

Macro Facility Enhancements for Release 6.09E and Release 6.11

Jeff Polzin, SAS Institute Inc., Cary, NC
 Susan O'Connor, SAS Institute Inc., Cary, NC

ABSTRACT

This paper summarizes the changes and enhancements which have been added to the macro facility of the SAS® System in Release 6.09E and in Release 6.11. These macro enhancements are experimental. Examples of the enhancements will be given in the paper. Additionally, sample macro examples will be made available for access using SAS Institute's Internet gateway.

INTRODUCTION

A new macro statement, %SYSCALL, provides the ability to access SAS CALL routines and SAS/TOOLKIT® user-written CALL routines. There are three new macro functions. The %SYSEVALF macro function does floating point evaluation. %SYSFUNC and %QSYSFUNC macro functions provide macro facility interfaces to SAS functions and to functions written with SAS/TOOLKIT.

Base SAS software users will be able to use a set of functions for DATA step access, external file access, and system functions, similar to the SAS/AF® software Screen Control Language functions, through the macro facility using the new %SYSFUNC and %QSYSFUNC macro functions. The newly accessible functions are listed in an appendix.

For macro debugging, the %PUT macro statement has been enhanced to write macro variable names, scope, and values to the SAS log. Using the %PUT statement with one of the keywords _ALL_, _AUTOMATIC_, _USER_, _LOCAL_ or _GLOBAL_ give macro programmers the ability to list relevant information for debugging macro variables.

The ability to write macro generated text to an external file in addition writing it to the SAS log with OPTION MPRINT was ranked highly for macro on recent SASWARE ballots. This macro debugging feature has been added. Users can now write the macro generated text to an external file to aid in macro debugging.

There is a new SAS Institute supplied automatic macro variable. SYSSCP adds more detail to the SYSSCP automatic macro variable which gives the operating system you are using.

%SYSCALL STATEMENT

%SYSCALL *call-routine* <(call-routine argument list)>;

Where..	Is Type ...	And represents ...
<i>call-routine</i>		names a SAS System or user-written CALL routine.
<i>call-routine argument list</i>	N or C	a list of macro variable names separated by commas.
Details		

The %SYSCALL macro statement can access SAS System CALL routines or user-written CALL routines created with SAS/TOOLKIT. The %SYSCALL macro statement can be inside a macro definition or in open code.

CALL routine invocation passes the value of a macro variable by reference. Call by reference means that the CALL routine can modify the value of the macro variable. Each value of each macro variable argument is fetched and passed to the CALL routine. Upon return from the CALL routine, the values for each argument are written back to their respective macro variables.

This behavior is similar to the behavior of the %SUPERQ macro function. Like %SUPERQ, the macro variable in the argument list must be the names of macro variables with no leading ampersands or macro expressions that produce the names of macro variables with no leading ampersands. But unlike %SUPERQ, the arguments to %SYSCALL are not quoted. Note that this syntax of no leading ampersands is not the traditional macro syntax. No result value is produced upon return from the execution of a %SYSCALL macro statement; the values of macro variables that were passed by %SYSCALL are altered.

OPTION MLOGIC will issue an informational message for the %SYSCALL statement during macro execution, as it does for other macro statements.

The LABEL, VNAME, SYMPUT, and EXECUTE CALL routines are not accessible using the %SYSCALL macro statement.

%SYSCALL Example Using the RANUNI Call Routine

The following macro statements illustrate the %SYSCALL statement

```
%let a = 123456;
%let b = .;
%syscall ranuni (a,b);
%put &a &b;
```

In this example, the value of the macro variables A and B are character strings, 123456 and '.' respectively.

The macro statement %SYSCALL RANUNI(A,B) invokes the SAS CALL RANUNI routine. The SAS CALL RANUNI routine takes two arguments: a seed with an integer value and an X variate that is numeric.

In this example, the %SYSCALL macro statement converts the A and B macro variables' values from character strings 123456 and '.' to the data types as required for each argument to the RANUNI CALL routine. The character string 123456 which is the seed is converted to an integer and character string period is converted to a numeric missing value for the x variate. A numeric value is stored in floating point format. The CALL RANUNI routine executes and returns an integer value for the seed and a numeric value for the x variate to the %SYSCALL macro routine. %SYSCALL converts the newly returned arguments back from integers and numerics to character strings, the standard macro data type. The macro variable values are updated with these new character strings. In this example, the macro variable A is updated

Applications Development

to the character string 1587033266 and the macro variable B has the character string value of 0.739019954.

This %PUT statement

```
%put &a &b;
```

writes these macro variable values for macro variables A and B to the SAS log

```
1587033266 0.739019954
```

If there are problems converting the macro variable character string to the appropriate data type for the CALL routine, the %SYSCALL macro statement will issue errors or warnings.

For example, the following program:

```
%let a1 = 1.23;  
%let b1 = .8885;  
%syscall ranuni(a1,b1);  
%put &a1 &b1;
```

will have character strings for A1 and B1 macro variable values of 1.23 and .8885.

