Reading and Processing Mystery Data Sets

Jimmy DeFoor
Benbrook, Texas

Introduction

Don’t want to know and especially don’t want to code the variable names of a SAS data set? Want that data set to remain mystery in the details? Then, use the data dictionary for that data set as a source for SAS macro variables and then use those macro variables in SAS arrays. Just sort the variable names according to their common purpose or use and by their character or numeric and types. Then read those SAS variable names into macro variables using Call Symput. Next, retrieve those SAS variable names into SAS array definitions by using the macro variables there. Finally, use those SAS arrays to perform the desired actions with or on the SAS variables.

This paper describes the process outlined above. It begins with an overview of SAS arrays, macros, and the data dictionary view ‘vcolumn’. Next, it describes how the arrays, macros, and the vcolumn view are used to investigate the contents of data provided by the three credit bureaus: Epsilon, Equifax, and Transunion.

Vcolumn View

This SAS view contains the names of the libraries, data sets, and variable names associated with the session in which SAS is executing. The work library is placed there automatically. Other libraries are added as their library names are referenced in the SAS code.

Also stored in the data dictionary are the lengths, labels, formats, types (numeric or character) of the variables, informats, order of the variables, and positions of the variable.

The vcolumn view is located in the SASHELP catalog. You can view its contents in the SAS explorer window or pull its contents into a SAS data set. Proc SQL is the most efficient way to get the metadata. Below is an example of the data that could be retrieved. This was pulled from the data set named CARS that is also located in SASHELP.

libname	**memname**	**memtype**	**name**	**type**	**length**	**npos**	**format**
SASHELP	CARS	DATA	Make	char	13	80	
SASHELP	CARS	DATA	Model	char	40	93	
SASHELP	CARS	DATA	Type	char	8	133	
SASHELP	CARS	DATA	Origin	char	6	141	
SASHELP	CARS	DATA	DriveTrain	char	5	147	
SASHELP	CARS	DATA	MSRP	num	8	0	DOLLAR8.
SASHELP	CARS	DATA	Invoice	num	8	8	DOLLAR8.

Arrays

In SAS, an array is just a convenient way of temporarily identifying a group of variables.
It is not a data structure that stands independently of a group of variables. It exists only during the execution of a single Data Step and is used to execute a common action against a group of variables. The array must be specified again in a subsequent Data Step if the same variable group is to be processed together again.

Arrays are specified with an array name, a count of the number of variables, and then a listing of the variables to be included in the array.

```
Array charvars (5) charvar1 charvar2 charvar3 charvar4 charvar5;
```

After being tied to an array, each variable can be accessed through the array name and the order in which it was specified in the array. For example, charvar4 was listed 4th in the array, so its array position is 4. Its specific array reference is, thus, charvars (4).

Variables must already exist when assigned to an array or they must be created during the array specification. Variables created by an array statement must be given a list of initial values. The variables can either be permanent or temporary.

```
Array numvars  (5) numvar1 numvar2 numvar3 numvar4 numvar5 (1 5 8 3 6);

Array tempvars (5) _temporary_ (1 5 8 3 6);
```

The variables of a temporary array cannot be accessed directly. They can only be accessed via the array name and the array position; e.g., tempvars(3) = 10. Plus, they cannot be written to a data set.

The values of the variables defined in an array are retained during each read of the input data set just as if they had been initially defined with a Retain statement.

Array variables can also be specified so that they are either defined with a leading 0 in their names or created with the leading zero in their names.

```
Array oldvars(10) A01-A10;

Array newvars(10) N01-N10 (1 7 3 5 9 2 8 5 6 10);
```

The usual method of accessing an array is to reference it within a Do loop so that the variables of the array are accessed in the order of their specification in the array.

```
Do J = 1 to 10;
  Oldvars(j) = newvars(j);
End;
```

Macro Variables and Macro Definitions

Macro variables are simply holders of text strings. Even if the text is an integer, it is a character when handled by the macro processor unless seen within the %Eval function.

