

## MACROS: TIPS, TECHNIQUES, AND EXAMPLES

Andrew M. Traldi

Advanced Quantitative Solutions, Inc.

Most SAS® users are aware that SAS has a macro facility but are unsure of how they can use it or are fearful that macros are too difficult. Although macros can be complex, they can be very helpful in writing general-purpose SAS programs; in some instances, they are absolutely critical to an application.

### WHAT IS THE SAS MACRO FACILITY?

The purpose of the SAS macro language is to generate text which is used in SAS programs; this text can be any valid SAS code: statements, variables, strings, PROC steps, etc. In its simplest form, a macro can be used for text substitution in SAS code that is needed often. Consider the following example:

```
%macro hello(name,day,topic);
DATA _NULL_;
PUT "Hello, &name.. How are you today?
Welcome to &day's class on &topic..";
RUN;
%mend hello;
%hello(Andrew,Friday,SAS macros)
```

Here is the resulting SASLOG:

```
Hello, Andrew. How are you today?
Welcome to Friday's class on SAS macros.
NOTE: The DATA statement used 0.25
seconds.
```

### MACRO VARIABLES

The macro language, like most languages, involves constants and variables. It is possible to use macro variables without actually using macros. For example:

```
%let mthname=July 1996;
%let region=Northeast;
```

```
/* Select desired observations and put
```

```
in dataset MTHSALES */
PROC PRINT DATA=MTHSALES;
TITLE "Sales report for &mthname --
Region: &region";
RUN;
```

These statements could be useful if you provide reporting by month and region and you want to be able to generate reports for each. Since we have automated the title generation of the PROC PRINT, couldn't we also automate the process by which observations are selected as well? The answer is – of course – yes, by adding an additional macro variable and adding a WHERE clause to the PROC PRINT step:

```
%let mth=5; /* month #;
PROC PRINT DATA=IN.YTDSALES
(WHERE=(MONTH(SALEDATE) = MDY(&mth,1,96)
AND UPCASE(REGION) =
UPCASE("&region")));
TITLE "Sales report for &mthname --
Region: &region";
RUN;
```

This example assumes that there is a dataset IN.YTDSALES that contains all sales for the current year with a variable SALEDATE and a character variable REGION that we can use for record selection. Note that we haven't even used a macro here, just macro variables.

### ENVIRONMENT

A short discussion of the macro variable environment is probably in order. The environment can be explicitly specified with either the %global or the %local statement. A global macro variable is available throughout the SAS program (open code as well as all macros). A local macro variable is available only in the macro where it is defined.

Consider the following example:

```
%global var1 var2;
%let var1=hello;
%let var2=world;

%macro test2(x);
%put * in macro test2, x=%x *;
%mend test2;

%macro test;
%put * test: var1=%var1 var2=%var2 *;
%local var2;
%let var1=hi;
%let var2=universe;
%put * test: var1=%var1 var2=%var2 *;

%let var3=%str(hello, world);

%put * test: var3=%var3 *;
%test2(%var3);
%mend test;

%put * open code: var1=%var1 var2=%var2
*;
DATA _NULL_;
PUT "Before invoking macro test,
var1=%var1 var2=%var2";
RUN;
%test
DATA _NULL_;
PUT "After invoking macro test,
var1=%var1 var2=%var2";
RUN;
%put * open code: var3=%var3 *;
```

Here is the resulting SASLOG:

```
* open code: var1=hello var2=world *
13582 DATA _NULL_;
13583 PUT "Before invoking macro test,
var1=%var1 var2=%var2";
13584 RUN;
```

```
Before invoking macro test, var1=hello
var2=world
NOTE: The DATA statement used 0.28
seconds.
```

```
13585 %test
* test: var1=hello var2=world
* test: var1=hi var2=universe *
* test: var3=hello, world *
* in macro test2, x=hello, world *
MPRINT(TEST): ;
13586 DATA _NULL_;
13587 PUT "After invoking macro test,
var1=%var1 var2=%var2";
13588 RUN;
```

```
After invoking macro test, var1=hi
var2=world
NOTE: The DATA statement used 0.36
seconds.
```

```
WARNING: Apparent symbolic reference
VAR3 not resolved.
13589 %put * open code: var3=%var3 *;
* open code: var3=%var3 *
```

The default environment for a macro is what I would call "downwardly global." That is, the value of the macro variable can be referenced (and changed!) by the macro in which it is defined as well as in macros which are invoked by the defining macro, but not in other macros or in open code. This may seem a little cumbersome at first, but it allows you a great deal of flexibility.

