SESUG Paper 117-2019

Macroception: Maximizing the Capacity of your Macro Variables

Connor Cosenza, Ankura Consulting Group LLC

ABSTRACT

SAS® imposes a maximum character limit of 65,534 on macro variables. Running up against this limit can be frustrating and time-consuming. The author has run into this problem when using macro variables to search long lists of values or to dynamically generate code. This paper provides a program which solves this problem by partitioning the values to be encoded in a given macro variable, assigning those values to sub-macro variables, and encoding a parent macro variable with the names of the sub-macro variables. When the parent macro variable is called, all raw data values are decoded. This significantly increases the maximum number of characters which can be encoded into a single macro – specifically, this theoretical upper-limit increases from 65,534 to 623,883,680 characters.

INTRODUCTION

Sixty-Five Thousand Five Hundred Thirty-Four. Why is this number significant? This is the maximum character length allotted to macro variables in SAS®. When using macro variables to perform large searches or dynamically generate code, it is not difficult to run up against this limit. Doing so can be very frustrating and can require the programmer to either fundamentally alter the logic of the program or otherwise arbitrarily break up the contents of the macro variable into multiple chunks. This can make the code less intuitive and dynamic. This paper provides a solution to this problem which increases this maximum character length to over 600 million.

PROOF OF CONCEPT

Throughout the author's career, he has often had to write SAS® code which queries large data sets against large lists of values provided externally. It is possible to improve the elegance and efficiency of this type of code by using dynamically generated macro variables. Specifically, it is easy to encode a list of search terms into a macro variable and then condition a query on the contents of that macro variable. For example, consider SASHELP.CLASS. Let's say we would like to subset the data by only those students aged 14 and 15. Of course this could be done very simply with the code below:

```
data want;
set sashelp.class;
where Age in (14 15);
run;
```

In certain situations, though, it may be beneficial to separate the search values from the rest of the code. This can be achieved by using a macro variable in place of the actual search values, as shown below:

```
data ValueList;
input ValueList;
datalines;
14
15
;run;

proc sql noprint;
select ValueList into :SearchMacro separated by ' ' from ValueList;
quit;

data want;
set sashelp.class;
where Age in (&SearchMacro.);
run;
```

This produces an identical result but allows for the separation of query and search value selection. This is sufficient most of the time but can break down if the list of search values grows too big. Consider a simple data set that only includes the natural numbers from 1 to 1,000,000:

```
data FullList;
do List = 1 to 1000000;
output;
end;
run;
```

Let's say we would like to subset the data by only the values of List which are multiples of 100. We can try to use our earlier method to achieve this result:

```
data ValueList;
do ValueList = 100 to 1000000 by 100;
output;
end;
run;

proc sql noprint;
select ValueList into :SearchMacro separated by ' ' from ValueList;
quit;
```

But this produces the following error:

```
ERROR: The length of the value of the macro variable SEARCHMACRO (65540) exceeds the maximum
length (65534). The value has been truncated to 65534 characters.
```

Our list of search values includes too many characters (65,540) for SAS® to encode into a macro variable. Only the first 65,534 have been encoded and the remaining 6 characters have been left out. For this reason, we will not be able to directly apply our earlier method to this situation. We are ultimately constrained by a character limit of 65,534 for any given macro variable, but we could greatly increase the maximum output of a single macro variable by encoding that macro variable not with raw data but rather with *other* macro variables into which we encode chunks of the raw data (i.e.: "macroception"). The code given below is one way of achieving this result:

```
proc sql noprint;
select ValueList into :SearchMacro1 separated by ' ' from ValueList
      where ValueList <= 500000;
select ValueList into :SearchMacro2 separated by ' ' from ValueList
      where ValueList > 500000;
quit;

data _null_;
ParentMacro='&SearchMacro1 &SearchMacro2';
call symputx('ParentMacro',ParentMacro);
run;

data SearchResult;
set FullList;
where List in (&ParentMacro.);
run;
```

The code above places half of the search values into &SearchMacro1 and the other half into &SearchMacro2, and then creates the macro &ParentMacro which contains the text '&SearchMacro1 &SearchMacro2'. When we call &ParentMacro in the final DATA step, SAS® decodes this as &SearchMacro1 &SearchMacro2, which it further decodes as the raw data that is encoded in each of the two sub-macro variables. This achieves the desired result and all multiples of 100 are successfully queried from the original data.

