Version 6 Macro Features for the PC

Jeff McDermott, SAS Institute Inc., Cary, NC
Bruce Tindall, SAS Institute Inc., Cary, NC

INTRODUCTION

The macro facility, part of base SAS® software, is a programming tool for extending and customizing SAS software and reducing the amount of text required to do common tasks. This paper focuses on macro language features new to Release 6.03 of base SAS software including macro variables, macro statements, macro functions, interactive macro windows, DATA step interfaces, and enhanced macro debugging tools.

MACRO VARIABLES

Macro variables include those you create and those created by the macro processor (automatic macro variables).

Support for many Version 5 macro variables has been added to Release 6.03 including

SYSBUFFR	SYSDSN	SYSMENV
SYSDATE	SYSENV	SYSSCP
SYSDAY	SYSINDEX	SYSTIME
SYSDEVIC	SYSJOBID	SYSVER

Support for macro variables new to the SAS System includes the following:

- SYSCMD contains the last command from the command line of a macro window that was not recognized by the SAS Display Manager System unless you make a direct assignment to the macro variable. You can use this variable to check for a user-defined command on the command line during the execution of a macro window. The %WINDOW statement is discussed in the following section, **MACRO STATEMENTS.**

```
%* The following is a SYSCMD example;
%if %upcase(&syscmd)=QUIT %then ....

%* You can nullify the value in SYSCMD with the following;
%let syscmd=;
```

- SYSERR contains the return code set by SAS procedures. You can use this variable to decide whether to execute DATA or PROC steps depending on whether an earlier step ran correctly or failed (for example, because of a missing data set). Values for SYSERR include

0	Execution completed successfully without warning messages.
1	Execution was cancelled by the user with a RUN CANCEL statement.
2	Execution was cancelled by the user with a control-break signal.
4	Execution completed successfully but with warning messages.
Values greater than 4	An error occurred. The number will indicate the type of error in future releases of SAS software.

For interactive procedures (DATASETS, PLOT, IML, and so on), SYSERR contains the highest return code that occurred during execution. The procedure as a whole may complete successfully, but if one or more RUN groups fail SYSERR is still greater than 4.

```
%* If the procedure runs successfully, SYSERR is 0;
%if &syserr = 0 %then %str(endsas;);

%* A control-break signal sets SYSERR to 2;
%if &syserr=2 %then %str(dm 'recall';);
```

- SYSINFO contains return code information provided by some SAS procedures. Values of SYSINFO are documented with the procedures that use it. Currently the COMPARE procedure makes use of SYSINFO by turning on different bits depending on the outcome of the comparison. The coded values are ordered and scaled to permit you to tell the degree to which the data sets differ. **Table 1** gives SYSINFO codes for PROC COMPARE.

Table 1 PROC COMPARE SYSINFO Codes

Code	Description
1	Data set LABEL= options differ
2	Data set TYPE= options differ
4	Variable has different informat
8	Variable has different format
16	Variable has different length
32	Variable has different label
64	Base DS has OBS not in compare DS
128	Compare DS has OBS not in base DS
256	Base DS has BY group not in compare DS
512	Compare DS has BY group not in base DS
1024	Base DS has variable not in compare DS
2048	Compare DS has variable not in base DS
4096	A value comparison was unequal
8192	Conflicting variable types
16384	BY variables do not match
32768	Fatal error: comparison not done

```
%* The following statements illustrate SYSINFO with PROC COMPARE;
data a b;
   x=1;
   output a b;
   output a;
   y=1; output a;
   x=2; output b;
run;
proc compare data=a comp=b out=c;
run;
%* Base DS has obs. not in compare DS-code 64;
%* A value comparison was unequal-code 4096;
%* Sysinfo=64+4096;
%* Writes value in print report title;
%put &sysinfo; /* writes value to the log */
title "Error code is &sysinfo";
proc print data=c;
run;
```

The value for SYSINFO is 4160 in the example above and 4096 + 64 = 4160, so the numbers that sum to equal SYSERR correspond to the errors that occurred in PROC

COMPARE. SYSINFO is reset to null or to a new value after each step boundary.