#### TIPS:

- define all macro variables as either global or local.
- set aside specific macro variables for %do loop indices (for example, &i, &j, &k) and define them as %local to avoid resetting their values in other macros.

#### SYMPUT AND SYMGET

The CALL SYMPUT routine is used to assign values to macro variables during DATA step execution, while the SYMGET function is used to retrieve values from macro variables during DATA step execution (macro variables can also be resolved directly, during DATA step compilation). This example will illustrate the use of SYMPUT and SYMGET as well as how macro variables are resolved directly.

```
%let mth=7;
%let region=Northeast;
%global mthname;
* Generate month name for title;
DATA _NULL_;
LENGTH TEST MTHNAME $ 16;
* These two statements are equivalent;
MTHNAME=PUT (MDY (&mth,1,96),MONNAME9.) ||
' 1996';
MTHNAME=PUT (MDY (SYMGET ('MTH'),1,96),MONN
```

```

AME9.) || ' 1996';
* Load month name into a macro variable;
CALL SYMPUT ('MTHNAME', TRIM(MTHNAME));
* These two statements are not
equivalent;
TEST=SYMGET ('MTHNAME');
PUT '&mthname=' "&mthname " TEST=;
RUN;
%put ** after DATA step,
mthname=&mthname **;

```

Note how we integrated the macro variable into a regular SAS statement, first by resolving it directly and then by using the SYMGET function. It is important to remember that macro variables that are set using the SYMPUT function are not available to the macro compiler until AFTER the DATA step is finished; this distinction is noted by the second pair of statements. Here is the resulting SASLOG:

```

13595 DATA _NULL_ ;
13596 LENGTH TEST MTHNAME $ 16;
13597 * These two statements are
equivalent;
13598
MTHNAME=PUT (MDY (&mth,1,96),MONNAME9.) ||
' 1996';
13599
MTHNAME=PUT (MDY (SYMGET ('MTH'),1,96),MONNAME9.) ||
' 1996';
13600 * Load month name into a macro
variable;
13601 CALL
SYMPUT ('MTHNAME', LEFT (TRIM (MTHNAME)));
13602 * These two statements are not
equivalent;
13603 TEST=SYMGET ('MTHNAME');
13604 PUT '&mthname=' "&mthname "
TEST=;
13605 RUN;

```

NOTE: Character values have been converted to numeric values at the places given by: (Line): (Column).

```

13599:17
&mthname= TEST=July 1996
NOTE: The DATA statement used 0.61 seconds.

```

```

13606 %put ** after DATA step,
mthname=&mthname **;
** after DATA step, mthname=July 1996 **

```

Note that even though the macro variable was numeric, SAS has to perform a character to numeric conversion when the

SYMGET function is used. That is because the macro compiler treats everything as text, even macro variables or constants that are valid numeric values<sup>1</sup>. Also note that the macro variable mthname cannot be resolved directly during DATA step execution, but can be retrieved via the SYMGET function.

## PARAMETERS

The macro language allows for passing of parameters in much the same way as other programming languages; indeed, those of you who develop SAS/AF applications with SAS Screen Control Language are probably used to passing parameters.

A macro can have two types of parameters: positional and keyword. Positional parameters should only be used for relatively simple macros that have few parameters. Here is an example of a macro with keyword parameters:

```

%macro smartprt(dsn=_LAST_, title=, by=, id=, var=, dsnopt=, options=);

```

```

%* Print the specified dataset, using
the specified variables in the BY, ID,
and VAR statements, using dataset
options, and print options

```

```

&dsn: LIB.DATASET or DATASET
&title: Title for PROC PRINT
&by, &id, &var: list of variables
&dsnopt: dataset options (i.e. OBS=
WHERE= FIRSTOBS=, etc.)
&options: print options (i.e. NOOBS
LABEL N, etc.);

```

```

%* We could have error-checking here
(i.e. ensure all variables in &var are
actual variables, &dsn exists, etc.);

```

```

%if &by ne %then %let by=BY &by;
%if &title ne %then

```

---

<sup>1</sup> In SAS/AF<sup>®</sup> applications, it is possible to differentiate between character and numeric macro variables by using the SYMGETN, SYMGETC, SYMPUTC, and SYMPUTN functions (these are not available in base SAS).

```

    %let title=TITLE "&title";
    %if &id ne %then %let id=ID &id;
    %if &var ne %then %let var=VAR &var;
    %if %quote(&dsnopt) ne %quote() %then
        %let dsnopt=%str( (&dsnopt) );

