PharmaSUG 2011 - Paper TT04

Using a HASH Table to Reference Variables in an Array by Name

John Henry King, Hopper, Arkansas

ABSTRACT

Array elements are referenced by their index value using a constant, e.g. a[1], a[2], or with a variable whose value contains the index a[i] where i=1,2. This works well enough for most applications but there are situations where you need to reference an array element but you only know the name of the variable not the array index. In a situation like that a HASH table can be used to provide a lookup table for the array index using the variable name as a key. An example will be shown where this technique is applied to an ODS OUTPUT data set created by PROC FREQ, to normalize the table variables. Another example will show that many CSV files with a varying number of fields and with the fields in any order can be read in a single data step.

INTRODUCTION

Usually when we write a data step program we know the names of the variables. In effect the variable names are constant data and are compiled with the programs various other parts. Of course this works well enough most of the time. However there are situations, perhaps when we write a dynamic program, were we don't know the names of the variables to process. An example of this type of data set would be a stacked frequency counts data set created by PROC FREQ using ODS OUTPUT.

Obs	Table	Sex	Frequency	Percent	Cum Frequency	Cum Percent	Age
1	Table Sex	F	9	47.37	9	47.37	.
2	Table Sex	M	10	52.63	19	100.00	.
3	Table Age		2	10.53	2	10.53	11
4	Table Age		5	26.32	7	36.84	12
5	Table Age		3	15.79	10	52.63	13
6	Table Age		4	21.05	14	73.68	14
7	Table Age		4	21.05	18	94.74	15
8	Table Age		1	5.26	19	100.00	16

This data set contains the one-way frequency tables for SEX and AGE from SASHELP.CLASS. If there had been more variables named in the TABLES statement this data would have more observations and more variables.

EXAMPLE 1: NORMALIZING OUTPUT DATA SET FROM PROC FREQ.

In a response to a question on the SAS® Discussion Forums PROC FREQ -- getting both unformatted and formatted values into CSV file I posted a program that uses the technique of referencing an array element by name using a HASH table. The program normalizes the table variables into either CVALUE or NVALUE depending on the variable type. The following example is similar to the program I suggested for the SAS Discussion Forum.

```
proc format;
   value age  0-12='Tween' 13-19='Teen';
   value $sex  'M'='Male'    'F'= 'Female';
   run;
data class;
   set sashelp.class;
   attrib age format=age.  label='Age Group';
   attrib sex format=$sex. label='Gender';

   run;
ods listing close;
proc freq data=class;
   ods output OneWayFreqs=OneWayFreqs;
```

```
   run;
ods listing;
```

These statements create the data set, “ONEWAYFREQS” used in the example. This is a data set of stacked frequency tables see Exhibit 1 for contents and complete data listing. The following data step will normalized the table variables; the program is dynamic in that any ONEWAYFREQS output from PROC FREQ can be processed with this data step. All the information about which variables to process are contained in the data set and processed accordingly. The numbered statements are described in detail below. See Exhibit 2 for the output from this data step.

```
data Normalized;                                                          1
   length vName $32 vLabel $256 vType $1 vLength 8 vFormat $32            2
          _i_ nValue 8 cValue $64 Formatted $256;
   if 0 then set OneWayFreqs;                                             3

   array _c _character_;                                                  4
   array _n _numeric_;

   if _n_ eq 1 then do;                                                   5
      declare hash vr();                                                  6
      vr.definekey('vName');
      vr.definedata('vName','vLabel','vType','VLength','vFormat','_i_');
      vr.definedone();
      do _i_ = 1 to dim(_c);                                              7
         vName    = vName(_c[_i_]);
         vType    = vType(_c[_i_]);
         vLength  = vLength(_c[_i_]);
         vFormat  = vFormat(_c[_i_]);
         vLabel   = vLabel(_c[_i_]);
         vr.add();
         end;
      do _i_ = 1 to dim(_n);                                              7
         vName    = vName(_n[_i_]);
         vType    = vType(_n[_i_]);
         vLength  = vLength(_n[_i_]);
         vFormat  = vFormat(_n[_i_]);
         vLabel   = vLabel(_n[_i_]);
         vr.add();
         end;
      vr.output(dataset:'vr');                                            8
      end;
   set OneWayFreqs;                                                       9
   vName     = scan(table,-1,' ');                                        10
   formatted = vvalueX(vName);                                            11
   vr.find();                                                             12
   select(vType);                                                         13
      when('N') nValue = _n[_i_];
      when('C') cValue = _c[_i_];
      end;
   keep vName vLabel vType Vlength vFormat                                14
        nValue cValue Formatted
        Frequency Percent CumFrequency CumPercent;
   run;                                                                   15
```

