

## Macro Facility Enhancements for Release 6.11

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

### Abstract

This paper summarizes the major changes and enhancements that have been added to the macro facility of the SAS® System in Release 6.11. These macro enhancements are experimental in Release 6.11. Examples of these enhancements will be given in the paper. Additionally, sample macro examples are 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.

The %PUT macro statement has been enhanced to write all macro variable names, their referencing environments, and their values to the SAS log. Using the %PUT statement with one of the keyword's `_ALL_`, `_AUTOMATIC_`, `_USER_`, `_LOCAL_` or `_GLOBAL_` will give macro programmers the ability to list all macro variable names, all automatic macro variable names, all user-defined macro variable names, all macro variables in a local scope, or all macro variables in a global scope.

Base SAS software users will be able to access a set of functions similar to the SAS/AF® software Screen Control Language functions through the macro facility using the new %SYSFUNC and %QSYSFUNC macro functions. Some of the newly accessible functions are included in an appendix.

### %SYSCALL Macro Statement

The %SYSCALL macro statement can access SAS System CALL routines or user-written CALL routines created with SAS/TOOLKIT. In order to avoid macro name collisions with the names of users' existing macros, the SAS System uses the reserved prefix letters SYS. The %SYSCALL macro statement can be inside a macro definition or in open code.

The form of the %SYSCALL statement is:

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

where

*call-routine*                      names a SAS System or user-written CALL routine.

*call-routine argument list*      a list of macro variable names separated by commas.

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.

### Examples of the %SYSCALL Macro Statement

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. As a reminder, the SAS CALL RANUNI routine takes two arguments:

RANUNI(*seed*,*x*)

where

*seed*                      is an integer. If *seed* is less than 0 or equal to 0, the time of day is used to initialize the seed stream. A new value for *seed* is returned each time CALL RANUNI is executed.

*x*                          is a numeric value. A new value for the random variate *x* is returned each time CALL RANUNI is executed.

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 is converted to an integer and the period is converted to a numeric missing value. 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 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 Macro Floating Point Evaluation Function

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. In order to avoid macro name collisions with the names of users' existing macros, the SAS System uses the reserved prefix letters SYS. The %SYSEVALF macro function can be used inside a macro definition or in open code.

The form of the %SYSEVALF function is:

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

where

*expression* may be a floating point number.  
*specification* may be the following conversion operation specifications: BOOLEAN, CEIL, FLOOR, INTEGER, TRUNCATE.

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 %EVAL. 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.

Explicit use of the %SYSEVALF function will allow floating point arithmetic.

### Example Using the %SYSEVALF Function

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 generate 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.

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

The %SYSFUNC and %QSYSFUNC macro functions can be used to access SAS System functions or user-written functions generated with SAS/TOOLKIT.

In order to avoid macro name collisions with the names of users' existing macros, the SAS System uses the reserved prefix letters SYS for the %SYSFUNC function. Beware of the potential for a macro name collision with %QSYSFUNC. 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. 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.

The form of the functions is:

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

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

where

*function* is the name of the SAS function or user-written function to be executed.

*function argument list* is a list of arguments to the referenced function.

*format* optionally specifies a SAS or user-defined format or user-written format to be used to format the result of the function referenced.

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, the standard macro variable value type. 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 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 that overrides the function's default format.

The number of arguments required by the referenced function is checked, and an appropriate error messages are generated when necessary. Type conversion errors are also generated when detected, and execution of the function terminates.

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 '.' character string. Character error values are null and converted to a " 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.

### Simple Examples of %SYSFUNC

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 the macro facility using the SAS DATE(), TIME(), and TRANSLATE() functions as macro functions.

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 "&sysdate Is the Date This Session Began";
```

would produce the title

```
15APR95 Is the Date This 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 April 15, 1995, use DATE() and the WORDDATE. format.

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

would produce the title

```
April 15, 1995 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.));
```

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.