```

```

PROC PRINT &options DATA=&dsn &dsnopt;
&title;
&by ;
&id ;
&var ;
RUN;
%mend smartprt;

```

Here is the macro invocation and the resulting SASLOG:

```

%smartprt (var=SALESREP CUSTOMER SALEDATE
AMOUNT,dsn=IN.YTDSALES,
dsnopt=%str(WHERE=(YEAR(SALEDATE) =
1996)),options=NOOBS,title=1996 Sales)

```

```

MPRINT(SMARTPRT): PROC PRINT NOOBS
DATA=IN.YTDSALES ( WHERE=(YEAR(SALEDATE)
= 1996)) ;
MPRINT(SMARTPRT): TITLE "1996 Sales";
MPRINT(SMARTPRT): ;
MPRINT(SMARTPRT): ;
MPRINT(SMARTPRT): VAR SALESREP
CUSTOMER SALEDATE AMOUNT ;
MPRINT(SMARTPRT): RUN;

```

NOTE: The PROCEDURE PRINT used 0.25 seconds.

Note how the order of the parameters is not important and that we do not have to specify the &by and &id macro variables since there are no BY or ID variables to be used in the PROC PRINT step. If we had used positional parameters instead, our invocation would look like this:

```

%smartprt (IN.YTDSALES,1996
Sales,,,SALESREP CUSTOMER SALEDATE
AMOUNT,WHERE=YEAR(SALEDATE) =
1996),NOOBS)

```

Here, we need to include commas to mark the places of the &by and &id macro variables as well as remembering the required ordered of the parameters. For more complex macros, it is almost always better to use keyword parameters.

Let us continue to build on our print macro. Suppose you would like to be able to

produce individualized sales reports for each sales rep. Without a macro, we would probably use SAS code like this:

```

** Producing individualized sales
reports for each sales rep. No macro --
first we sort the dataset ;
PROC SORT DATA=IN.YTDSALES;
BY SALESSSN SALEDATE ACCOUNT;
RUN;

```

```

PROC PRINT DATA=IN.YTDSALES;
TITLE 'SALES REPORT -- JULY 1996';
BY SALESSSN REGION;
PAGEBY SALESSSN;
ID SALESSSN REGION;
VAR SALESREP SALEDATE CUSTOMER AMOUNT;
SUM AMOUNT;
SUMBY SALESSSN;
RUN;

```

We can see that although this is easy to code, we do not have the ability to generate the desired report titles nor do we have any flexibility. By continuing to expand our print macro, we can generate individual reports for each sales rep:

```

%macro prt(dsn=, sortvar=, classvar=,
var2=, var3=, mv1=mlv, mv2=m2v,
mv3=m3v);
/* This sets up macro variables for the
title
input parameters:
dsn=dataset name sortvar=variable(S)
to sort by
class=variable for BY group
var2-var3=two addl vars to use
mv1-mv3=prefix for macro variables
output macro variable:
numitems=# of classvar levels

**** no error checking done ****
;
%local i var1;
%let var1=&classvar;
%global numitems;
/* Sort dataset if necessary;
%if &sortvar ne &classvar %then %do;
PROC SORT DATA=&dsn;
BY &sortvar;
RUN;
%end;
/* Get all levels of Class variable;
PROC SORT DATA=&dsn NODUPKEY OUT=_TEMP_;
BY &classvar;
RUN;

```

```

DATA _NULL_;
SET _TEMP_ END=LAST;
/* Load macro variables values;
%do i=1 %to 3;

```

```

    %if &&var&i ne %then %do;
        CALL SYMPUT("&&mv&i" ||
COMPRESS(PUT(_N_,5.)),TRIM(&&var&i));
        %end;
    %end;
IF LAST THEN CALL SYMPUT
('NUMITEMS',COMPRESS(PUT(_N_,5.)));
RUN;

/* Print the dataset using the macro
title to generate levels of class
variable and variable values for titles;

%do i=1 %to &numitems;
    OPTIONS PAGENO=1;
    PROC PRINT DATA=&dsn
(WHERE=(&classvar = "&&ssn&i"));
    TITLE1 "SALES REPORT FOR &&rep&i,
DISTRICT: &&m3v&i";
    VAR ACCOUNT CUSTOMER AMOUNT;
    SUM AMOUNT;
    RUN;
%end;
%mend prt;

%prt(dsn=IN.YTDSALES, classvar=
SALESSSN, sortvar=SALEDATE CUSTOMER,
var2=SALESREP, var3=REGION, mv1=SSN,
mv2=REP)

