Paper 4-25

Using Macro Functions

Arthur L. Carpenter, California Occidental Consultants

ABSTRACT

Many macro functions are very analogous to those of the DATA step. The differences are in how they are used and applied. While DATA step functions are applied to values on the Program Data Vector, PDV, macro functions are used to create, modify, and work with text strings. Initially this may seem like a minor difference, but because macro functions deal with text, they can be used to build SAS code. This becomes the powerful advantage of the macro language.

The use of various macro functions will be presented. Some of these functions return quoted results, and macro quoting functions will be introduced. Examples will include functions like %SYSEVALF and %SYSFUNC, which are new to V6.11 and V6.12, as well as, various Autocall macros. Several of the examples are adapted from "*Carpenter's Complete Guide to the SAS® Macro Language*".

KEYWORDS

MACRO, MACRO FUNCTIONS, %SYSFUNC, MACRO QUOTING, SCL

MACRO CHARACTER FUNCTIONS

Macro character functions either change or provide information about the text string that is included as one of the arguments. Many of these functions are analogous to similar functions in the DATA step. Some of the macro character functions that have analogous DATA step functions include:

Macro Function(s)	Analogous DATA Step Function	Task
%INDEX	index	First occurrence of a text string is located
%LENGTH	length	Character count
%SCAN %QSCAN	scan	Search for the nth word in a text string
%SUBSTR %QSUBSTR	substr	Select text based on position
%UPCASE	upcase	Convert to upper case

It is important that you remember the differences between these macro functions and their DATA step equivalents. DATA step functions work on character strings, numeric values, and DATA step variable values. Macro functions are applied to text strings that **NEVER** contain the values of DATA step variables.

Several of these functions have two forms, with and without a Q at the start of the function name. Functions with names that start with Q (quoting) remove the meaning from special characters including the ampersand (&), percent sign (%), and mnemonic operators in returned values.

➤%INDEX

The %INDEX function searches the first argument (ARGUMENT1) for the first occurrence of the text string which is contained in the second argument (ARGUMENT2). If the target string is found, the position of its first character is returned as the function's response (0 if not found).

SYNTAX

```
%INDEX(argument1,argument2)
```

EXAMPLE

This example stores three words in the macro variable &X. The %INDEX function is then used to search in &X for the string TALL and the result is then displayed using the %PUT statement.

```
%LET X=LONG TALL SALLY;
%LET Y=%INDEX(&X,TALL);
%PUT TALL CAN BE FOUND AT POSITION &Y;
```

Notice that the TALL as the second argument is not in quotes. The %PUT results in the following text being written to the LOG:

```
TALL CAN BE FOUND AT POSITION 6
```

➤%LENGTH

The %LENGTH function determines the length (number of characters) of it's argument. The number of detected characters is then returned. When the argument is a null string the value of 0 is returned.

SYNTAX

```
%LENGTH(argument)
```

EXAMPLE

In the macro %LOOK the name of the incoming data set is checked to see if it exceeds 8 characters.

```
%MACRO LOOK(dsn,obs);
%if %length(&dsn) gt 8 %then
  %put Name is too long - &dsn;
%else %do;
PROC CONTENTS DATA=&dsn;
  TITLE "DATA SET &dsn";
  RUN;

PROC PRINT DATA=&dsn (OBS=&obs);
  TITLE2 "FIRST &obs OBSERVATIONS";
  RUN;
%end;
%MEND LOOK;
```

The LOG shows that the following data set name exceeds 8 characters:

```
53   %look(demographics, 5)
Name is too long - demographics
```

➤%SCAN and %QSCAN
The %SCAN and %QSCAN functions both search a text string (ARGUMENT1) for the nth word (ARGUMENT2) and returns its value. If ARGUMENT3 is not otherwise specified the same word delimiters are used as in the DATA step SCAN function. For an ASCII system these include the following (for EBCDIC the ¬ is substituted for the ^):

```
blank . < ( + | & ! $ * ) ; ^ - / , % > \
```

%QSCAN removes the significance of all special characters in the returned value.

SYNTAX
```
%SCAN(argument1,argument2[,delimiters])
%QSCAN(argument1,argument2[,delimiters])
```

EXAMPLE
The macro variable &X below can be broken up using the %SCAN function.

```
%LET X=XYZ.ABC/XYY;
%LET WORD=%SCAN(&X,3);
%LET PART=%SCAN(&X,1,Z);
%PUT WORD IS &WORD AND PART IS &PART;
```

The %PUT returns the following:

```
WORD IS XYY AND PART IS XY
```

Notice that the word delimiter (third argument) is not enclosed in quotes as it would be in the DATA step SCAN function.