Macro variables are often created and assigned a string via a %Let statement. If specified in open code, such as the first statement in a SAS program, the macro variable and its string are loaded into the global symbol table.

```
Let yr = 2010;
```

But they can also be created and assigned in the call of a compiled macro. This code would call the compiled macro definition named Report and then create and assign the macro variables yr and bur into the local symbol table of Report.

```
%Report(yr=2011,bur=tu);
```

Macro variables are always defined in symbol table. If the table is available to any macro or to the SAS code, the variable is defined in the global symbol table. If it is accessible only within a compiled macro, it is in the local symbol table of that macro. The same name could be defined in two symbol tables, but it would be two different macro variables.

One of the most useful ways to define and load a macro variable is with the Call Symput statement. It is executed within a Data Step and passes strings into the macro variables each time the Call is executed.

The Call Symput can pass a quoted string to a macro variable or it can pass the contents of a variable. In addition, the macro variable may be specified as a quoted string or as the contents of a variable. Here are some examples:

```
/* Assign ‘red’ to the macro variable test */
Call Symput(‘test’,’red’);
/* Assign the data value of color to the macro variable test. */
Call Symput(‘test’,color);
/* Assign ‘red’ to the macro variable created from the data value of test. */
Call Symput(test,’red’);
/* Assign the data value of color to the macro variable created from the */
/* data value of test. */
Call Symput(test,color);
```

Regardless of how they are created, the content of the macro variable is retrieved when the variable is used with a leading ampersand (&), such as in &test. The SAS processor passes the & strings to the macro processor and the macro processor returns the related character string.

Combinations of ampersands are often used to allow macro code to retrieve multiple macro variable strings within a %Do Loop. Such code allows multiple strings of SAS code to be generated by the code. %Do loops cannot be used in open code like macro variables can be. They must be used in a compiled macro which will be stored and retrieved from a catalogue.

```
%macro create;

Length
%Do J = 1 %to 5;
   &&test&j 4
%end;
;
%mend create;
*;
```

When compiled successfully, the create macro would be called and executed at the spot in the SAS code where the %create was located.

```
Data work1;
  %create
  Set srcedata;
Run;
```

For example, the length statement would be inserted by the macro processor below Data work1. Then, the processor would resolve &&test&j five times and retrieve the associated string from the symbol table. Last, it would place a semicolon after the 4 that is the length of var5.

```
Data work1;
  Length var1 4 var2 4 var3 4 var4 4 var5 4;
  Set srcedata;
Run;
```

The resolution of &&test&j will be indirect (multiple pass). On the first pass of the first loop, the && will become a single & and the &j will become a 1, establishing a new macro variable of &test1. On the second pass, the symbol table will then be searched for &test1 and, in this example, the string of var1 will be retrieved. This indirect or multiple resolution will occur on each loop through the code.

Employ These Tools to Investigate Bureau Data.

Now I will show you how to use these tools to read the combined Bureau data set and to investigate the contents of each field.

Access Data Dictionary Vcolumn to Get the Names of Variables

The Vcolumn table has the names, lengths, types, and formats for each variable in a SAS data set. Also in the table are the names of the SAS data sets and the names of the library in which the variables may be found. The table entries can be retrieved using Proc SQL or a SAS data step, but retrieval is much faster using Proc SQL.

First, I retrieved the dictionary content I wanted by specifying the memname (data set), its memtype, and its libname in the where clause of the Select statement. I then read the work1 data set created from the Proc Sql into a Data Step, where I identified the names of the variables that were from each Bureau file. All three bureau files had been joined into the same SAS data set in Proc SQL through a join by indiv_id, with the first one group of attributes being from Transunion, the second from Experian, and the third from Equifax.