The %SYSCALL macro statement will be unable to convert the A1 macro variable character string 1.23 to an integer value as required for the first argument, the seed, to CALL RANUNI and this warning message will be issued to the log:

```
WARNING: Argument 1 to function RANUNI  
referenced by the %SYSFUNC or %QSYSFUNC macro  
function is missing or out of range.
```

The value of macro variables A1 and B1 will remain unaltered and the %PUT statement will generate unaltered strings 1.23 and .8885.

%SYSEVALF FLOATING POINT EVALUATION FUNCTION

%SYSEVALF(*expression*<*specification*>)

Where ...	Is type ...	And represents ...
<i>expression</i>	N	may be a floating point number.
<i>specification</i>	C	may be the following conversion operation specifications: BOOLEAN, CEIL, FLOOR, INTEGER, TRUNCATE.

Details

The new %SYSEVALF macro function evaluates arithmetic and logical expressions in the macro language performing floating point arithmetic. The result of the %SYSEVALF function arithmetic evaluation is formatted using BEST12. format. The result of the %SYSEVALF function arithmetic evaluation can be optionally converted by specifications of Boolean, ceil, floor, integer, or truncate functions. The %SYSEVALF macro function can be used inside a macro definition or in open code.

The %SYSEVALF macro function differs from the traditional

%EVAL macro function in its evaluation of floating point numbers. The traditional way to evaluate an expression in the macro facility is the explicit or implied use of the %EVAL function. All features in the macro language that evaluate expressions contain an implied use of the %EVAL function, for example, the %IF statement or the %DO %UNTIL statement. The %EVAL function performs integer arithmetic. Therefore, with %EVAL, calculations on fractions are not allowed, and the division operation resulting in a fraction is truncated to an integer. However, explicit use of the %SYSEVALF function will allow floating point arithmetic.

Simple Examples Using %SYSEVALF Macro

Suppose the following macro variables are defined

```
%let a = 100;  
%let b = 200;  
%let c = 1.597;
```

Then the following statement writes the value 300 to the SAS log:

```
%put %eval(&a + &b);
```

The character strings A and B have been converted to integers, the integer operation performed, and the resulting integer is converted to a character string. However, conversions of the character string value of the macro variable C from 1.597 to an integer value would fail. The following macro expression:

```
%put %eval (&a + &c);
```

would generated an error message since the value of C, the character string 1.597, could not be converted to an integer. The following message would be written to the log:

```
ERROR: A character operand was found in the %EVAL  
function or %IF condition where a numeric  
operand is required. The condition was:  
100 + 1.597
```

By using the %SYSEVALF function instead of the %EVAL function the following expression would write 101.597 to the log:

```
%put %sysevalf(&a + &c);
```

This evaluation with the %SYSEVALF macro function is successful because the character values for both macro variables A and C can be converted to numeric values. The %SYSEVALF macro function does floating point arithmetic and converts the result back to a character string 101.597.

%SYSEVALF and Optional Result Formatting Specification

Optional specifications to the %SYSEVALF function convert the results accordingly. For example,

```
%put %sysevalf(&a + &c, boolean);
%put %sysevalf(&a + &c, ceil);
%put %sysevalf(&a + &c, floor);
%put %sysevalf(&a + &c, integer);
%put %sysevalf(&a + &c, truncate);
```

would write 1, 102, 101, 101, and 101 respectively to the log.

%SYSFUNC AND %QSYSFUNC MACRO FUNCTIONS

%SYSFUNC(function(function argument list)<format>)

%QSYSFUNC(function(function argument list)<format>)

Where ...	Is type ...	And represents ...
function		is the name of the SAS function or user-written function to be executed.
function argument list	N or C	is a list of arguments to the referenced function.
format	C	optionally specifies a SAS or user-defined format or user-written format to be used to format the result of the function referenced.

Details

The %SYSFUNC and %QSYSFUNC macro functions can be used to access SAS System functions or user-written functions generated with SAS/TOOLKIT. The %SYSFUNC and %QSYSFUNC macro functions can be used inside a macro definition or in open code.

These additional macro functions provide support for constructs that users of the macro facility may have had difficulty overcoming in the past. For example, the new macro functions can be used to obtain the option names and values for SAS host, base, and graphics options. They can also be used to open SAS DATA sets, to obtain attribute information, to close SAS DATA sets, and to read and write to external files.

With %SYSFUNC and %QSYSFUNC, each argument to the referenced SAS function is evaluated and converted to the appropriate data type, numeric or character, as required by the referenced function. If necessary, an implied %SYSEVALF function will convert the macro character string to a floating point number. The SAS or user-written function executes and returns the results to the macro facility. The returned result of the referenced function is converted to a character string. Numeric results will be converted to a character string using the BEST12. format. Character function results are taken as is; no formatting or translation is necessary.

All results from functions accessed by either the %SYSFUNC or %QSYSFUNC macros can be formatted using one of the SAS System formats, any user-defined format generated by PROC FORMAT, or any user-written format created with SAS/TOOLKIT. This optional second parameter to these SAS functions specifies a format.