EXPLANATION OF CODE

Although this works, it is not particularly elegant or dynamic. Quite a bit of hard-coding is required. Additionally, the example shown above is one of the simplest situations. This code can get significantly more complicated as the number of required sub-macro variables increases. In order to increase code functionality, we must be able to dynamically generate a few measurements and objects such as:

1. The number of rows of raw data
2. The number of sub-macro variables necessary to house the raw data
3. The names of the sub-macro variables that will house the raw data
4. Code that places the names of the sub-macro variables into a single parent macro variable
5. Code that subsets the raw data into mutually exclusive and exhaustive subsets and places each of those subsets of data into a single sub-macro variable

To illustrate this code, consider our ValueList data set defined earlier:

```
data ValueList;
do ValueList = 100 to 1000000 by 100;
output;
end;
run;
```

The code described in this section will use these five steps to ultimately encode a single macro variable with all values from the ValueList data set defined above.

1. DETERMINE THE NUMBER OF VALUES TO BE SEARCHED OVERALL

We can easily determine the number of values to be searched by adding a “*varname* + 1” line to our DATA step and using CALL SYMPUTX() to place the final value into a macro variable, as shown below:

```
data ValueList2;
set ValueList;
Count + 1;
call symputx('Count',Count);
run;
```

At this point, the macro variable &Count contains the number 10,000 because the data set ValueList2 contains 10,000 rows of data.

2. DETERMINE THE NUMBER OF SUB-MACRO VARIABLES NECESSARY

In a similar fashion, we can dynamically determine the number of sub-macro variables required to house all our data. For this example, let’s allow a maximum of 1,000 rows of data to be entered into each sub-macro variable (we will make this more dynamic later). The code given below achieves this result:

```
data _null_;
NumberBins=floor(&Count./1000);
call symputx('NumberBins',NumberBins);
run;
```

In this DATA _NULL_ step, we create the &NumberBins macro variable which is defined as the floor function of &Count divided by 1,000. At this point, the macro variable &NumberBins contains the number 10 because floor(10,000/1,000) = 10.

3. GENERATE THE NAMES OF THE SUB-MACRO VARIABLES

Next, we will create the full list of sub-macro variables names. We will actually generate a total of 11 sub-macro variables, ranging from 0 to 10. These sub-macro variables are named &L0, &L1, &L2, … , &L10 ("L" for List). The code given below achieves this result.

```
data ListOfBins;
do Count = 0 to &NumberBins.;
Base = cats('&L',Count);
output;
end;
run;
```

4. ENCODE THE PARENT MACRO VARIABLE WITH THE NAMES OF THE SUB-MACRO VARIABLES

We will then place the names of these sub-macro variables into our parent macro variable, called &AllBins, as shown below:

```
proc sql noprint;
select Base into :AllBins separated by ' ' from ListOfBins;
quit;
```

To be clear, at this point, &AllBins is an empty shell ready to be filled with raw data. It only contains the text &L0 &L1 &L2 … &L10. The sub-macro variables have not yet been encoded with any data.

