PharmaSUG 2020 - Paper AP-093

One Macro to create more flexible Macro Arrays and simplify coding

Siqi Huang, Boehringer Ingelheim Pharmaceuticals, Inc.

ABSTRACT

The purpose of using macro array is to make it easier to repetitively execute SAS® code. Macro array is defined as a list of macro variables sharing the same given prefix and a numeric suffix, such as A1, A2, A3, etc., plus an additional macro variable with a suffix of "N" containing the length of the array.

In this paper, I will introduce a %MAC_ARRAY macro, which provides a more flexible way to create a macro array from any given list of values, or from any selected variable in a dataset, either character or numeric.

The application of this macro array is broad as well, including but not limited to: 1) creating similar datasets; 2) stacking multiple datasets; 3) repeating same calculation among multiple variables in the datasets; 4) automatically updating parameters used in other macros. To sum up, %MAC_ARRAY macro can easily keep your code neat and improve program efficiency.

INTRODUCTION

As programmers, you would face many situations where you have to write repeated codes or have to execute commands with similar patterns. For example, sometimes you need to append multiple datasets and typing the dataset names individually would be tedious.

Also in clinical trials, you may have the same analyses applied to multiple endpoints, writing a macro may save you a lot of time and energy by avoiding repeated coding. However, with many endpoints, you still want an easier way to update the parameters in the macro automatically instead of calling the macro multiple times. Therefore, a standard macro has been developed to hide the repetitive machinery and to create shorter and more readable programs.

In this paper, I will introduce a standard and simple SAS® macro on functionality, parameters and usage.

MACRO CALL AND ITS PARAMETERS

You can call simply this macro by using the following command:

```
%mac_array (array_name =, values =);
```

Where:

array_name	Name for the macro array to be defined **Required:** Yes **Default:** No default value
values	An explicit list of character strings to put in the array (separated by space) **Required:** Yes **Default:** No default value

MACRO USAGE

This macro can create macro arrays more flexibly compared to the traditional one, which assigns the array values individually.

For example, if we have below content:

Macro variable name	Macro variable value
Fruit1	apple
Fruit2	pear
Fruit3	orange
Fruit4	banana
nFruit	4

Table 1. Sample content for creating a macro array

In the following sections, you will find how we create the macro array in two ways.

TRADITIONAL WAY

In the traditional way, you usually need following commands:

```
<%global Fruit1 Fruit2 Fruit3 Fruit4 nFruit;>
%let Fruit1 = apple;
%let Fruit2 = pear;
%let Fruit3 = orange;
%let Fruit4 = banana;
%let nFruit = 4;
```

This process would be annoying when we have a long list of values to populate into the macro array.

THE MORE EFFICIENT WAY

There are different ways to create a macro array and assign values to the variables by using this new method.

1. Get values from an explicit list and get arrays automatically

   To create the same macro variables as above, this time you can simply call the macro %MAC_ARRAY:

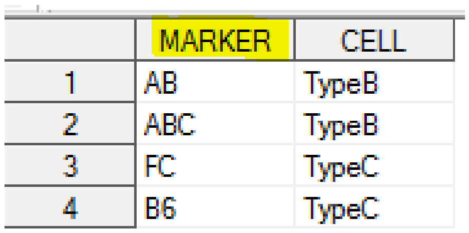
   ```
   %mac_array(array_name = Fruit, values =apple pear orange banana);
   ```

   In this case, same macro variables will be created with one command instead of typing 5 lines of commands (if global macro variables are needed, then 6 lines); and it can save more energy when the list of values is longer.