The %QSCAN function is needed when you want to return a value that contains an ampersand or percent sign. This is demonstrated below:

```
%let dsn = clinics;
%let string =
%nrstr(*&stuff*&dsn*&morestuf);

%let wscan = %scan(&string,2,*);
%let wqscan = %qscan(&string,2,*);

%put &wscan &wqscan;
```

The %PUT writes:

```
clinics &dsn
```

Both functions return the value &DSN, but since the meaning of the & is not masked by %SCAN, the &DSN in &WSCAN is resolved to clinics.

➤%SUBSTR and %QSUBSTR
Like the DATA step SUBSTR function these macro functions return a portion of the string in the first ARGUMENT. The substring starts at the POSITION in the second argument and optionally has a LENGTH of the third argument.

SYNTAX
```
%SUBSTR(argument,position[,length])
%QSUBSTR(argument,position[,length])
```

As is the case with most other macro functions, each of the three arguments can be a text string, macro variable, expression, or a macro call. If a value for LENGTH is not specified, a string containing the characters from POSITION to the end of the argument is produced.

EXAMPLE
```
%LET CLINIC=BETHESDA;
%IF %SUBSTR(&CLINIC,5,4) = ESDA %THEN
       %PUT *** MATCH ***;
%ELSE %PUT *** NOMATCH ***;
```

The LOG would contain `*** MATCH ***` since &CLINIC has the value ESDA in characters 5 through 8.

As is shown in the following example, the %QSUBSTR function allows you to return unresolved references to macros and macro variables.

```
%let dsn = clinics;
%let string =
%nrstr(*&stuff*&dsn*&morestuf);

%let sub  = %substr(&string,9,5);
%let qsub = %qsubstr(&string,9,5);

%put &sub &qsub;
```

The %PUT will write `clinics* &dsn*` in the LOG.

➤%UPCASE
The %UPCASE macro function converts all characters in the ARGUMENT to upper case. This function is especially useful when comparing text strings that may have inconsistent case.

SYNTAX
```
%UPCASE(argument)
```

EXAMPLE
The following code allows the user to differentially include a KEEP= option in the PROC PRINT statement.

The %UPCASE function is used to control for variations in the text that is supplied by the user in the macro call.

```
%macro printit(dsn);
* use a KEEP for CLINICS;
%if %upcase(&dsn)=CLINICS %then
  %let keep=(keep=lname fname ssn);
%else %let keep=;
proc print data=&dsn &keep;
title "Listing of %upcase(&dsn)";
run;
%mend printit;

%printit(cLinICs)
```

The macro call to %PRINTIT produces the following code.

```
proc print data=cLinICs (keep=lname fname
ssn);
title "Listing of CLINICS";
run;
```

MACRO EVALUATION FUNCTIONS

Since in the macro language there are no numbers or numeric variables (there is only text and macro variables that contain text), numeric operations such as arithmetic and logical comparisons become problematic. Evaluation functions are used to bridge the gap between text and numeric operations.

Prior to Release 6.12 only the %EVAL function was available. Because this function has some limitations, the %SYSEVALF function was added to the SAS System starting with Release 6.12.

The evaluation functions are used:

- to evaluate arithmetic and logical expressions
- inside and outside of macros
- during logical comparisons to specify TRUE or FALSE - a value of 1 is returned for logical expressions if the condition is true, 0 (zero) if it is false
- perform **integer** and **floating point** arithmetic

The requests for these functions are either explicit (called by the user by name) or implicit (used automatically without being directly called during comparison operations).

➤%EVAL
The %EVAL function **always** performs integer arithmetic. Regardless of the requested operation the result will always be an integer.

SYNTAX
`%EVAL(`*`argument`*`)`

EXAMPLE
In the following example the %EVAL function is called to perform arithmetic operations. The code uses %EVAL to add the value of 1 to &X which in this case contains 5.

```
%LET X=5;
%LET Y=&X+1;
%LET Z=%EVAL(&X+1);
%PUT  &X  &Y  &Z;
```

The %PUT writes the following to the LOG:

```
5  5+1  6
```

Non-integer arithmetic is not allowed. Use of the statement

```
%LET Z=%EVAL(&X+1.8);
```

would result in the following message being printed in the LOG:

```
ERROR: A character operand was found in
the %EVAL function or %IF condition where
a numeric operand is required. The
condition was: 5+1.8
```

The 'integer only' requirement is taken quite literally. If the above argument had been `&X+1.0` or even `&X+1.`, the error message would still have been generated. In this case the decimal point is enough to prevent a numeric interpretation of the argument.

