Macro Arrays Make %DO-Looping Easy

Ted Clay, Clay Software & Statistics, Ashland, Oregon

ABSTRACT

The purpose of the macro array concept is to make it easier to repetitively execute SAS® code. A macro array is defined as a list of macro variables sharing the same prefix and a numeric suffix, such as AA1, AA2, AA3, etc., plus an additional macro variable with a suffix of "N" containing the length of the array. The %MACARRAY macro creates a macro array from an explicit list of values, or from a character or numeric variable in a data set. Then the fun begins. The %DO_OVER macro loops over the macro array, substituting macro array values wherever you put a "?" in your code. For example, $%DO_OVER(AA, phrase="?")$ generates a list with double-quotes around each element. Need to rename variables adding a suffix? RENAME $%DO_OVER(AA, phrase=?=?suf)$; Or store which variable is true? $%DO_OVER(AA, phrase=if? then x="?"; between=else)$. Or concatenate a series of data sets? SET $%DO_OVER(AA, phrase=mylib.?(in=in?))$; Or repeatedly execute an externally-defined macro? $%DO_OVER(AA, macro=mymacro)$. Multiple arrays can be defined and looped-over simultaneously. The combination of %MACARRAY and %DO OVER may become your macro equivalent of vice-grip pliers.

INTRODUCTION

This paper explains what a macro array is, how to put text into it, and how to loop (iterate) over the text values stored in it. Two macros, %MACARRAY and %DO_OVER, make the use of macro arrays painless and fun. Macro %DO-looping is common practice in many programs using the macro language. Unfortunately, the usual solutions involve somewhat awkward macro syntax, with double ampersands and macro definitions which can break up the continuity of programs. These two macros allow you to hide the repetitive machinery, resulting in programs that are shorter and more readable.

These macros significantly lower the threshold for using the macro language in powerful ways. The %MACARRAY and %DO_OVER macros are analogous to the ARRAY and DO OVER statements in the SAS data step language, which define and loop over implicitly subscripted arrays. Because these macros are self-contained and use global macro variables, you can use them freely in "open code". The macros use regular characters as much as possible, freeing you from the need for macro quoting functions.

QUICK OVERVIEW

A *macro array* is a list of macro variables sharing the same prefix and a numerical suffix. The suffix numbers run from 1 up to a highest number. The value of this highest number, or the *length* of the array, is stored in an additional macro variable with the same prefix, plus the letter "N". The prefix is also referred to as the *name* of the macro array.

Example: The macro array "DAYS" containing the first three days of the work-week.

Macro Variable Name	Contents
DAYS1	Monday
DAYS2	Tuesday
DAYS3	Wednesday
DAYSN	3

Most people who have used the macro language have created a list similar to this. By also storing the length of the array, to loop over the array you only need to specify the name of the array, in this case "DAYS."

THE SLOW WAY

The simple but tedious way to get data into a macro array is with %let statements.

```
%let DAYS1=Monday;
%let DAYS2=Tuesday;
%let DAYS3=Wednesday;
%let DAYSn=3;
```

Now suppose you want to concatenate 3 data sets, named for days of the week, stored in library MYLIB. You need to generate the following set statement:

```
Set mylib.Monday
  mylib.Tuesday
  mylib.Wednesday;
```

The macro to produce it would look like this:

THE FAST WAY

Using the %MACARRAY and %DO_OVER macros, all of the above can be written in just two lines:

```
%MACARRAY(days, VALUES=Monday Tuesday Wednesday)
set %DO_OVER(days, PHRASE=mylib.?);
```

Notice the question mark "?" in the PHRASE parameter. The %DO_OVER macro substitutes the days of the week for the question mark. Furthermore, we can use the VALUES= parameter of %DO_OVER to use just one line:

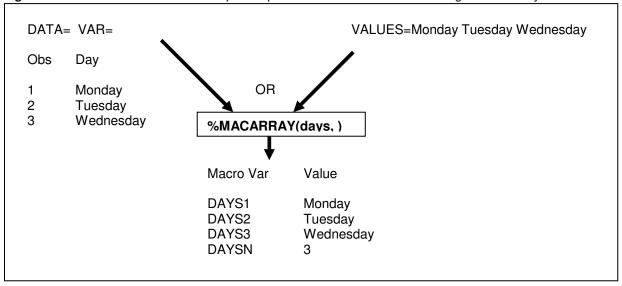
```
set %DO_OVER(VALUES=Monday Tuesday Wednesday, PHRASE=mylib.?);
```

Next we explain the syntax and capabilities of the %MACARRAY and %DO OVER macros.

