SAS ® ARRAYS: A BASIC TUTORIAL

Bruce Gilsen, Federal Reserve Board

INTRODUCTION

SAS ® arrays provide a simple, convenient way to process a group of variables in a SAS DATA step. As a SAS software consultant at the Federal Reserve Board for the last ten years, I have often observed that SAS users avoid arrays, even when their applications would greatly benefit from array use. This paper is targeted at users who have avoided SAS arrays.

This paper presents a basic introduction to SAS arrays. It attempts to demonstrate that arrays are easy to use and can greatly simplify SAS DATA step processing. The examples are intentionally simple, using SAS data sets with only a few observations and a few variables, and using as few SAS programming statements as possible.

OVERVIEW

The following topics are included in this paper:

- SAS arrays: basic concepts
- SAS arrays: basic usage
- SAS arrays compared to FORTRAN-style arrays
- Implicit arrays
- Defining arrays: examples
- Defining and referencing arrays: details
- Array-oriented functions
- Functions that accept arrays as arguments
- Array applications: examples
  - Using arrays in an IF-THEN statement
  - Setting missing values to zero
  - Reading data from input files
  - Looping through a two-dimensional array
  - Using the same array in more than one DATA step
  - Retrieving and displaying variable names with CALL VNAME

SAS ARRAYS: BASIC CONCEPTS

This section introduces the following basic concepts: array definition, array element, subscript, one-dimensional array, multi-dimensional array, lower bound, upper bound, array reference, index variable, explicit array, and implicit array. The following DATA step is used in this section to illustrate these concepts.

```
(1)  data new;
(2)    set old;
(3)    array gnp (4) consume invest gov taxes;
(4)    do i = 1 to 4;
(5)      gnp(i) = gnp(i) + 100;
(6)    end;
       more SAS statements
```

- Array definition

An array must be defined with an ARRAY statement prior to its use. An ARRAY statement names the array and specifies the variables to be included in the array. A simple ARRAY statement has the following syntax.

ARRAY *array-name (number-of-elements) list-of-variables ;*

Statement (3) defines the array GNP that contains the 4 variables CONSUME, INVEST, GOV, and TAXES.

- Array element

Each variable in an array is known as an array element. Array GNP has the 4 elements CONSUME, INVEST, GOV, and TAXES. These variables are contained in SAS data set OLD.

- Subscript

In an ARRAY statement, a subscript specifies the number of array elements. In statement (3), the number 4 inside the parentheses is a subscript.

When an array is referenced, a subscript specifies which array element to process. In statement (5), the index variable I is a subscript.

- One-dimensional and multi-dimensional arrays

Simple arrays have one dimension, and are known as one-dimensional arrays. Arrays with more than one dimension are known as multi-dimensional arrays. Two-dimensional arrays can be thought of as providing row and column arrangement of a group of variables. Arrays can be defined with as many dimensions as necessary. Array GNP is one-dimensional.

- Lower bound and upper bound

For a one-dimensional array, the lowest subscript is known as the lower bound of the array, and the highest subscript is known as the upper bound of the array. The lower bound of a one-dimensional array defaults to 1. Array GNP has a lower bound of 1 and an upper bound of 4.

Multi-dimensional arrays have lower and upper bounds for each dimension.

- Array reference

Specifying an array element or elements in a SAS statement is known as referencing an array. To reference a single array element, use the following syntax.

array-name (subscript)

Statement (5) references an element of the GNP array.

- Index variable

DO loops are frequently used to process array elements one at a time. A variable used to specify a range of values for DO loop processing is called an index variable. In statements (4) - (6), the DO loop uses the index variable I.

- Explicit arrays and implicit arrays

The arrays used in this paper are known as explicit arrays. Implicit arrays, an earlier type of SAS array, are still available in Version 6 of SAS software. Explicit arrays are strongly recommended for all array usage.

SAS ARRAYS: BASIC USAGE

To understand arrays, it is important to understand the following concepts.