- SYSLAST gives the name of the most recently created SAS data set in the form *libref.datasetname*. SYSLAST is easier to use than SYSDSN because you can insert it directly into SAS code without changing it.

```
%* printing a data set name in a title;
%* inserting &SYSLAST after PROC PRINT;
libname a "c:\mydir\sugi";
data a.one;
   x=1;
run;
title "Printout of data set &syslast";
proc print data=&syslast;
run;
```

- SYSMSG contains a message you specify to be displayed in the message area (upper-left corner) of a macro window. The text is displayed only during the next displayed window after the assignment is made.

```
%* The following is an example using SYSMSG;
%if %upcase(&syscmd) ne QUIT %then
    %let sysmsg=Invalid response, &syscmd;
%display window1;
```

MACRO STATEMENTS

The following Version 5 macro statements are supported under Release 6.03:

```
%*comment       %END              %INPUT       macro invocation
%DO             %GLOBAL           %label       %MACRO
iterative %DO   %GOTO|%GO TO      %LET         %MEND
%DO %UNTIL      %IF-%THEN/%ELSE   %LOCAL       %PUT
%DO %WHILE
```

Support for macro statements new to the SAS System includes the following:

- The %WINDOW statement defines a macro window. Within the %WINDOW statement there are options that enable you to specify the background color and size; the color, location, and attributes of different fields; user-defined keys; and different window groups.

- The %DISPLAY statement displays a macro window. It includes the options NOINPUT, BLANK, and BELL.

Both %WINDOW and %DISPLAY statements can appear in macro definitions as well as in open code (outside any macro definition). The WINDOW and DISPLAY statements have been available in the DATA step since the SAS System was first released for personal computers. In Release 6.03 the DATA step WINDOW statement has added support for several new options. The complementary new macro %WINDOW options are as follows:

DISPLAY=YES|NO
: determines whether the macro processor displays the characters typed as you enter them in a macro variable field. The default value is YES. This option can be used with variable fields because text fields are always displayed. This option is useful for entering a secret password field. It should not be confused with the DISPLAY or %DISPLAY statements.

REQUIRED=YES|NO
: determines whether a value must be entered for a macro variable before continuing processing. You cannot enter a null value in a required field, but you can get around it during execution by entering an appropriate command on the command line to exit that window or to display another window. To do so you define the command (for example, QUIT in the second example below) and then check for its use with the SYSCMD variable. The default value of the REQUIRED= option is NO.

```
%* The following shows a macro window outside a macro definition;
%* It demonstrates the %WINDOW options REQUIRED= and DISPLAY=;
%window example color=red
   #4 @7 'You must enter a character here to continue'
      +1 reqvar 1 a=underline required=yes
   #5 @7 'You may simply hit <CR> to continue this time'
      +1 noreqvar 1 a=underline required=no
   #8 @7 'Nothing is displayed as you type in this field'
      +1 hideit 7 a=underline display=no
   #9 @7 'Characters appear as you type in this field'
      +1 flauntit 7 a=underline display=yes;
%display example;
```

```
%* The following example illustrates a macro window
   within a macro definition, (it could go in open code);
%* It uses %WINDOW, %DISPLAY, SYSCMD, and SYSMSG;
%macro macroex;
%window example
   group=prompt
      #3 @2 'Please enter "1" or "2" to display a message:'
         +1 choose 1 a=underline
      #5 @2 'Or enter "QUIT" on the command line to stop.'
   group=one
      #11 @5 'This is groop #1' color=yellow
   group=two
      #13 @5 'This is group #2' color=green a=highlight;
   %let display=; /* initialize variables */
   %let syscmd=;
   %let sysmsg=;
   %display example.prompt;
   %do %while (%upcase(&syscmd) ne QUIT);
      %if &choose=1 %then %do;
         %display example.one noinput;
         %let choose=;  /* reinitialize */
      %end; /* %if %then %do */
      %else %if &choose=2 %then %do;
         %display example.two noinput;
         %let choose=;
      %end; /* %else %if %then %do */
      %else %do;
         %let sysmsg=Invalid response, &syscmd..;
         %let choose=;
         %end; /* %else %do */
         %let syscmd=;
         %display example.prompt;
   %end;  /* %do %while */
   %mend macroex;
%macroex;
```

