Array: Construction and Usage of Arrays of Macro Variables

Ronald Fehd

Centers for Disease Control, and Prevention, Atlanta GA USA

ABSTRACT

The SAS® software data step statement `array V (3) $ V1-V3 ('A' 'B' 'C');` produces three character variables named V1, V2, and V3 with corresponding initial values, 'A', 'B', and 'C' and a function, `dim(V)` which returns a value of 3. Programmers can write simple macro tools for use in larger macro procedures. These tools can duplicate SAS software data step constructions that the programmer is comfortable using and make reading and comprehension easier. The macro statement `%Array(V,A B C)` produces three macro variables, V1, V2, and V3, with corresponding values: A, B, and C and macro variable `Dim_V` with the value 3. These variables can then be used in the macro iterative loop statement `%Do I = 1 %to &Dim_V.;` This paper examines the SAS data step array statement and discusses the issues in constructing and using arrays of macro variables. The macro `Array` takes parameters of either a list of elements or a data set.

Macro Array is a basic utility used in two other macros that address analysis of multiple-response data. See [3], [4].

INTRODUCTION

A common task for an experienced programmer is to recognize a recurring pattern of code and encapsulate that pattern in a routine which simplifies the processing presentation, while still enabling later readers of the program to grasp the complex concepts that have been coded.

The SAS software macro language is a simple yet powerful programming language. This article examines the SAS software array and associated do loop statements with the idea of translating those concepts into SAS software macro language usage.

SAS ARRAY STATEMENT AND THE DIMENSION (DIM) FUNCTION

The explicit array statement in SAS software has seven phrases; we will examine the four that are most commonly used:

1. `array`: SAS statement key-word

2. `array name`: required

3. `subscript`: required, either of a number or asterisk, enclosed in parenthesis

   - `(3)`: a number supplied for creation of a series of variables
   - `(*)`: asterisk indicating subscript is determined by SAS software by counting the supplied array-elements

4. `array elements`: an optional list of variable names

The dimension function — `dim(ABC)` — has two parts:

1. `dim` is the SAS function name

2. `parameter` is an array name defined in the same data step

A typical usage of the array statement would consist of accessing one set of variables in order to repeat some processing on each variable. The example in `Program 1` below reads in three Fahrenheit temperatures and converts them to Celsius. Note that the `PROC Contents` listing shows that SAS has created a series of variables based on the absence of the array-elements in the `array Celsius` statement. Their names — `Celsius1`, `Celsius2`, and `Celsius3` — correspond to the way the variables are accessed in the iterative loop by the array convention of `Celsius(I)`, i.e., `Celsius(1)`, `Celsius(2)`, and `Celsius(3)`.

```
Program 1
DATA  Temperatures;
array Celsius  {3};*note no array-elements, see Contents;
array Farnheit {*} Low Med Hi;
drop  I;
input Low Med Hi;
do I = 1 to dim(Farnheit); Celsius{I} = (Farnheit{I}-32) * 5/9;
                           end;
cards;*<deleted for brevity>;
PROC Contents;


- - - SAS output: - - -
#    Variable    Type    Len    Pos
-    --------    ----    ---    ---
4    Celsius1    Num      8     24
5    Celsius2    Num      8     32
6    Celsius3    Num      8     40
3    Hi          Num      8     16
1    Low         Num      8      0
2    Med         Num      8      8
```

SAS SOFTWARE MACRO LANGUAGE ITERATIVE LOOP

To replicate the SAS software iterative loop in the macro language we use a sequentially numbered series of macro variables and a macro variable containing the dimension:

```
%Let Var1 = Q04A;
%Let Var2 = Q04B;
%Let Var3 = Q04C;
%Let Dim_Var = 3;
```

The macro iterative loop and usage of the macro variables can then be written in a form that is visually similar to the SAS software iterative loop.

```
%Do I = 1 %to &Dim_Var.; %Put Var&I. :: &&Var&I.; %end;
```

This loop writes the following note to the SAS log:

```
Var1 :: Q04A
Var2 :: Q04B
Var3 :: Q04C
```

This is a construction used regularly in certain types of macros. The purpose of this paper is to construct a macro that supports this iterative loop. Such a macro would be named `Array`, and would have two of the SAS array statement phases as parameters: array name, and array-element values. This macro would return a sequentially-numbered series of macro variables and the dimension of the array. The array-element values could be either a provided list or the values of a variable in a data set. This second option of providing the array-element values in a data set would enable macro procedures to be completely data-driven. A `where` parameter has been added to facilitate subsetting of data sets used as parameters. See [3], [4] for examples.

