Paper 028-30

Storing and Using a List of Values in a Macro Variable

Arthur L. Carpenter
California Occidental Consultants, Oceanside, California

ABSTRACT

When using the macro language it is not at all unusual to need to manipulate a list of values. These may be a list of variables in a data set or even a list of data set names. Macro variables are often used to hold this list and there are a couple of storage options. Either each word of the list can be stored in its own indexed macro variable or, as discussed in this paper, one macro variable can be used to store the entire list.

The macro variable that contains this list of values can be created in a number of different ways including through the use of the DATA step or PROC SQL. Once created, the user will often want to step through the list performing one or more operations on each of the elements in the list, and this is often accomplished using a combination of the %DO statement and the %SCAN function.

This paper shows how to write a series of values to a macro variable and then how to take advantage of the list by utilizing it from within the macro language.

KEYWORDS

Macro variables, %SCAN, &&VAR&I, PROC SQL, INTO, CALL SYMPUT, %DO

INTRODUCTION

Since the macro language does not truly support the concept of the ARRAY, a number of methods have been developed to allow the programmer to overcome this limitation. These include the use of the %INCLUDE, the use of the CALL EXECUTE, lists of macro variables that are accessed using the &&VAR&I construct, and the topic of this paper, lists stored within a single macro variable. Each of these approaches has both advantages and disadvantages. This author has found the latter two solutions to be the most practicable as well as easier and more logical to implement.

The use of the &&VAR&I construct to implement macro arrays is a fairly common solution. Once the user understands the basic concepts involved this solution, with practice it is fairly straight forward. Like arrays in the DATA step, this approach utilizes one variable per array element. These array elements are linked together not with an ARRAY statement, but by a common naming convention (usually a common named prefix) and a numeric suffix. This results in a list of macro variables with names such as: &SUBJ1, &SUBJ2, &SUBJ3, &SUBJ4, etc. The numeric suffix (such as &I) becomes the array index and the array element is addressed using &&SUBJ&I.

The advantage of this approach is that the array elements can hold most anything and the number of elements in the array is limited only by the available memory. The disadvantage is that the number of macro variables can become quite enormous, especially if there are a number of macro arrays being constructed at the same time. An alternative approach, that is sometimes practical, is to store the array elements in a single macro variable instead of a series of macro variables.

The use of a single macro variable to hold an entire array of values is especially useful when the array values make up single words, and the complete list of elements is less than 64K characters. In fact the array elements don't even have to be single words as long as there is some character that can be used to distinguish or separate the array elements. The %SCAN function can then be used to identify and breakout the individual array elements.

For discussion purposes let's say that we would like to create one or more macro arrays that contain information about a SAS table. We can build a data set with this information by using PROC CONTENTS.

```
proc contents data=sashelp.class noprint out=metaclass;
   run;
```

The resulting data set (METACLASS) will have one row for each variable that was found in the original data set (SASHELP.CLASS).

```
options nocenter nodate;
```

```
title 'Meta Data for SASHELP.CLASS';
proc print data=metaclass;
   var name type length varnum;
   run;
```

Meta Data for SASHELP.CLASS

Obs	NAME	TYPE	LENGTH	VARNUM
1	Age	1	8	3
2	Height	1	8	4
3	Name	2	8	1
4	Sex	2	1	2
5	Weight	1	8	5

BUILD THE LIST IN A DATA STEP

While the macro arrays of the form &&VAR&i are often created in the DATA step using successive calls to the CALL SYMPUT routine, this technique is not as practical for building a single macro variable that is to contain all of the individual values. The primary problem is that the list is usually first built in a character variable which can only contain 32K characters (200 in V6 and before), before it is transferred to the macro variable.

In the following DATA _NULL_ step, the character variable ALLVAR is used to accumulate the list of variable names, and it is only after all the values are stored in ALLVAR is the macro variable (&VARLIST) created.

```
data _null_;
   length allvars $1000;
   retain allvars ' ';
   set metaclass end=eof;
   allvars = trim(left(allvars))||' '||left(name);
   if eof then call symput('varlist', allvars);
   run;

%put &varlist;
```

Because of timing issues between the macro language and the DATA step, it is more difficult to successively append values directly to the macro variable without first creating the intermediate DATA step variable (ALLVARS above). It can be done and the technique requires the use of the RESOLVE function, which is itself a bit tricky.