1. The familiar DATA statement to start the definition of a data step program and name the output data set.
2. LENGTH used to define variables. Variables beginning with V will contain attributes of the normalized variables. CVALUE and NVALUE will hold the values of the character and numeric normalized variables respectively. FORMATTED is the formatted value of each variable created and _I_ is the array index variable.
3. The unexecuted SET statement defines all the variables in the input data. This is used at this location in the program in order to have these variables included in the arrays defined in “3”.
4. Define two arrays one of all the character variables another of all the numeric variables. These arrays will include all the variables that were summarized by PROC FREQ plus all other variables that have been named in the LENGTH statement. We are interested in the ones summarized by PROC FREQ but we don’t know the names.
5. Do the next statements only one time when _N_=1.

6. Declare a HASH object named VR and define the attributes of the HASH object using the methods
    a. DEFINEKEY defines VNAME as the variable that the HASH is keyed on.
    b. DEFINEDATA defines the variables that are stored as data in the HASH.
    c. DEFINEDONE signals no more DEFINEKEY or DEFINEDATA method statements follow.
7. Fill the HASH with the data about the variables defined by the arrays _C and _N. The V functions are used to extract the relevant metadata about each variable and the array index _I_ associated the each variable. This data is stored VR.ADD() in the hash VR indexed by VNAME.
8. The OUTPUT method creates a data set of the data stored in the HASH. This is not an important part of the program and is only done to make it easier to see what data is contained in the hash object. See Exhibit 3 for contents and complete data listing.
9. Read the data created by PROC FREQ.
10. Parse the variable TABLE for the name of the variable that the current observation is associated with.
11. Using VNAME create a formatted version of the variable using VVALUEX. If we were just making a stacked data summary table this would probably be the only normalized value we would need.
12. Look up the metadata for the variable identified by VNAME using the FIND method. This sets the values of all the V variables defined above and the all important array index _I_.
13. Now that we know the type and array index assign NVALUE or CVALUE accordingly.
14. KEEP the variables of interest.
15. Signal the end of the step.

EXAMPLE 2: READING MULTIPLE CSV FILES WITH CHANGING NUMBER OF VARIABLES.

This question was posted on SAS-L Reading in Multiple CSV files with changing number of variables regarding reading CSV files. The user mentions that there are thousands of files and the files do not have exactly the same structure, all fields are not present in all files or in the same order. The only thing we know is the names of the fields we want to read and the variable type for each field. I suppose we could write a macro loop program to call PROC IMPORT for each CSV file but did I mention there are thousands of files to read. Macro looping on PROC IMPORT would probably take a very long time and then we still need to combine the SAS data sets. By using the technique demonstrated here we can write a single data step program that will read all the files and create a single SAS data set.

Consider three small CSV files TT04a.csv, TT04b.csv, and TT04c.csv, they will provide input to the example.

```
TT04A.csv          TT04B.csv          TT04C.csv
x,y,z              z,x,a,s            x,z,dob
1,3,2              5,55,5,M           3,99,03/29/2008
.a,2,3             6,2,6,F            .b,1,05/22/1985
1,9,7
```

For this example we will define a data set that has variables X, Y, DOB, and S. Notice that one of the fields Z will not be included in the final data set. To communicate the variable information to the data step that reads the CSV files a data set is defined that has all the variables of interest including any INFORMATS, FORMATS and LABELS. INFORMATS are particularly import for reading any field that cannot be read with a default character of numeric INFORMAT. The following data step is used in our example; it creates a data set with zero observations. CALL MISSING is used to keep the step from generating uninitialized variable messages.

```
data csvData0;
   attrib x          length=  8;
   attrib y          length=  8;
   attrib dob        length=  8 informat=mmddyy10. format=date9.;
   attrib s          length= $1 informat=$1.;
   stop;
   call missing(of _all_);
   run;
```

After reading the CSV files a data set with the following data will be created.