```

Here is the resulting SASLOG:

```

MPRINT(PRT):  PROC SORT
DATA=IN.YTDSALES;
MPRINT(PRT):  BY SALEDATE CUSTOMER;
MPRINT(PRT):  RUN;

```

NOTE: The data set IN.YTDSALES has 74 observations and 11 variables.  
NOTE: The PROCEDURE SORT used 0.4 seconds.

```

MPRINT(PRT):  PROC SORT
DATA=IN.YTDSALES NODUPKEY OUT=_TEMP_;
MPRINT(PRT):  BY SALESSSN;
MPRINT(PRT):  RUN;

```

NOTE: 64 observations with duplicate key values were deleted.  
NOTE: The data set USER.\_TEMP\_ has 10 observations and 11 variables.  
NOTE: The PROCEDURE SORT used 0.34 seconds.

```

MPRINT(PRT):  DATA _NULL_;
MPRINT(PRT):  SET _TEMP_ END=LAST;
MPRINT(PRT):  CALL SYMPUT("SSN" ||
COMPRESS(PUT(_N_,5.)),TRIM(SALESSSN));
MPRINT(PRT):  CALL SYMPUT("REP" ||
COMPRESS(PUT(_N_,5.)),TRIM(SALESREP));
MPRINT(PRT):  CALL SYMPUT("m3v" ||
COMPRESS(PUT(_N_,5.)),TRIM(REGION));
MPRINT(PRT):  IF LAST THEN CALL
SYMPUT('NUMITEMS',COMPRESS(PUT(_N_,5.)))
;
MPRINT(PRT):  RUN;

```

NOTE: The DATA statement used 0.56 seconds.

```

MPRINT(PRT):  OPTIONS PAGENO=1;
MPRINT(PRT):  PROC PRINT
DATA=IN.YTDSALES (WHERE=(SALESSSN =
"018448368"));
MPRINT(PRT):  TITLE1 "SALES REPORT FOR
GUY LEWIS, DISTRICT: SOUTHEAST";
MPRINT(PRT):  VAR ACCOUNT CUSTOMER
AMOUNT;
MPRINT(PRT):  SUM AMOUNT;
MPRINT(PRT):  RUN;

```

NOTE: The PROCEDURE PRINT used 0.19 seconds.

```

MPRINT(PRT):  OPTIONS PAGENO=1;
MPRINT(PRT):  PROC PRINT
DATA=IN.YTDSALES (WHERE=(SALESSSN =
"042641999"));
MPRINT(PRT):  TITLE1 "SALES REPORT FOR
SHARON FREDERICKS, DISTRICT: NORTHEAST";
MPRINT(PRT):  VAR ACCOUNT CUSTOMER
AMOUNT;
MPRINT(PRT):  SUM AMOUNT;
MPRINT(PRT):  RUN;

```

NOTE: The PROCEDURE PRINT used 0.19 seconds.

...

Suppose that you want even more flexibility, such as:

- Using a numeric, rather than character, variable for the sort variable
- Using more than three variables in the title
- Using more than one classification variable

We can accomplish all of these, and more, by modifying the macro. However, there is a point of diminishing returns. Although we want our programs to be as flexible as possible, it does not make sense to spend countless programming hours so that we won't have to rewrite a few PROC steps at some point in the future.

## MACRO STYLE AND COMMENTS

There are style issues when writing macro code, just as there are for regular SAS code.

- Use good, clean style. This is especially important since macro code is typically less readable than base SAS code. Some examples of good macro style include: indenting %do groups, using white space, and – most importantly – using comments liberally.
- Use keyword parameters and define macro variables as needed.

Every person has programming conventions that he or she likes to use. Here are a few that I use to help keep my macro code as readable as possible:

- Use lower case for macro code and upper case for regular SAS code.
- Set up most global macro variable assignments in open code at the beginning of the program so that they can be found and changed easily.

Here is an important note regarding macro comments:

- Note that %\* indicates a macro comment, while \* is a SAS comment. The macro compiler ignores a %\* comment but treats a \* comment as constant text. This can cause problems, in some instances, when the \* comment is used instead of the %\* comment (for example, in between two %do groups). Another distinction is that \* comments inside macros will appear in the SASLOG (if OPTIONS MPRINT is set) while %\* comments will not.