THE %MACARRAY MACRO

The purpose of the %MACARRAY macro is to create a macro array and load data into it from either a SAS data set or an explicit list of values.

Figure 1: The %MACARRAY macro accepts two possible sources of data for creating a macro array:



SYNTAX

The syntax for %MACARRAY reads as follows:

```
%MACRO MACARRAY(arraypos, array=, data=, var=, values=, delim=%str());
```

Parameter	Definition	
ARRAYPOS	Positional alternative to the ARRAY keyword parameter.	
ARRAY	Name(s) for the macro array(s) to be defined.	
DATA	Dataset or view source of text data. Data set options are allowed. If this parameter is used, the MACARRAY macro must be called between other steps of the program.	
VAR	Character or numeric variable(s) containing values to put in the array(s). Required with DATA=. If multiple arrays are defined, the VAR= list must correspond one-for-one with the macro names given in the ARRAY or ARRAYPOS parameter.	
VALUES	An explicit list of character strings to put in the array. It is an alternative to the DATA parameter. It can only be used when defining a single array. Using this parameter, the macro can be placed anywhere in your program, because no SAS code is executed.	
DELIM	A single-character separator for parts of the VALUES parameter. Default=space.	

The %MACARRAY macro starts with the ARRAY or ARRAYPOS parameter, which gives the name of the macro array being defined, or names of the multiple arrays. %MACARRAY(array=x,etc.) is absolutely equivalent to %MACARRAY(x,etc.). Having both a positional and keyword version of the same parameter gives you the choice of how you want to use this macro. Some prefer the brevity of the positional parameter, while others may have a policy of always using keyword parameters for clarity. In this paper we consistently use only the positional parameter.

Next comes either the DATA parameter or the VALUES parameter, depending on where the array contents are coming from. If the DATA parameter is used, the VAR parameter is required. The DELIM parameter is used when the VALUES parameter contains anything other than blanks to separate the values.

EXAMPLES

Suppose we have an existing data set called THISWEEK.

Proc Print of THISWEEK data set

Obs	Day	Flavor	Name
1	Monday	Chocolate	Alice
2	Tuesday	Raspberry	Bobby
3	Wednesday	Rocky Road	Cindy

Example 1: Create a single macro array using data in a data set.

%MACARRAY (days, DATA=thisweek, VAR=Day);

Example 2: Create a single macro array using an explicit list of values.

%MACARRAY(days, VALUES=Monday Tuesday Wednesday)

This example, as seen before, creates the same result as Example 1.

Example 3: Create multiple macro arrays using data in a data set.

%MACARRAY(days kids flavors, DATA=thisweek, VAR=Day Name Flavor)

The values of the variable "Day" become the values of macro array "days", the values of the variable "Name" become the values of macro array "kids", and Flavors loads the "flavors" array.

Example 4: Create a macro array with VALUES requiring the DELIM parameter.

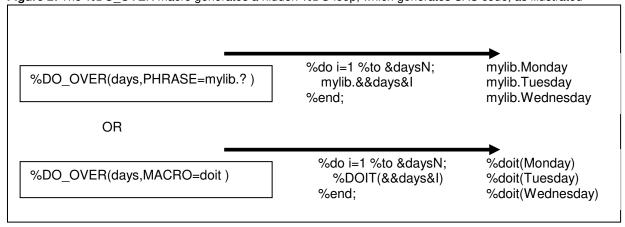
%MACARRAY(flavors, VALUES=Chocolate/Raspberry/Rocky Road, DELIM=/)

The blank space in "Rocky Road" created the problem.

THE %DO OVER MACRO

The purpose of the %DO_OVER macro is to execute SAS code repetitively, inserting macro array values where you specify. The SAS code can be either internal, in the PHRASE= parameter, or external, in the macro named in the MACRO= parameter.

Figure 2: The %DO OVER macro generates a hidden %DO loop, which generates SAS code, as illustrated



In the discussions that follow, we will concentrate on the use of the PHRASE= parameter.

