SESUG Paper 138-2018

Easily Updating Programs with the SAS® Macro Language: an ED*Facts* Example

Aaron Brown, South Carolina Department of Education

ABSTRACT

Many times, a programmer may have to generate the same report several times, such as weekly, monthly, or annually. If these reports have minor changes, making the necessary updates can be time-consuming and cause bugs or errors due to typos. This paper describes examples of how to easily update programs by using the SAS® macro language, using the example of the annual ED*Facts* data submissions (which state Departments of Education must are required to send to the federal Department of Education). This paper utilizes both macro variables (using the *%let* statement) and macros.

INTRODUCTION

The primary purpose of this paper is to display various ways a programmer could use macro variables and macros to effectively make updates to programs with minimal chance of introducing typos or bugs. To this end, we will look at the example of how I, as the ED*Facts* Coordinator for South Carolina, use the macro language to update my programs for our annual ED*Facts* submissions.

If I were manually making the manual changes without macro variables, it would mean finding and hard-coding changes in a dozen or more places. Using macros, I generally can make all updates by updating one or two macro variables. This paper shows how to either 1) consolidate updates to macro variables at the start of the program, or 2) use macros or macro variables to dynamically make the needed updates. The end result is having one section at the start of the program that requires edits, keeping the rest of the program static. See the appendices at the end of this program for an example with and without macro variables, using a simplified version of the ED*Facts* student enrollment file (ED*Facts* File #052).

The topics we will cover include:

1. Using macro variables to minimize the number of places a program needs to be updated.
2. Using macro variables to generate the year in different formats.
3. Using macro variables to generate a count of the number of records
4. Putting code that is re-used multiple times into a macro.
5. Using macro variables to create a CSV file with a dynamically-generated header row

Generally, when processing student data, it is with datasets containing 20,000 to 300,000 records. For this paper, dummy data will be used to protect student privacy. The dummy data we will use is shown below, with variables for our district identifier, school identifier, gender (M=Male, F=Female), and Race-Ethnicity codes (here, W=White, H=Hispanic). For the sake of simplicity, I am using 5 students between two schools within a single district, with a limited range of race/ethnicity values.

For this paper, we will assume our data is saved to a SAS library called EDFacts.

DistrictCode	SchoolCode	Gender	Race
0001	001	M	W
0001	001	M	W
0001	001	F	H
0001	002	M	W
0001	002	F	W

Table 1. Contents of the *MyData* Data Set

The rest of the Introduction gives context related to ED*Facts* and why the changes are required. Readers can skip to Topic 1 if they are eager to get to the SAS code.

US ED provides a data file specification for each type of file. These specifications go over what data should be included and what format it should be in. It also includes whether we submit the data at the state (SEA), district (LEA), and/or school (SCH) level. Thus, for one file, it is possible we will have to submit up to 3 CSV files. For example, file 052 is submitted at all three levels.

ED*Facts* also requires us to use a header row containing metadata instead of column names. Column names are not part of the CSV files that are submitted. The metadata include the name of the CSV file, the school year, and the number of observations (i.e., rows).

In South Carolina, we have decided to store our ED*Facts* data in a systematic way by school year. Thus, the folder structure from year to the next is mostly identical except for a change in school year, such as from 1617 for the 2016-17 year to 1718 for the 2017-18 school year.

TOPIC 1: USING MACRO VARIABLES TO MINIMIZE THE NUMBER OF UPDATES

My preference is to create macro variables at the start of my program, which capture all the hard-coded changes that are needed to update the program for a new year.

These changes include the reporting year, the date the data are generated, the location of input and output files, and the name of the output file(s). Since this file is reported at the state-, district-, and school-level, there are three different output variables. See below:

```
*year and date;
%let yr=1617;
%let today=%sysfunc(today());
*input location;
%let input=EDFacts.mydata;
*output directory and name of the CSV files to be generated;
%let outputdir=O:\EDFacts\&yr\052\output\;
%let SEA=SCSEAMEMBERSHP052%substr(&yr,3,2)AB.CSV;
%let LEA=SCLEAMEMBERSHP052%substr(&yr,3,2)AB.CSV;
%let SCH=SCSCHMEMBERSHP052%substr(&yr,3,2)AB.CSV;
```

Note how the yr (short for year) variable is used within other variables. In this instance, we actually only have to hard-code the year and, if it changes, the input. The others are generated automatically by the macro language. I keep them near the top of the program in case a change is needed. For example, if I had to regenerate the submitted files, I could change my initials at the end (“AB”) to something like “v2” for version 2.

If we execute this code and use the %PUT _USER_ statement, we can see their values in the SAS log. This can be helpful to check that your macro variables were coded correctly and for debugging purposes.

```
%put _USER_;
```

```
33          %put _USER_;
GLOBAL INPUT EDFacts.mydata
GLOBAL LEA SCLEAMEMBERSHP05217AB.CSV
GLOBAL OUTPUTDIR O:\EDFacts\1617\052\output
GLOBAL SCH SCSCHMEMBERSHP05217AB.CSV
GLOBAL SEA SCSEAMEMBERSHP05217AB.CSV
GLOBAL TODAY 21284
GLOBAL YR 1617
```

Output 1. SAS Log Excerpt, Showing Macro Variables

TOPIC 2: USING MACRO VARIABLES TO REFORMAT THE YEAR

The year has been stored in XXYY format for years 20XX-20YY. However, for the header row, the data is required as 20XX-20YY. Instead of hard-coding both ways, a second macro variable can be created from the first one.