MACRO FUNCTIONS

The following Version 5 macro functions are supported under Release 6.03:

```
%BQUOTE      %NRQUOTE      %STR
%EVAL        %NRSTR        %SUBSTR
%INDEX       %QUOTE        %UNQUOTE
%LENGTH      %SCAN         %UPCASE
%NRBQUOTE
```

DATA STEP INTERFACES

The DATA step interfaces SYMGET and SYMPUT allow you to retrieve macro variables and to create new macro variables, respectively, during DATA step execution rather than when the DATA step is being constructed. Both are available in Versions 5 and 6.

OTHER FEATURES

More features supported in the Version 6 macro language include the following:

- %NRBQUOTE and %BQUOTE now quote mnemonic operators (AND, NOT, and so on). This means that you no longer have to use both %QUOTE and %BQUOTE (or %NRQUOTE and %NRBQUOTE) around an expression to make sure that mnemonics are quoted.
- The length of a macro variable can be up to 32,767 characters.

```
%* The following example demonstrates
   %BQUOTE quoting the mnemonic operator AND;
%* Version 5 would interpret the string AND as mnemonic;
%* It would then issue an error message for the %IF statement;
%macro macroex2;
   %let bq=black and white;
   %if %bquote(&bq)$c¬= %then %put %bquote(&bq);
      %else %put value of bq is null;
   %mend macroex2;
%macroex2;


%* The following example demonstrates
   %NRBQUOTE quoting &, %, and mnemonic operators;
%* Version 5 would flag the %IF statement because of the mnemonics;
%macro macroex3;
   %let nrbq=try % and & not ne or le;
   %if %nrbquote(&nrbq)$c¬= %then %put %nrbquote(&nrbq);
      %else %put value of nrbq is null;
   %mend macroex3;
%macroex3;
```

CURRENT DIFFERENCES BETWEEN VERSION 5 AND RELEASE 6.03 MACRO LANGUAGE

A few differences exist between current releases of Version 5 and Release 6.03.

- Release 6.03 does not support the DQUOTE option but behaves as if the option is turned on. A future release will support it with the default set to ON.
- The default %SCAN delimiter list (used if you do not specify your own list) differs between Version 5 and Version 6. In Version 5 the list contains the cent sign, but in Version 6 it does not; in Version 6 the list contains the greater-than and backslash symbols, but in Version 5 it does not.
- Only two values (S and D) of SYSMENV are supported in Release 6.03. The value P (supported in Version 5) will be added in a future release.
- The autocall facility and remaining Version 5 features will also be added in a later release.

FEATURES FOR DEBUGGING YOUR MACROS

In Release 6.03, the SAS macro facility has some new debugging features that will help you develop your macro applications more easily and more quickly.

The debugging features in Release 6.03 are the MPRINT option, the SYMBOLGEN option, and the new MTRACE option. In future releases there will be further debugging tools, as well as enhancements to these.

The MPRINT Option

You may be familiar with the MPRINT and SYMBOLGEN options from Version 5. The output from these options is easier to read in Release 6.03.