PARAMETERS AND CONSTRAINTS

The simplicity of the macro language both allows and requires construction of a routine that has the appearance of the SAS software `array` statement. Since this is a routine and not a SAS software implementation, there are relations among the parameters that are constraints.

The first and most obvious is that the array-name parameter must follow SAS naming conventions. SAS names may be up to 32 characters in length. For this routine, some number of characters must be reserved for the sequential

numbering of the suffix. As the magnitude of the number of array elements increases, the length of the array name must decrease in order for the combined length to be less than or equal to 32.

A second constraint on the array name parameter is that the macro variable used for the dimension has the form: `Dim_<array-name>`. This construction was chosen to appear visually similar to the usage of the dimension function: `dim(<array-name>)`. This convention reduces the length of the array-name as suffix to 28 characters.

The array-name parameter is both prefix and suffix. As suffix to the name of the returned value of dimension, it can be no more than 28 characters in length. As prefix to the series of macro variables 28 characters in the array name allows a maximum of 9,999 sequentially numbered macro variables to be created without suffering a `SAS name too long` error. For larger arrays, the length of the array name can be as small as one character.

Array elements in the SAS software data step array statement are assumed to be delimited by spaces. When array-element values are provided to this routine as a list, the macro `%scan` function is used to pick out each value. The delimiters of the macro function `%scan` are the set of non-alpha-numeric characters. For special cases where, for instance, an array-element value may contain two or more words, the delimiter parameter may be supplied.

A data set and variable name may be supplied as parameters, instead of a list. This routine was written to handle various series of variable names, which were subsets of a `PROC Contents` output data set. Review the test data supplied with the macro.

CASE 1: SCANNING MACRO VALUES FROM A LIST

The macro function `%scan` operates the same as the SAS software function. In order to construct a loop which has a data-dependent termination, it is necessary to use and test a temporary variable for the exit condition. Here is pseudo-code and macro statements for a loop that converts a list to array elements:

```
initialize: I := 1                    %Let I = 1;
        get I-th Item from List  %Let Item    = %scan(&List,&I);
loop:                                 %Do %until(&Item. = );
        assign Item              %Let &Name.&I = &Item.;
        increment I              %Let I       = %eval(&I + 1);
        get I-th Item from List  %Let Item    = %scan(&List,&I);
until      Item is blank          %end;
assign dimension                      %Let Dim_&Name. = %eval(&I - 1);
```

Whereas the pseudo-code shows that the test is done at the bottom of the loop, SAS attaches the `%until` function to the iterative `%Do` at the top of the loop. The index is incremented using the `%eval` function. At the loop exit the index is off by one; the dimension is therefore `index - 1`.

CASE 2: SQL SELECT INTO MACRO VALUES FROM A DATA SET VARIABLE

The SQL select into statement can allocate as many macro variables as there are rows in the data set. Here is an example of the basic syntax:

```
PROC SQL noprint;
        select <variable name>
        into   :Name1 - :Name9999
        from   <<libref.>><data set name>
        ;quit;
        %Let   Dim_Name = &SqlObs.;
```

This usage replaces the symput function which was used to transfer values from a data set variable to the macro environment. See [2] for an explanation of symput usage to create a macro array.

USAGE OF %ARRAY IN OTHER MACROS

The code for creating a macro array from a list was first written as part of the `%CheckAll` macro. This macro analyzes multiple-response data, a series of variables which contain answers to survey questions with the instructions 'check all that apply'. After typing in hundreds of variables as lists for the various series, I wrote the second section which uses a previously prepared subset of a `PROC Contents` data set. This addition allows both research and production usage of the `%CheckAll` macro. See Fehd [3], [4] and test data with the macro.

DiIorio [1] discusses macro arrays of data set names.

CONCLUSION

The SAS software `array` and `do` statements are a simple programming tool which allow a programmer to access a list of variables. The macro language allows a programmer to access a list of items with a `%Do;` statement but lacks a specific `%Array` statement. This paper has presented a macro array which converts either a list or values of a variable into a sequentially-numbered series of macro variables with common prefix and sequential numeric suffix and also returns a macro variable with the dimension. This routine hides complexity and simplifies readability of programs which contain macro loops.

The SAS software macro language is a simple language. Its simplicity leaves many advanced programming concepts apparently unavailable. Its simplicity is an asset in that, with some forethought and planning, generic tools can be relatively easily written. This macro was initially developed to take a list of variable names as a parameter. After some usage it became apparent that adding the option to accept a data set as parameter would eliminate tedious typing of the variable lists, and, in addition, since the routine was then data-driven, guarantee the accuracy of the data thus processed.

