**PO-18**

# Array, Hurray, Array; Consolidate or Expand Your Input Data Stream Using Arrays

William E Benjamin Jr – Owl computer Consultancy, LLC Phoenix AZ

## Abstract

You have an input file with one record per month, but need an output file with one record per year. But you cannot use PROC TRANSPOSE because other fields need to be retained or the input file is sparsely populated. The techniques shown in this paper will enable you to be able to either consolidate or expand your output stream of data by using arrays. Sorted files of data records can be processed as a unit using "BY Variable" groups and building an array of records to process. This technique allows access to all of the data records for a "BY Variable" group and gives the programmer access to the first, last and all records in between at the same time. This will allow the selection of any data value for the final output record.

## Introduction

Arrays are everywhere, look no further than the pop machine in the nearest break room. It may have a column of buttons, or several rows of buttons, or even a key pad that allows you to choose a drink from row A and Column 5. Your Parking lot has lots of rows of spaces to park your car, and may even have levels assigned to different groups or companies. So why don't programmers use arrays more often? Sometimes, because the first program they were assigned to modify or write did not have an array, and they have been able to get along without them just fine, thank you! But look at their code again, and you may see groups of variables like one for each month of a year, or age groups between 18 and 85.

Some will argue that it takes too much CPU time to process all of the empty slots in an array. However, even that argument will fall by the wayside if the programmer uses a few simple counters to limit the number of array entries that need to be cleared after they are no longer needed. Arrays have many uses which include table look-up, consolidation of data, summing numbers, transposing data values and many others. This paper will show examples from simple to complex to explain the use of arrays.

## What is an Array?

An array is a concept first, then when applied it becomes a structure. I contend that the SAS® DATA Step, Microsoft Excel, Microsoft Access, and many other software tools process "Arrays" by design. What makes the concept of an array so universal is that an array is simple it has a name, a subscript, value or a set of values. On a large scale a SAS dataset, an Excel spreadsheet, and an Access database table are all arrays.

- They have a name.
- Each record can be referenced by an observation number, row number, or a database key.
- They have variables or columns.

Collectively these features of the file types listed above allow them to qualify as arrays. Most programmers do not consider files to be arrays; this is easy to do because programmers rarely point to specific records of a file for processing. But the SAS SET statement has a "POINT=" option that allows just that for the selection of a specific data record within a random access file. SAS also has the options OBS= and FIRSTOBS= which will restrict records used as input to SAS processing routines. SAS Also uses Drop and Keep commands to limit variables. Excel files can have highlighted rows, columns, or ranges that are processed, and Access Tables are frequently processed using the keys.

Arrays are not evil devices designed to confuse and befuddle programmers, they are structures we use every day. Relational Database designers use small tables to store repeated data values to reduce the overall size of the database. These small files are little more than arrays used to aid the processing and storage requirements of the database. Microsoft Excel 2007 and beyond store arrays of repeated values to conserve space, and a pop machine has an array of buttons to allow you to pick your favorite drink today.

With this idea in mind, let's just think smaller.

### Array Definition Syntax

One of the simplest syntax forms for an array definition is the following:

SYNTAX -- `ARRAY array_name {subscript_range} $ n variable_list (values);`

### Simple Arrays

One simple task an array can do is to convert a numeric month number into a character month value. Many programs try to conserve space by using the fewest number of bytes or characters to represent what would take more space. Appendix-A shows you how to use the array in example (1) to convert a two digit month value to the name of the month. The code in Appendix-A reads a data file of months, sales amounts and returns. After sorting by the month variable the code then summarizes the data file for printing by using retained variables to store totals for each month of the year and outputting one record for each month. The following is an array definition that could be used to perform that task.

Example (1)--
```
ARRAY months {12} $ 12
mth1-mth12 ('January'   'February' 'March'    'April'
            'May'       'June'     'July'     'August'
            'September' 'October'  'November' 'December');
```

This definition has seven parts, other definitions can have more (or less when using numeric values).

1. The SAS Code statement "`ARRAY`" begins the definition of an array
2. Next a valid SAS name ("months") defines the array name reference
3. The "`{subscript_range}`" is a numeric expression defining the number of array values.
4. This definition is a list of character variables (denoted by the "$")

5. The length of each variable in this array is specified in bytes (12)
6. A list of 12 character variable names is provided (`mth1-mth12`). The number of variable names must match the number of elements defined by the subscript range.
7. This definition also has an optional list of initial values that populate the array variables.