During the join, Proc SQL differentiated the common names of the variables from the different sources. To do that, it added 0 to the second group of bureau variables, and 1 to the end of the third group. All three data sources had the same variables in the same order and the first variable in each group started with the string BURSEQ.

```
Proc SQL;
   Create Table work1 as
    Select *
        From sashelp.vcolumn
          Where libname = 'MINE' and
                memtype = 'DATA' and
                memname = ‘BUREAU’;
  quit;
*;
data names;
  length group $2;
  retain group;
  keep group libname memname name type length varnum;
  set work1;
  if      upcase(name) = 'BURSEQ' then
    group = 'TU';
  else if upcase(name) = 'BURSEQ0' then
    group = 'EX';
  else if upcase(name) = 'BURSEQ1' then
    group = 'EQ';
  output;
run;
*;
```

Create Macro Variables with the Names of Each of the Variables

My goal was to put the variables into the same order within each group (bureau) and type (num or char). This would enable me to create an array for each group and type and allow me to process the same attributes at the same time. For example, I would be able to process all of the char variables separate from the num variables, which array processing requires. I would also be able to process one attribute variable for one bureau at the same time I was processing the same variable for another bureau because they would be at the same position in each array, such as position 10 (Attr64, for example).

There isn't enough room (or value) for showing all of the attribute variables in each array, but each array had a form like below and were composed of either numeric or character variables.

```
Array tuchar (10)
       Tu_attr55  Tu_attr56  Tu_attr57 ....... Tu_attr64;
Array tunum (10)
       Tu_attr45  Tu_attr46  Tu_attr47 ....... Tu_attr54;
```

To create these arrays, I sorted the names of variables by their bureau source and their types (char or num), order, and name. I sorted 'by descending group' so that the sort order would match the order of the bureau data in the SAS data set (TU, EX, EQ).

```
proc sort data = names1;
  by descending group type varnum name;
run;
```

Thereafter, I created a macro variable for each SAS variable and a macro variable for the count in each group of variables. I did this by using a Call Symput to load each SAS variable name into a unique macro variable. The code I used, with comments is below.

```
data _null_;
  length var $11;
  retain j 0;
  set names;
  by descending group type;
  if first.type then
    j = 0;
  /* count the rows in data set */
  j = j + 1;
  /* load current row of type of data into cnt variable */
  cnt = left(put(j,3.0));
  /* create the macro variable name to contain SAS variable name */
  /* the values will be tu_char_1, tu_num_1, etc. */
  var = group||'_'||trim(type)||'_'||cnt;
  *put var= ; /* use put statement to show var values in log */
  call symput(var,name);
```

 The macro variable to hold the name of a unique SAS variable was created by concatenating the group (bureau) and type (char or num) with separating strings of underscores ('_') and with an ending value equivalent to order of that variable in the variables of that group and type.

```
var = group||'_'||trim(type)||'_'||cnt;
call symput(var,name);
```

Into each macro variable was loaded the name of the SAS variable on that row. For example, the macro variable **Tu_num_1** was loaded with the SAS variable name **Tu_attr45**. Other macro variables created included Tu_char_1, Eq_num_8, and Ex_char_15.

I also used Call Symput to capture the count of the number of variables that would be assigned to each group and type, such as **Tu_num** and **Tu_char.** This count and the associated macro variables for that group and type were used to write out the num array for the each numeric group and the char array for each character group.

```
  if last.type then
    do;
       /* create the macro variable to contain count of variables within */
       /* each group and type, such as tu_char_cnt and tu_num_cnt */
       var = group||'_'||trim(type)||'_'||'CNT';
       call symput(var,cnt);
       *put var= ; /* use put statement to show var values in log */
    end;
run;
```

Thus, from this process I could create macro variables Tu_num_1 – Tu_num_10 that would contain the first 10 numeric variables in the Tu grouping, such as **Tu_attr45** through **Tu_attr54**, which would be assigned to the **Tunum** array.

This enabled me to use the macro variables in a %Do loop and assign the variables of the array using indirect or multiple resolution.

```
/* array for Tu num variables */
Array tunum (&tu_num_cnt)
 %Do j = 1 %to &tu_num_cnt;
    &&tu_num_&j
 %end;
   ;
```

Upon resolution, this macro code became

```
/* array for Tu num variables */
Array tunum (10)
      tu_attr45 tu_attr46 tu_attr47 tu_attr48 tu_attr49
      tu_attr50 tu_attr51 tu_attr52 tu_attr53 tu_attr54
      ;
```

Macro variables are most often resolved using direct reference. For example, the Macro statement **%Let city = Fort Worth** would load the string 'Fort Worth' into the macro variable 'city'. The macro processor could use this info to resolve the statement **Retain City "&city"** into **Retain City "Fort Worth"** before sending the statement to the SAS processor. The macro processor resolves all of a macro statement before passing the results to the SAS processor.