If an error condition has occurred, the appropriate return value is generated based on whether the result of the function is numeric

or character. Numeric error values are missing and converted to a missing, "." character string. Character error values are null and converted to a null, "", character string.

The %QSYSFUNC macro function is identical to the %SYSFUNC macro function except that the result of the function is quoted.

All SAS functions are accessible with %SYSFUNC or %QSYSFUNC except: DIM, LAG, DIF, HBOUND, LBOUND, PUT, INPUT, SYMGET, and RESOLVE.

EXAMPLES OF %SYSFUNC AND %QSYSFUNC

The %SYSFUNC and %QSYSFUNC macro functions allow users to access SAS functions not previously available through a simple interface with the macro facility. This section contains simple examples of macro using a variety of system functions to perform specific operations that required complex solutions in the past.

Embedding Formatted DATE/TIME Values into TITLE Example

The macro facility readily provides the ability to produce the date on which a SAS session began to execute. For example, the automatic macro variable SYSDATE will produce a title with the date on which your SAS session started executing in DATE6. or DATE7. format.

```
title "On &sysdate the SAS Session Began";
```

would produce the title

On 10MAR96 the SAS Session Began

If you wanted the current date, not the date on which your session began, the following macro statement:

```
%sysfunc(date());
```

would execute the SAS date function and produce the current SAS date value.

To produce a title with the current date formatted into a form such as March 10, 1996, use the DATE function and the WORDDATE. format.

```
title "%sysfunc(date(),worddate.)
Is the Current Date";
```

would produce the title

March 10, 1996 Is the Current Date

The automatic macro variable SYSTIME produces the time the SAS program started executing in TIME5. format, as in 14:17. But if you want the current time, the following macro code would write the current time to the log formatted as hh:mm:ss:

```
%let time = %sysfunc(time());
%put %sysfunc(putn(&time, time.));
```

Applications Development

The first %SYSFUNC function executes the TIME function that returns the current time of day as a SAS time value such as 53526.08. %SYSFUNC converts the SAS time value to a character string as the value of the macro variable TIME. The %PUT statement will write to the log the character string that is returned by the second %SYSFUNC function. In this case, the %SYSFUNC converts the character value of the macro variable TIME to a numeric value. The system function PUTN returns the formatted value of the numeric, using the numeric format TIME. as a character value that the %PUT statement writes to the log as the current time, for example, 14:52:06.

TRANSLATE Function Example

Using the %SYSFUNC function, a user can interface with the TRANSLATE function. If you wish to translate N's to P's in an arbitrary string using a macro (e.g. V01N01 should be translated to V01P01), you could use a more complicated algorithm writing a sophisticated macro considering special characters, such as a -, or you could simply use the %SYSFUNC function to access TRANSLATE function.

```
%let string1 =  
  %sysfunc(translate(&string1,P, N));
```

This simple statement would convert the value of the macro variable STRING1 from V01N01 to V01P01. It could also convert more complex values for STRING1, such as V01N01-V01N10 to V01P01-V01P10.

Example to Determine the Existence of an External File

To easily determine the existence of an external file, two system functions can be used, FEXIST or FILEEXIST. Select the appropriate system function based on the manner in which the external file reference is to be used.

If the external file reference uses a SAS fileref use the FEXIST system function.

```
%macro usage1;  
  %if %sysfunc(fexist(myref)) %then  
    %put External file for filref does exist.  
  %else  
    %put External file for filref does NOT exist.  
%mend usage1;
```

If the external file reference is based on the physical name of the external file, the FILEEXIST system function should be used.

```
%macro usage2;  
  %if %sysfunc(fileexist(myfile.dat)) %then  
    %put The external does exist.  
  %else  
    %put The external file does NOT exist.  
%mend usage2;
```

Example Determine Existence of a SAS DATA Set or Catalog

It may be useful to determine the existence of a SAS DATA set or catalog for initialization or to conditionally bypassed code. Two system functions are now available to determine the existence of a SAS DATA set or catalog, EXIST and CEXIST.

For example, to determine if the SAS DATA set EXAMPLE exists in the library GLOBAL

```
%macro usage3;  
  %if %sysfunc(exist(GLOBAL.EXAMPLE)) %then  
    %put Data set GLOBAL.EXAMPLE does exist.  
  %else  
    %put Data set GLOBAL.EXAMPLE does NOT exist.  
%mend usage3;
```

or to determine if the catalog member SASHELP.CBT exists in the catalog BASE in the library SASHELP

```
%macro usage4;  
  %if %sysfunc(exist(SASHELP.BASE.SASHELP.CBT))  
    %then  
    %put SASHELP.BASE.SASHELP.CBT does exist.  
  %else  
    %put SASHELP.BASE.SASHELP.CBT does NOT exist.  
%mend usage4;
```

Example for Number of Variables and Observations in a DATA Set

Many solutions have been generated in the past to obtain the number of variables and/or observations present in a SAS DATA set. Most past solutions have utilized a combination of _NULL_ DATA steps, SET statement with NOBS=, and arrays to obtain this information. With the introduction of the OPEN and ATTRN system function this information can be obtained more readily and without interfering with step boundary conditions.