Obs	x	y	dob	s	File Name
1	1	3	.		TT04A
2	A	2	.		TT04A
3	1	9	.		TT04A
4	55	.	.	M	TT04B
5	2	.	.	F	TT04B
6	3	.	29MAR2008		TT04C
7	B	.	22MAY1985		TT04C

The data step that follows is a dynamic CSV reader. It will read the file(s) defined by the file reference using the variable information passed to it by the data set CSVDATA0.

```
filename FT16F001 'TT04*.csv';                                          1
data csvData;                                                           2
   length cdummy $1 ndummy 8;                                           3
   call missing(cdummy,ndummy);                                         4
   if 0 then set csvData0;                                              5
   array _cvars[*] _character_;                                         6
   array _nvars[*] _numeric_  ;

   length vname $32 vtype $1 k j 8;                                     7
   declare hash vinfo();                                                8
   vinfo.definekey('vname');
   vinfo.definedata('vname','k','vtype');
   vinfo.definedone();

   do k = 1 to dim(_cvars);                                             9
      vname = upcase(vname(_cvars[k]));
      vtype = vtype(_cvars[k]);
      vinfo.add();
      end;
   do k = 1 to dim(_nvars);
      vname = upcase(vname(_nvars[k]));
      vtype = vtype(_nvars[k]);
      vinfo.add();
      end;
   vinfo.output(dataset:'vinfo');                                       10

   declare hash finfo(ordered:'a');                                     11
   finfo.defineKey('j');
   finfo.defineData('vname','j');
   finfo.defineDone();
   declare hiter fI('finfo');                                           12

   length fname $256 FileName $64;                                      13
   infile FT16F001 dsd missover filename=fname eof=eof eov=eov;         14
   do _n_ = 1 by 1;                                                     15
      call missing(of _cvars[*] _nvars[*]);                             16
      input @;                                                          17
      if _n_ eq 1 or eov then do;                                       18
         finfo.clear();                                                 19
         FileName = scan(Fname,-2,'\.');                                20
         eov = 0;                                                       21
         do j = 1 by 1;                                                 22
            input vname:$upcase32. @;                                   23
            if missing(vname) then leave;                               24
            finfo.add();                                                25
            end;
         finfo.output(dataset:csvfilename);                             26
         end;
      else do;                                                          27
         do while(fi.next() eq 0);                                      28
            rc = vinfo.find();                                          29
```

```
            if rc eq 0 then select(vtype);                          30
               when('C') input _cvars[k] @;
               when('N') input _nvars[k] @;
               end;
            else input cdummy:$1. @;                                31
            end;
         output;                                                    32
         end;
      input;                                                        33
      end;
eof:                                                                34
  stop;                                                             35
  drop vname vtype k j cdummy ndummy rc;                            36
  run;
```

