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**
- **SYSDATE**
- **SYSDAY**
- **SYSDEVIC**
- **SYSDSN**
- **SYSENV**
- **SYSINDEX**
- **SYSJOBID**
- **SYSMENV**
- **SYSSCP**
- **SYSTIKE**
- **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.

```sas
%* The following is a SYSCHD example;
%if upcase(syscmd)='QUIT'
 ...
%* You can nullify the value in SYSCHD 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

```sas
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 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.

- **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.

```sas
Table 1 PROC COMPARE SYSINFO Codes

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

For interactive procedures, 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.
The following Version 5 macro statements are supported under Release 6.03:

### MACRO STATEMENTS

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

- **%WINOOW**: Defines a macro window. The text is displayed only during the next displayed window after the assignment is made.
- **%DISPLAY**: Displays a macro window. It includes the options NOINPUT, BLANK, and BELL.

Support for macro statements new to the SAS System includes:

- 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 6.0 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.

### MACRO FUNCTIONS

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

- **$QUOTE**
- **$SQUOTE**
- **$STR**
- **$TOKEN**
- **$LENGTH**
- **$SCAN**
- **$NRBQUOTE**
- **$BOUOTE**
- **$SCAN**
- **$SUBSTR**
- **$UPCASE**
- **$UNQUOTE**
- **$LEVAL**
- **$JINDEX**
- **$RINDEX**
- **$RINDEX**
- **$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:

• `%NRQUOTE 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 quoting the mnemonic operator AND;

  %macro macroex2;
  data _null_;
  if %bquote(AND)=" THEN PUT %bquote(AND);
  else PUT value of AND is null;
  %end;
  %macroend;
  %macroex2;

The output from this SAS program looks like this:

MPRINT(TTODAY): TITLE "Today is Monday,":
MPRINT(TTODAY): TITLE2 "SYSDATE";
NOTE: The SAS System stopped processing this step because of errors.
NOTE: The DATA statement used 5.00 seconds.

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 Ssysday,";
  title2 "Ssysdate";
  %mend ttoday;
  %ttoday;

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 _null_;
  if dsn= %then _null_;
  %else dsn;
  put 'This is a data step';
  run;
  %mend;
  %abc(xxx, yyy);

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 1.60 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.YEE 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 idsn=
then put(STR(_NULL_));
%else put(STR(idsn));
```

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 x=3;
let aaa=xyz;
put The value is &aaa;
```

The program produces this output:

```
SYMBOLGEN: Macro variable AAA 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 x='xyz'=str(data a; x=1; run);
put &x=
```

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.
```

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 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_:

```
%macro datastep(dsn);
    data hf
        %if idsn=
        then put(STR(_NULL_));
        %else put(STR(idsn));
%end;
%datastep( work, xyz)
```
When you run this macro with the MTRACE option turned on, the following output appears in the SAS log:

```
MTRACE(STRCOUNT): Beginning execution.
MTRACE(STRCOUNT): Parameter STRING has value ontogeny recapitulates phylogeny
MTRACE(STRCOUNT): Parameter CHAR has value y
MTRACE(STRCOUNT): %LET (variable name is WORD)
MTRACE(STRCOUNT): %LET (variable name is CHAR)
MTRACE(STRCOUNT): Beginning execution.
```

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 CHAR)
MTRACE(STRCOUNT): Beginning 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 5; 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.

• \%end 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.

• global abc;

MTRACE(ABC): \%GLOBAL ABC;  

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 \%keyword;

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

MTRACE(ABC): \%GOTO XYZ TGT (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 \%until{\%j>0}; ... \%end;

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

or

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

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

• \%do \%while{\%j<0}; ... \%end;

MTRACE(ABC): \%DO \%WHILE(\%j<0) loop beginning; condition is FALSE. Loop will not iterate again.

or

MTRACE(ABC): \%DO \%WHILE(\%j>0) loop beginning; condition is TRUE; loop will iterate again. etc.

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

• include \%include / source;

MTRACE(ABC): \%INCLUDE file is \%INCLUDE

The file ref or file name is printed unresolved.

• \%global a b c;

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

\%global a b c;

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.

SAS is a registered trademark of SAS Institute Inc., Cary, NC.