There is both direct and indirect reference in the above code. The macro variable **&tu_num_cnt** resolves to 10 through direct reference by the macro processor, becoming **Array tunum (10)** from **Array tunum (&tu_num_cnt)**.

Next, the macro processor executes the %Do Loop, which it can only process within a compiled macro. Macro variables can be resolved directly or indirectly in open SAS code or within a compile macro, but indirect reference is most useful when managed within a %Do Loop because the loop alters the value of &j incrementally and, thus, allows multiple indirect resolutions of the same macro reference.

```
 %Do j = 1 %to 10;
    &&tu_num_&j
 %end;
```

The macro processor would execute the %Do Loop ten times, resolving **&&tu_num&j** ten times.

In the first loop, the macro processor would resolve &**&tu_num_&j** to **&tunum_1** on its first pass and then to **Tu_attr45** on its second pass. Next, it would add **Tu_attr45** to the SAS code **Array tunum (10)** and increment the value of j by one (1). Then, it would resolve &**&tu_num_&j** to **&tunum_2** on the first pass and to **Tu_attr46** on the second pass. After two loops through the macro code, the array statement sitting in the SAS processor would be:

```
Array tunum (10) Tu_attr45 Tu_attr46
```

After all passes through the %Do Loop, the complete SAS statement would be:

```
Array tunum (10)
      tu_attr45 tu_attr46 tu_attr47 tu_attr48 tu_attr49
      tu_attr50 tu_attr51 tu_attr52 tu_attr53 tu_attr54
      ;
```

I also used the same technique to create new variables that I would use for storing the results of my comparisons of the bureau variables. I built code that retrieved the SAS variable name and then created a new name by adding '_ck' to the end of it. For example, a new variable of Tu_attr45_ck was created from Tu_attr45. I added the **%trim** function to drop any trailing blanks from the resolved string before adding the '_ck'. I used this technique in creating a Length statement that would define the _ck variables.

```
Length
     %Do j = 1 %to &tu_char_cnt;
       %let var = %trim(&&tu_char_&j)_ck;
       &var 4
     %end;
```

After 10 loops, the SAS processor would have the statement.

```
Length tu_attr45_ck 4 tu_attr46_ck 4 tu_attr47_ck 4 tu_attr48_ck 4
       tu_attr49_ck 4 tu_attr50_ck 4 tu_attr51_ck 4 tu_attr52_ck 4
       tu_attr53_ck 4 tu_attr54_ck 4
       ;
```

To do this I slightly varied the technique that I used in creating the earlier arrays. I loaded the resolved string into a macro variable named ‘var’ using a %Let statement. Below is how the macro code would resolve through each pass of the macro processor before being assigned to ‘var’.

```
1)  %let var = %trim(&&tu_char_&j)_ck;
2)  %let var = %trim(&tu_char_1)_ck;
3)  %let var = %trim(tu_attr45 )_ck;
4)  %let var = tu_attr45_ck;
```

I did this because &&tu_char_&j.ck resolved incorrectly. The ‘_ck’ was spaced one character to the right of the variable name, creating an error. An example would be ‘tu_attr45 _ck’. By the way, a period must be used to end a macro variable if it bumps directly against another string, such as _ck unless wrapped in a function. Using %Trim to drop any blanks at the end of the string tu_attr45 eliminated the need for the period and also removed the ending blank.