1. Define a FILEREF with wildcard which allows reading any number of file as if they are one stream of data. INFILE statement options will provide variables to differentiate the files as they are being read.
2. Signal the start of a data step program and name the output data set.
3. Define two “dummy” variables one character and one numeric to insure the ARRAY statements don’t fail if the data set CSVDATA0 has no character or numeric variables.
4. CALL MISSING insures there are no uninitialized variable messages.
5. This unexecuted SET statement defines the variables in CSVDATA0 to this data step.
6. Now define two arrays of all the variables that are defined at this point in the complication; one for all the character variables and another of all the numeric variables. It is important to understand that variables defined following these statements are not included in these two arrays.
7. Define utility variables:
    a. VNAME, VTYPE and K are used in the VINFO hash they provide the reference to the index to the arrays defined in “6”.
    b. J is the index to the HASH of column names read from the first record of each CSV file
8. Declare a HASH object named VINFO and define the attributes of the HASH object using methods
    a. DEFINEKEY defines VNAME as the variable that the HASH is keyed on.
    b. DEFINEDATA defines the variables that are stored as data in the HASH.
    c. DEFINEDONE signals no more DEFINEKEY or DEFINEDATA method statements follow.
9. Fill the variable information hash VINFO with each variables type, VTYPE, and array index K.
10. Output the data in the VINFO hash to a SAS data set. The data shown here for our example shows the information needed to reference the data in the arrays defined in “6”. If we want to reference variable X this hash tells us the array is the numeric array, VTYPE=N with array index K=2.

Obs	vname	k	vtype
1	X	2	N
2	Y	3	N
3	CDUMMY	1	C
4	NDUMMY	1	N
5	DOB	4	N

11. Define another hash, FINFO, to hold the CSV file field information. This hash is indexed on J the position of the field name in the field names record of each CSV. The data item is the field name. As the values of this hash are read the data values stored in VNAME provide the lookup to the array index in the VINFO hash.
12. Declare a hash iterator object. The iterator object allows the program to use the hash like an array and easily process the hash based on the lookup keys. For each data record in the CSV files the program will iterate over the field information hash to read each field from the CSV.
13. Declare two variables to facilitate working with the CSV filenames.
14. The input statement opens the FILEREF FT16F001 and specifies the various options needed.
    a. DSD (delimiter-sensitive data) used to specify how the delimiters in the CSV file are handled.
    b. MISSOVER specifies that attempts to read past the end of record cause the variable to be assigned a missing value while not flowing to the next record.

c. FILENAME= specifies a variable that will receive the full path name of the file where the current records are being read from.
d. EOF= specifies a statement label where the program will jump to when the program reaches end of file.
e. EOV= specified a flag variable that is set to 1 when the first record of the concatenated files are read. The program will use this flag to determine when to read the field names for each file.

15. The program uses and iterative DO instead of the tradition DATA step loop to read the file. This is an infinite loop but the INFILE statement option EOF will save us.
16. Since the program is not using the data step loop CALL MISSING is used to insure that the values being read are initialized to missing.
17. Each time the program reads a record it needs to determine if the record is a data record or field name record so each record is read and held with the "trailing at".
18. If the record is the first one _N_=1 or the first record in the concatenation EOV=1 process the field names. EOV is not set to one for first file read that is why we need to check _N_ EQ 1.
19. The field information hash is cleared because each file in the concatenation has potentially different fields in a different order.
20. Scan the relevant information from FNAME to create a variable to identify the source file for each record. FNAME is set to the full path of the file being read.
21. Reset EOV, this variable is automatically set by the INFILE statement but the user has to set it back to zero or the value stays one until it is reset to one again by the INFILE statement.
22. Start a loop and count by 1.
23. Read a field name from the field names record of the CSV.
24. Check if VNAME is missing and exit loop if it is. The MISSOVER option sets VNAME to missing when the fields are exhausted, i.e. there is an attempt to read past the end of the record.
25. Using the ADD method add the J^{th} field name to the field information hash. This hash is essentially an array that will change dimension for each new CSV.
26. Using the OUTPUT method output the FINFO hash to a data set, this is for the example only, but makes it easy to see the result of reading the field name record. Here you can see that there are three fields X, Y and Z. With the data in this hash we know the name of each field which in turn can be used to lookup the information need to read into a like named variable.