SYNTAX

The syntax for %DO_OVER reads as follows:

Parameter	Definition		
ARRAYPOS	Positional alternative to the ARRAY keyword parameter		
ARRAY	Name(s) for the macro array(s) to iterate over. If multiple macro arrays are given, they must have the same length.		
PHRASE	SAS code into which to substitute the macro variable values. The PHRASE parameter may contain semicolons and extend to multiple lines. The default value of PHRASE is a single . (See Example 5 for the default case.) Single Array Case: Put a "?" (the ESCAPE character) wherever you want the macro values to be placed. Optionally, the "?" can be followed by the name of the macro array, for clarity. Multiple Array Case: Put a "?" immediately followed by the name of the macro array. The macro will replace the "?" and name with the value of that particular macro array. References to macro names are case sensitive and must agree with the ARRAY parameter values. Obtaining the Loop Index: Put "?_I_" wherever you want the value of the looping index to appear. It is replaced by "1", "2", "3", etc. up to the length of the array.		
ESCAPE	A single character to be replaced by macro array values. Default is "?". If followed by a macro array name, it plus the array name are replaced by the array values.		
BETWEEN	Code to generate between iterations of the main phrase or macro. Because there is often a need to put a comma between elements of an array, the special parameter value COMMA is recognized for programming convenience. BETWEEN=COMMA is equivalent to BETWEEN=%STR(,).		
MACRO	The name of an externally-defined macro to execute on each value of the array. It overrides the PHRASE parameter. The macro must have positional parameters defined, in the same order and meaning as the macro arrays specified in the ARRAY or ARRAYPOS parameter.		
VALUES	An explicit list of character strings to iterate over. This parameter is an alternative to giving an array name in the ARRAYPOS or ARRAY= parameter. A single hidden internal array is created and used. When VALUES= are given, the PHRASE parameter may only use single "?" to be replaced by macro array values. If a MACRO= is supplied, it must be a single-parameter macro.		
DELIM	A single-character separator for parts of the VALUES parameter. Default=space.		

The %DO_OVER macro, like its companion %MACARRAY, starts with the ARRAY or ARRAYPOS parameter, which gives the name of the macro array being looped-over, or names of the multiple arrays. %DO_OVER(x,etc.) is absolutely equivalent to %DO_OVER(array=x,etc.). As an alternative to ARRAY or ARRAYPOS, you may use the VALUES= parameter to supply explicit values. Next, you must specify either the PHRASE= or MACRO= parameter to define your SAS code. Finally, the BETWEEN parameter lets you specify code that will be generated between each repetition of the PHRASE code or the external MACRO code.

EXAMPLES

Example 5: Using the default PHRASE value, which is a single question-mark.

%DO_OVER(kids)

generates

Alice Bobby Cindy

This macro execution can be used inside a macro assignment statement, creating an easy way to get one macro variable with all the kids names. Recall from Example 3 that these names were originally in a data set. Many times the author has written dozens of lines to accomplish the following:

```
%MACARRAY(kids,DATA=thisweek,VAR=Name)
%let allkids=%DO_OVER(kids);
```

Example 6: A simple PHRASE with one macro array used one time.

In this line of code in a data step, we are testing whether the value of the character variable NEWKID is found in our list of kids stored in the macro array.

```
if newkid in (%DO_OVER(kids,PHRASE="?") ) then put newkid 'found in list';
```

Example 7: PHRASE with ? plus the array name.

```
Same as above, just that we have put "?kids" instead of "?". The purpose is program clarity.

if newkid in (%DO_OVER(kids,PHRASE="?kids")) then put newkid `found in list';
```

Example 8: PHRASE with one macro array used in two places.

```
Shows renaming a macro array "demog" of demographic variables ("age" "sex" "race") to all have the suffix "_1995".

RENAME %DO_OVER (demog, phrase=?=?_1995);

which generates code like this:

Age=age_1995 sex=sex_1995 race=race_1995

Or suppose we want to use the same variables to create a data set for each frequency table from Proc Freq

Proc freq data=mydata;

%DO_OVER (demog, PHRASE=table ? / out=?;)

Run;

which generates code like this:

Table age / out=age;

Table sex / out=sex;

Table race / out=race;
```

Example 9: Using VALUES=, and PHRASE containing a reference to the Array Index

Recalling the example of concatenating data sets containing the days of the week, suppose we wanted to keep track of which data set a given observation came from. The following code would create variables IN1, IN2 and IN3 for the three input data sets. This example also illustrates the use of the VALUES= parameter.

```
Set %DO_OVER(VALUES=Monday Tuesday Wednesday,PHRASE=mylib.?(in=in?_I_));
```

The string "?_I_" gets replaced by "1", "2", "3". Notice that no macro array is named -- only the values appear.