5. ENCODE THE SUB-MACRO VARIABLES WITH MUTUALLY EXCLUSIVE AND EXHAUSTIVE SETS OF THE RAW DATA

The final section of this program performs the heavy-lifting of actually dividing the raw data into mutually exclusive and exhaustive sets and encoding each of the sub-macro variables with one of these sets. All code is enclosed in the macro program %AddToBins(), as shown below:

```
%macro AddToBins();
%do BinCount = 0 %to &NumberBins.;

data _null_;
L=cats('L',&BinCount.);
call symputx('L',L);
run;

%global &L.;

proc sql noprint;
select ValueList into :&L. separated by ' '
from ValueList2
where &BinCount.*1000 <= Count <= (&BinCount.*1000)+(1000-1) ;
quit;
%end;
%mend AddToBins;

%AddToBins;
```

Let's take this macro step-by-step.

```
%do BinCount = 0 %to &NumberBins.;
```

The AddToBins macro loops through the data as many times as there are sub-macro variables necessary to exhaust the raw data. In this case, the macro loops 11 times (from 0 to 10). For the next few steps, let's only consider the first loop (when &BinCount = 0).

```
data _null_;
L=cats('L',&BinCount.);
call symputx('L',L);
run;
```

This DATA _NULL_ step generates the macro variable &L which contains the text L0.

```
%global &L.;
```

In this step, SAS® decodes the macro variable &L created in the previous step and creates a new macro variable called &L0 which is null and ready to be filled with raw data.

```
proc sql noprint;
select ValueList into :&L. separated by ' '
from ValueList2
where &BinCount.*1000 <= Count <= (&BinCount.*1000)+(1000-1) ;
quit;
```

This final block of code is analogous to when we created &SearchMacro1 &SearchMacro2 earlier. Here, we place the first 999 rows of raw data from ValueList into the null macro variable &L0 generated in the previous step. Only the first 999 rows of data are included in this first loop because &BinCount is currently equal to 0, so the where statement conditions the ValueList2 data set where Count (which, remember, simply counts the rows from 1 to 10,000) is between 0 (since 0 x 1,000 = 0) and 999 (since 0 x 1,000 + 1,000 – 1 = 999). We can see that in the second loop, &L1 will be encoded with the values from the next 1,000 rows of data (where 1,000 <= Count <= 1,999), and so on. We can test that this works by running the code below, which successfully queries all desired data.

```
data SearchResult2;
set FullList;
where List in (&AllBins.);
run;
```

MAKING THE CODE MORE DYNAMIC

There are two major limitations to this code as it is described above:

1. There is no way to modify the Values of ValueList
2. There is no way to alter the maximum number of rows encoded into a sub-macro variable

This section describes how the code can be modified to allow for these alterations. The full code, with these modifications, is provided in the Appendix.

1. ALLOWING MODIFICATIONS TO VALUELIST

The most common modification to ValueList is the addition of quotation marks. If using the contents of ValueList to search character data, each value must be surrounded by quotation marks. The full code is changed in three places in order to allow for this modification:

1. A Macro variable called &VarManipulation is added to define the required modification
2. A variable called ValueList2 is generated with the defined modification
3. The final PROC SQL step which encodes the sub-macro variables with the raw data uses ValueList2,

defined above, instead of ValueList

2. ALLOWING ALTERATIONS TO THE MAXIMUM NUMBER OF ROWS WHICH CAN BE ENCODED IN A SUB-MACRO VARIABLE

The programmer may find it helpful to alter the maximum number of rows which can be encoded into the sub-macro variables. Increasing this maximum limit will increase the capacity of the final &AllBins macro variable. This can be achieved by generating a macro variable called &BinSize which contains a number equal to the maximum number of rows the programmer would like added to the sub-macro variables. We would then only have to replace all instances of “1,000” with the &BinSize macro variable.

CONCLUSION

This paper has presented a means by which a programmer can obviate the 65,534 maximum character limit on macro variables imposed by SAS®. It should be noted that there are other methods through which large searches can be achieved. For instance, the programmer could use an INNER JOIN within the SQL procedure to obtain a similar result. The method described in this paper is superior when dealing with code which requires querying the data by use of a DATA step as opposed to PROC SQL. This code can also be used for other purposes. The author has used this program in the past to dynamically generate large amounts of code. For example, if the programmer would like to run 1,000 separate DATA steps, the code used to generate these steps can be placed in the values of ValueList and then run using the &AllBins macro variable. This can be much more succinct and elegant than writing all 1,000 DATA steps (assuming there is some pattern by which the code for these DATA steps can be dynamically generated). The code provided has been generalized as much as possible to broaden its applicability.