➤%SYSEVALF
The floating point evaluation function, %SYSEVALF, is new with Release 6.12 of the SAS System (although it was available but undocumented in Release 6.11). This function can be used to perform non-integer arithmetic and will even return a non-integer result from an arithmetic operation.

SYNTAX
`%SYSEVALF(`*`expression[,conversion-type]`*`)`

The EXPRESSION is any arithmetic or logical expression which is to be evaluated and it may contain macro references.

The second argument, CONVERSION-TYPE, is an optional conversion to apply to the value returned by %SYSEVALF. Since this function can return non-integer values, problems could occur in other macro statements that use this function but expect integers.

When you need the result of this function to be an integer, use one of the CONVERSION-TYPEs. A specification of the CONVERSION-TYPE converts a value returned by %SYSEVALF to an integer or Boolean value so it can be used in other expressions that require a value of that type. CONVERSION-TYPE can be:

- BOOLEAN — 0 if the result of the expression is 0 or missing,

		1 if the result is any other value.
•	CEIL	round to next largest whole integer
•	FLOOR	round to next smallest whole integer
•	INTEGER	truncate decimal fraction

The CEIL, FLOOR, and INTEGER conversion types act on the expression in the same way as the DATA step functions of the same (or similar) names i.e. CEIL, FLOOR, and INT.

EXAMPLE
The following table shows a few calls to %SYSEVALF and the resulting values.

Example	Result (%put &x;)
%let x=%sysevalf(7/3)	2.333333333
%let x=%sysevalf(7/3,boolean)	1
%let x=%sysevalf(7/3,ceil)	3
%let x=%sysevalf(7/3,floor)	2
%let x=%sysevalf(1/3)	0.333333333
%let x=%sysevalf(1+.)	.
%let x=%sysevalf(1+.,boolean)	0

MACRO QUOTING FUNCTIONS

Quoting functions allow the user to pass macro arguments while selectively removing the special meaning from characters such as &, %, ;, ‘, and “. Most of these functions are **not** commonly used and are even less commonly understood. Although they are powerful and can even be necessary, programming solutions are usually available that do not require the use of the quoting functions.

All quoting functions are **not** alike. Consult the documentation to get the gory details, however the following three functions should solve most of your quoting problems.

➤%STR
The most commonly used macro quoting function is %STR ('most commonly used' does not necessarily make it the best, see %BQUOTE below). Often it is used along with the %LET statement to mask semicolons that would otherwise terminate the %LET.

In the following example we want to create a macro variable &P that contains two SAS statements;

```
%LET P=PROC PRINT DATA=DSN; RUN;;
```

Because the semicolon following DSN terminates the %LET statement, the macro variable &P contains

```
PROC PRINT DATA=DSN
```

which will almost certainly result in a syntax error due to the missing semicolon.

The %STR function masks the semicolon by quoting it.

```
%LET P=%STR(PROC PRINT DATA=DSN; RUN;);
```

This results in the macro variable &P being correctly assigned the two statements.

```
PROC PRINT DATA=DSN; RUN;
```

➤%BQUOTE
The %BQUOTE function is probably the best choice as a overall quoting function. It eliminates many of the limitations of the %STR function (Carpenter, 1999), and it will also remove the meaning from unmatched symbols that are normally found in pairs such as quotes and parentheses. The following %LET will cause all sorts of problems because the apostrophe will be interpreted as an unmatched quote.

```
 %let a = Sue's new truck;
```

The %STR function will not help because %STR does not mask quote marks, however %BQUOTE does.

```
%let a = %bquote(Sue's new truck);
%put &a;
```

➤%UNQUOTE
Once a quoting function has been used, the text remains quoted. Since these "*quotes*" are hard to see, even in the LOG, this can cause problems for the programmer that does not anticipate that quoting functions may have been used. If you need to remove or change the effects of any of the other quoting functions, the %UNQUOTE is used.