## USING SYSTEM MACRO VARIABLES

One big advantage of using macro programming in SAS is that it allows you access to system information, such as internal return codes, operating system information, SAS version information, etc. For example, you might have a SAS program that runs on more than one operating system (multi-platform architecture is a big advantage of the SAS system) and need to allocate external files depending on the operating system being used currently. Without using a macro, you would have to write a DATA step like the following, using the system macro variables to allocate the files:

```
DATA _NULL_;
IF "&sysscp" eq 'OS2' THEN DO;
  PUT 'ALLOCATING OS2 FILES';
  LIBNAME SALES "Y:\SAS610\SALES";
  LIBNAME USER "Y:\SAS610\USER";
  FILENAME LISTING
  'Y:\SAS610\LISTING.TXT';
END;
ELSE DO; * MVS;
  PUT 'ALLOCATING MVS FILES';
  LIBNAME SALES
  "D4.U.NOR.AUDIT.SASDATA.SALES";
  LIBNAME USER "&userid..SASDATA";
  FILENAME LISTING "&userid..SASLIST";
END;
RUN;
```

But, this technique will not work! SAS library and external file allocations are not executable code and are therefore compiled, even if they are within a conditional DO group. The preceding example was submitted while running SAS under the OS/2® operating system:

```
13714
13715 DATA _NULL_;
13716 IF "&sysscp" eq 'OS2' THEN DO;
13717   PUT 'ALLOCATING OS2 FILES';
13718   LIBNAME SALES
  "Y:\SAS610\SALES";
NOTE: Libref SALES was successfully
assigned as follows:
      Engine:          V610
      Physical Name: Y:\SAS610\SALES
13719   LIBNAME USER "Y:\SAS610\USER";
```

NOTE: Libref USER was successfully assigned as follows:

```
Engine: V610
Physical Name: Y:\SAS610\USER
13720 FILENAME LISTING
'Y:\SAS610\LISTING.TXT';
13721 END;
13722 ELSE DO; * MVS;
13723 PUT 'ALLOCATING MVS FILES';
13724 LIBNAME SALES
"D4.U.NOR.AUDIT.SASDATA.SALES";
NOTE: Library SALES does not exist.
13725 LIBNAME USER
"&userid..SASDATA";
WARNING: Apparent symbolic reference
USERID not resolved.
WARNING: Apparent symbolic reference
USERID not resolved.
NOTE: Library USER does not exist.
13726 FILENAME LISTING
"&userid..SASLIST";
WARNING: Apparent symbolic reference
USERID not resolved.
WARNING: Apparent symbolic reference
USERID not resolved.
13727 END;
13728 RUN;
```

ALLOCATING OS2 FILES

NOTE: The DATA statement used 1.42 seconds.

Even though the macro variable syssep resolved to OS2 – as we would expect – the MVS® LIBNAME and FILENAME statements were still compiled and caused an error! A macro solves this problem because the SAS code is conditionally generated, not conditionally executed:

```
%macro allocate;
/* allocate external files and SAS
libraries depending on which OS
and userid (in case it is MVS);
%if %syssep eq OS2 %then %do;
  %let root=Y:\SAS610;
  %put ALLOCATING OS2 FILES;
  LIBNAME SALES "&root\SALES";
  LIBNAME USER "&root\USER";
  FILENAME LISTING "&root\LISTING.TXT";
%end;
%else %do; /* MVS;
  %put ALLOCATING MVS FILES;
  %let root=D4.U.NOR.AUDIT;
  LIBNAME SALES "&root..SASDATA.SALES";
  LIBNAME USER "&userid..SASDATA";
  FILENAME LISTING "&userid..SASLIST";
%end;
%mend allocate;

13749 %allocate
ALLOCATING OS2 FILES
```

```
MPRINT (ALLOCATE): LIBNAME SALES
"Y:\SAS610\SALES";
NOTE: Libref SALES was successfully
assigned as follows:
Engine: V610
Physical Name: Y:\SAS610\SALES
MPRINT (ALLOCATE): LIBNAME USER
"Y:\SAS610\USER";
NOTE: Libref USER was successfully
assigned as follows:
Engine: V610
Physical Name: Y:\SAS610\USER
MPRINT (ALLOCATE): FILENAME LISTING
"Y:\SAS610\LISTING.TXT";
```

Note that only the allocations for the OS/2 section are generated, so the SAS compiler never sees the allocation statements that are required for MVS.