The simple SAS macro in this example generates two TITLE statements containing today's day and date:

```
option mprint;
%macro ttoday;
   title "Today is &sysday,";
   title2 "&sysdate";
   %mend ttoday;
%ttoday
```

The output from this SAS program looks like this:

```
MPRINT(TTODAY):   TITLE "Today is Monday,";
MPRINT(TTODAY):   TITLE2 "28MAR88";
```

When the MPRINT option is in effect, each SAS statement that the macro generates appears on a separate line of the SAS log. Each line begins with the word "MPRINT," followed by the name of the macro that generated the statement, followed by the SAS statement itself.

Each statement is printed after all macro calls, macro variable references, and macro functions are resolved. So, in this example, you do not see the macro variable names SYSDAY and SYSDATE in the output; instead, you see the actual values to which they resolved. That is, you see the actual statements as the SAS System eventually executed them.

Using the MPRINT Option to Debug a Macro

This is a macro that contains an error that the MPRINT option can help you find. The macro is supposed to generate a DATA step using the data set name you pass to it (or _NULL_ if you do not pass it a data set name).

```
%macro abc(dsn);
   data
      %if &dsn=
          %then _null_;
      %else &dsn;
      put 'This is a data step';
   run;
   %mend;
```

But when you invoke the macro by entering

```
%abc(xxx.yyy)
```

these messages appear in the log:

```
ERROR: Syntax error detected.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: The DATA statement used 5.00 seconds.
```

If you turn on the MPRINT option and invoke the macro again, you see this in the log:

```
ERROR: Syntax error detected.
MPRINT(ABC):   DATA XXX.YYY PUT 'This is a data step';
MPRINT(ABC):   RUN;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: The DATA statement used 1.00 seconds.
```

Instead of the three statements you expect, you see only two; the DATA and PUT statement appear as one statement because there is no semicolon ending the DATA statement.

To correct this error, change the %IF statement in the macro so that its %THEN and %ELSE clauses generate a semicolon in addition to the data set name:

```
%if &dsn=
    %then %str(_null_;);
%else %str(&dsn;);
```

The SYMBOLGEN Option

In addition to being easier to read, the output from the SYMBOLGEN option contains more information in Release 6.03 than in Version 5.

This SAS program uses the SYMBOLGEN option:

```
option symbolgen;
%let n=3;
%let aaa3=xyz;
%put The value is &&aaa&n;
```

The program produces this output:

```
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable N resolves to 3
SYMBOLGEN:  Macro variable AAA3 resolves to xyz
The value is xyz
```

Whenever the macro facility successfully resolves a macro variable, the SYMBOLGEN option prints a line in the SAS log starting with the word SYMBOLGEN and giving both the variable's name and its value.

This option also prints a message when it resolves a double ampersand into a single ampersand, which helps you to follow the resolution of indirect macro variable references like the one in the example above.

If the macro variable's value contains characters that have been quoted by one of the macro quoting functions such as %STR, the SYMBOLGEN option prints those characters as if they were not quoted, but it also prints an informatory message stating that some of the characters in the value were quoted.

In the next example, the semicolons and the equals sign that appear in the output from the SYMBOLGEN option are not stored in the macro variable as real semicolons and equals signs, but as quoted characters. The SYMBOLGEN option, however, prints them as if they had not been quoted so that you can read them. The SAS statements

```
option symbolgen;
%let xyz=%str(data a; x=1; run;);
%put &xyz;
```

produce this output:

```
SYMBOLGEN:  Macro variable XYZ resolves to data a; x=1; run;
SYMBOLGEN:  Some characters in the above value which were subject
            to macro quoting have been unquoted for printing.
data a; x=1; run;
```

Using the SYMBOLGEN Option to Debug a Macro

Here is a macro that contains an error that the SYMBOLGEN option can help you find. The macro prints a message telling whether its two arguments, when concatenated, become the string "hi there" (with a blank between the two words).

```
%macro concat(a,b);
   %let c=&a&b;
   %if &c=hi there
       %then %put It says hi there;
   %else %put It does not say hi there;
   %mend concat;
```

If you invoke the macro by entering

```
%concat(hi, there)
```

it prints the message

```
It does not say hi there
```

even though there seems to be a blank at the beginning of the word "there" in the invocation, which should cause a blank to occur between "hi" and "there" in the concatenated value — or should it?