- An array associates a group of **variables** so that the variables can be more easily processed in a SAS DATA step.
- Array references always refer to values in the **current observation.**

Consider the following DATA step.

```
data new ;
  set old ;
  array gnp (4)  consume invest gov taxes ;
  more SAS statements
```

To reference a variable in the GNP array, use either its name or an array reference. The following specifications are equivalent.

Variable name	Array reference
consume	gnp(1)
invest	gnp(2)
gov	gnp(3)
taxes	gnp(4)

The following two statements are equivalent.

```
put gnp(1) ;
put consume ;
```

The following four statements are equivalent.

```
gnp(3)= gnp(3) + 100 ;
gnp(3)= gov + 100 ;
gov= gnp(3) + 100 ;
gov= gov + 100 ;
```

Suppose the values of the variables CONSUME, INVEST, GOV, and TAXES for the first 6 observations of a SAS data set are as follows.

consume	invest	gov	taxes
100	200	300	400
110	210	310	410
120	220	320	420
130	230	330	430
140	240	340	440
150	250	350	450

When the first observation of the data set is processed, GNP has the following values.

- gnp(1) = 100 (because consume = 100)
- gnp(2) = 200 (because invest = 200)
- gnp(3) = 300 (because gov = 300)
- gnp(4) = 400 (because taxes = 400)

When the second observation of the data set is processed, GNP has the following values.

- gnp(1) = 110
- gnp(2) = 210
- gnp(3) = 310
- gnp(4) = 410

SAS ARRAYS COMPARED TO FORTRAN-STYLE ARRAYS

Some SAS software users find SAS arrays confusing because they differ from the *FORTRAN-style arrays* used in programming languages such as FORTRAN, PASCAL, and C. This section provides a brief description of FORTRAN-style arrays and contrasts them with SAS arrays.

Purpose of an array

A SAS array associates a **group of variables** so that the variables can be processed together. This was detailed in the previous section,

A FORTRAN-style array associates a **group of values** so that the values can be processed together.

FORTRAN-style arrays: basic usage

Consider a FORTRAN-style array called GNP with six elements. Let's assume that array subscripts are surrounded by parentheses and that GNP(1) references the first value of the array, GNP(2) references the second value of the array, and so on. (In some languages, such as C, array subscripts are surrounded by braces and GNP(0) references the first value of the array.)

If array GNP contains time-series data, then each array element might contain the value of GNP in a different time period. For example, the values in the GNP array could be the values of GNP in six periods, as follows.

100
110
120
130
140
150

GNP(1) references the first value of GNP, 100. GNP(2) references the second value of GNP, 110, and so on. Note how FORTRAN-style array references refer to different values of the same variable, whereas SAS array references refer to the values of different variables in the same observation.

IMPLICIT ARRAYS

This section provides some information about implicit arrays.

- Implicit arrays are an older type of array. It is strongly recommended that explicit arrays always be used instead of implicit arrays.
- Implicit arrays are still available in Version 6 of SAS software. They are retained for compatability with older systems.
- An easy way to identify implicit arrays in SAS code is that implicit array definitions do not include the number of array elements. An example of an implicit array definition is as follows.

  ```
  array gnp consume invest gov taxes ;
  ```

- The DO OVER statement can only be used with implicit arrays.

DEFINING ARRAYS: EXAMPLES

1. Define an array containing 4 variables. The number 4 inside the parentheses specifies the number of variables.

```
array gnp (4)  consume invest gov taxes ;
```

2. Define an array identical to the array in the first example. The asterisk specifies that the SAS system should count the number of variables listed and size the array accordingly. Since the number of variables in the array can be changed without changing the subscript, this array definition is more flexible than the first array definition.

```
array gnp (*)  consume invest gov taxes ;
```