Three macro variables are defined below, but the second, &OTH, is defined using the %NRSTR function. This means that &CITY can not be resolved when &OTH is resolved. When the %UNQUOTE function is applied to &OTH its value (&CITY) is seen as a macro variable which is also resolved.

```
%let city = miami;
%let oth = %nrstr(&city);
%let unq = %unquote(&oth);

%put &city &oth &unq;
```

The LOG shows:

```
miami &city miami
```

Although &OTH looks like any other macro variable in the %PUT statement, it will not be fully resolved because

it is quoted, thus preventing &CITY from being resolved.

USING DATA STEP FUNCTIONS IN THE MACRO LANGUAGE

Two macro tools were introduced with Release 6.12 that allow the user to execute virtually all of the functions and routines available in the DATA step as part of the macro language. The %SYSCALL macro statement calls DATA step routines and the %SYSFUNC macro function executes DATA step functions. Through the use of these two elements of the macro language, the need to use the single pass DATA _NULL_ step to create macro variables is virtually eliminated.

More importantly these two macro functions greatly increase the list of functions available to the macro language by making available almost all DATA step and user-written functions.

SYNTAX

```
%SYSFUNC(function-name(function-
      arguments)[,format])
%QSYSFUNC(function-name(function-
      arguments)[,format])
```

EXAMPLE

The following example shows three ways to add the current date to a TITLE. The automatic macro variable &SYSDATE is easy to use but cannot be formatted. Prior to Release 6.12 most users created a DATA _NULL_ step with an assignment statement and a CALL SYMPUT to create a formatted macro variable. The DATA step can now be avoided by using the %SYSFUNC macro function.

```
data _null_;
today = put(date(),worddate18.);
call symput('dtnull',today);
run;

title1 "Automatic Macro Variable SYSDATE
      &sysdate";
title2 "From a DATA _NULL_ &dtnull";
title3 "Using SYSFUNC
      %sysfunc(date(),worddate18.)";
```

The following three titles are produced:

```
Automatic Macro Variable SYSDATE 10APR00
 From a DATA _NULL_      April 10, 2000
   Using SYSFUNC      April 10, 2000
```

The leading spaces before the date in the second two titles is caused by the date string being right justified. The LEFT and TRIM functions can be used to remove the space, however care must be exercised or a couple of problems can be encountered.

The first is that function calls cannot be nested within a %SYSFUNC. Fortunately this is rather easily handled because %SYSFUNC requests **can** be nested.

Secondly the resolved values of interior calls to %SYSFUNC are used as arguments to the outer calls. When the resolved value contains special characters (especially commas), they can be misinterpreted. The following revised TITLE3 will not work because the interior %SYSFUNC uses a formatted value which contains a comma.

```
title3 "Using SYSFUNC
%sysfunc(left(%sysfunc(date(),worddate18.)
))";
```

After the inner %SYSFUNC is executed the result is:

```
title3 "Using SYSFUNC %sysfunc(left(April
10, 2000))";
```

Because of the comma, the LEFT function will see two arguments (it is expecting exactly one), and the message 'too many arguments' is generated.

The %QSYSFUNC function can be used to mask special characters in the text string that is passed to the next function. Rewriting the TITLE statement using %QSYSFUNC as is shown below eliminates the problem with the comma.

```
title3 "Using SYSFUNC
%sysfunc(left(%qsysfunc(date(),worddate18.
)))";
```

TITLE3 from above becomes:

```
Using SYSFUNC April 10, 2000
```

Starting with Release 6.12, functions originally included in the Screen Control Language (SCL) library, became available in the DATA step and so are also now available to %SYSFUNC.

Starting with Nashville Release of the SAS System (Version 7), SAS/ACCESS® utilizes the engine option in the LIBNAME statement to designate the type of data that is to be read or written. This allows the user to directly read non-SAS data forms such as Excel® and dBase®. Other third party vendors have been using technique for some time.

The following macro uses the PATHNAME function to retrieve the path associated with a LIBREF. This path is then used to build a new LIBREF with a different engine.

```
%macro engchng(engine,dsn);
* engine - output engine for this &dsn
* dsn    - name of data set to copy
*;

* Create a libref for the stated Engine;
libname dbmsout clear;
libname dbmsout &engine
      "%sysfunc(pathname(sasuser))";

* Copy the SAS data set using the
* alternate engine;
```

```
proc datasets;
copy in=sasuser out=dbmsout;
select &dsn;
run;

%mend engchng;

****************;
* convert to alt. engine;
%engchng(v604,classwt)
```

Notice that the LIBREF in the PATHNAME function call is **not** in quotes. Remember that arguments to macro functions are always text so the quotes are not necessary. As you use DATA step functions with %SYSFUNC, you should expect the behavior of many of the arguments of the DATA step functions to vary slightly in ways such as this.