2. Get values from given columns

   By using this macro, we can also populate the values from a dataset into the array, together with SELECT INTO:

   For example, we have the following dataset:



	MARKER	CELL
1	AB	TypeB
2	ABC	TypeB
3	FC	TypeC
4	B6	TypeC

Figure 1. Sample dataset with content to create a macro array

Now you can create a macro array with the values in column "marker" with following code:

```
proc sql noprint;
      select distinct marker into: markers separated by " "
```

```
        from testdata;
quit;

%mac_array(array_name = marker, values =&markers.);Fruit1 = apple;
```
In this case, below macro variables will be created:

Macro variable name	Macro variable value
marker1	AB
marker2	ABC
marker3	FC
marker4	B6
nmarker	4

Table 2. Sample result for a macro array

APPLICATION

In this section, I will illustrate four examples to help you better understand how to combine this macro tool with you daily tasks.

To be noticed, the usage of %MAC_ARRAY should be embedded within a higher level macro to enable the use of %do loop.

EXAMPLE 1: CREATE DATASETS WITH THE SAME STRUCTURE

In the first example, I will show you how to create 5 datasets named from tDay01 to tDay05. Each dataset will contain one variable "Day", which equals to the day information in the dataset name. You will use %DO LOOP outside DATA STEP to increase efficiency:

```
%macro create_ds();
    %mac_array(array_name = day, values =1 2 3 4 5);
    %do i=1 %to &nday.;
          data tDay0&&day&i..;
                Day=&&day&i..;
          run;
    %end;
%mend;
%create_ds();
```

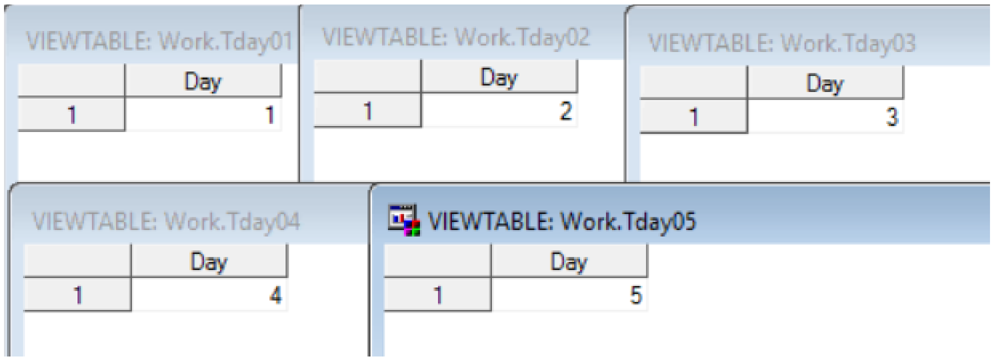
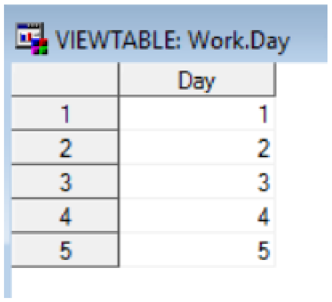


Figure 2 Output datasets for Example 1

EXAMPLE 2: STACK MULTIPLE DATASETS WITH SIMILAR NAMES

In the second example, you can use below code to stack the 5 datasets created in the above example. %DO LOOP is embedded within DTAT STEP this time:

```
%macro stack_ds();
    data day;
          set
          %do i=1 %to &nday.;
                tDay0&&day&i..
          %end;
          ;
    run;
%mend;
%stack_ds();
```



VIEWTABLE: Work.Day

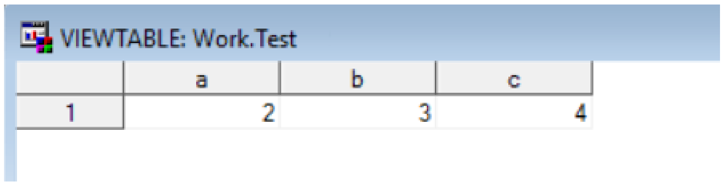
	Day
1	1
2	2
3	3
4	4
5	5

Figure 3 Output dataset for Example 2

Since the macro variables created by %MAC_ARRAY are global, you just need to call the macro once in the same program.