WORK.TT04A

Obs	vname	j
1	X	1
2	Y	2
3	Z	3

WORK.TT04B

Obs	vname	j
1	Z	1
2	X	2
3	A	3
4	S	4

WORK.TT04C

Obs	vname	j
1	X	1
2	Z	2
3	DOB	3

27. When the record is not a field information record but a data record do the work to read the fields.
28. Using the hash iterator created in "12" process all the records in the hash. If we were using an ARRAY this would be "DO J=1 TO DIM(FINFO);" the difference here is we don't know the dimension.
29. Using the FIND method and the key stored in VNAME returned from iterating over the FINFO hash look-up the array index information stored in the VINFO hash. RC will contain information on the success of this operation. If the field is not found in the VINFO hash the value of RC will not be zero, indicating a field that will not be read into the output data set.
30. If the FIND was successful RC=0 then we can read the field based on data type and array index.
31. Otherwise read the field as CDUMMY to advance the input pointer to the next field.
32. Write the record to the output data set named in "2".
33. Release the held input record.
34. This is the statement label named it the INFILE statement option EOF. This is object of an implied GOTO statement, the program branches here on end of file.
35. When the program jumps to EOF: STOP.
36. Drop the utility variables.

CONCLUSION

This paper has described two programs where the variables in an array are reference by name. This is accomplished using a hash keyed on variable name that stores the array index in the data section of the hash. This technique is useful for processing ODS output data sets like those created by PROC FREQ. It is especially useful where an application reads CSV files when the order of the fields is not consistent from file to file, or there are fields that are unwanted or unexpected.

CONTACT INFORMATION

John Henry King
Ouachita Clinical Data Services, Inc.
1769 Hwy 240 West
Hopper, AR 71935
PH: 501-351-0432

EXHIBIT 1: ODS OUTPUT ONEWAYFREQS

This is a listing of most the variables used in the Example 1 variables not relevant to the example are omitted.

Table	Name	Frequency	Percent	Sex	Age	Height	Weight
Table Name	Alfred	1	5.2632		.	.	.
Table Name	Alice	1	5.2632		.	.	.
Table Name	Barbara	1	5.2632		.	.	.
Table Name	Carol	1	5.2632		.	.	.
Table Name	Henry	1	5.2632		.	.	.
Table Name	James	1	5.2632		.	.	.
Table Name	Jane	1	5.2632		.	.	.
Table Name	Janet	1	5.2632		.	.	.
Table Name	Jeffrey	1	5.2632		.	.	.
Table Name	John	1	5.2632		.	.	.
Table Name	Joyce	1	5.2632		.	.	.
Table Name	Judy	1	5.2632		.	.	.
Table Name	Louise	1	5.2632		.	.	.
Table Name	Mary	1	5.2632		.	.	.
Table Name	Philip	1	5.2632		.	.	.
Table Name	Robert	1	5.2632		.	.	.
Table Name	Ronald	1	5.2632		.	.	.
Table Name	Thomas	1	5.2632		.	.	.
Table Name	William	1	5.2632		.	.	.
Table Sex		9	47.3684	F	.	.	.
Table Sex		10	52.6316	M	.	.	.
Table Age		7	36.8421		11	.	.
Table Age		12	63.1579		13	.	.
Table Height		1	5.2632		.	51.3	.
Table Height		1	5.2632		.	56.3	.
Table Height		1	5.2632		.	56.5	.
Table Height		1	5.2632		.	57.3	.
Table Height		1	5.2632		.	57.5	.
Table Height		1	5.2632		.	59.0	.
Table Height		1	5.2632		.	59.8	.
Table Height		2	10.5263		.	62.5	.
Table Height		1	5.2632		.	62.8	.
Table Height		1	5.2632		.	63.5	.

Table	Height	1	5.2632	.	64.3	.
Table	Height	1	5.2632	.	64.8	.
Table	Height	1	5.2632	.	65.3	.
Table	Height	2	10.5263	.	66.5	.
Table	Height	1	5.2632	.	67.0	.
Table	Height	1	5.2632	.	69.0	.
Table	Height	1	5.2632	.	72.0	.
Table	Weight	1	5.2632	.	.	50.5
Table	Weight	1	5.2632	.	.	77.0
Table	Weight	1	5.2632	.	.	83.0
Table	Weight	2	10.5263	.	.	84.0
Table	Weight	1	5.2632	.	.	84.5
Table	Weight	1	5.2632	.	.	85.0
Table	Weight	1	5.2632	.	.	90.0
Table	Weight	1	5.2632	.	.	98.0
Table	Weight	1	5.2632	.	.	99.5
Table	Weight	2	10.5263	.	.	102.5
Table	Weight	2	10.5263	.	.	112.0
Table	Weight	2	10.5263	.	.	112.5
Table	Weight	1	5.2632	.	.	128.0
Table	Weight	1	5.2632	.	.	133.0
Table	Weight	1	5.2632	.	.	150.0