Turning on the SYMBOLGEN option and invoking the macro causes this output:

```
SYMBOLGEN:  Macro variable A resolves to hi
SYMBOLGEN:  Macro variable B resolves to there
SYMBOLGEN:  Macro variable C resolves to hithere
It does not say hi there
```

There is not a blank in the concatenated value after all. This is correct because leading and trailing blanks in the arguments to a macro are ignored unless they are made significant by a macro quoting function such as %STR.

The following macro invocation gives the desired result:

```
%concat(hi,%str( there))
```

The MTRACE Option

The new debugging feature in the Version 6 macro facility is the MTRACE option, which tells you the same things the MLOGIC option tells you in Version 5, but the MTRACE option also tells you much more.

The MTRACE option prints a message in the SAS log every time a macro programming statement (%LET, %PUT, %IF, %GOTO, %DO, %INCLUDE, %GLOBAL, %LOCAL, %WINDOW, or %DISPLAY) is executed, as well as whenever a macro begins or ends execution.

Here is a simple macro named DATASTEP that generates the first statement of a SAS DATA step. If you do not pass it a data set name, it generates the statement

```
data _null_;
```

If you do pass it a data set name, it uses that name in the DATA statement instead of _NULL_:

```
option mtrace;
%macro datastep(dsn);
   data
      %if &dsn=
          %then %str(_null_;);
      %else %str(&dsn;);
   %mend;
%datastep(work.xyz)
```

When you run this macro with the MTRACE option turned on, the following output appears in the SAS log:

```
MTRACE(DATASTEP):  Beginning execution.
MTRACE(DATASTEP):  Parameter DSN has value work.xyz
MTRACE(DATASTEP):  %IF condition &dsn = is FALSE
MTRACE(DATASTEP):  Ending execution.
NOTE: The data set WORK.XYZ has 1 observations and 0 variables.
```

First, the MTRACE option prints a message saying that the macro named DATASTEP is beginning.

Next, when the %IF statement is executed, the MTRACE option shows you the condition in the %IF statement (&DSN=<blank>, in this case) and tells you whether the condition is true or false. In this example, the condition is false, so the macro skips the %THEN clause and goes on to the %ELSE clause.

Notice that the MTRACE option prints the condition of the %IF statement exactly as you code it in the macro. It does not resolve any macro variable references that occur in the condition. Seeing it printed this way helps you identify which %IF statement is being executed. This can be helpful in a large macro that has many %IF statements. If you need to see the values of the macro variables, you can use the SYMBOLGEN option along with the MTRACE option.

Finally, when the macro reaches the %MEND statement, the MTRACE option tells you that the macro is ending.

Notice that the MTRACE option does not show you the SAS statements that the macro generates. You can use the MPRINT option to see them.

Examples of all the different SAS macro statements for which the MTRACE option prints a message, along with the output that the MTRACE option prints when the statement is executed, are in the appendix to this paper.

A Bug in the MTRACE Option and How to Avoid It

There is a bug (which will be fixed in the next release) that causes the MTRACE option to print "garbage" characters where it should print the word TRUE or FALSE in its output for the %IF statement.

To prevent this from happening, place the following code at the beginning of your SAS program, before any DATA or PROC step:

```
option mtrace;
%macro abc;
%mend abc;
%abc
option nomtrace;
```

After running this, you can turn the MTRACE option back on when you actually need it, and its output for %IF statements will then be correct.

Using the MTRACE Option to Debug a Macro

Suppose you want to write a macro that splits a character string into individual words and then reports how many times a given character occurs in each word. Also suppose that a friend of yours has already written a macro that returns the number of times a given character occurs in a character string.