EXAMPLE 3: APPLY SAME OPERATION AMONG SELECTED VARIABLES

Suppose you have a dataset "TEST" with three variables as shown below:



VIEWTABLE: Work.Test

	a	b	c
1	2	3	4

Figure 4 Sample dataset "TEST"

Now you can create a new dataset "TEST2", which contains additional variables:

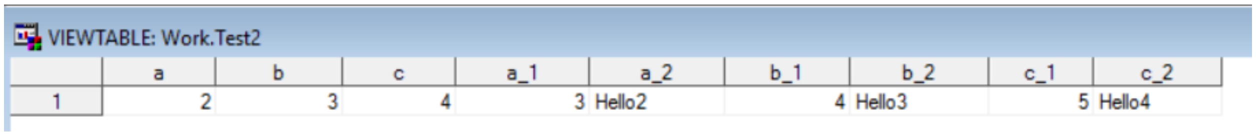
- A1, B1 and C1, which equal to (A+1), (B+1) and (C+1)
- A2, B2, and C2, which have prefix "Hello" followed by the value of A, B and C

with below code:

```
%mac_array(array_name = column, values=a b c);

%macro op();
data test2;
    set test;
    %do i=1 %to &ncolumn.;
          &&column&i.._1=&&column&i..+1;
          &&column&i.._2=cats("Hello",&&column&i..);
    %end;
run;
%mend;

%op();
```


VIEWTABLE: Work.Test2

	a	b	c	a_1	a_2	b_1	b_2	c_1	c_2
1	2	3	4	3	Hello2	4	Hello3	5	Hello4

Figure 5 Output dataset for Example 3

EXAMPLE 4: UPDATE PARAMETERS IN EXISTED MACROS AUTOMATICALLY

Assuming you already have a macro to generate a summary table for certain parameters:

```
 %macro sum_table(param=);
   proc summary data=sampledata print;
         var &param.;
   run;
%mend;
```

To generate reports for all candidate parameters, you have to call the macro multiple times by typing them individually and change the parameters manually in traditional way.

However, with %MAC_ARRAY, you can simply type the command once and SAS® will help you to call the macro with the parameters updated automatically. Shown as below:

```
%mac_array(array_name = par, values =age height weight);
%macro run_sum();
    %do i=1 %to &npar.;
          %sum_table(param=&&par&i..);
    %end;
%mend;
%run_sum();
```

Alternatively, with few efforts, we can modify the original macro and make the code more efficient:

```
%macro sum_table(param=);
    %mac_array(array_name = par, values =&param.);
    %do i=1 %to &npar.;
          proc summary data=sampledata print;
                var &&par&i..;
          run;
    %end;
%mend;
```

CONCLUSION

The %MAC_ARRAY macro offers a more convenient way to create macro arrays, which enable you to explode the functions of SAS® Macro to a larger extent. Along with the use of %do loop, programmers can save more time and energy from redundant work and make life easier.

APPENDIX

Source code for %MAC_ARRAY set-up:

```
%macro mac_array(array_name = ,  values = );
    %let i =1;
    %let value = %scan((&values.), &i.);

    %do %until (&value. = );
          %global &array_name.&i;
          %let &array_name.&i = &value.;
          %let i = %eval(&i. +1);
```

```
            %let value = %scan((&values.), &i.);
        %end;
        %global n&array_name.;
        %let n&array_name. = %eval(&i. - 1);
    %mend;
```

ACKNOWLEDGMENTS

I would like to gratefully acknowledge the valuable feedback and support from my colleagues at Department of Biostatistics and Data Science in Boehringer Ingelheim inc, and special thanks goes to Zhuoran Wu for her support throughout this project.

CONTACT INFORMATION

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

Siqi Huang
Boehringer Ingelheim Pharmaceuticals, Inc.
Siqi.huang@boehringer-ingelheim.com