I often write applications that read several related external flat files; for example, they might differ only by month and region number. Here is an example of a simple macro which would allow a user to specify the month(s) and region(s) to read:

```
* specify starting month, ending month,
and regions (separate by space, leave
blank
for all);
%global startmth endmth regions;
%let startmth=DEC95;
%let endmth=FEB96;
%let regions=2 4;

%macro tokens(string,sep);
/* This macro returns the number of
tokens in a string, with a given
delimiter (default=blank);
%local i result;
%if %quote(&sep) eq %then
  %let sep=%str( );
%let i=1;
%do
  %while(%quote(%scan(&string,&i,%str(&sep)
))) ne %str( );
    %let i=%eval(&i+1);
%end;
%let i=%eval(&i-1);
%let result=&i;
%result
%mend tokens;

%macro allocate;
/* DATA step to load in desired months;
%local i j mth;
DATA _NULL_;
ENDMTH=INPUT("&endmth",MONYY5.);
MTH=INPUT("&startmth",MONYY5.);
N=0;
```

```

DO WHILE (MTH LE ENDMTH);
  N+1;
  CALL SYMPUT ('MTH' ||
COMPRESS (PUT (N, 5.)), PUT (MTH, MONYY5.));
  MTH=INTNX ('MONTH', MTH, 1);
END;
CALL SYMPUT
('NUMMTHS', COMPRESS (PUT (N, 5.)));
RUN;

FILENAME IN (
%do i=1 %to &nummths;
  %let numregs=%tokens(&regions);
  %do j=1 %to &numregs;
    %let reg=%scan(&regions, &j);

"D4.U.NOR.AUDIT.REGION&reg..&month&i"
  %end;
%end;
) DISP=SHR;
%mend allocate;

13796 %allocate
MPRINT (ALLOCATE): DATA _NULL_;
MPRINT (ALLOCATE):
ENDMTH=INPUT ("FEB96", MONYY5.);
MPRINT (ALLOCATE):
MTH=INPUT ("DEC95", MONYY5.);
MPRINT (ALLOCATE): N=0;
MPRINT (ALLOCATE): DO WHILE (MTH LE
ENDMTH);
MPRINT (ALLOCATE): ;
MPRINT (ALLOCATE): N+1;
MPRINT (ALLOCATE): CALL SYMPUT ('MTH' ||
COMPRESS (PUT (N, 5.)), PUT (MTH, MONYY5.));
MPRINT (ALLOCATE):
MTH=INTNX ('MONTH', MTH, 1);
MPRINT (ALLOCATE): END;
MPRINT (ALLOCATE): CALL
SYMPUT ('NUMMTHS', COMPRESS (PUT (N, 5.)));
MPRINT (ALLOCATE): RUN;

```

NOTE: The DATA statement used 0.44 seconds.

```

MPRINT (ALLOCATE): FILENAME IN (
MPRINT (ALLOCATE):
"D4.U.NOR.AUDIT.REGION2.DEC95"
"D4.U.NOR.AUDIT.REGION4.DEC95"
MPRINT (ALLOCATE):
"D4.U.NOR.AUDIT.REGION2.JAN96"
"D4.U.NOR.AUDIT.REGION4.JAN96"
MPRINT (ALLOCATE):
"D4.U.NOR.AUDIT.REGION2.FEB96"
"D4.U.NOR.AUDIT.REGION4.FEB96" )
DISP=SHR;

```

There are several general-purpose utility macros that I use often. The macro tokens, listed above, is a good example. Although I could easily have written the file allocation macro above without using another macro – by simply using the %do %while group directly rather than calling another macro – I

wrote that macro because often times I need a function to return the number of tokens in a string. This macro allows me to do that and has the flexibility to search for tokens in a string using a delimiter other than a blank space.

There are several macros described in *SAS Guide to Macro Processing, Version 6, Second Edition* which you will find very useful. Some examples of these include:

|          |                                                                                       |
|----------|---------------------------------------------------------------------------------------|
| %exist:  | returns whether a dataset exists;                                                     |
| %numobs: | returns the number of observations in a dataset; and                                  |
| %words:  | counts the number of words in a string (similar to my macro tokens discussed earlier) |

## CONCLUSION

I hope that this paper gives you a little insight into the technique of writing SAS macros. There are many issues which I did not cover, including the difficulties which can arise when trying to deal with complicated text strings (those which include special characters, such as: "%&"). I highly recommend referring to the *SAS Guide to Macro Processing* when you begin to use macros; you will find it indispensable.

## **Trademarks**

**SAS and SAS/AF are registered trademarks of SAS Institute, Inc. in the USA and other countries.**

**OS/2 and MVS are trademarks of IBM Corporation.**