The SAS Array definition has other syntax parts that we will discuss later. The array used in Appendix-A looks something like Figure 1. and was just used to replace one value with another.

| index | Data Value |
|---|---|
| 1 | January |
| 2 | Febuary |
| 3 | March |
| 4 | April |
| 5 | May |
| 6 | June |
| 7 | July |
| 8 | August |
| 9 | September |
| 10 | October |
| 11 | November |
| 12 | December |

Figure 1. (to the left) Is a one dimensional array of month names (the index values are not part of the array) . The actual array definition above shown as Example (1) would consist of the SAS variables mth1, mth2, mth3, …, mth12.

Appendix –A uses this array to replace a month number with the name of the month. Additionally since the input file is sorted, the "BY Variable" statement in the SAS program and the "list.variable" reduce the output to one line per month in the output file.

## Using Arrays to Collect, Consolidate, and Summarize Data

Another use of arrays is to summarize data. Appendix-B shows an example of this use of arrays. The SASHELP dataset "Stocks" is used as input for the example code in Appendix-B. The file has stock market data for three companies listed at the beginning of the month for 19 years and 5 months, along with high and low prices during the month. The program uses a "WHERE" clause on the set statement to read only the full 19 years of data then keeps two variables (the high and low stock value for each month) in the arrays defined below. The indices of the array are derived from the variables "STOCK" and "DATE" in the source file. Two formats were used to allow the definition and order of the companies to be external to the data step that reads the file. The year and the month were extracted from the "DATE" variable and used as the other two indices. The following array definitions set up these data structures using temporary variables. :

Example—2:

```
Array monthly_high   {3,19,12}  _temporary_ ;
Array monthly_low    {3,19,12}  _temporary_ ;
```

This array would be similar in structure to two Excel workbooks with 3 pages (one for each company) any each page would have 19 rows with 12 columns. The rows would store data for each year and the

columns would store data for each month. Each workbook would store the data for one variable (high stock price or low stock price).
This definition has six parts.

1. The SAS Code statement "ARRAY" begins the definition of an array
2. Next a valid SAS name ("monthly_high" or "monthly_low") defines the array name reference
3. The "{subscript_range}" is a numeric expression defining the number of values in the array. This definition has three parts. The number three represents three companies, the 19 is for each of the 19 full years of stock market data, and the 12 is for the 12 months in each year. The total number of variable required for this definition is 3*19*12=684.
4. This definition is a list of numeric variables (denoted by the absence of a "$")
5. The length of each variable in this array is the default of 8 bytes for a numeric variable
6. The SAS keyword "_temporary_" is used to define the array variables as a list of exactly as many variables as are needed to fill all dimensions of the array. The variables are automatically retained.

The output file from the code in Appendix-B is a file with 19 records for each of three companies. The PROC PRINT output is shown listed by company. The records are shown in the newest to oldest order, but could be output in the other order by using the alternate "DO loop" shown in the code.

**Using Arrays to Expand Data**

The next example is a little shorter and simpler. Example-3 (See Appendix-C for a listing of the code) uses the SASHELP.PRICEDATA file with 17 variables that have different prices. A file like this would be great for a back office accounting application but it might not be useful as a public price sheet. The code that follows expands the file by creating one record for each different price. That file could then be sorted and separated into public price sheets.

Example—3:

```
Array prices {17} price1-price17;
```

This definition has six parts.

1. The SAS Code statement "ARRAY" begins the definition of an array
2. Next a valid SAS name ("prices") defines the array name reference
3. The "{subscript_range}" is a numeric expression defining the number of array values.
4. This definition is a list of numeric variables (denoted by the absence of a "$")
5. The length of each variable in this array is the default of 8 bytes for a numeric variable
6. The SAS variable name range "price1-price17" defines the list of array variables. These variables are part of the input file, but are dropped from the output file.

The output file produced by Example-3 is a file with 17,340 lines generated from the input file that had 1020 lines. The file changed from 28 variables to 13 variables (17 variables were dropped and two were added). The code that follows will generate the new output file.