Executing the following code:

```
%macro obsnvars(ds,nvarsp,nobsp);  
  %global &nvarsp &nobsp;  
  %let dsid = %sysfunc(open(&ds));  
  %if &dsid %then %do;  
    %let &nobsp = %sysfunc(attrn(&dsid,NOBS));  
    %let &nvarsp = %sysfunc(attrn(&dsid,NVARS));  
    %let rc = %sysfunc(close(&dsid));  
  %end;  
  %else  
    %put Open for data set &ds failed -  
    %sysfunc(sysmsg());  
%mend obsnvars;  
  
data example;  
  array a a1-a13;  
  do i = 1 to 8;  
    output;  
  end;  
run;  
  
%obsnvars(example,nvars,nobs)  
%put Data set example has &nvars variable(s)  
  and &nobs observation(s).;
```

will generate the following output

Data set example has 14 variable(s)
and 8 observation(s).

Using the GETOPTION Functions to Obtain the Values of SYSTEM or GRAPHICS Options

In report generation, it is convenient to be able to save and to restore a set of system options around a section code responsible for the report. This can be accomplished using a combination of the GETOPTION system function and the OPTION statement to save current option settings, to reset the options to new values and to set then option back to their original values once the report has been generated.

The following macro code:

```
%macro genrep;
  %let ps=%sysfunc(getoption(ps,keyword));
  %let ls=%sysfunc(getoption(ls,keyword));
  %put Saved settings for &ps and &ls;

  options ps=32 ls=65;
  %let nps=%sysfunc(getoption(ps,keyword));
  %let nls=%sysfunc(getoption(ls,keyword));
  %put New settings for &nps and &nls;

  /* Report Generation occurs here */

  options &ps &ls;
  %put Restored settings for &ps and &ls;
%mend;

%genrep
```

will generated the following output:

```
Saved settings for PS=60 and LS=132
New settings for PS=32 and LS=65
Restored settings for PS=60 and LS=132
```

%SYSFUNC Using Formatting Specification

The following code produces a user-defined format X using the FORMAT procedure:

```
proc format;
  value x
    Low-<0 = 'Less Than Zero'
    0 = 'Equal To Zero'
    0<-high = 'Greater Than Zero'
    other = 'Missing';
run;
```

The following macro below is defined to take a parameter PARM and write the value of the parameter to the log using the PUTN function.

```
%macro try(parm);
  %put &parm is %sysfunc(putn(&parm,x.));
%mend try;
```

When the macro %TRY is invoked, the %SYSFUNC converts the character value of the PARM macro variable to a numeric value for the PUTN function. The PUTN function returns as a character value the formatted value of the numeric using the user-defined format X.

Invoking the %TRY macro as

```
%try(1.02)
%try(.)
%try(-.38)
```

would write the following to the log:

```
1.02 is Greater Than Zero
. is Missing
-.38 is Less Than Zero
```

Anonymous FTP File Using New Macro Function Interface

We have created a set of macro routines using the %SYSFUNC macro function that correspond in name and argument specifications to the functions supplied by SAS. This file defines macros from %ABS to %ZIPSTATE, which are compatible with the SAS functions documented in Chapter 11, "SAS Functions," in SAS Language: Reference, Version 6, First Edition. These macro routines are available in a file, functions.sas, for users who can use anonymous file transfer protocol (FTP). See "Appendix 1" for details.

Examples from this file include the following macro definitions:

```
%macro abs(arg1);
  %sysfunc(abs(&arg1))
%mend abs;

%macro zipstate(arg1);
  %sysfunc(zipstate(&arg1))
%mend zipstate;
```

which, when invoked as follows:

```
%abs(-3);
%zipstate(27511)
```

will produce the strings 3 and NC, respectively, and appear to be macro functions fully compatible with SAS functions.

For the case of functions that contain a varying number of arguments, such as the SUM function, the ancillary file contains a macro definition for VARARGS.

```
%macro varargs/parmbuff;
  %let arglist = &syspbuff;
  %let argn = 1;
  %let argv = %qscan(&arglist,
    &argn, %str(,));
  %do %while (%eval(&argv ne));
    %let arg&argn = &argv;
    %let argn = %eval(&argn+1);
    %let argv = %qscan(&arglist, &argn,
      %str(,));
  %end;
  %let argn = %eval(&argn - 1);
  %do i = 1 %to &argn;
    &&arg&i
    %if %eval(&i < &argn) %then ,;
  %end;
%mend varargs;
```

When creating a macro definition for %SUM, the %VARARGS macro will handle the issue of a varying number of arguments. For example,

```
%macro sum/parmbuff;
  %let arglist = &syspbuff;
  %sysfunc(sum(%varargs(&arglist)))
%pend sum;
```

Invoking the SUM macro

```
%sum(1,2,3,4,5,5)
%sum(5000, 100000, 70, 1.0, 3000, -1)
```

would produce the values 12.8 and 108070.

