

Macro Functions: How to Make Them - How to Use Them

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

The SAS[®] macro language includes a number of macro functions, such as %UPCASE, %QSCAN, and %SUBSTR. In addition a number of Autocall macros, such as %LEFT, %TRIM, and %VERIFY, are also supplied that act like macro functions. Macro functions are especially useful as programming tools, because the function call is replaced directly by the result of the call. Many macro programmers are unaware that they too can write useful macro functions.

Fortunately macro functions are not difficult to program. Not difficult, that is, if you know the simple techniques required to turn a macro into a macro function. This paper shows you how to create a macro function, the statements that you must avoid, and the technique used to “pass” the result of the function back to the calling program.

INTRODUCTION

Macros are often used as programming tools to answer specific questions about the programming environment. A typical type of macro solution has the macro create one or more macro variables that are added to the global macro symbol table. Since these variables are then available throughout the SAS session, they can later be accessed as required. The disadvantage of this approach is that the macro variables must be global and therefore the programmer risks collisions with other global macro variables that may be in use by the program or application. A further disadvantage is the need to use macro variables to pass information in the first place. These disadvantages can often be eliminated by converting the macro into a macro function.

The following macro, which was published in the *SAS[®] Guide to Macro Processing, Version 6*, can be used to determine if a data set exists. It utilizes a DATA_NULL_ step, the automatic macro variable &SYSERR, and a global macro variable (&EXIST). When the macro is called, the DATA step is compiled including the SET statement. During compilation SAS determines if the data set (&DSN) exists, and if it does not exist, the macro variable &SYSERR will be loaded with a value greater than zero.

```
%macro exist(dsn);
  %global exist;
  %if &dsn ne %then %do;
    * An unknown data set causes a
    * compile error that is reflected
    * in the SYSERR macro variable;
    data _null_;
      stop;
      if 0 then set &dsn;
      run;
  %end;
  %if &syserr=0 %then %let exist=YES;
  %else %let exist=NO;
%mend exist;
```

After the macro has been called, branching and execution decisions can be made based on the value stored in the global macro variable &EXIST. For example, assume that in a DATA step a user wants to conditionally execute a block of code depending on the existence of the data set SASUSER.BIGDAT. The macro call and associated statements might be:

```
%exist(sasuser.bigdat)

data .....;
set.....;
if "&exist"="YES" then do;
... more SAS statements ...
```

The macro is first called (this creates and loads the global macro variable &EXIST) and then the macro variable is later tested. The user is responsible for not having another global macro variable named &EXIST, the user must know

that the macro variable created by the macro is &EXIST, and that EXIST will take on the values of YES and NO. The macro works, and it is reasonably efficient, but it is NOT a macro function. The following discussion demonstrates how this macro differs from a macro function, and what needs to be done in order to convert it into a macro function.

BUILDING THE FUNCTION

In order for a macro to be classified as a macro function, it must be usable within macro statements, and it must be able to build base language code directly. In the above example of %EXIST, the macro MUST be called outside of the DATA step, since it generates its own DATA step. The following macro call would cause problems in the DATA step logic.

```
data .....;
set.....;
%exist(sasuser.bigdat)
if "&exist"="YES" then do;
... more SAS statements ...
```

The attributes required of a macro function include:

- all statements in the macro must be macro statements
- the macro should create NO macro variables other than those that are local to that macro
- the macro should resolve to the value that is to be returned

USE ONLY MACRO STATEMENTS

The %SYSFUNC macro function is a wonderful tool when building your own macro functions. It allows you to use DATA step functions without using the DATA step. In the previous version of the macro %EXIST the DATA step is used to ascertain whether or not the data set named in &DSN exists. Since the DATA step reads no data and is only used to build a macro variable, it can be completely replaced with a call to %SYSFUNC. The following version of the macro %EXIST eliminates the use of the

DATA step.

```
%macro exist(dsn);
%global exist;
/* Check if &DSN has been created;
%if %sysfunc(exist(&dsn)) %then
    %let exist=YES;
%else %let exist=NO;
%mend exist;
```

Since this version of %EXIST does not utilize a DATA step the macro call could now be placed within the DATA step that uses &EXIST. Although the macro call from within the DATA step caused problems in the previous example, the following code will now work.

```
data .....;
set.....;
%exist(sasuser.bigdat)
if "&exist"="YES" then do;
... more SAS statements ...
```

Notice that even the comments should be converted to macro comments. In this version of %EXIST it really does not matter whether or not the comment is a macro comment, but as we continue to build the macro function, it becomes crucial that we use only macro style comments.

CREATE ONLY LOCAL MACRO VARIABLES

Since the previous version of %EXIST still creates a global macro variable (&EXIST), the macro is still not a macro function. The %LOCAL statement causes macro variables to be assigned to the local symbol table, and should be used for **all** of the macro variables that are defined within the macro. Use of the %LOCAL statement ensures that there will be no collisions between macro variables defined within the macro and those that may already exist on the global symbol table.

The problem, of course, is that these local macro variables cannot be accessed outside of the macro. How then does one pass values out of a macro without creating global macro variables? The answer to this question is at the heart of the solution of how to build macro functions.