There is no simple TRANSLATE macro function available in the macro facility. However, 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 ():

```
%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.

### Simple Example of %SYSFUNC Using User-defined Format

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;
```

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 *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 2" 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;
    %let arg&i = %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)))
%mend sum;
```

### Invoking the SUM macro

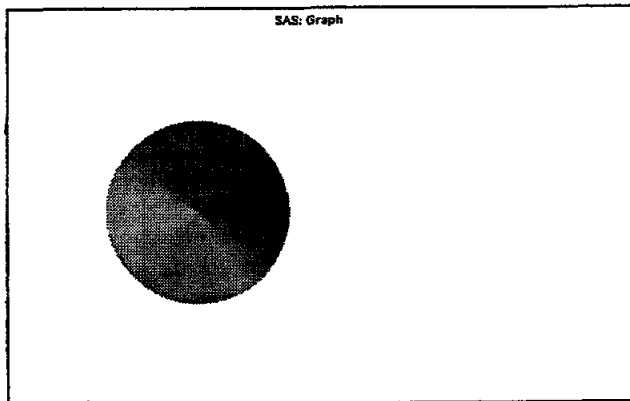
```
%sum(1,2,3,4,5,5)
%sum(5000, 100000, 70, 1.0, 3000, -1);
```

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 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.

## Using Graphics Functions with %SYSFUNC

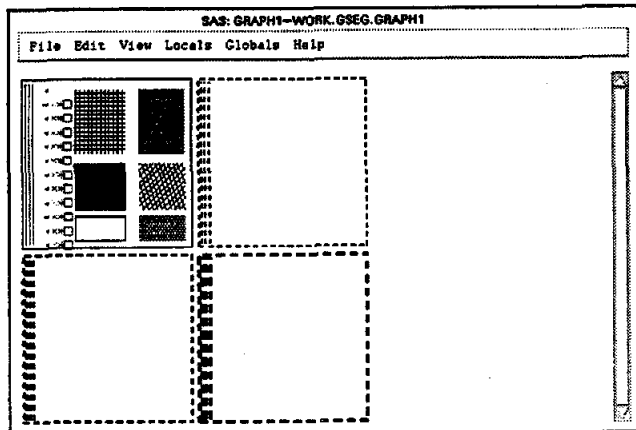
The %SYSFUNC macro function can execute SAS/GRAPH graphics functions. In the following example, the macro %DRAWCIR uses %SYSFUNC to execute the GKS functions to draw a circle where each degree of arc is colored with a different color band.



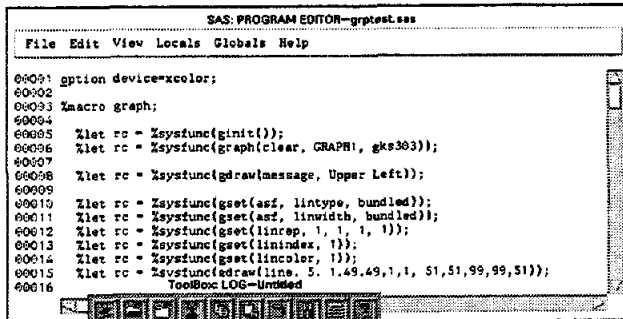
```
goption device=xcolor;

%macro drawcir;
    %let rc = %sysfunc(ginit());
    %let rc = %sysfunc(graph(clear));
    %let rc = %sysfunc(gset(filtype,solid));
    %do i = 0 %to 180;
        %let color =
            %sysfunc(putn(%eval(%i*2),hex3.))8080;
        %let rc = %sysfunc(gset(colrep,
            %eval(%i+1),&color));
        %let rc = %sysfunc(gset(filcolor,
            %eval(%i+1)));
        %let rc = %sysfunc(gdraw(pie, 50, 50, 25,
            %eval(%i*2),%eval(%i*2+2)));
    %end;
    %let rc = %sysfunc(graph(update));
    %let rc = %sysfunc(gterm());
%mend drawcir;