With the same method, I was also created an array of the _ck variables so that I could store the results of evaluating each set of attributes.

```
Array charmatch (&tu_char_cnt)
     %Do j = 1 %to &tu_char_cnt;
        %let var = %trim(&&tu_char_&j)_ck;
        &var
     %end;
       ;
```

This macro code resolved to:

```
Array charmatch (10) tu_attr55_ck tu_attr56_ck tu_attr57_ck tu_attr58_ck
                     tu_attr59_ck tu_attr60_ck tu_attr61_ck tu_attr62_ck
                     tu_attr63_ck tu_attr64_ck;
```

Use Arrays to Investigate and Classify Attribute Values

With all of the arrays created, I could now write the SAS code that would inspect each bureau attribute and store the results in the associated **_ck** variable. I used an array reference within a SAS Do loop. The Do loop used the macro variable &tu_char_cnt to set the upper boundary of the loop. The look up of &tu_char_cnt in the symbol table retrieved the value of 10.

```
do j = 1 to &tu_char_cnt; /* &tu_char_cnt becomes 10 */
  if      eqchar(j) eq ' ' and
          tuchar(j) eq ' ' and
          tuchar(j) eq ' ' then
     charmatch(j) = 1;     /* missing all */
  else if eqchar(j) eq ' ' and
          tuchar(j) eq ' ' then
     charmatch(j) = 2;     /* missing two  */
  else if exchar(j) eq ' ' and
          tuchar(j) eq ' ' then
     charmatch(j) = 2;     /* missing two  */
  else if exchar(j) eq ' ' and
          eqchar(j) eq ' ' then
     charmatch(j) = 2;     /* missing two */

  /* Intermediate code omitted */

end;
```

After storing the results of each evaluation of the _ck variables in a SAS data set, I then created another Data Step to classify and sum those results.

In that Data Step, I used length statements like those in the Before column to establish the variables that I would use for classifying the results stored in the _ck variables. The macro variable `&tu_char_cnt` was used to define the last variable in each group. The After column shows the statements after resolution.

Before	**After**
`length co1 - co&tu_char_cnt 4;`	`length co1 - co10 4;`
`length cw1 - cw&tu_char_cnt 4;`	`length cw1 - cw10 4;`
`length ch1 - ch&tu_char_cnt 4;`	`length ch1 - ch10 4;`
`length cf1 - cf&tu_char_cnt 4;`	`length cf1 - cf10 4;`
`length cv1 - cv&tu_char_cnt 4;`	`length cv1 - cv10 4;`
`length cx1 - cx&tu_char_cnt 4;`	`length cx1 - cx10 4;`
`length cs1 - cs&tu_char_cnt 4;`	`length cs1 - cs10 4;`
`length ce1 - ce&tu_char_cnt 4;`	`length ce1 - ce10 4;`
`length cn1 - cn&tu_char_cnt 4;`	`length cn1 - cn10 4;`
`length ct1 - ct&tu_char_cnt 4;`	`length ct1 - ct10 4;`

I arrayed the variables created from the length statements with this code.

```
 array charone  (&tu_char_cnt) co1 - co&tu_char_cnt;
 array chartwo  (&tu_char_cnt) cw1 - cw&tu_char_cnt;
 array charthre (&tu_char_cnt) ch1 - ch&tu_char_cnt;
 array charfour (&tu_char_cnt) cf1 - cf&tu_char_cnt;
 array charfive (&tu_char_cnt) cv1 - cv&tu_char_cnt;

/* Intermediate code omitted */

 array charten  (&tu_char_cnt) ct1 - ct&tu_char_cnt;
```

Again, using direct reference, these statements resolve as follows:

```
 array charone  (10) co1 - co10;
 array chartwo  (10) cw1 - cw10;
 array charthre (10) ch1 - ch10;
 array charfour (10) cf1 - cf10;

/* Intermediate code omitted */

 array charten  (10) ct1 - ct10;
```