Your macro (call it STRCOUNT) can split the string into words and pass each word to your friend's macro (call it COUNT) to find out how many times the character occurs in it. Then your macro can print a line telling how many times the character appears in that word.

```
%include "countmac.sas";  /* Include your friend's COUNT macro. */
%macro strcount(string,char);
   %do i=1 %to 5;  /* Assume no more than 5 words in string. */
      %let word=%scan(&string,&i);
      %let c=%count(&word,&char);
      %put There are &c &char%str(%')s in &word;
      %end;
   %mend;
```

This looks as if it should work, but when you invoke your macro to find out how many y's are in each word of a phrase, as in

```
%strcount(ontogeny recapitulates phylogeny , y)
```

you find only one line of output on the log where you should find three (one for each word):

```
There are 1 y's in ontogeny
```

If you turn on the MTRACE option and invoke your macro again, you see the following output:

```
MTRACE(STRCOUNT):  Beginning execution.
MTRACE(STRCOUNT):  Parameter STRING has value ontogeny recapitulates
                   phylogeny
MTRACE(STRCOUNT):  Parameter CHAR has value y
MTRACE(STRCOUNT):  %DO loop beginning; index variable I; start value
                   is 1; stop value is 5; by value is 1.
MTRACE(STRCOUNT):  %LET (variable name is WORD)
MTRACE(STRCOUNT):  %LET (variable name is C)
MTRACE(COUNT):.  Beginning execution.

/* Here there are several lines of output from the COUNT macro
we will ignore for the moment. */

MTRACE(COUNT):  Ending execution.
MTRACE(STRCOUNT):  %PUT There are &c &char's in &word
There are 1 y's in ontogeny
MTRACE(STRCOUNT):  %DO loop index variable I is now 10; loop will
                   not iterate again.
MTRACE(STRCOUNT):  Ending execution.
```

Notice that when the %DO loop in your macro begins executing, the loop control variable, I, is set to 1 and is supposed to increase by 1 each time the loop iterates. But after the loop has iterated one time, the value of I is 10 according to the MTRACE output. So the loop stops after only one iteration.

Your code and the MTRACE output both show that your macro did not change the value of the variable I during that iteration of the %DO loop. It must have been the other macro, COUNT.

A look at the MTRACE output from COUNT that was omitted above confirms this assumption. One of the MTRACE lines from COUNT contains the following:

```
MTRACE(COUNT):  %DO loop beginning; index variable I; start value
                is 1; stop value is 8; by value is 1.
```

The author of the COUNT macro forgot to make the loop control variable, I, a local variable by naming it in a %LOCAL statement. (Variables that you use internally, within a single macro, should always be local variables to avoid interference with other macros' similarly named variables.)

Therefore, when COUNT changed the value of the variable I, it was the same variable that your macro, STRCOUNT, was also using. By the time macro COUNT returned to macro STRCOUNT, the variable had been increased to 10, which caused STRCOUNT's %DO loop to end prematurely.

CONCLUSION

We hope that you find these new macro features useful. As always, and as with all features of the SAS System, if you have any suggestions for improving them or if you have any suggestions for completely new features, please send them to the Technical Support department at SAS Institute.

APPENDIX

Here are examples of all the different SAS macro statements for which the MTRACE option prints a message, each followed by the output that the MTRACE option prints when the statement is executed. The "ABC" in each message (after the "MTRACE") represents the name of the currently executing macro.

- `%macro abc(x,y);`

  Assuming that you call the macro as

  ```
  %ABC(hi,BYE)
  ```

  the following messages are printed:

  ```
  MTRACE(ABC):  Beginning execution.
  MTRACE(ABC):  Parameter X has value hi
  MTRACE(ABC):  Parameter Y has value BYE
  ```

  The values of the parameters are fully resolved before being printed and are not converted to uppercase.