3. Define an array containing ten variables, X1 - X10.

```
array xall (*)  x1 - x10 ;
```

4. Define an array containing three variables, X1 - X3. If X1 is not already defined, initialize it to 10 in each observation. If X2 is not already defined, initialize it to 20 in each observation. If X3 is not already defined, initialize it to 30 in each observation.

```
array xall (*)  x1 - x3 (10 20 30) ;
```

5. Define an array containing the three character variables CHARV1, CHARV2, and CHARV3.

```
array charall (*)  $  charv1 charv2 charv3;
```

6. Define a two-dimensional array containing five rows and two columns. TWODIM(1,1) references the value of X1, TWODIM(1,2) references the value of X2, TWODIM(2,1) references the value of X3, and so on.

```
array twodim (5,2)  x1 - x10 ;
```

7. Define an array containing all numeric variables in the current DATA step.

```
array gnp (*)  _numeric_ ;
```

8. Define an array containing all character variables in the current DATA step.

```
array gnpchar (*)  _character_ ;
```

9. Define an array containing ten variables, X1 - X10. Make the lower and upper bounds of the array 0 and 9, respectively, instead of the default bounds of 1 and 10. Programs execute faster if the lower bound of an array is set to 0 instead of 1. Note that the lower and upper bound are separated by a colon (:).

```
array xall (0:9)  x1 - x10 ;
```

10. Define a temporary array with three variables. See the section, "DEFINING AND REFERENCING ARRAYS: DETAILS," for more information about temporary arrays.

```
array temp1 (3)  _temporary_ ;
```

DEFINING AND REFERENCING ARRAYS: DETAILS

- An array can contain either numeric or character variables, but not both.
- The ARRAY statement that defines an array must appear in the DATA step before the array is referenced.
- Array definitions are not stored in SAS data sets. As a result, an array definition is in effect only in the DATA step in which it is defined. To use an array in multiple DATA steps, define it in each of the DATA steps.
- When an array is defined or referenced, the subscript can be surrounded by parentheses **()**, brackets **{}**, or braces **[]**. Parentheses are used in this paper.
- Index variables are frequently used in DO loops to reference array elements one at a time. Index variables are included in the SAS data set being created unless they are excluded with the KEEP or DROP statement or the KEEP= or DROP= data set option.
- Do not use an index variable with the same name as a variable already in the data set.
- Do not define an array with the same name as a variable in the data set. One convention that makes this error less likely is to begin array names with an underscore (_).

- The RETAIN statement causes a DATA step variable to retain its value from one observation to the next. If an array name is included in a RETAIN statement, all the variables in the array retain their values. Temporary array elements are automatically retained.

- If an ARRAY statement includes variables that are not defined in the current DATA step or initialized in the ARRAY statement, the variables are created with initial values of missing (.) for numeric arrays and blank for character arrays.

- Since the SAS DATA step processes one observation at a time, arrays always reference values in the current observation.

- Temporary arrays can be used to improve program efficiency. They can be employed if an array contains no previously defined variables and none of the variables in the array need to be written to an output data set. Temporary arrays differ from other SAS arrays in that

  - Temporary arrays cannot be defined using an asterisk to specify the size of the array.

  - Temporary array elements can only be referenced by their array name and a subscript.

  - Temporary array elements are automatically retained.

- If an array is referenced in a PUT statement that uses named output, the SAS system displays the name of the DATA step variable that corresponds to the array element being referenced. Consider the following array.

  array gnp(*) consume invest gov taxes;

  For an observation in which the value of INVEST is 50, the statements PUT INVEST=; and PUT GNP(2)=; both generate the following output.

  INVEST=50

ARRAY-ORIENTED FUNCTIONS

The examples in this section assume that the following arrays have been defined.

- array gnp (*) consume invest gov taxes ;

- array twodim(3,2) x1 - x6;

1. DIM *(array-name)*

- For a one-dimensional array, the DIM function returns the number of elements (variables).

- For a multi-dimensional array, the DIM function returns the number of elements in an array dimension. Specify the array dimension either of the following ways.

  - Specify the dimension as the second argument of the DIM function.