## Conclusion

This paper has shown several techniques that use arrays to perform different tasks without using PROC TRANSPOSE. The first task replaced a numeric month value for the name of the month, and summed values to reduce a sparsely populated variable number of monthly records into a set of records with one record per month. The second example read all of the data in a file into an array and then summarized the records looking for the months with maximum and minimum stock prices. This example then wrote out annual records for each annual set of values by company. The third job split one file with 17 price values into a file with records that had only one price value per record. Arrays are powerful data structures and have many uses. In fact many programmers use some form of an array without realizing or calling the structure a formal array. This paper has attempted to point out some of the structure and uses of an array in conjunction with other features of the SAS Language like “BY Variable” processing, end of file processing, using the output statement to control the final datasets.

## CONTACT INFORMATION

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

Name: William E Benjamin Jr
Enterprise: Owl Computer Consultancy, LLC
Address: P.O.Box 42434
City, State ZIP: Phoenix AZ, 85080
Work Phone: 623-337-0269
E-mail: William@owlcomputerconsultancy.com

## Appendix A – One Dimensional Array Example

This example uses a one dimensional array to convert a two digit month value into a text string that is the month name. The data is sorted and summarized before being printed.

```
data monthly_sales;
infile cards;
input month 2. sales returns ;
datalines;
01  5678   9
04 27875  30
02  9078  75
05 29118  50
04  8634  32
07  4833  11
09 25644 270
03  5668 350
02   665 700
05  5673  90
06 10622 330
10  5615 120
05  5699  10
11  5676  50
08   674  50
09  8651 240
11   683  18
04  5845 490
03 16289 820
01   450  40
02  6647 280
12  4645 560
11  9672 920
;
run;

proc sort data=monthly_sales;
by month;
run;

data net_sales;
retain net_sales monthly_sales monthly_returns 0 this_month '            ';
* The array is defined here, initial values for each month name I defined here;
ARRAY month_names {12} $ 12 mth1-mth12
       ('January'   'February' 'March'    'April'
        'May'       'June'    'July'     'August'
        'September' 'October' 'November' 'December');

set monthly_sales;
by month;
if first.month then do;
       net_sales        = 0;         * Total sales less returns;
       monthly_sales    = 0;         * Total monthly sales;
       monthly_returns  = 0;         * Total monthly returns;
       record_count     = 0;         * Count of monthly records;
```

```
end;

* The array is used here to convert the month number to the month name;
this_month       = month_names(month);                    * Convert to Character month name;

net_sales        = sum(net_sales,sales,(returns*-1));     * Total sales less returns;
monthly_sales    = sum(monthly_sales,sales);              * Total monthly sales;
monthly_returns  = sum(monthly_returns ,returns);         * Total monthly returns;
record_count + 1;                                         * Count of monthly records;

if last.month then output;
drop mth:; * Drop all variables that begin with mth, this would be mth1-mth12;
run;

proc print data=net_sales noobs;
format monthly_sales monthly_returns net_sales Dollar10.;
var month record_count this_month monthly_sales monthly_returns net_sales;
sum monthly_sales monthly_returns net_sales;
run;

* Note then month numbers in the input dataset were converted to the values in the 'this_month'
variable;
```

The SAS System 1

| month | record_ count | this_ month | monthly_ sales | monthly_ returns | net_sales |
|---|---|---|---|---|---|
| 1 | 2 | January | $6,128 | $49 | $6,079 |
| 2 | 3 | February | $16,390 | $1,055 | $15,335 |
| 3 | 2 | March | $21,957 | $1,170 | $20,787 |
| 4 | 3 | April | $42,354 | $552 | $41,802 |
| 5 | 3 | May | $40,490 | $150 | $40,340 |
| 6 | 1 | June | $10,622 | $330 | $10,292 |
| 7 | 1 | July | $4,833 | $11 | $4,822 |
| 8 | 1 | August | $674 | $50 | $624 |
| 9 | 2 | September | $34,295 | $510 | $33,785 |
| 10 | 1 | October | $5,615 | $120 | $5,495 |
| 11 | 3 | November | $16,031 | $988 | $15,043 |
| 12 | 1 | December | $4,645 | $560 | $4,085 |
| | | | $204,034 | $5,545 | $198,489 |