ACKNOWLEDGEMENTS

Thank you to Frederick Pape and Paul-Harvey Weiner for providing feedback on early drafts of this paper, and to Ankura Consulting Group, LLC (Ankura) which has provided me with both the resources and training to use SAS®. Ankura is an expert services and business advisory firm headquartered in Washington, DC, focusing on expert witness, bankruptcy and corporate restructuring, litigation support, forensic accounting, geopolitical risk assessment, and general management consulting services.

APPENDIX

The full code is given below:

```
/*===============================================================
   Macroception: Maximizing the Capacity of Your Macro Variables

                        Full Code

 Inputs: 1. Alter VarManipulation Macro Variable
                so that the values in "ValueList" are
                formatted in a manner that is
                searchable for your given situation.
         2. Choose the maximum number of values
                which can be placed in each Bin.
         3. Add the list of values to be searched
                to the ValueList Data Step.

 Output:  Macro Variable &AllBins. which contains
                all formatted values separated by spaces.
                &AllBins. can then be used outside this
                program to call and search the original
                set of values.
 -----------------------------------------------------------------
 Notes:
    After running this code, the Macro Variable
    &AllBins. can be used to perform searches on
    lists that exceed the maximum Macro Variable
    character length of 65534. This also will have
    a character limit (determined by the number of
    bins used), but this limit is orders of
    magnitude higher than the original limit.

    If using values within a custom dataset,
    simply make sure that the custom dataset is
    named "ValueList" and contains a variable called
    "ValueList" with the relevant values and then do
    NOT run INPUT 3.
=================================================================*/


/*======================== INPUT 1. ==============================*/
*Program the appropriate manipulation on the variable ValueList
so that it is searchable;
%Let VarManipulation=catt('"',ValueList,'"');

/*======================== INPUT 2. ==============================*/
*Choose the maximum number of values to be placed in each Bin.
Note, in the very unlikely situation where this code still hits
a character limit (determined not by the character length of the
values but instead by the number of bins used), the BinSize can
be increased so as to reduce the number of Bins overall, thereby
increasing the maximum capacity of this code;
%Let BinSize=1000;

/*======================= INPUT 3. ===============================*/
*Place the list of searches you would like to perform. If using
values within a custom dataset, simply make sure that the custom
dataset is named "ValueList" and contains a variable called
"ValueList" with the relevant values and then do not run INPUT 3;
data ValueList;input ValueList;datalines;
100
200
300
400
;run;

/*======================= PROGRAM ================================*/
data ValueList2;
set ValueList;
ValueList2 = &VarManipulation.;
Count + 1;
call symputx('Count',Count);
run;

%put Number of Values: &Count.;

data _null_;
NumberBins=floor(&Count./&BinSize.);
call symputx('NumberBins',NumberBins);
run;
```

```
%put Bins will range from 0 to &NumberBins.;

data ListOfBins;
do Count = 0 to &NumberBins.;
Base = cats('&L',Count);
output;
end;
run;

proc sql noprint;
select Base into :AllBins separated by ' ' from ListOfBins;
quit;

%macro AddToBins();
%do BinCount = 0 %to &NumberBins.;

data _null_;
L=cats('L',&BinCount.);
call symputx('L',L);
run;

%put Currently Creating &L.;

%global &L.;

proc sql noprint;
select ValueList2 into :&L. separated by ' '
from ValueList2
where &BinCount.*&BinSize.<=Count<=(&BinCount.*&BinSize.)+(&BinSize.-1);
quit;
%end;
%mend AddToBins;

%AddToBins;

*Now the macro variable &AllBins can be used;
```

CONTACT INFORMATION

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

Connor Cosenza
Ankura Consulting Group, LLC
2000 K St. NW 12th Floor
Washington, DC 20006
Phone: (202) 481-1333
Email: connor.cosenza@ankura.com