%drawcir
```



Another example of %SYSFUNC using the graphics functions is provided in an anonymous FTP file, grptest1.sas. This graph is created with the %GRAPH macro in Release 6.11 with the following code.



## Error Handling

Whenever the %SYSCALL macro statement or %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 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 %SYSCALL error conditions, no result is returned. In all cases, macro processing continues.

The following error messages may be written to the log:

The minimum number of arguments required by the function or call routine is not specified.

The number of arguments passed to the function is greater than the maximum allowed.

An argument to the call routine or functions that cannot be properly resolved, evaluated, or converted.

An error condition was detected during the execution of the requested call routine or function.

## New Access to Macro Variables, Referencing Environments, and Values

In Release 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.

The %PUT statement writes text to the SAS log. It can be used inside a macro definition or in open code. The syntax is:

**%PUT** text;

where *text* is any text.

In release 6.11, you can use the following arguments as text to the %PUT statement: **\_ALL\_**, **\_AUTOMATIC\_**, **\_GLOBAL\_**, **\_LOCAL\_**, or **\_USER\_**.

These arguments will write the 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 referencing environment 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 referencing environment requested, nothing will be written to the log and execution will continue.



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 environment of ARRGH1, that is not empty because it contains the parameter PARAM1. The local referencing environment of ARRGH1 contains the macro variable MACVAR1, that does not exist in the global referencing environment 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:

```
SAS LOG--Untitled
File Edit View Global Help

NOTE: Copyright (c) 1989-1995 by SAS Institute Inc., Cary, NC, USA
NOTE: SAS (r) Proprietary Software Release 6.11.0001D123199
      Licensed to SAS Institute Inc., Site 0000000001.

NOTE: Running on DEVHOST Model 9900/715 Serial Number 20121119.

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)

NOTE: The DATA statement used 0:00:05.0 real 0:00:00.40 cpu.

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.
NOTE: The DATA statement used 0:00:01.0 real 0:00:00.10 cpu.
```

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 referencing environment; but since the DATA step is executed only when the RUN statement is encountered in open code, the current environment is the global environment when CALL SYMPUT executes. The CALL SYMPUT routine creates MACVAR2 in the global referencing environment. The macro variable MACVAR2 is available in the global environment for the next DATA step, and no warning is written to the log.

```
SAS LOG--Untitled
File Edit View Global Help

NOTE: Copyright (c) 1989-1995 by SAS Institute Inc., Cary, NC, USA
NOTE: SAS (r) Proprietary Software Release 6.11.0001D123199
      Licensed to SAS Institute Inc., Site 0000000001.

NOTE: Running on DEVHOST Model 9900/715 Serial Number 20121119.

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;

NOTE: The DATA statement used 0:00:02.0 real 0:00:00.10 cpu.

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.
NOTE: The DATA statement used 0:00:01.0 real 0:00:00.10 cpu.
```

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:

```
ARRGH2 PARAM2 20
```

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

The second %PUT \_USER\_ statement in the global referencing environment will write