- `%mend abc;`

  ```
  MTRACE(ABC): Ending execution.
  ```

- `%if &A=hello`

  Assuming that the value of &A is Hello, the following message is printed:

  ```
  MTRACE(ABC):  %IF condition &A = Hello is TRUE
  ```

  Assuming that the value of &A is Goodbye, the following message is printed:

  ```
  MTRACE(ABC):  %IF condition &A = Hello is FALSE
  ```

  The condition of the %IF statement is printed unresolved and is not converted to uppercase.

  ```
  %put Today is &sysdate;
  MTRACE(ABC): %PUT Today is &sysdate
  ```

  The string in the %PUT statement is printed unresolved and is not converted to uppercase.

- `%let uvw=xyz;`

  ```
  MTRACE(ABC): %LET (variable name is UVW)

  %let &uvw=A Value;
  MTRACE(ABC): %LET (variable name is XYZ)
  ```

  The variable name is resolved, if necessary, before being printed so that you see the name of the actual variable being created or assigned.

- `%include &fn / source2;`

  ```
  MTRACE(ABC): %INCLUDE file is &FN
  ```

  The fileref or file name is printed unresolved.

- `%global a b c;`

  ```
  MTRACE(ABC): %GLOBAL A B C;

  %global &vars;
  ```

  Assuming the value of VARS is XX YY ZZ, the following message is printed:

  ```
  MTRACE(ABC):  %GLOBAL XX YY ZZ;
  ```

  The list of variables is resolved, if necessary, before being printed. The %LOCAL and %INPUT statements are treated similarly to %GLOBAL.

- `%goto &&xyz&i;`

  Assuming the value of I is 3 and the value of XYZ3 is TGT, the following message is printed:

  ```
  MTRACE(ABC):  %GOTO &&XYZ&I (label resolves to TGT)
  ```

  The target of the %GOTO is printed in both unresolved and resolved forms so that you can see which %GOTO statement is being executed and which label it will branch to.

- `%do i=1 %to 10; ...`
  `%end;`

  ```
  MTRACE(ABC): %DO loop beginning; index variable I; start value
               is 1; stop value is 2; by value is 1.
  MTRACE(ABC): %DO loop index variable I is now 2; loop will
               iterate again. <etc.>
  MTRACE(ABC): %DO loop index variable I is now 5; loop will
               not iterate again.
  ```

  The name of the index variable is resolved.

- `%do %while(&j>0); ...`
  `%end;`

  ```
  MTRACE(ABC): %DO %WHILE(&j>0) loop beginning; condition is
               TRUE.
  MTRACE(ABC): %DO %WHILE(&j>0) condition is TRUE; loop will
               iterate again. <etc.>
  MTRACE(ABC): %DO %WHILE(&j>0) condition is FALSE; loop will
               not iterate again.
  ```

  or

  ```
  MTRACE(ABC): %DO %WHILE(&j<0) loop beginning; condition is
               FALSE. Loop will not be executed.
  ```

  The condition is printed unresolved and is not converted to uppercase.

- `%do %until(&j>0); ...`
  `%end;`

  ```
  MTRACE(ABC): %DO %UNTIL(&j>0) loop beginning.
  MTRACE(ABC): %DO %UNTIL(&j>0) condition is FALSE; loop will
               iterate again. <etc.>
  MTRACE(ABC): %DO %UNTIL(&j>0) condition is TRUE; loop will
               not iterate again.
  ```

  The condition is printed unresolved and is not converted to uppercase.

- `%window abc #1 #1 This is a macro window ... ;`

  ```
  MTRACE(ABC): %WINDOW statement beginning.
  ```

  In future releases of the SAS System, the MTRACE option will show more information about the %WINDOW statement.

- `%display abc;`

  ```
  MTRACE(ABC): %DISPLAY statement beginning.
  ```

  In future releases of the SAS System, the MTRACE option will show more information about the %DISPLAY statement.