MACRO CALL RESOLVES TO THE RETURNED VALUE

Rather than creating a macro variable that is made available later, we can let the macro call itself resolve to the value of interest. This way the %EXIST(SASUSER.BIGDAT) macro call will be literally replaced in the code stream by whatever value is to be tested. In the following version of %EXIST the macro call will be replaced by either YES or NO.

```
%macro exist(dsn);
  /* Check if &DSN has been created;
  %if %sysfunc(exist(&dsn)) %then YES;
  %else NO;
%mend exist;
```

Notice the *action* for the %IF-%THEN statement (YES) and the %ELSE statement (NO). Since a YES or NO is not a macro statement it cannot be executed by the macro processor (as a %LET statement would be), it is just 'left behind'. When the macro expression is true (%SYSFUNC and the EXIST function find the data set named in &DSN, the whole %IF-%THEN-%ELSE resolves to YES. Since this is not a macro statement it is left behind where it must be dealt with by the base language.

Now we can ask the IF-THEN question without referring to the macro variable &EXIST and without first calling the macro.

```
data .....;
set.....;
if "%exist(sasuser.bigdat)"="YES" then do;
... more SAS statements ...
```

If the data set SASUSER.BIGDAT exists the DATA step becomes:

```
data .....;
set.....;
if "YES"="YES" then do;
... more SAS statements ...
```

When we can anticipate how the functions that are called from within the macro by %SYSFUNC will behave, we can write more sophisticated macro functions. The EXIST

DATA step function, for instance, returns a non-zero positive value when the data set is located. That means, of course, that we can test for that value directly, and this is what was done in the previous example - within the macro. The following example takes this one step further by performing the check outside of the macro rather than within it, and this eliminates the %IF-%THEN-%ELSE.

Like the above example, the following version of %EXIST contains only macro statements and it establishes no macro variables - global or local. It further reduces what the user needs to know in order to use the macro by eliminating the YES / NO and in the process the macro %IF-%THEN-%ELSE.

```
%macro exist(dsn);
  /* The following sysfunc call results
  /* in a non-zero value when the data
  /* set exists;
  %sysfunc(exist(&dsn))
%mend exist;
```

It is the last line of the above macro that establishes it as a macro function. The call to %SYSFUNC will execute the EXIST function. The result of that execution will be a number *i.e.* 0 or 1028. Since this value is not a macro statement it will be left behind, and this value can then be tested for directly by the calling program.

This allows the use of the macro as part of either a DATA step statement or a macro statement. In a DATA step IF-THEN-ELSE we might use the following statement.

```
if %exist(sasuser.bigdat) > 0 then do;
```

Or better yet just:

```
if %exist(sasuser.bigdat) then do;
```

If the data set exists the IF-THEN statement might become something like:

```
if 1028 then do;
```

And of course 1028 is true.

USING THE FUNCTION

The first definition of the %EXIST macro shown above in the Introduction has several limitations that have already been discussed. In addition, this macro might not be usable in some macro environments. Again this is a direct consequence of the fact that it must run a DATA step before it can build the macro variable &EXIST. This means that this macro could never be used in a macro that was itself being used as a macro function. Further the user would run the risk that the sequence of execution of macro statements and DATA steps might cause problems (base language statements are executed after macro language statements and consequently the macro variable &EXIST might not have been created when it is needed).

Excluding the non-macro statements and building a macro function such as the one shown in the final version of %EXIST, allows its use in both the macro language and the base language. As a macro function %EXIST can also be used in macro language statements such as:

```
%if %exist(sasuser.bigdat) %then %do;
```

Because of this flexibility the macro function becomes a building block that can be used in the construction of other macro functions. For instance a function to return the number of observations in a data set would first need to ascertain whether or not the data set exists.

SUMMARY

Macro functions are not that hard to build. You need to remember to use all macro statements, to create only local macro variables, and to allow the macro to resolve to the value that is to be 'passed' back to the calling routine.

Take a look at your macro tools. Some of them might be candidates for conversion to macro functions. Try it. Macro functions are fun to

write, they reduce the possibilities of macro variable collisions, and best of all, they can be used to impress your date after the movies.

REFERENCES

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

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

Hamilton, Jack, "How Many Observations Are In My Data Set?", *Proceedings of the Twenty-Sixth SAS® User Group International Conference*, Cary, NC: SAS Institute Inc., 2001.

Lund, Pete, "My Favorite Functions", *Proceedings of the Twenty-Fifth SAS® User Group International Conference*, Cary, NC: SAS Institute Inc., 2000.

Lund, Pete, "Make Your Life a Little Easier: A Collection of SAS Macro Utilities", *Proceedings of the Twenty-Sixth SAS® User Group International Conference*, Cary, NC: SAS Institute Inc., 2001.

SAS® Guide to Macro Processing, Version 6, Second Edition, Cary, NC, SAS Institute Inc., 1990, 319pp.

ABOUT THE AUTHOR

Art Carpenter's publications list includes three books, and numerous papers and posters presented at SUGI, PharmaSUG, NESUG, MWSUG, and WUSS. Art has been using SAS since 1976 and has served in various leadership positions in local, regional, national, and international user groups.

Art is a SAS Certified Professional™ and 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



TRADEMARK INFORMATION

SAS and SAS Certified Professional are registered trademarks of SAS Institute, Inc. in the USA and other countries.

® indicates USA registration.