EXHIBIT 2: DATA NORMALIZED

This is the data created in Example 1, the normalized output from PROC FREQ.

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	vName	Char	32		
2	vLabel	Char	256		
3	vType	Char	1		
4	vLength	Num	8		
5	vFormat	Char	32		
6	nValue	Num	8		
7	cValue	Char	64		
8	Formatted	Char	256		
9	Frequency	Num	8	BEST8.	
10	Percent	Num	8	6.2	
11	CumFrequency	Num	8	BEST8.	Cumulative Frequency
12	CumPercent	Num	8	6.2	Cumulative Percent

Variables 9 through 12, Frequency, Percent, CumFrequency, and CumPercent and some observations are omitted from the listing below.

Obs	vName	vLabel	vType	vLength	vFormat	nValue	cValue	Formatted
1	Name	Name	C	8	$8.	.	Alfred	Alfred
2	Name	Name	C	8	$8.	.	Alice	Alice
3	Name	Name	C	8	$8.	.	Barbara	Barbara
...								
19	Name	Name	C	8	$8.	.	William	William
20	Sex	Gender	C	1	$SEX6.	.	F	Female
21	Sex	Gender	C	1	$SEX6.	.	M	Male

Obs	vName	vLabel	vType	vLength	vFormat	nValue	cValue	Formatted
22	Age	Age Group	N	8	AGE5.	11.0		Tween
23	Age	Age Group	N	8	AGE5.	13.0		Teen
24	Height	Height	N	8	BEST12.	51.3		51.3
25	Height	Height	N	8	BEST12.	56.3		56.3
…								
40	Height	Height	N	8	BEST12.	72.0		72
41	Weight	Weight	N	8	BEST12.	50.5		50.5
42	Weight	Weight	N	8	BEST12.	77.0		77
…								
55	Weight	Weight	N	8	BEST12.	150.0		150

EXHIBIT 3: DATA SET VR

This is data set VR created by the OUTPUT method in Example 1, it is the data contain in the hash VR.

Variables in Creation Order

#	Variable	Type	Len
1	vName	Char	32
2	vLabel	Char	256
3	vType	Char	1
4	vLength	Num	8
5	vFormat	Char	32
6	_i_	Num	8

Obs	vName	vLabel	vType	vLength	vFormat	_i_
1	nValue	nValue	N	8	BEST12.	2
2	vLabel	vLabel	C	256	$256.	2
3	F_Height	Height	C	6	$6.	13
4	vLength	vLength	N	8	BEST12.	1
5	CumPercent	Cumulative Percent	N	8	F6.2	6
6	Name	Name	C	8	$8.	9
7	vType	vType	C	1	$1.	3
8	vFormat	vFormat	C	32	$32.	4
9	F_Weight	Weight	C	6	$6.	14
10	Weight	Weight	N	8	BEST12.	9
11	vName	vName	C	32	$32.	1
12	F_Name	Name	C	7	$7.	8
13	Table	Table	C	256	$256.	7
14	Formatted	Formatted	C	256	$256.	6
15	Height	Height	N	8	BEST12.	8
16	cValue	cValue	C	64	$64.	5
17	CumFrequency	Cumulative Frequency	N	8	BEST8.	5
18	Percent	Percent	N	8	F6.2	4
19	Age	Age Group	N	8	AGE5.	7
20	F_Sex	Sex	C	6	$6.	10
21	F_Age	Age	C	5	$5.	12
22	Frequency	Frequency	N	8	BEST8.	3
23	Sex	Gender	C	1	$SEX6.	11