REFERENCES

[1] DiIorio, Frank (1996), *MACArray: a Tool to Store Dataset Names in a Macro 'Array',* Proceedings of the Fourth Annual Conference of the SouthEast SAS Users Group, 1997.

[2] Fehd, Ronald (1997), *%Array: construction and usage of arrays of macro variables,* Proceedings of the 22nd Annual SAS[®] Users Group International Conference, 1997. `http://www2.sas.com/proceedings/sugi22/CODERS/PAPER80.PDF`

[3] Fehd, Ronald (1997), *%CHECKALL, a macro to produce a frequency of response data set from multiple-response data* Proceedings of the 22nd Annual SAS[®] Users Group International Conference, 1997. `http://www2.sas.com/proceedings/sugi22/POSTERS/PAPER236.PDF`

[4] Fehd, Ronald (1997), *%ShowComb: a macro to produce a data set with frequency of combinations of responses from multiple-response data* Proceedings of the 22nd Annual SAS[®] Users Group International Conference, 1997. `http://www2.sas.com/proceedings/sugi22/POSTERS/PAPER204.PDF`

SAS[®] is a registered trademark of SAS Institute, Inc. In the USA and other countries, ® indicates USA registration.

Author: Ronald Fehd   **bus: 770/488-8102**
Centers for Disease Control MS-G23
4770 Buford Hwy NE
Atlanta GA 30341-3724   **e-mail: RJF2@cdc.gov**

To receive the latest edition of the macro Array send an e-mail to the author with the subject: request Array.