Once you understand the intricacies of the RESOLVE function, you can use it to avoid the creation of the intermediate variable and to help with the timing issues. Since the RESOLVE function accesses the macro symbol table at **DATA step execution**, it can retrieve values that were just added to the symbol table with the SYMPUT routine. The following example concatenates the list of variable names, one at a time, onto a single macro variable. Notice that the argument to the RESOLVE function is quoted with single quotes and contains an &. The single quotation marks are used to prevent the macro processor from resolving the argument before or while the DATA step is compiled.

```
%let varlist =;
data _null_;
   set metaclass;
   call symput('varlist',trim(resolve('&varlist'))||' '||trim(name));
   run;

%put &varlist;
```

CREATING THE LIST WITH SQL

The list of values can also be created using PROC SQL. The INTO modifier along with the colon (:) is used to identify the macro variables that are to be created. Here the list of variables, their type, and their individual lengths are stored in macro variables.

```
proc sql noprint;
   select name ,type, length
      into :varlist separated by ' ',
           :typlist separated by ' ',
           :lenlist separated by ' '
        from metaclass;
   quit;
%let cntlist = &sqlobs;
%put &varlist;
%put &typlist;
%put &lenlist;
%put &cntlist;
```

The `separated by` clause tells SQL to append succeeding values. In this code the word separator is a blank, however if you have values that contain blanks you would need to select a character that is not otherwise used.

PROC SQL automatically counts the number of observations that it processes and places that number in the macro variable &SQLOBS. In this case that number is also the number of array values in each of the lists. Knowing how many elements there are in a list is often helpful when that list is to be used element by element, consequently the value is saved in &CNTLIST. Be careful if you use &SQLOBS directly as the counter, since that value could change with each PROC SQL.

USING A MACRO LOOP

In each of the examples above we are not really looking at data, but really we are examining the meta-data (variable names, etc.) of the data set of interest. Consequently it is often possible in this type of situation to be able to avoid the DATA step and PROC SQL steps altogether. In the following macro, %GETVARS, which was written by Michael Bramley of Trilogy Corporation and is described more fully in Carpenter(2004), the macro %DO loop is used to access the meta data directly.

The macro %GETVARS is a macro function and returns the list of variables in a data set (&DSET). This macro makes use of the VARNAME function to return the variable name.

```
%Macro GetVars(Dset) ;
   %Local VarList ;
   /* open dataset */
   %Let FID = %SysFunc(Open(&Dset)) ; ❶
   /* If accessable, process contents of dataset */
   %If &FID %Then %Do ;
      %Do I=1 %To %SysFunc(ATTRN(&FID,NVARS)) ; ❷
        %Let VarList= &VarList %SysFunc(VarName(&FID,&I));❸
      %End ;
      /* close dataset when complete */
      %Let FID = %SysFunc(Close(&FID)) ; ❹
   %End ;
   &VarList ❺
%Mend ;
```

❶ The data set of interest is opened for inquiry.
❷ The ATTRN function with the NVARS argument returns the number of variables in the data set.
❸ The VARNAME function returns the name of the &Ith variable. This variable is appended to the growing list of variables stored in &VARLIST.
❹ Once the data set is no longer needed, it is closed.
❺ The list of variables is 'left behind' and effectively returned by the macro.

USING THE %SCAN FUNCTION

Since it is designed to return a specific word or element from a list of words, the %SCAN macro function is often used when you want to step through the list of elements in a macro list. The %SCAN function has an analogue version %QSCAN which returns a quoted value.

Both the %SCAN and %QSCAN functions have two required and one optional argument and they take the form of:

```
%scan(macro_var_list,  word_number,  word_delimiter)
```

where	macro_var_list	macro variable containing the list of elements or words
	word_number	the number associated with the word of interest
	word_delimiter	one or more characters (the default includes a blank) that are used to separate the words - this argument is optional if the default list is to be used.

The %SCAN function lends itself to the process of working with lists of array elements because the second argument (the word number) is numeric. If this word number is controlled through the use of an iterative %DO or a %DO %WHILE loop it is possible to step through this list as one would step through a list of array elements.

COUNTING THE WORDS IN A LIST

Very often before you can step through the list you will need to know how many elements or words it contains. Usually this number is saved when the list is first constructed, but this is not always the case. The %GETVARS macro just shown returns the list of variables but not the number of variables (although it calculates and uses the number internally).