Example 10: PHRASE using multiple arrays

Suppose we already have defined macro arrays VNAMES and LBLS containing a list of variable names and labels. To generate the LABEL statements:

```
%DO_OVER(vnames lbls,PHRASE=label ?vnames="?lbls";) for separate label statements LABEL %DO_OVER(vnames lbls,PHRASE=?vnames="?lbls"); for one big label statement.
```

Notice that you are required to put the array names after the "?" to show where you want each array's values to go.

Example 11: MACRO parameter with a single array.

The most common reason for using the MACRO parameter is that the quantity of SAS code that you want to repeatedly execute becomes too large. Suppose, for example, you want to execute a long series of statistical procedures on a variable, then execute the same series of steps on another variable, etc. perhaps putting the name of the current variable in a title statement. Using the same VNAMES macro array used in Example 10, we have

```
%MACRO DOSTATS(var);
Title "Analyses of variable &VAR";
Proc means data=x;
   var &VAR;
Proc Reg data=x;
   model count = age sex &VAR;
run;
%mend;
%DO_OVER(vnames,MACRO=dostats);
```

Example 12: MACRO parameter with multiple arrays.

The example is the same as above, except that we also wish to put the variable labels in the title.

```
%MACRO DOSTATS(var,lbl);
Title "Analyses of variable &VAR which is &LBL";
Proc means data=x;
   var &VAR;
Proc Reg data=x;
   model count = age sex &VAR;
run;
%mend;
%DO_OVER(vnames lbls,MACRO=dostats);
```

In this case it is critical that the order of the macro array names ("vnames lbls") be the same as the order of the corresponding positional parameters in the external macro ("var,lbl"). They do not have to be the same names.

HOW IT WORKS

The following is the essential code from the %DO_OVER macro. Comments and parameter-checking logic have been removed. The fully robust version of this and %MACARRAY is available from the author.

```
%MACRO DO_OVER(arraypos, array=, phrase=?, escape=?, between=, macro=,
                   values=, delim=%str());
%LOCAL prefixes MAnum MAcount arraynotfound j did frc crc iter i TP;
%* Get macro array name (prefix) from either keyword or positional parameter;
        %str(&arraypos) ne %then %let prefixes=&arraypos;
%else %if %str(&array) ne %then %let prefixes=&array;
%else %if %str(&values) ne %then %let prefixes=_Internal;
                                                                                   (1)
%else %put ERROR: No macro array name or values specified for DO_OVER macro;
%* Parse the macro array names;
%do MAnum = 1 %to 999;
%let prefix&MANUM=%scan(&prefixes,&MAnum,' ');
                                                                                  (2)
%if &&prefix&MAnum ne %then %let MAcount=&MAnum;
%else %let MAnum=1000;
%end;
%* Load the VALUES into macro array 1 (only one is permitted);
%if %length(%str(&VALUES)) >0 %THEN
%do iter=1 %TO 9999;
 %let val=%scan(%str(&VALUES),&iter,%str(&DELIM));
                                                                                   (3)
 %if %str(&VAL) ne %then
     %let &PREFIX1&ITER=&VAL;
                                                                                   (4)
     %let &PREFIX1.N=&ITER;
                                                                                   (5)
    %end;
  %else %let iter=10000;
%end;
%if %quote(%upcase(&BETWEEN))=COMMA %then %let BETWEEN=%str(,);
%if %length(%str(&MACRO)) ne 0 %then %do;
     %let TP = %nrstr(%&MACRO)(;
     %do J=1 %to &MACOUNT;
         %let currprefix=&&prefix&J;
         %IF &J>1 %then %let TP=&TP%str(,);
                                                                                  (6)
        %LET TP=&TP%nrstr(&&)&currprefix%nrstr(&I);
         %END;
     %let TP=&TP);
     %end:
%else %do;
     %let TP=&PHRASE;
     %let TP = %qsysfunc(tranwrd(&TP, &ESCAPE._I_, %nrstr(&I.)));
                                                                                  (7)
     %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE._i_,%nrstr(&I.)));
     %do J=1 %to &MACOUNT;
         %let currprefix=&&prefix&J;
                                                                                  (8)
         %LET TP = %gsysfunc(tranwrd(&TP,&ESCAPE&currprefix,
                                    %nrstr(&&)&currprefix%nrstr(&I..)));
                                                                                  (9)
         %if &MACOUNT=1 %then %let TP = %qsysfunc(tranwrd(&TP,&ESCAPE,
                                    %nrstr(&&)&currprefix%nrstr(&I..)));
         %end:
    %end;
%do I=1 %to &&&prefix1.n;
%if &I>1 and %length(%str(&between))>0 %then ≬
                                                                                 (10)
%unquote(&TP)
%end;
%MEND;
```