When the macro must not contain any non-macro statements, *librefs* can be defined by using the LIBNAME DATA step function.

```
%let rc = %sysfunc(libname(classwt,
      %sysfunc(pathname(sasuser)),
      v604));
```

This single macro statement will create the *libref* CLASSWT with the V604 engine using the same path information as SASUSER.

%SYSFUNC can also be used to check to see if a data set exists by using the EXIST function. The following %IF will execute the %RPT macro only when the data set identified by &DSN exists.

```
%if %SYSFUNC(exist(&dsn)) %then %RPT(&dsn);
```

The GAMMA function can be used to calculate factorials. In a macro a factorial can be calculated by calling the GAMMA function with %SYSFUNC. The following macro acts like a function and returns the factorial value of the argument, N.

```
%macro fact(n);
  %sysfunc(gamma(%eval(&n+1)))
%mend fact;

%let fact5 = %fact(5);
```

The macro variable &FACT5 will contain 120.

AUTOCALL MACROS THAT MIMIC FUNCTIONS

The AUTOCALL facility allows the user to call macros that have been defined outside of the execution of the current program. A number of these macros are provided with the base macro language and are described in the Macro Language Elements section of the *SAS® Macro Language: Reference, First Edition* reference manual. Although these are, strictly speaking, macros, they act like functions.

Commonly used Autocall macros include:

- %CMPRES
- %LEFT
- %LOWCASE
- %TRIM
- %VERIFY

➤%LEFT

This macro can be used to left justify a macro argument. In the earlier example for %QSYSFUNC the DATA step LEFT function was used, this title can be further simplified by using %LEFT.

```
title3 "Using SYSFUNC
%left(%qsysfunc(date(),worddate18.))";
```

➤%VERIFY

While %INDEX and its variations search for specific strings, %VERIFY determines the position of the first character that is **NOT** in a text string. The following example subsets a string starting at the first character that is not a number.

```
%let code = 2000SUGI25;
%let part =
%substr(&code,%verify(&code,1234567890));
```

&PART will contain:

```
SUGI25
```

➤%CMPRES

The %CMPRES macro removes **multiple** blanks (as well as leading and trailing blanks) from a text string. This macro is similar to the COMPBL DATA step function. In the following example a numeric value is placed into a macro variable using the SYMPUT routine. In the conversion process a series of leading blanks are added to &FIVE.

```
data _null_;
x=5;
call symput('five',x);
run;

%put *&five*;
%let five = *%cmpres(&five)*;
%put &five;
```

The resulting LOG shows:

```
138
139  %put *&five*;
*           5*
140  %let five = *%cmpres(&five)*;
141  %put &five;
*5*
```

SUMMARY

Macro functions extend the capabilities and power of the DATA step to the macro language. Many of the DATA step functions have analogous macro functions, and other DATA step functions can now be directly accessed from within the macro language through the use of macro functions such as %SYSFUNC. Since recent changes to the base SAS System allow the use in the DATA step of functions that were originally a part of SCL, these functions are also now available in the macro language.

Quoting functions and functions that return quoted results, are available and can be used to remove the special meaning of selected characters.

The Autocall library that is supplied with the SAS System also contains a number of macros that act like functions. These macros further extend the range of the macro language by providing even more power and functionality.

ABOUT THE AUTHOR

Art Carpenter's publications list includes three books (*Annotate: Simply the Basics*, *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language*), two chapters in *Reporting from the Field*, and over three dozen papers and posters presented at SUGI, PharmaSUG, and WUSS. Art has been using SAS since 1976 and has served in a variety of positions in user groups at the local, regional, and national level.

Art is a SAS Quality Partner™ and SAS Certified Professional™. Through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199

(760) 945-0613
art@caloxy.com
www.caloxy.com

REFERENCES

Burlew, Michele M., *SAS® Macro Programming Made Easy*, Cary, NC: SAS Institute, Inc., 1998, 280 pp.

Carpenter, Arthur L., *Carpenter's Complete Guide to the SAS® Macro Language*, Cary, NC: SAS Institute Inc., 1998, 242 pp.

Carpenter, Arthur L., "Macro Quoting Functions, Other Special Character Masking Tools, and How to Use Them", *Proceedings of the Twenty-Fourth Annual SUGI Conference*, 1999, pp237-241.

SAS Institute Inc., *SAS® Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc., 1997, 304 pp.

TRADEMARK INFORMATION