If you use the functions.sas file available from SAS Institute you should check for macro name collisions with autocall or compiled stored macros used at your site. For example, if you use as a function the %SUM macro just described, you should not have a %SUM autocall macro in an autocall library referenced by the SASAUTOS= options.

Error Handling

Whenever the %SYSCALL macro statement or %SYSEVALF, %SYSFUNC and %QSYSFUNC macro functions encounter an error condition, an error message is written to the log.

For %SYSFUNC and %QSYSFUNC, a default type of the invoked system function determines the return value. Functions that return character strings have an error default value of a null string, ". Functions that return numeric values have an error default value of a missing value, ".".

For %SYSEVALF error conditions return an error default value of a missing value, ".".

For %SYSCALL error conditions, no result is returned. In all cases, macro processing continues.

In error conditions, the following error messages may be written to the log:

- Argument <argument number> to function <name> referenced by the %SYSFUNC or %QSYSFUNC macro function is

missing or out of range.

- An error occurred while executing function <name> referenced by the %SYSFUNC or %QSYSFUNC macro function.
- The SAS System could not load the <name> function referenced by the %SYSFUNC or %QSYSFUNC macro function. The system may be out of memory. An error value of missing (".") or a null string (") will be supplied in lieu of the the result of the function. Results of this macro may be incorrect.
- The <name> function referenced in the %SYSFUNC or %QSYSFUNC macro function is not found.
- The function <name> referenced by the %SYSFUNC or %QSYSFUNC macro function has too many arguments.
- The function <name> referenced by the %SYSFUNC or %QSYSFUNC macro function has too few arguments.
- Argument <argument number> to function <name> referenced by the %SYSFUNC or %QSYSFUNC macro function is not a number.
- %SYSEVALF must be followed by a parenthesized expression.
- The function <name> referenced by %SYSFUNC, %QSYSFUNC, or %SYSCALL cannot be used within the MACRO function/call-routine interfaces.
- Invalid data for format <format name>.
- A number has become too large in %SYSEVALF evaluation.
- Division by zero in %SYSEVALF is invalid.
- Unknown %SYSEVALF conversion operand '<conversion name>' specified; conversion is terminated.
- The %SYSEVALF ROUND conversion operation is not supported.
- A floating point overflow has occurred during the %SYSEVALF floating point conversion operation requested; conversion is terminated.
- %SYSEVALF detected an invalid number, <invalid number>, during the conversion operation requested; conversion is terminated.
- Missing values were generated as a result of performing an operation on missing values during %SYSEVALF expression evaluation.
- Mathematical operations could not be performed during %SYSFUNC function execution. The result of the operations have been set to a missing value.
- %SYSEVALF detected a missing value during the conversion operation requested; conversion is terminated.
- %SYSEVALF function has no expression to evaluate.
- The %SYSFUNC or %QSYSFUNC macro function has too many arguments. The excess arguments will be ignored.

%PUT USAGE FOR MACRO VARIABLES, VALUES, AND SCOPE

%PUT <text> | _ALL_ | _AUTOMATIC_ | _GLOBAL_ | _LOCAL_ | _USER_ >;

Where ...	And represents ...
text	is any text.
ALL	writes all the scopes, macro variable names, and macro variable values which currently exist in all nested scopes, including SAS Institute supplied base automatic macro variables, to the log. The scopes are listed in the order of innermost to outermost scope.

Where ...	And represents ...
<u>_AUTOMATIC_</u>	writes all SAS Institute supplied base automatic macro variables and their values to the log. An automatic macro variable is defined by the base SAS System rather than by the user. The scope is listed as AUTOMATIC.
<u>_GLOBAL_</u>	writes all globally scoped macro variables and their values, except SAS Institute supplied base automatic macro variables, to the log. The scope is listed as GLOBAL. The global scope is the outermost referencing environment that exists until the end of the SAS session or job.
<u>_LOCAL_</u>	writes all local macro variables and their values as they currently exist during the execution of an individual macro. The local scope ceases to exist when the macro creating it completes. The scope is listed as the macro name of the macro that creates the macro variable.
<u>_USER_</u>	writes all the scopes, macro variable names, and macro variable values which exist in all nested scopes, except SAS Institute supplied base automatic macro variables, to the log.

In Release 6.09E and 6.11, a new experimental feature added to the macro facility is designed to assist in debugging macro variables, macro variable values, and macro variable scope.

These arguments will write the scope (referencing environment), the macro variable name, and the macro variable value to the SAS log. If the macro variable's value is null, the value will appear blank, but the scope and the macro variable name will be written to the log. If the macro variable value has some characters that are subject to macro quoting, the value will not be unquoted when it is written to the log. If there is no macro variable in the scope requested, nothing will be written to the log and execution will continue.