```
GLOBAL MACVAR2 a token
```

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

## Conclusion

SAS Institute has added some exciting new experimental features to the macro facility in release 6.11 that you may 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 System functions. The Institute has provided access with anonymous FTP to sample programs using these new features.

In version 6.11, there are new keywords that can be optionally added to the %PUT statement to allow macro variable names, referencing environments, and values to be printed to the SAS log. Perhaps the most useful to the macro programmer will be %PUT \_USER\_, that will list all the user-defined macro variables in all nested referencing environments that are active when it is executed.

ed. %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 keyword's \_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.

## References

SAS Institute Inc. (1991), *SAS/GRAPH® Software: Usage, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

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. We would especially like to thank Oita Coleman, the manager of Core Testing, and Bruce Tindall of the Core Supervisor Group who devoted additional time for testing and code review. Also thanks go to Russ Tyndall and Tina Keene, our macro support staff in Technical Support, who gave us feedback from users as these macro features were developed. Thanks to Marian Moore, the manager of the Core Supervisor Group, who gave technical assistance and review for this paper.

## Appendix 1

The following describes an additional set of SAS system functions that are supplied with the 6.11 SAS System. These functions are similar in behavior as those found in the SCL programming environment with the same names. These new functions are accessible using the new macro features as well as in other SAS programming language environments. Almost all of the functions available to the SAS DATA step are also available to the new macro enhancements, %SYSFUNC and %SYSCALL.

**ATTRC** Returns the value of a character attribute for a SAS data set.

`attr-value = ATTRC(data-set-id, attr-name);`

where

**attr-value** The value of the character attribute.

**data-set-id** The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**attr-name** The attribute name. If *attr-name* is invalid, a missing value is returned for it *attr-value*.

**ATTRN** Returns the value of a numeric attribute for the specified SAS data set.

`attr-value = ATTRN(data-set-id, attr-name);`

where

**attr-value** The value of the numeric attribute.

**data-set-id** The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**attr-name** The numeric attribute. If *attr-name* is invalid, a missing value is returned for *attr-value*.

**CLOSE** Closes a SAS DATA set.

`sysrc = CLOSE(data-set-id);`

where

**sysrc** Whether the operation was successful:

0 if successful

non-zero if not successful.

**data-set-id** The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**CUROBS** Returns the observation number of the current observation.

`obs_number = CUROBS(data-set-id);`

where

**obs-number** The current observation number.

**data-set-id** The data set identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**DROPNOTE** Deletes a note marker from a SAS data set and/or an external file.

`rc = DROPNOTE(data-set-id / file-id, note-id);`

where

**rc** Whether the operation was successful:

0 if successful

non-zero if not successful

**data-set-id / file-id** The identifier assigned when the data set or external file was opened. If *data-set-id* or *file-id* is invalid, the function returns an error condition.

**note-id** The identifier assigned by the NOTE or FNOTE function.

**DSNAME** Returns the data set name associated with a data set identifier.

`dataset-name = DSNAME(data-set-id);`

where

**dataset-name** Data set name associated with the specific identifier value.

**data-set-id** The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**EXIST** Verifies the existence of a SAS data library member.

`rc = EXIST(member-name<,member-type>);`

where

**rc** Whether the library member exists.

**member-name** Name of the SAS data library member.

**member-type** The type of data library member.

**ACCESS** member is an access descriptor using SAS/ACCESS software.

**CATALOG** member is a SAS catalog.

**DATA** member is a SAS data file. This is the default.

**VIEW** member is a SAS data view.

**FETCH** Reads the next nondeleted observation from a SAS data set.

*sysrc* = **FETCH**(*data-set-id* <,NOSET>);

where  
*sysrc* Whether the operation was successful:  
0 if successful  
non-zero if not successful  
-1 if end of the data set has been reached (no more observations to fetch)  
*data-set-id* The data set identifier returned by the OPEN function. If the *data-set-id* argument is invalid, the function returns an error condition.  
'NOSET' The instruction to prevent the automatic passing of SAS data set variable values to macro variables.

**FETCHOBS** Reads a specified observation from a SAS data set into the Data Set Data Vector.

*sysrc* = **FETCHOBS**(*data-set-id*,*obs-number* <,options>);

where  
*sysrc* Whether the operation was successful:  
0 if successful  
non-zero if not successful  
-1 if end of the data set has been reached (no more observations to fetch).  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*obs-number* The number of the observation to read.  
options One or more options, separated by blanks:  
'ABS' The value of *obs-number* is absolute; that is, deleted observations are counted.  
'NOSET' The instruction to prevent the automatic passing of SAS data set variable values to MACRO variables.

