Advanced Macro Topics: Utilities and Examples

Arthur L. Carpenter
California Occidental Consultants

KEY WORDS

autocall, function, macro, macro functions, SASHELP, utility, views

ABSTRACT

Numerous utilities have been created using the SAS® Macro Language and many of the newer utilities take advantage of recent changes and enhancements to the SAS® System. The examples of macro utilities presented in this paper include the use of:

- SASHELP views
- DATA step and SCL functions using %SYSFUNC
- the AUTOCALL library
- system supplied AUTOCALL macros

The macros detailed in this paper include examples of:

- copying members of a catalog
- subsetting data sets
- working with lists of data set variables
- counting observations
- converting macro text to lower case

Each of the examples presented in this paper is taken from the author's latest BBU book "*Carpenter's Complete Guide to the SAS® Macro Language*".

COPYING MEMBERS OF A CATALOG

This macro is used to copy selected members of a catalog in the TEST environment to a production area. The names of the members do not need to be known prior to execution, however a filter is available to select members beginning with certain types of names.

The SASHELP.VSCATLG view ❶ is used to create a list of the members in the TEST library. A DATA _NULL_ step is used with a subsetting IF ❷ to select the members of interest. The SELECT statement ❺ used within PROC DATASETS is built using the macro variables (&&CNAME&I) that contain the member names ❸ and the number of names (&CATCNT) ❹.

```
* Copy catalogs from the TEST to the
* PRODUCTION areas.;

%macro catcopy(test,prod);
* test  - libref for the test area
* prod  - libref for the production
*         area
*;

* Determine catalogs in TEST area;
data _null_;
set sashelp.vscatlg
  (where=(libname="%upcase(&test)"));❶
length ii $2;

* Select only some of the catalog
* members;
if memname in: ('DE', 'ED', 'PH');❷

* Create macro variable for each
* member;
i+1;
ii=left(put(i,2.));
call symput('cname'||ii,memname);❸
call symput('catcnt', ii);❹
run;

proc datasets ;
copy in=&test out=&prod
memtype=catalog;
```

```
select
  %do i = 1 %to &catcnt;❺
      &&cname&i
  %end;
;
quit;
%mend catcopy;

%catcopy(appls,work)
```

The Technical Support section of *SAS Communications* Volume xxii 4th Qtr. 1996, p. 43 has a similar example that builds the new catalogs with modifications using PROC BUILD.

SUBSETTING DATA SETS

The macro %SELPCNT can be used to select a specified percentage of the observations of a data set. In this example we would like to base the subset on the value of a variable *e.g.* select the observations with the largest values. The POINT and NOBS options on the SET statement can be used in the selection process. In the following macro, the data are sorted first ❶ and then subsetted in the following DATA step. The NOBS option ❷ creates a variable on the PDV which is equal to the number of observations in the data set. This allows us to calculate the number of observations to read (IDPCNT) ❸. The count is based on total observations not total number of distinct values of the ID variable (&IDVAR).

```
%macro selpcnt(dsn,idvar,pcnt);

* Sort the incoming data set in
descending order;
proc sort data=&dsn ❶
          out=items;
by descending &idvar;
run;

* Read the first IDPCNT observations
* from ITEMS;
data topitems;
idpcnt = nobs*&pcnt; ❸
do point = 1 to idpcnt;
   set items point=point nobs=nobs ❷;
   output;
end;
stop;
run;
%mend selpcnt;

%selpcnt(sasclass.biomass,bmtotl,.25);
```

CHECKING THE EXISTENCE OF SAS DATA SETS

At times we would like to be able to determine if a data set exists before we execute a procedure such as PROC PRINT against it. When creating systems dynamically some data sets may not exist under certain circumstances and we need to be able to determine their status at execution time.

In Release 6.11 and later the macro function %SYSFUNC can be combined with functions such as EXIST to check for the existence of a data set.

```
%macro exist(dsn);
%global exist;
%if %sysfunc(exist(&dsn)) ❶ %then
      %let exist=YES;
%else %let exist=NO; ❷
%mend exist;
```

❶ The SCL EXIST function returns a value that is equal to 1 when the data set exists. Otherwise it returns a 0 causing the %IF to be false ❷ which results in &EXIST being set to NO.