WALKING THROUGH THE MACRO EXECUTION

Using Example 9.

```
Set %DO_OVER(VALUES=Monday Tuesday Wednesday,PHRASE=mylib.?(in=in?_I_));
```

- (1) The macro detects that the VALUES parameter is used. The automatic macro array name "_Internal" is inserted.
- (2) The names of the macro arrays are parsed and counted. In this case, there is only one name ("Internal").
- (3) The text of the VALUES parameter is parsed.
- (4) The individual values are assigned in turn to new macro variables Internal1, Internal2, etc.
- (5) The length of the array is stored in a macro variable ending with "N". In this case, _InternalN=3.
- (6) If the MACRO= parameter had been used, this section would be used, but is not used in this example.
- (7) In processing the PHRASE parameter, first any instances of "?_I_" (in upper or lower case) are replaced by the string "&I.". The value of TP (Translated Phrase) is mylib.?(in=in&I.). Eventually the &I resolves to "1", "2", "3", etc.
- (8) Next, the macro translates any instances of "?" followed by a macro name. This would be necessary when more than one macro array name is being used, or is an option when only one array name is used. It would not be used in this example, which uses the automatic array name created with the VALUES= parameter.
- (9) Because it is true that we are using only one macro array, MACOUNT=1 and this statement executes. It translates any instance of "?". The value of TP (Translated Phrase) becomes:

 mylib.&&_Internal&I..(in=in&I.) If the original %DO_OVER call had used %DO_OVER(days,) instead of the VALUES parameter, the value of TP would have been: mylib.&&days&I..(in=in&I.)

A CASE STUDY OF CODE SHRINKAGE

We start with a data set called "mydata.population" containing geographical areas and their populations. The population counts are categorized into a "large number" of numeric variables (nine of them). Our task is to obtain new variables with the percent that each count variable is out of a Total variable. We want the new variable names to be the old variable name plus "_pct".

VERSION 1: ORIGINAL OLD-STYLE CALCULATION OF PERCENTS;

The above program has the disadvantage that the "_pct" suffixes had to be manually typed into the program. Perhaps somewhere in the middle of typing this list, you would realize this is a perfect application for a macro array.

VERSION 2: USE %DO_OVER TO ADD THE "_PCT" SUFFIX;

The above accomplished your purpose, which was to make it easy to put on the variable suffix "_pct", but there is no need for using data step ARRAY statements and a DO loop. Let macro arrays and macro %DO-looping accomplish the same task.

VERSION 3: ELIMINATE THE DATA STEP ARRAY STATEMENTS:

Suppose you had no other reason to create the "counts" macro array. You can bypass the need for creating a macro array by feeding the "VALUES=" list directly to the DO_OVER macro.

VERSION 4: NO NEED FOR USING THE MACARRAY MACRO;

An internal macro array is automatically created and its values are used to replace the "?" where it occurs in the phrase. This achieves the maximum code reduction and program clarity.

CONCLUSION

The concept of macro arrays closes an important gap in the SAS system. The %MACARRAY macro strongly links the world of data to the world of macro variable values. Once defined, you do not have to be concerned with how many elements a macro array contains, only referring to the macro array by its name. Then, in a style similar to implicit arrays and the "DO OVER" statement in the SAS data step language, the macro %DO_OVER allows you to execute SAS code plugging in the values from macro arrays. Feel free to take advantage of these macros. They extend the power and scope of the SAS programming language.

ACKNOWLEDGEMENTS

David Katz, David Katz Consulting, for the macro array concept and its use with an externally defined macro. Art Carpenter, California Occidental Consultants, for aid in coding at the 2003 WUSS Code Clinic.

ABOUT THE AUTHOR

Ted Clay, M.S. is a statistical consultant and data analyst. His clients have included pharmaceutical companies, manufacturing companies, and grass-roots organizations, as well as research projects in epidemiology and health policy at the University of California San Francisco.

Your comments and questions are valued and encouraged. Contact the author at:

Ted Clay Clay Software & Statistics 168 Meade St. Ashland, OR 97520 Work Phone: 541-482-6435

Fax: Same

Email: tclay@ashlandhome.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.