**GETVARC** Assigns the value of a SAS data set variable to a character variable.

*cval* = **GETVARC**(*data-set-id*, *var-num*);

where  
*cval* The value of the character variable identified by the *var-num* argument.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* The number of the variable in the dataset. This value can be obtained using the VARNUM function. In addition, this value is listed next to the variable when you use the CONTENTS procedure.

**GETVARN** Assigns the value of a SAS data set variable to a numeric variable.

*nval* = **GETVARN**(*data-set-id*, *var-num*);

where  
*nval* The value of the numeric variable identified by the *var-num* argument.  
*data-set-id* The identifier assigned when the data set was opened.  
*var-num* The number of the variable in the dataset. This value can be obtained using the VARNUM function. In addition, this value is listed next to the variable when you use the CONTENTS procedure.

**NOTE** Returns an observation ID for the current observation of a SAS data set.

*note-id* = **NOTE**(*data-set-id*);

where  
*note-id* The identifier assigned to the observation.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**OPEN** Opens a SAS data set.

*data-set-id* = **OPEN**(<*data-set-name*<, mode>>);

where  
*data-set-id* The data set identifier or 0 if the data set could not be opened.  
*data-set-name* The SAS data set to be opened. The default value for *data-set-name* is *\_LAST\_*, which is the last data set created in the current SAS session. If the libref is omitted, WORK is assumed.

**POINT** Locates an observation identified by the NOTE function.

*sysrc* = **POINT**(*data-set-id*, *note-id*);

where  
*sysrc* Whether the operation was successful:  
0 if successful  
non-zero if not successful  
*data-set-id* The identifier assigned when the data set was opened.  
*note-id* The identifier assigned to the observation by the NOTE function.

**REWIND** Positions the data set pointer to the beginning of a SAS data set.

*sysrc* = **REWIND**(*data-set-id*);

where  
*sysrc* Whether the operation was successful:  
0 if successful  
non-zero if not successful  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**SET** Links SAS data set variables to variables of the same name and data type.

**CALL SET**(*data-set-id*);

where  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.

**SYSMSG** Returns the text of error messages.

*cval* = **SYSMSG**();

where  
*cval* Text of the error message produced by these functions.

**SYSRC** Returns a system error number or the exit status of the most recently called entry.

*rc* = **SYSRC**();

where  
*rc* The SAS System return code for the most recent error.

**VARFMT** Returns the format assigned to a SAS data set variable.

*format* = **VARFMT**(*data-set-id*, *var-num*);

where  
*format* The format assigned to the specified variable.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* Number of the variable's position in the SAS data set. This number is next to the variable in the list produced by the CONTENTS procedure. The VARNUM function returns this number.

**VARINFMT** Returns the informat assigned to a SAS data set variable.

*informat* = **VARINFMT**(*data-set-id*, *var-num*);

where  
*informat* The informat assigned to the variable.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* The number of the variable's position in the SAS data set. This number is next to the variable in the list produced by the CONTENTS procedure. The VARNUM function returns this number.

**VARLABEL** Returns the label assigned to a SAS data set variable.

*cval* = **VARLABEL**(*data-set-id*, *var-num*);

where  
*cval* The label assigned to the specified variable.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* The number of the variable's position in the SAS data set. This number is next to the variable in the list produced by the CONTENTS procedure. The VARNUM function returns this number.

**VARLEN** Returns the length of a SAS data set variable.

*nval2* = **VARLEN**(*data-set-id*, *var-num*);

where  
*nval2* The length of the variable.  
*data-set-id* The identifier assigned when the data set was opened.  
*var-num* The number of the variable in a SAS data set.

**VARNAME** Returns name of a SAS data set variable.

*cval* = **VARNAME**(*data-set-id*, *var-num*);

where  
*cval* A variable's name.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* The number of the variable in a SAS data set.

**VARNUM** Returns the number of a SAS data set variable.

*var-num* = **VARNUM**(*data-set-id*, *cval*);

where  
*var-num* The number of the specified SAS data set variable or 0 if the variable is not in the SAS data set.  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*cval* The variable's name.

**VARTYPE** Returns the data type of a SAS data set variable.

*type* = **VARTYPE**(*data-set-id*, *var-num*);

where  
*type* The variable's type:  
C for a character variable  
N for a numeric variable  
*data-set-id* The identifier assigned when the data set was opened. If *data-set-id* is invalid, the function returns an error condition.  
*var-num* The number of the variable in a SAS data set.

## Appendix 2

Appendix 2 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 appendix 1. */
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 20 files are stored in the following directory:

`/pub/sugi20`

There is one subdirectory for each SUGI 20 paper that has files. Download the following file in `/pub/sugi20` for a complete index of all files in `/pub/sugi20`:

`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.

SAS, SAS/AF, SAS/GRAPH, 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.