  - Use the DIM*n* function, where *n* is a dimension.

- Examples.

  - Set Y to 4, the number of elements in the GNP array.

    y = dim(gnp) ;

  - Execute a DO loop from 1 to 4.

    do i = 1 to dim(gnp) ;

  - Set D1 to 3, the size of the first dimension of the TWODIM array. The following three statements are equivalent.

    d1 = dim(twodim, 1) ;
    d1 = dim(twodim) ;
    d1 = dim1(twodim) ;

  - Set D2 to 2, the size of the second dimension of the TWODIM array. The following two statements are equivalent.

    d2 = dim(twodim, 2) ;
    d2 = dim2(twodim) ;

2. HBOUND *(array-name)*

- For a one-dimensional array, the HBOUND function returns the upper bound of the array.

- For a multi-dimensional array, the HBOUND function returns the upper bound of an array dimension. Specify the array dimension either of the following ways.

- Specify the dimension as the second argument of the HBOUND function.
- Use the HBOUND*n* function, where *n* is a dimension.

- For a one-dimensional array, the DIM and HBOUND functions return the same value if the lower bound of the array is 1.
- Examples.
  - Set Y to 4, the upper bound of the GNP array.

    ```
    y = hbound(gnp) ;
    ```

  - Execute a DO loop from 1 to 4.

    ```
    do i = 1 to hbound(gnp) ;
    ```

  - Set HIGH1 to 3, the upper bound of the first dimension of the TWODIM array. The following three statements are equivalent.

    ```
    high1 = hbound(twodim, 1) ;
    high1 = hbound(twodim) ;
    high1 = hbound1(twodim) ;
    ```

  - Set HIGH2 to 2, the upper bound of the second dimension of the TWODIM array. The following two statements are equivalent.

    ```
    high2 = hbound(twodim, 2) ;
    high2 = hbound2(twodim) ;
    ```

3. LBOUND *(array-name)*

- For a one-dimensional array, the LBOUND function returns the lower bound of the array.
- For a multi-dimensional array, the LBOUND function returns the lower bound of an array dimension. Specify the array dimension either of the following ways.
  - Specify the dimension as the second argument of the LBOUND function.
  - Use the LBOUND*n* function, where *n* is a dimension.

- Examples.
  - Set Y to 1, the lower bound of the GNP array.

    ```
    y = lbound(gnp) ;
    ```

  - Execute a DO loop from 1 to 4. The following two statements are equivalent.

    ```
    do i = lbound(gnp) to dim(gnp) ;
    do i = lbound(gnp) to hbound(gnp) ;
    ```

  - Set LOW1 to 1, the lower bound of the first dimension of the TWODIM array. The following three statements are equivalent.

    ```
    low1 = lbound(twodim, 1) ;
    low1 = lbound(twodim) ;
    low1 = lbound1(twodim) ;
    ```

  - Set LOW2 to 1, the lower bound of the second dimension of the TWODIM array. The following two statements are equivalent.

    ```
    low2 = lbound(twodim, 2) ;
    low2 = lbound2(twodim) ;
    ```

FUNCTIONS THAT ACCEPT ARRAYS AS ARGUMENTS

The following SAS functions accept arrays as arguments, and treat each array element as an argument to the function: CSS, CV, MAX, MEAN, MIN, N, NMISS, ORDINAL, RANGE, SKEWNESS, STD, STDERR, SUM, USS, and VAR. Except for ORDINAL (which has an additional argument prior to the argument list), these functions all have the following syntax.

*function(*argument*1*, argument2,...,argument*n*);

To specify a one-dimensional or multi-dimensional array as an argument to a function, use the OF keyword and an asterisk.

of *array-name* (*)

- Examples.

In the following examples, arrays are specified as arguments to functions. These examples assume that the following arrays have been defined.

- array gnp (*) consume invest gov taxes ;
- array twodim(3,2) x1 - x6;

  - Set SUMGNP to the sum of CONSUME, INVEST, GOV, and TAXES in each observation. The following two statements are equivalent.