The EXIST function is combined with the %SYSFUNC macro function in a similar example in *SAS® Macro Language, Reference, First Edition*, pp. 242.

WORKING WITH LISTS OF DATA SET VARIABLES

This example assumes that you have created a macro variable that contains a list of data set variables. In this case the macro variable (&KEYFLD) contains the variables that form the key fields (variables used in a BY statement).

A sample definition of the macro variable &KEYFLD might be:

```
%let keyfld = investid subject treatid;
```

In the program that will use this list we might expect to see a BY statement such as:

```
by &keyfld;
```

In order to use FIRST. and LAST. processing, however we need to know the name of the last variable in the list. This allows us to write a statement such as:

```
if last.treatid then do;
```

To do this using &KEYFLD we need to know its component parts (variable names in the list) and the number of names. The following DATA step can be used to create a series of macro variables (&KEY1, &KEY2, ..), one for each name in &KEYFLD.

```
*determine the list of key vars;
data _null_;
* count the number of keyvars
* save each for later;
str="&keyfld"; ❶
do I = 1 to 6;
 key = scan(str,i,' '); ❷
 if key ne ' ' then do;
  ii=left(put(i,1.));
  call symput('key'||ii,
       trim(left(key))); ❸
  call symput('keycnt',ii); ❹
 end;
end;
run;
```

❶ The variable STR is created to hold the list using an assignment statement. SYMGET could also have been used.

❷ This string is then broken into words using the SCAN function.

❸ Macro variables are then created using the SYMPUT routine.

❹ The number of key variables is also counted and assigned to &KEYCNT.

Once the macro variables have been established statements that require FIRST. or LAST. processing can be rewritten as:

```
by &keyfld;
if last.&&key&keycnt then do;
```

Since &KEYCNT is the number of key variables &&KEY&KEYCNT will resolve to the name of the last variable in the list stored in &KEYFLD.

Since the same result can be accomplished using only macro statements, the DATA step in the previous example is not as efficient as it could be. In the following example the %DO %UNTIL loop is used to step through and count the elements in &KEYFLD.

```
%Macro doit;

%let I = 1; ❶
%do %until
 (%scan(&keyfld,&I,%str( ))=%str()); ❷
   %let key&I=
          %scan(&keyfld,&I,%str( )); ❸
   %let I = %eval(&I + 1); ❹
%end;
%let keycnt = %eval(&I-1); ❺
%mend doit;
```

❶ Initialize &I which will be used as the counter in the %DO %UNTIL loop.

❷ Scan &KEYFLD for the i^{th} word using a blank as the separator and check to see if the function results in a null string. Notice

that the first %STR contains a blank space while the second has no space.

❸ The previous line ❷ determined that the i^{th} word exists retrieve it and store it in &KEYi.

❹ Increment the counter by 1 in preparation for the next scan.

❺ The counter was incremented one time too many. Reduce the value by one and save the count in &KEYCNT.

COUNTING OBSERVATIONS

In the following example the macro %OBSCNT acts like a macro function in that the macro call resolves to a value that is the number of observations in the stated data set.

```
%macro obscnt(dsn); ❶
%local nobs;
%let nobs=.;

%* Open the data set of interest;
%let dsnid = %sysfunc(open(&dsn)); ❷

%* If the open was successful get the;
%* number of observations and CLOSE;
%* &dsn;
%if &dsnid %then %do; ❸
  %let nobs =
       %sysfunc(attrn(&dsnid,nobs)); ❹
  %let rc  =%sysfunc(close(&dsnid)); ❺
%end;
%else %do; ❻
  %put Unable to open &dsn - ;
  %put     %sysfunc(sysmsg());
%end;

%* Return the number of observations;
&nobs ❼
%mend obscnt;
```

❶ The user passes the name of the data set (&DSN) into the macro.

❷ The selected data set is opened and is assigned an identification number which is stored in &DSNID.

❸ If the data set was found and opened successfully &DSNID will be greater than 0 and this %IF expression will be true.

❹ The ATTRN function is used to determine the number of observations in the data set. The ATTRN function can be used to make a number of queries on the data set once it is opened. These include password and indexing information as well as the number of variables and the status of active WHERE clauses.

❺ The data set should be closed after retrieving the desired information.

❻ When the open is unsuccessful we may want to write a message to the LOG. The SYSMSG() function returns the reason the OPEN failed.