%PUT _AUTOMATIC_ Example

Invoking the following statement:

```
%put _automatic_;
```

writes the following automatic macro variables to the log but with your own site information, of course.

```
1          %put _automatic_;
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPEDV
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 10MAR96
AUTOMATIC SYSDAY Thursday
AUTOMATIC SYSDEVIC
AUTOMATIC SYSDSN      _NULL_
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 0
AUTOMATIC SYSINFO 0
AUTOMATIC SYSJOBID polzin
AUTOMATIC SYSLAST _NULL_
AUTOMATIC SYSLCKRC 0
AUTOMATIC SYSLIBRC 0
AUTOMATIC SYSMENV S
AUTOMATIC SYSMMSG
AUTOMATIC SYSPARM
AUTOMATIC SYSPBUFF
AUTOMATIC SYSRC 0
AUTOMATIC SYSSCP DEVHOST
AUTOMATIC SYSSCPL DEVHOST
AUTOMATIC SYSSITE 0000000001
AUTOMATIC SYS TIME 16:41
AUTOMATIC SYSVER 6.11
AUTOMATIC SYSVLONG 6.11.0001P123199
```

%PUT _USER_ Example

The following macro statements:

```
%let testit = Today is &sysday;
%macro here(parm1);
  title "Report for &parm1. &testit";
  %put _user_;
%mend here;
```

when invoked as

```
%here(John Doe)
```

writes the following to the log:

```
HERE PARM1 John Doe
GLOBAL TESTIT Today is Wednesday
```

When the macro HERE executes, the %PUT _USER_ statement determines all user-defined macro variables currently in existence. The innermost scope of the executing macro HERE contains the macro variable PARM1 with a value of John Doe. The macro variable TESTIT was created in open code in the global scope, that is written to the log as GLOBAL scope with a value of Today is Wednesday.

Applications Development

Example of CALL SYMPUT with %PUT_USER_

The %PUT_USER_ statement should be most useful in determining scopes of macro variables as well as their values in debugging situations or with complex macro scoping, such as with CALL SYMPUT.

In a more complicated example, the user may be confused about the CALL SYMPUT rule that creates a macro variable value in the current scope available while the DATA step is executing, provided that environment is not empty. An environment is not empty if it has at least one other macro variable or macro statement label stored in it. The scope available to the DATA step is that scope available when the step boundary is hit, for example, the RUN statement. If the current macro referencing environment is empty, the CALL SYMPUT routine places the variable in the closest available scope that is not empty.

In the following example, macro ARRGH1 contains a complete DATA step with the CALL SYMPUT statement:

```
%macro arrgh1(param1);
  data _null_;
    x = "a token";
    call symput('macvar1',x);
  run;
%mend arrgh1;
```

When the macro is invoked and followed by a DATA step as follows:

```
%arrgh1(10)
data temp;
  y = "&macvar1";
run;
```

a message is generated by the WORK.TEMP DATA step to indicate there is no resolution of MACVAR.

WARNING: Apparent symbolic reference MACVAR1 not resolved.

This is because the DATA step is complete within the local referencing scope of ARRGH1, that is not empty because it contains the parameter PARAM1. The local scope of ARRGH1 contains the macro variable MACVAR1, that does not exist in the global scope when it is referenced.

You use the %PUT_USER_ statement inside the macro and before the DATA step that references MACVAR1 to illustrate this

```
%macro arrgh1(param1);
  data _null_;
    x = "a token";
    call symput('macvar1',x);
  run;
  %put _user_;
%mend arrgh1;
```

When ARRGH1 is invoked

```
%arrgh1(10)
%put _user_;
data temp;
  y = "&macvar1";
run;
```

the following is printed to the log:

```
1   %macro arrgh1(param1);
2     data _null_;
3       x = "a token";
4       call symput('macvar1',x);
5     run;
6     %put _user_;
7   %mend arrgh1;
8
9   %arrgh1(10)
ARRGH1 PARAM1 10
ARRGH1 MACVAR1 a token
10  %put _user_;
11  data temp;
12    y = "&macvar1";
WARNING: Apparent symbolic reference
        MACVAR1 not resolved.
13  run;

NOTE: The data set WORK.TEMP has
      1 observations and 1 variables.
```

indicating that the local scope ARRGH1 contains two macro variable names, PARAM1 and MACVAR1, that will not exist when MACVAR1 is referenced in the global scope.

The second %PUT_USER_ statement in the global scope will print nothing to the log since there is no user-defined macro in the currently active scope, the global scope.