## Appendix B – Multi-Dimensional Array Example

```
proc format fmtlib;
value $comp_n
'IBM'       =  1
'Intel'     =  2
'Microsoft' =  3;

value comp_c
1 = 'IBM'
2 = 'Intel'
3 = 'Microsoft';
run;

data stocks;

        retain cmp1 cmp2 cmp3 '          ' Max_month Min_month '            ';
        * Array to store the text version of the company name;
        Array Company {3} $ 9 cmp1-cmp3;

        * array of month names;
        ARRAY months {12} $ 12
                              mth1-mth12 ('January'   'February'  'March'   'April'
                                          'May'       'June'      'July'    'August'
                                          'September' 'October'   'November' 'December');

        ************************************************;
        ** Multi dimensional arrays for each variable **;
        ** All Companies, All years, All months       **;
        ** _temporary_ variables were used to avoid   **;
        ** a large number retained variables.         **;
        ** These arrays would each use 684 variables  **;
        ************************************************;
        Array monthly_high   {3,19,12}  _temporary_ ;
        Array monthly_low    {3,19,12}  _temporary_ ;

        format Annual_yearly_high Annual_yearly_low dollar16.2;

        * data in file for 1986 is Aug to Dec - only use full year's worth of data;
        set sashelp.stocks (where=(date gt mdy(12,31,1986))) end=eof;

        * calculate the array indices;
        company_index  = input(put(stock,$comp_n.),3.);      * IBM, Intel, and Microsoft;
        year_index     = input(put(date,year.),4.) - 1986;   * adjust to index value of 1;
        month_index    = input(put(date,month.),2.);         * index for month (1-12);

        *put the company names into an array;
        company(company_index) = put(company_index,comp_c.); * fill the company name array;

        * load the high and low monthly values into an array one record at a time;
        monthly_high   (company_index,year_index,month_index) = high;
        monthly_low    (company_index,year_index,month_index) = low;

        ************************************************;
        **  All data is read before processing any       **;
```

```
        **  information. The use of the output statement  **;
        **  in the "end of file" do/end structure will    **;
        **  suppress all automatic output to files. One   **;
        **  record is output for each company, for each   **;
        **  year with the stock highs and lows (and the   **;
        **  month of the high or low). The commands:      **;
        **  1) Do yr = 1 to 19                            **;
        **  2) Do yr = 19 to 1 by -1                      **;
        **  control the way records are output (ascending **;
        **  or descending by year)                        **;
        ************************************************;

        * when all data is in the arrays search for high and low months in each year by company;
        if eof then do;
                do comp = 1 to 3; * for each company;
                        company_name = company(comp);
                        *do yr = 1 to 19;       * in each year - oldest year first;
                        Do  yr = 19 to 1 by -1; * in each year - newest year first;
                            Annual_yearly_high  = 0;
                                Annual_yearly_low   = 9999999;
                                do mth = 1 to 12; * check every month;
                                        if Annual_yearly_high lt monthly_high(comp,yr,mth) then do;
                                           Annual_yearly_high =  monthly_high(comp,yr,mth);
                                           Max_month = months(mth);
                                        end;
                                        if Annual_yearly_low gt monthly_low(comp,yr,mth) then do;
                                           Annual_yearly_low =  monthly_low(comp,yr,mth);
                                           Min_month = months(mth);
                                        end;
                                        year = yr + 1986;
                                end;
                                output; * save the annual highs and lows;
                        end;
                end;
        end;

        drop open close volume high low date stock cmp1 cmp2 cmp3 company_index
                year_index month_index AdjClose comp yr mth mth1-mth12;
run;

proc print data=stocks noobs;
by company_name;
var year annual_yearly_high max_month annual_yearly_low min_month;
run;
```



| FORMAT NAME: COMP_C LENGTH: 9 NUMBER OF VALUES: 3<br>MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 9 FUZZ: STD | | |
|---|---|---|
| START | END | LABEL (VER. V7\|V8 31JUL2011:13:30:35) |
| 1 | 1 | IBM |
| 2 | 2 | Intel |
| 3 | 3 | Microsoft |

| FORMAT NAME: $COMP_N LENGTH: 1 NUMBER OF VALUES: 3<br>MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 1 FUZZ: 0 | | |
|---|---|---|
| START | END | LABEL (VER. V7\|V8 31JUL2011:13:30:35) |
| IBM | IBM | 1 |
| Intel | Intel | 2 |
| Microsoft | Microsoft | 3 |