    ```
    sumgnp = sum(of gnp(*)) ;
    sumgnp = sum(consume,invest,gov,taxes);
    ```

  - This statement generates an error because the OF keyword is omitted.

    ```
    sumgnp = sum(gnp(*)) ;
    ```

  - This statement generates an error because the asterisk is omitted.

    ```
    sumgnp = sum(of gnp) ;
    ```

  - Set MEANX to the mean of the variables X1 - X6 in each observation. The following two statements are equivalent.

    ```
    meanx = mean(of x1 - x6);
    meanx = mean(of twodim(*));
    ```

  - Set MAXX to the maximum of the variables X1 - X6 and the number 100 in each observation. The following two statements are equivalent.

    ```
    maxx = max(of x1 - x6,100);
    maxx = max(of twodim(*),100);
    ```

ARRAY APPLICATIONS: EXAMPLES

1. Using arrays in an IF-THEN statement

In this example, an IF-THEN statement checks the value of array elements, and increases them by 1000 if they are greater than 200.

```
data newdata (drop = i);
  set olddata;
  array gnp (*)  consume invest gov taxes ;
  do i = 1 to 3;
    if gnp(i) > 200 then gnp(i)= gnp(i)+1000;
  end;
run;
```

The values of the variables CONSUME, INVEST, GOV, and TAXES in data sets OLDDATA and NEWDATA are as follows.

Data set olddata:

consume	invest	gov	taxes
100	200	300	400
110	210	310	410
120	220	320	420
130	230	330	430
140	240	340	440
150	250	350	450

Data set newdata:

consume	invest	gov	taxes
100	200	1300	400
110	1210	1310	410
120	1220	1320	420
130	1230	1330	430
140	1240	1340	440
150	1250	1350	450

Notes for example 1

- Since the DO statement was executed for only the first three array elements, the variable TAXES is unchanged.
- The DROP= data set option prevents the index variable, I, from being included in the output data set.

2. Setting missing values to zero

In this example, missing values (.) are set to zero for all numeric variables in a data set.

```
data newdata (drop = i);
  set olddata;
  array allnum (*)  _numeric_ ;
  do i = 1 to dim(allnum);
    if allnum(i) = . then allnum(i) = 0;
  end;
run;
```

3. Reading data from input files

In this example, the array _INPTARR is used to read three variables into SAS data set TEST1.

```
data test1 (drop=i) ;
  array _inptarr (*)  firstvar nextvar lastvar;
  do i = 1 to dim(_inptarr);
    input  _inptarr(i)  @;
  end ;

  cards ;
  1 2 3
  4 5 6
  7 8 9
  ;
run ;
```

4. Looping through a two-dimensional array

This example illustrates how to reference the values in a two-dimensional array. Using nested DO loops, the values in a two-dimensional array are printed one at a time.

```
/* This DATA step creates sample data.    */
data one;
  input x1 - x6;
  cards;
  1 2 3 4 5 6
  7 8 9 10 11 12
  ;
run;

data two (drop = ii jj);
  set one;
  array twodim(3,2) x1 - x6;
  put 'Observation number '   _n_ ;
  do ii = 1 to 3;
    do jj = 1 to 2;
      put 'ii = '   ii  'jj = '   jj
        ' twodim(ii,jj) = '   twodim(ii,jj);
    end;
  end;
run;
```

The following PUT statements are printed in the SAS log.

```
Observation number 1
ii = 1 jj= 1  twodim(ii,jj) = 1
ii = 1 jj= 2  twodim(ii,jj) = 2
ii = 2 jj= 1  twodim(ii,jj) = 3
ii = 2 jj= 2  twodim(ii,jj) = 4
ii = 3 jj= 1  twodim(ii,jj) = 5
ii = 3 jj= 2  twodim(ii,jj) = 6
Observation number 2
ii = 1 jj= 1  twodim(ii,jj) = 7
ii = 1 jj= 2  twodim(ii,jj) = 8
ii = 2 jj= 1  twodim(ii,jj) = 9
ii = 2 jj= 2  twodim(ii,jj) = 10
ii = 3 jj= 1  twodim(ii,jj) = 11
ii = 3 jj= 2  twodim(ii,jj) = 12
```