In the following example, the DATA step is not complete within the macro ARRGH2 because there is no RUN statement in the macro:

```
%macro arrgh2(param2);
  data _null_;
    x = "a token";
    call symput('macvar2',x);
  %mend arrgh2;
```

When ARRGH2 is invoked, followed by the RUN statement after the invocation, and the DATA step

```
%arrgh2(20)

run;

data temp;
  y = "&macvar2";
run;
```

the macro variable PARAM2 is created in the ARRGH2 local scope; but since the DATA step is executed only when the RUN statement is encountered in open code, the current scope is the

global scope when CALL SYMPUT executes. The CALL SYMPUT routine creates MACVAR2 in the global scope. The macro variable MACVAR2 is available in the global scope for the next DATA step, and no warning is written to the log.

```

1  %macro arrgh2(param2);
2      data _null_;
3          x= "a token";
4          call symput('macvar2',x);
5  %mend arrgh2;
6
7  %arrgh2(20)
8
9  run;
10
11 data temp;
12     y = "&macvar2";
13 run;

```

NOTE: The data set WORK.TEMP has 1 observations and 1 variables.

You can use the %PUT _USER_ statement inside the macro and before the DATA step referencing MACVAR2 to illustrate this.

```

%macro arrgh2(param2);
    data _null_;
        x= "a token";
        call symput('macvar2',x);
        %put _user_;
    %mend arrgh2;

%arrgh2(20)

run;

%put _user_;

data temp;
    y = "&macvar2";
run;

```

When ARRGH2 is invoked, the following is printed to the log:

```

1          %macro arrgh2(param2);
2              data _null_;
3                  x= "a token";
4                  call symput('macvar2',x);
5                  %put _user_;
6              %mend arrgh2;
7
8              %arrgh2(20)
ARRGH2 PARAM2 20
9
10             run;

```

indicating that the local scope ARRGH2 contains one macro variable name PARAM2 with a value of 20. When the macro finishes execution, this local scope will cease to exist.

The second %PUT _USER_ statement in the global scope will write

GLOBAL MACVAR2 a token

```

11
12         %put _user_;
13
14         data temp;
15             y = "&macvar2";
16         run;

```

NOTE: The data set WORK.TEMP has 1 observations and 1 variables.

to the log since this example creates the macro variable MACVAR2 in the global scope. The subsequent reference to the MACVAR2 macro variable will resolve in the temp DATA step.

WRITE MACRO GENERATED TEXT TO EXTERNAL FILE

In 6.09E and 6.11, there is the ability to write macro generated text to an external file. Traditionally, OPTION MPRINT displays SAS source statements generated by macro execution on the SAS log. These statements are built token by token in the SAS tokenizer. Each statement begins on a new line with a prefix with one space separating each token and ends with a semicolon.

MPRINT(macroname): token1 token2 token3 ;

OPTION RESERVEDB1, in conjunction with OPTION MPRINT, will write the macro generated SAS statements to an external file in addition to writing them to the SAS log. The external file must be assigned by the fileref MPRINT. In the external file, the prefix of MPRINT(macroname): will be stripped. This technique will be a useful macro debugging tool for users who wish to examine or separately run macro generated text. The use of a reserved binary option, OPTION RESERVEDB1, made backporting of this feature from 6.11 to 6.09E possible.

The syntax is

```

FILENAME MPRINT 'your-pathname';
OPTIONS MPRINT RESERVEDB1;

```

The external file must be assigned by the fileref MPRINT. All macro generated statements will be written to the external file until the SAS session terminates.

For example, if we had a macro named %FILENAME which generated a filename statement appropriate for unique operating systems, the following code:

```

FILENAME MPRINT 'scratch/external.sas';
OPTIONS MPRINT RESERVEDB1;
%FILENAME(test1, here)

```

writes to the SAS log the macro generated text from the macro %FILENAME using the MPRINT prefix and a note indicating the external file.

```

MPRINT(FILENAME):  FILENAME TEST1 "here" ;
NOTE: The macro generated output from MPRINT
will also be written to external
file /local/u/xxx/scratch/external.sas
while OPTIONS MPRINT and RESERVEDB1
are set.

```

Applications Development

Also, an external file will be created, in this case `/local/u/xxx/scratch/external.sas`, which will contain the macro generated text and all subsequent macro generated text in the session.

```
FILENAME HO "here" ;
```

If MPRINT is not assigned as a fileref or if the file cannot be accessed, warnings will be written to the log and RESERVEDB1 will be set to off. To attempt to use the feature again one must reset RESERVEDB1. The first message will be from the HOST operating system, for example,

```
WARNING: No logical assign for filename MPRINT.
```

or

```
ERROR: Physical file does not exist.  
/local/u/fake/foolish/nothere/place.
```

Macro will issue the following note and turn off RESERVEDB1

```
NOTE: To store macro generated text in an external  
file, assign the filename MPRINT to a valid  
external file. To attempt to access this  
macro debugging feature again, reset OPTION  
RESERVEDB1.
```

By default OPTIONS MPRINT and RESERVEDB1 are off. To see the value of RESERVEDB1 use PROC OPTIONS INTERNAL.

SYSSCP MACRO VARIABLE

The automatic macro variable SYSSCP returns an abbreviation of the operating system you are using. The new automatic macro variable SYSSCPL returns the name of the specific operating system you are using. Both values are read only.