Next, I used the arrays to sum the number of ways each attribute was classified. The code finds the classification assigned to each attribute on each record and then adds one (1) to the correct variable for that classification. For example, if a record had an attribute that was classified with a value of five (5), a one (1) would be added to the counts in charfive(J). Another attribute on that record that had been marked with value of two (2) would have a one (1) added to the counts in chartwo(J).

```
do j = 1 to &tu_char_cnt;  /* &tu_char_cnt becomes 10 */
  if      charmatch(j) =  1 then
      charone(j)  = charone(j) + 1;
  else if charmatch(j) =  2 then
      chartwo(j)  = chartwo(j) + 1;
  else if charmatch(j) =  3 then
      charthre(j) = charthre(j) + 1;
  else if charmatch(j) =  4 then
      charfour(j) = charfour(j) + 1;

  /* Intermediate code omitted */

  else if charmatch(j) = 10 then
      charten(j)  = charten(j) + 1;
end;
```

Finally, two more arrays and multiple output statements are used to transpose the counts on the last record of the input data set. This code writes out the counts so that the totals for each attribute are in a single field and on a separate record. Otherwise, the totals for each attribute would be in ten fields.

```
array numtot (&tu_num_cnt)
  %Do j = 1 %to &tu_num_cnt;
    %let var = %trim(&&tu_num_&j)_tot;  /* example: attr45_tot */
    &var
  %end;
   ;
array chartot (&tu_char_cnt)
  %Do j = 1 %to &tu_char_cnt;
    %let var = %trim(&&tu_char_&j)_tot; /* example: attr55_tot */
    &var
  %end;
   ;
if eof then
  do;
     cat = '01';
     do j = 1 to &tu_char_cnt;
       chartot(j) = charone(j);
     end;
     do j = 1 to &tu_num_cnt;
       numtot(j)  = numone(j);
     end;
     output;
     cat = '02';
     do j = 1 to &tu_char_cnt;
       chartot(j) = chartwo(j);
     end;
     do j = 1 to &tu_num_cnt;
       numtot(j)  = numtwo(j);
     end;
     output;

     /* Intermediate code omitted */
     cat = '10';
     do j = 1 to &tu_char_cnt;
       chartot(j) = charten(j);
     end;
     do j = 1 to &tu_num_cnt;
       numtot(j)  = numten(j);
     end;
     output;
  end;
```

For example, the counts for Attr55 could have been in charone(1) through charten(1) on one row. Now, they will be on ten rows in chartot(1). This will allow the counts each attribute to be viewed separately. Referenced with the user format on the left, the data could be displayed as on the right.

User Format

```
Proc Format;
  value $cat
    '01' = 'Missing all'
    '02' = 'Missing twice'
    '03' = 'Zero all'
    '04' = 'Zero twice'
    '05' = 'One all'
    '06' = 'One twice'
    '07' = 'Equal all'
    '08' = 'Equal twice'
    '09' = 'Equal none'
    '10' = 'Not assigned'
   ;
 run;
```

Category **Attr51_tot**

```
Missing all          100
Missing twice         55
Zero all             230
Zero twice           386
One all              900
One twice            321
Equal all             12
Equal twice          123
Equal none            87
Not assigned         205
```

Summary and References

I hope this information has been useful. You can find more out about arrays and Proc SQL in prior papers of mine.

Proc SQL – A Primer for SAS® Programmers
Proceedings of the Thirty-First SAS Users Group International Conference
http://www2.sas.com/proceedings/sugi31/250-31.pdf

Using Do Statements, Links, and Arrays
Proceedings of the Second Annual SAS Global Forum (SGF) 2008 Conference
http://www2.sas.com/proceedings/forum2008/179-2008.pdf

Kirk Paul Lafler also has many good papers on SAS Macros, Proc SQL and the use of the data dictionary tables. Here are several.

Exploring DICTIONARY Tables and Views
Proceedings of the 4th Annual SAS Global Forum (SGF) 2010 Conference
http://support.sas.com/resources/papers/proceedings10/155-2010.pdf

SAS® Macro Programming Tips, Tricks and Techniques
Proceedings of the Midwest SAS Users Group 2010 Conference
http://www.mwsug.org/proceedings/2011/sas101/MWSUG-2011-S102.pdf

DATA Step versus PROC SQL Programming Techniques
Proceedings of the Southeast SAS Users Group 2009 Conference
http://analytics.ncsu.edu/sesug/2009/FF003.Lafler.pdf