Notes for example 4

- To make the program more automated, calculate the upper bounds of the array with the DIM1 and DIM2 functions instead of hard coding them.

```
do ii = 1 to dim1(twodim);
  do jj = 1 to dim2(twodim);
```

- The DROP= option is not required, but is used to prevent the index variables II and JJ from being included in the output data set.

- The automatic variable _N_ is created in every DATA step, but is not included in the output data set. It contains the current observation number.

5. Using the same array in more than one DATA step

An array definition is only in effect in the DATA step in which it is defined. To define the same array in more than one DATA step, you can use the %LET statement to create a macro variable that contains the array definition, and use the macro variable in each of the DATA steps.

```
/* array definition, used in subsequent steps */
%let arraydef= array sugi1 (*)  invest gov tax;
run;

data one ;
  &arraydef;    /* define array sugi1 */
  more SAS statements

data two ;
  &arraydef;    /* define array sugi1 */
  more SAS statements
```

6. Retrieving and displaying variable names with CALL VNAME

The CALL VNAME routine assigns the name of any variable, including an array element, to the value of another variable.

In this example, the array ARRAY1, which contains the numeric variables VAR1 and VAR2, is tested for negative values. If the value of an array element is less than zero, CALL VNAME sets the value of the character variable VARNAME to the name of the current array element, and the observation number and name and value of the array element are printed.

```
/* This DATA step creates sample data.    */
data old ;
  input var1 var2 ;
  cards;
  1 -100
  -2 200
  ;
run ;

data new (drop=i varname);
  set old ;
  length varname $8;
  array array1(*)  var1 - var2;

  do i=1 to dim(array1);

    /* check if array value is negative */
    if array1(i) < 0 then do;

      /* Negative value found, call vname */
      /* sets variable VARNAME to name  */
      /* of current array element.      */
      call vname(array1(i),varname);
      put 'Observation number ' _n_ /
        'Variable name is ' varname
        ' variable value is ' array1(i);
    end;
  end;

run;
```

The following output is generated.

```
Observation number 1
Variable name is VAR2 variable value is -100
Observation number 2
Variable name is VAR1 variable value is -2
```

Notes for example 6

- Incorrect results are generated if the LENGTH statement for VARNAME is omitted. In this case, the SAS system defines VARNAME as a numeric variable, every call to CALL VNAME sets VARNAME to a numeric missing value (.), and the following note is printed in the SAS log.

  Numeric variables have been converted to character values.

- The DROP= option is not required, but prevents extraneous variables from being included in the output data set.

- The automatic variable _N_ is created in every DATA step, but is not included in the output data set. It contains the current observation number.

CONCLUSION

This paper has presented a basic introduction to SAS arrays. It has attempted to demonstrate that arrays are easy to use and can greatly simplify SAS DATA step processing.

Simple array examples have been presented. More sophisticated examples can be found in the manuals listed in the reference section of this paper, other papers presented at SAS Users Group International (SUGI) Conferences, and the sample program libraries distributed by SAS Institute.

For more information, contact

Bruce Gilsen
Federal Reserve Board, Mail Stop 171
Washington, DC 20551 202-452-2494
email: m1bfg00@frb.gov

REFERENCES

SAS Institute Inc. (1991), "SAS Language and Procedures: Usage2, Version 6, First Edition," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), "SAS Language Reference, Version 6, First Edition," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The following Federal Reserve Board employees contributed extensively to the development of this paper: Mike Bradicich, Donna Hill, Julia Meredith, Steve Schacht, and Peter Sorock. Their support is greatly appreciated.

TRADEMARK INFORMATION

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.