company_name=IBM

| year | Annual_yearly_high | Max_month | Annual_yearly_low | Min_month |
|---|---|---|---|---|
| 2005 | $99.10 | January | $71.85 | April |
| 2004 | $100.43 | February | $81.90 | August |
| 2003 | $94.54 | October | $73.17 | March |
| 2002 | $126.39 | January | $54.01 | October |
| 2001 | $124.70 | December | $83.75 | January |
| 2000 | $134.94 | September | $80.06 | December |
| 1999 | $246.00 | May | $89.00 | October |
| 1998 | $189.94 | December | $95.62 | January |
| 1997 | $179.25 | May | $81.75 | June |
| 1996 | $166.00 | December | $83.12 | January |
| 1995 | $114.62 | August | $70.25 | January |
| 1994 | $76.37 | October | $51.38 | March |
| 1993 | $59.88 | December | $40.63 | August |
| 1992 | $100.37 | July | $48.75 | December |
| 1991 | $139.75 | February | $83.50 | December |
| 1990 | $123.12 | July | $94.50 | January |
| 1989 | $130.88 | January | $93.37 | December |
| 1988 | $129.50 | July | $104.25 | March |
| 1987 | $175.88 | August | $100.00 | October |

company_name=Intel

| year | Annual_yearly_high | Max_month | Annual_yearly_low | Min_month |
|---|---|---|---|---|
| 2005 | $28.84 | July | $21.89 | January |
| 2004 | $34.60 | January | $19.64 | September |
| 2003 | $34.51 | November | $14.88 | February |
| 2002 | $36.78 | January | $12.95 | October |
| 2001 | $38.59 | January | $18.96 | September |
| 2000 | $147.50 | July | $29.81 | December |
| 1999 | $143.69 | January | $50.13 | June |
| 1998 | $126.19 | December | $65.66 | June |
| 1997 | $169.75 | May | $67.37 | December |

| | | | | |
|---|---|---|---|---|
| 1996 | $141.50 | December | $49.81 | January |
| 1995 | $119.00 | May | $55.19 | December |
| 1994 | $73.50 | March | $56.00 | April |
| 1993 | $121.25 | March | $49.25 | July |
| 1992 | $91.50 | December | $46.50 | May |
| 1991 | $59.25 | June | $37.75 | January |
| 1990 | $52.00 | July | $28.00 | October |
| 1989 | $36.00 | December | $22.87 | January |
| 1988 | $37.25 | June | $19.25 | November |
| 1987 | $62.75 | October | $20.25 | November |



company_name=Microsoft

| year | Annual_yearly_ high | Max_month | Annual_yearly_ low | Min_ month |
|---|---|---|---|---|
| 2005 | $28.25 | November | $23.82 | March |
| 2004 | $30.20 | November | $24.01 | March |
| 2003 | $57.32 | January | $22.55 | March |
| 2002 | $70.62 | January | $41.41 | July |
| 2001 | $76.15 | June | $42.88 | January |
| 2000 | $118.62 | January | $40.31 | December |
| 1999 | $180.38 | March | $75.50 | May |
| 1998 | $160.06 | February | $79.25 | March |
| 1997 | $150.75 | July | $80.75 | January |
| 1996 | $159.50 | December | $76.37 | December |
| 1995 | $109.25 | July | $58.25 | January |
| 1994 | $99.25 | May | $46.88 | July |
| 1993 | $98.00 | June | $70.37 | August |
| 1992 | $133.25 | January | $65.50 | July |
| 1991 | $117.50 | April | $60.50 | July |
| 1990 | $122.00 | April | $50.75 | August |
| 1989 | $89.25 | November | $45.75 | March |
| 1988 | $70.50 | July | $45.25 | November |
| 1987 | $128.25 | May | $37.25 | October |

## Appendix C – One Dimensional Array Example Used to Expand a Data File

```
* output file, with the 17 variables of the original array dropped;
data pricedata (drop=price1-price17);

        * output file, with the 17 variables of the original array dropped;
        array prices {17} price1-price17;
        set sashelp.pricedata;

        * output 17 records for every rodord on the input file;
        do i = 1 to 17;
                price = prices(i);
                price_number = i;
                output;
        end;
        drop i;
run;
```