❼ Since this is the last statement in the macro, the resolved value of &NOBS will be effectively 'returned' to the calling program and its value will be a period (.) if the data set was not opened successfully.

The following program creates the data set A and then calls %OBSCNT to write the number of observations to the LOG.

```
data a;
do i = 1 to 10;
  x=i**i;
  output;
end;
run;

%put number of obs is %obscnt(a);
```

The following line is written to the LOG.

```
number of obs is 10
```

CONVERTING MACRO TEXT TO lower CASE

The AUTOCALL macro facility allows

previously written macros to be saved and reused without including their definition in the current program. In addition to any macros that you might write and add to your own AUTOCALL library, the SAS System comes supplied with a number of macros in its own autocall library. These can be found in your default SASAUTOS location, under windows this might be:

```
!sasroot\core\sasmacro
```

Depending on your operating system and other SAS products leased additional autocall macros may be available.

Not only are these macros useful in-and-of-themselves, but since the code is available you can modify these macros for your own purposes or use them as patterns for new macros.

SAS® Macro Language: Reference, First Edition, briefly describes these macros (pp. 158-160), and includes their description with the other macro language elements in the Macro Language Dictionary, Chapter 13 (pp.161-270). In addition Chapter 7 (pp. 185-187) in the *SAS® Guide to Macro Processing, Version 6* lists many of the standard SAS System Autocall macros in one place.

One of the macros that is supplied with the SAS System AUTOCALL library is %LOWCASE. The %LOWCASE macro can be used to convert all upper case characters to lower case. The macro accepts a single argument which is translated using the %INDEX and %SUBSTR functions. This macro is demonstrated in the following example.

```
%let mixed = SAS Macro Language;
%let lower = %lowcase(&mixed);
%put &lower;
```

The resulting LOG shows the macro variable &LOWER to be all lower case.

```
175  %let mixed = SAS Macro Language;
176  %let lower = %lowcase(&mixed);
177  %put &lower;
sas macro language
```

The code for this macro is interesting because of the way it finds upper case characters and then translates them.

```
%macro lowcase(string);

... documentation removed ...

%local i length c index result;
%let length = %length(&string);
%do i = 1 %to &length;
   %let c = %substr(&string,&i,1);❶
   %if &c eq %then %let c = %str( );
   %else %do;
      %let index = ❷
%index(ABCDEFGHIJKLMNOPQRSTUVWXYZ,&c);
      %if &index gt 0 %then ❸
           %let c =
%substr(abcdefghijklmnopqrstuvwxyz,
              &index,1); ❹
   %end;
   %let result = &result.&c; ❺
   %end;
&result ❻
%mend;
```

❶ The i[th] character in the string of interest (&STRING) is temporarily stored in the macro variable &C.

❷ We then check to see if &C contains an upper case letter using the %INDEX function. And the result (position) is stored in &INDEX.

❸ If an upper case letter is found (&INDEX >0), it is converted using the %SUBSTR function ❹.

❺ &C is then appended (converted or not) onto &RESULT which continues to grow until the entire string has been checked.

❻ The macro variable &RESULT contains the converted string and this will be the final resolved value of the macro call.

SUMMARY

The SAS macro language allows developers and users to create flexible utilities and programs. These can be further strengthened by taking advantage of the many features that have been added to the macro language in recent releases of the SAS System. Of special interest is the use of the SASHELP views, the AUTOCALL libraries, and the ability to access the DATA step functions through the use of %SYSFUNC.

TRADEMARK INFORMATION

ABOUT THE AUTHOR

Art Carpenter's publications list includes two chapters in *Reporting from the Field*, the two books *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language* and over two dozen papers and posters presented at SUGI, WUSS, and PharmaSUG. Art has been using SAS since 1976 and has served as a steering committee chairperson of both the Southern California SAS User's Group, SoCalSUG, and the San Diego SAS Users Group, SANDS; a conference cochair of the Western Users of SAS Software regional conference, WUSS; and Section Chair at the SAS User's Group International conference, SUGI.

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

AUTHOR CONTACT

Art Carpenter
California Occidental Consultants
PO Box 6199
Oceanside, CA 92058-6199

(760) 945-0613

art@caloxy.com
www.caloxy.com