Examples include

Platform...	SYSSCP value	SYSSCPL value
MAC68000	MAC	MAC_M68
MAC PowerPC	MAC	MAC_MPP
MVS	OS	MVS
SOLARIS2	SUN 4	Solaris
VM/CMS	CMS	VM_ESA
WINDOWS 32S	WIN	WIN_32S
WINDOWS 95	WIN	WIN_95
WINDOWS/NT	WIN	WIN_NT

CONCLUSION

We have added some exciting new experimental features to the macro facility in release 6.09E and 6.11 which we hope you will find useful. The %SYSCALL macro statement can be used to call SAS CALL and SAS/TOOLKIT CALL routines. %SYSEVALF is a new macro function that performs floating point arithmetic. %SYSFUNC and %QSYSFUNC are macro functions which provide access to SAS base functions, SCL functions and SAS/GRAPH functions, as well as SAS/TOOLKIT functions. We have provided access with anonymous FTP to sample programs using these new features.

In version 6.09E and 6.11, there are new keywords which can be optionally added to the %PUT statement to allow macro variable names, scope, and values to be printed to the SAS log. Perhaps the most useful to the macro programmer will be the %PUT _USER_ which will list all the user-defined macro variables in all nested referencing environments that are active when it is executed. %PUT _ALL_ will generate even more macro variable information by also listing the SAS automatic macro variables as well as the user-defined macro variables. The keywords _LOCAL_, _GLOBAL_, and _AUTOMATIC_ will write local scope, global scope and automatic macro variables and values to the SAS log, if this should be needed during development of macro systems.

The new macro debugging aid will write macro generated text to an external file. This was very popular on last year's SASWARE ballot and might be a useful macro debugging trick. The new automatic macro variable value SYSSCPL will add more specific operating system values.

REFERENCES

- SAS Institute Inc. (1990), SAS Guide to Macro Processing, Version 6, Second Edition, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1990), SAS Procedures Guide, Version 6, Third Edition, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1990), SAS Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1991), SAS Screen Control Language: Usage, Version 6, First Edition, Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1991), SAS/TOOLKIT Software: Usage and Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

We would like to thank our colleagues in the Core Development Division, especially Oita Coleman, manager of Core Testing, who devoted additional time for testing and code review. Also thanks to go to Russ Tyndall and Tina Hobbs, our macro support staff in Technical Support, who gave us feedback from users as these macro features were developed. Special thanks also to Randy Poindexter in Technical Support for his input for the ability to write macro generated text to an external file.

Appendix 1

Appendix 1 provides information for accessing the sample files we are making available which illustrate more macros using the new macro statement %SYSCALL and the new macro functions %SYSEVALF, %SYSFUNC, and %QSYSFUNC.

The following files have been stored on SAS Institute's Internet gateway (sas.com):

function.sas	Creates corresponding macro functions.
grptest.sas	Creates second graphics example in paper.
grptest1.sas	Creates second graphics example in paper.
mactest.sas	Gives simple examples of new macro functions.
mactest1.sas	Gives examples of new macro functions with functions in Chapter 2.
opttest.sas	Illustrates getting option values.

You can download these files if you have access to the Internet. To download these files, connect to ftp.sas.com. Once you are connected, enter the following responses as prompted:

Name (ftp.sas.com: userid): anonymous
 Password: your e-mail address

All SUGI 21 files are stored in the following directory:

/pub/sugi21

There is one subdirectory for each SUGI 21 paper that has files. Download the following file in /pub/sugi21 for a complete index of all files in /pub/sugi21:

README.index

The file README.index had a description of each directory. The description will contain the title of the paper and the directory name where the files are stored.

Appendix 2

Appendix 2 provides an alphabetical list of all DATA set access, external file access and system functions and call routine now available. It is not within the scope of this paper to document them all. Please contact SAS Institute for appropriate documentation in a system function and call routine dictionary:

ATTRC ATTRN CEXIST CLOSE CUROBS DCLOSE DINFO
 DNUM DOPEN DOPTNAME DOPTNUM DREAD DROPNOTE
 DSNAME EXIST FAPPEND FCLOSE FCOL FDELETE FETCH
 FETCHOBS FEXIST FGET FILEEXIST FILENAME FILEREF
 FINFO FNOTE FOPEN FOPTNAME FOPTNUM FPOINT FPOS
 FPUT FREAD FREWIND FRLEN FSEP FWRITE GETOPTION
 GETVARC GETVARN LIBNAME LIBREF MOPEN NOTE OPEN
 PATHNAME POINT REWIND RXCHANGE RXFREE RXMATCH
 RXPARSE RXSUBSTR SET SYMSG SYSRC VARFMT
 VARINFMT VARLABEL VARLEN VARNAME VARNUM
 VARTYPE

SAS, SAS/AF, and SAS/TOOLKIT are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Jeff Polzin, SAS Institute Inc., SAS Campus Dr., Cary NC 27513,
 (919) 677-8000 x7679, polzin@unx.sas.com

Susan O'Connor, SAS Institute Inc., SAS Campus Dr., Cary NC
 27513, (919) 677-8000 x7671, sassmo@unx.sas.com