The %WORDCOUNT macro, which uses the %QSCAN function, can be used to count the words in a list. This macro function returns the count and does not create any global macro variables.

```
%macro wordcount(list);
   %* Count the number of words in &LIST;
   %local count;
   %let count=0; ❶
   %do %while(%qscan(&list,&count+1❷,%str( )) ne %str()❸);
      %let count = %eval(&count+1); ❹
   %end;
   &count ❺
%mend wordcount;
```

❶ The count is initialized to 0.
❷ %QSCAN looks ahead to see if the next word (&COUNT + 1) exists.
❸ Check to see if there is a next word. If there is then enter the loop so that &COUNT can be incremented ❹.
❹ Since the &COUNT+1 word was found, increment the value of &COUNT.
❺ When the last word has been counted, exit the loop and return the number of words counted by passing it back.

The following use of %WORDCOUNT displays the number of variables in &VARLIST:

```
%put SASHELP.CLASS has %wordcount(&varlist) variables;
```

The LOG might show:

```
86
87      %put SASHELP.CLASS has %wordcount(&varlist) variables;
SASHELP.CLASS has 5 variables
```

STEPPING THROUGH THE LIST USING THE %SCAN FUNCTION

In the %WORDCOUNT macro the %QSCAN function is used to step through the list of words using a %DO %WHILE loop. Usually you will know the number of words that you are dealing with (or the number can be determined by using %WORDCOUNT) and you will be able to use the iterative %DO loop to step through the list.

This is done in the following macro %EMPTYDSN which is used to create a 0 observation data table with the variables (and the variable attributes) specified in the macro variables &VARLIST, &TYPLIST, and &LENLIST. A %DO loop steps through the &CNTLIST variables that are to be added and creates the entry in the LENGTH statement for each variable. Notice that, because all three uses of the %SCAN function use the same index (word number - &i), the entries in the three arrays are automatically coordinated .

```
%macro emptydsn(dsn);
data &dsn(keep=&varlist);
   length
   %do i = 1 %to &cntlist;
      %scan(&varlist,&i) %if %scan(&typlist,&i)=2 %then $; %scan(&lenlist,&i)
   %end;
```

```
   ;
   stop;
   run;
%mend emptydsn;

%emptydsn(dummyclass)
```

The macro %EMPTYDSN creates a 0 observation data set by generating and executing the following DATA step. The code generated by the macro language is in blue.

```
data dummyclass(keep=Age Height Name Sex Weight);
   length
       Age 8
       Height 8
       Name $ 8
       Sex $ 1
       Weight 8
   ;
   stop;
   run;
```

SUMMARY

Although the SAS macro language does not directly support the concept of an array, there are several ways to mimic a macro array. Two of the more common methods are to create lists of values. The elements in these lists can be stored in individual macro variables or as a list of values in a single macro variable. Once the list is created there are several ways to use the list. More commonly they are used inside of macro %DO loops such as %DO %WHILE, %DO %UNTIL, and the iterative %DO.

The lists of array elements can be gathered and stored using DATA step processing, PROC SQL, or even the macro language itself. Usually the number of elements in the array are noted and saved, but it is possible to create tools that will determine the number of elements in a list.

When the list of elements is to be stored in a single macro variable, the individual elements are retrieved using the %SCAN or %QSCAN macro functions. These functions allow the user to step through the list one element at a time, retrieve the value of interest, and to process it.

ABOUT THE AUTHOR

Art Carpenter's publications list includes three books, and numerous papers and posters presented at SUGI and other user group conferences. Art has been using SAS® since 1976 and has served in various leadership positions in local, regional, national, and international user groups. He 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 430
Oceanside, CA 92085-0430

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

REFERENCES

Carpenter, Art, 2004, *Carpenter's Complete Guide to the SAS® Macro Language 2nd Edition,* Cary, NC: SAS Institute Inc.,2004. Most of the examples in this paper were taken from this text.

Carpenter, Arthur L., 1997, *Resolving and Using &&var&i Macro Variables*, presented at the 22nd SAS User's Group International, SUGI, meetings (March, 1997) and published in the Proceedings of the Twenty-Second Annual SUGI Conference, 1997.

Carpenter, Arthur L. and Richard O. Smith, 2001, *Library and File Management: Building a Dynamic Application*, presented at the 9th Western Users of SAS Software Conference (September, 2001) and at the Pharmaceutical SAS User Group meetings, PharmaSUG, (May 2001).

Flavin, Justina and Arthur L. Carpenter, 2003, *Is the Legend in your SAS/GRAPH® Output Telling the Right Story?*, included in the proceedings and presented at: WUSS 11 (2003), PharmaSUG 2004, and SUGI29 (2004). This paper includes an example of a macro variable list and its use to construct SYMBOL statements.

TRADEMARK INFORMATION