End note: This paper was typeset in LATEX. For further information about using LATEXto write your SUG paper, consult the SAS-L archives:

```
http://www.listserv.uga.edu/cgi-bin/wa?S1=sas-l
Search for                : 
The subject is or contains: LaTeX
The author's address      : RJF2
Since                     : 01 June 2003
```

```
                                  ─────────────────── array.sas ───────────────────
1
2     /* macro: Array   returns a series of macro-variables
3                       named &Name.1   &Name.2 .. &Name.N
4                       and a macro-variable named Dim_&Name.
5                       i.e. %Array(Var,Q04A Q04B Q04C);
6                       returns:  Var1::Q04A Var2::Q04B Var3::Q04C, Dim_Var::3
7     Parameters: array-name
8               array-elements: either of
9                   horizontal list of array element values
10                  vertical   list: data set, variable
11    Notes   : must declare %local macro variables Dim_Array-Name
12            and Array-Name1 ... Array-NameN –before– calling Array
13            length(Array-Name)                        must be <= 28
14            length(Array-Name) + length(dim_Array-Name) must be <= 32
15    Usage   : within macro: see also test data
16            %Array(Array-Name,Item-List);                         run;
17            %Array(Array-Name,data=Data-Name,var=Var-Name);       run;
18    Usage   : in open code:
19            %Array(Array-Name,data=Data-Name,var=Var-Name,_global_=1);run;
20    Process : case 1: scan user-provided list of array-elements
21            case 2: data set: call symput of variable
22            case 3: error msg
23    KeyWords: array %Array Contents delimiter loop macro ParmBuff %scan
24            %Qscan symput SysPbuff %until SQL                           */
25    %Macro Array /* ------------------------ --------------------------- */
26    (Name       /*macro-var array-name: len(array-name) must be <= 28    */
27    ,List       /*horizontal list of array-elements                      */
28    ,Libname  = WORK/*vertical list: libref                              */
29    ,Data     = .   /*vertical list: data set name                       */
30    ,Var      = .   /*vertical list: variable in data set                */
31    ,Dim      = 9   /*vertical list: how many _global_ vars to allocate?  */
32    ,Where    = 1   /*vertical list: subset phrase use w/DICTIONARY.tables*/
33    ,Delimitr = %str( ).<%str(()]&!$*%str())^-/%>\/*horiz-list delimiter  */
34    ,_Global_ = 0  /*?make mac-vars %global?,                            **
35    ,I       = 1  /*local macro loop counter/index                       */
36    ,Item    = .  /*local temp var for Qscan                             */
37    ,Print    = noprint /*local: to see SQL select use ,PRINT=)          */
38    ,MaxLenMvarName = 28/*local max length mVar Name                     */
39    )/des = 'create array of mac-vars from list/data'/*                  */
40      parmbuff   /* ParmBuff: see SysPbuff                               */
41    ;/* ..................................... ........................... */
42    %If    "&Name." ne ""                        /*case 1: Name & List */
43       and "&List." ne ""  and %length(&Name.) le &MaxLenMvarName.
44                   %then %do; %Let I    = 1;
45                           %Let Item = %Qscan(&List,&I,&Delimitr);
46    %Do %until(&Item. = );
47       %If &_Global_ %then %do; %global &Name.&I;                    %end;
48                           %Let &Name.&I = &Item.;
49                           %Let I        = %eval(&I + 1);
50                           %Let Item     = %Qscan(&List,&I,&Delimitr);
51                                             %*do %until; %end;
52    %If &_Global_     %then %do; %global Dim_&Name.;              %end;
53                           %Let    Dim_&Name. = %eval(&I - 1);
54    %Put @@Array returns &Name., dim=&&Dim_&Name. list<&List.>;%*case1;%end;
```

```
                          ———————————— array.sas continued ————————————
 55
 56   %Else %If "&Name." ne ""                  /*case 2: Name & (Data & Var)*/
 57        and "&List." eq ""  and "&Data." ne "."
 58        and "&Var."  ne "." and %length(&Name.) le &MaxLenMvarName.
 59                    %then %do;
 60   %If &_Global_.       %then %do; %Global Dim_&Name.;
 61                               %Let    Dim_&Name. = &Dim.;
 62      %Do I = 1 %to &&Dim_&Name.; %Global    &Name.&I.;            %end;
 63                                               %*If _Global_; %end;
 64   proc SQL &Print.;
 65          select &Var.
 66          into   :&Name.1 – :&Name.9999
 67          from   &Libname..&Data.
 68          where  &Where.
 69          ;quit;
 70          %Let   Dim_&Name. = &SqlObs.;
 71   %Put @@Array returns &Name. dim=&&Dim_&Name. &Data..&Var.; %*case2;%end;
 72   %Else %Do;                                %*case 3: print error msg *;
 73        %Put ERROR: in macro Array:;
 74        %Put array–name : required;
 75        %Put array–elements may be list, or (data and var);
 76        %Put list    : Array(Array–Name,Item–List);
 77        %Put data+var: Array(Array–Name,Data=Data–Name,Var=Var–Name);
 78      %If %length(&Name.) gt &MaxLenMvarName. %then
 79        %Put length(Array–Name) must be <= &MaxLenMvarName.;
 80        %Put parmlist: <&SysPbuff.>;%* /ParmBuff macro statement option*;
 81        %Put parm: Array–Name = <&Name.> length = <%Length(&Name.)>;
 82        %Put parm: ItemList = <&List.>;
 83        %Put parm: Libname  = <&LibName.> ;
 84        %Put parm: Data     = <&Data.>    ;
 85        %Put parm: Var      = <&Var.>     ;
 86        %Put parm: _Global_ = <&_Global_.> in (0,1);      %*case3; %end;
 87   %* ...................................... .................. Array; %Mend;
 88    /*–test data  ——————— enable by ending this line with slash (/) **/
 89   DATA   VQ04;            *convention for %CheckAll setup;
 90   length Name $ 8;       *can use PROC Contents output data set;
 91   Name = 'Q04A'; output;Name = 'Q04B'; output;Name = 'Q04C'; output;  run;
 92   %Macro Demo(Test=0,Dim_Var=10);%*initialize: declare macro array names;
 93   %Do I = 1 %to &Dim_Var.; %local Var&I.;                       %end;
 94   %If      &Test. eq 1 %then %do;%Array(Var,Q03A Q03B Q03C Q03D);   %end;
 95   %Else %If &Test. eq 2 %then %do;%Array(Var,data=VQ04,var=Name);   %end;
 96   %Else %If &Test. eq 3 %then %do;%Array(Var,Libname = Dictionary
 97                                          ,Data    = Tables
 98                                          ,Var     = MemName
 99                                          ,Where   = LibName eq 'WORK'
100                                          );                   %end;
101   %Else %GoTo EXIT;run;%*illustrate usage in loop;
102   %Do I = 1 %to &Dim_Var.; %Put Var&I. :: &&Var&I.;           %end;
103   %EXIT: run;%* ........................... ................ Demo *; %Mend;
104   %Demo(Test = 1);
105   %Demo(Test = 2);
106   %Demo(Test = 3);
107   %Array(This_Mvar_Name_Is_Tooooo_Long,A B C D E);*see error msg;
108   %Array(Test,A B C D E,_Global_=1);*make mac–vars %global in open code;
109   %Put _User_;*show global list of macro–variables;
110   run; /* ............................... ................ Test Data end*/
```