I also sometimes need to check the prior year’s file, for comparison and quality control purposes. Since we store our files systematically, I can generally just create a new macro variable with the old year value. Remember that although SAS stores macro variables as text, you can perform math on them by using the *%sysevalf* function.

```
%let myyear=20%substr(&yr,1,2)-20%substr(&yr,3,2);
%let lastyear=%sysevalf(&yr-101); *e.g., 1617 - 101 = 1516;
%PUT my year is &myyear AND last year is &lastyear;
```

As before, you can check that the variables were rendered correctly by using the %PUT statement. The above statement rendered this line in the SAS log:

```
my year is 2016-2017 AND last year is 1516
```

Output 2. SAS Log Excerpt, Showing Dynamically Generated Year Variables

TOPIC 3: USING MACRO VARIABLES TO GET A COUNT OF OBSERVATIONS

Part of the metadata required in the header row is the number of records in the data file. If I were not using macros, I could generate the file, check the number of records, and then hard-code the metadata. On the other hand, using macros enables me to have SAS dynamically generate this count and put it into a macro variable that we can use later.

For context, let us say that we made some aggregated counts, by different subgroups, to generate the ED*Facts* file. Here is code that could be used to make the school-level report. (For the state- or district-level reports, DistrictCode and SchoolCode are removed as needed.)

```
data rawdata;
       set &input;
       counter=1; *so that PROC MEANS has something to count;
run;
proc means data=rawdata noprint;
       class DistrictCode SchoolCode Gender Race;
       types  DistrictCode*SchoolCode
              DistrictCode*SchoolCode*Gender
              DistrictCode*SchoolCode*Race
              DistrictCode*SchoolCode*Gender*Race
              ;
       output out=mystats n(counter)=StudentCount;
run;
```

Now we have a data file with counts, including a total count for each school, counts by individual demographics, and counts by Gender and Race. While the TYPES statement is not required, it does yield more organized (e.g., pre-filtered) data when you only need certain combinations.[1]

But we need the number of observations. Instead of looking at then hard-coding the value (which, with our raw data, is 13), I prefer to use a DATA step with the CALL SUMPUTX function, as shown below.

```
data _null_;
       set mystats end=lastobs;
       if lastobs then CALL SYMPUTX("obsnum",_n_);
run;
```

The END statement sets a temporary variable called lastobs to 1 if it is the last observation in the dataset. The IF statement uses CALL SYMPUTX to put the counter (_n_) into a new macro variable called *obsnum* when we are at the last observation in the dataset. Now we have our metadata.[2]

This metadata can also be useful for general purposes. Since bugs sometimes cause records to be lost when outputting data from or reading data into a software package, putting the expected number of observations in a file's metadata or name improves the chances of detecting errors early. For example, the SC Department of Education requests that the number of student records be in the file name of our major testing files, so that we can tell immediately if some records are missing.

TOPIC 4: PUTTING REUSED CODE IN A MACRO VARIABLE

Since we have to generate a CSV file for the state-, district-, and school-level, we need the code to generate these files to appear in the program three times. Thus, if a change occurs or a bug is found, we need to fix it three times. Such repetition generates more room for mistakes like typos or forgetting to fix it in one area.

---

[1] We are assuming that no CLASS variables had missing values. If they did, you may want to use the MISSING statement within the CLASS statement. You may also need to clean the output data, if blanks were not wanted in some groupings.

[2] The SAS Support page, in Sample 24671, has code for a macro that accomplishes this task. See the References in this paper. I prefer the DATA step method, but both work and are valid.

I find put and input statements to be rather bothersome to update and debug, so I prefer to put them in a macro variable. Depending on your preferences, you might want to put more or less code in a macro variable. Here is what I use for our example:[3]

```
%macro putStatement;
       (ctr)(:8.)
       (StateCode)(:$2.)
       (StateNum)(:$2.)
       (DistrictCode)(:$4.) /*blank for SEA*/
       (SchoolCode)(:$3.) /*blank for SEA and LEA*/
       (TableName)(:$6.) /*MEMBER*/
       (Grade)(:$2.)
       (Race)(:$3.)
       (Gender)(:$1.)
       (TotalInd)(:$1.)
       (Details)(:$100.)
       (StudentCount)(:10.)
%mend;
```

Note that this code works just as well for an INPUT as for an output (that is, PUT) statement. Thus, I can use it to read in last year's files for comparison purposes as well as outputting this year's submission files.

TOPIC 5: USING MACRO VARIABLES TO OUTPUT A FILE, INCLUDING THE HEADER ROW WITH METADATA

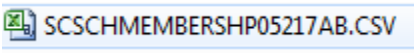
We have macro variables containing various metadata, such as the year, the file submission date, our file name, and the number of observations. EDFacts requires that all these be in the header row. We can dynamically generate the header row via a %let statement. We can then insert it into a CSV by using the DATA step.

We also have our put statement, used to generate the data, in a macro variable. By using these with the metadata, we can generate the CSV quickly. See the full code below. (LRECL is longer than is necessary. I tend to use an extra-long LRECL value as a safety precaution.)

```
%let firstline="SCHOOL MEMBERSHIP TABLE,&obsnum,&SCH,SCDE AB
%sysfunc(putn(&today,mmddyy6.)),&myyear";
data _null_;
       file "&outputdir\&SCH" dsd lrecl=500;
       set mystats;

       if _n_=1 Then Put &firstline;
       Put %putstatement;
run;
```

If I look at the output directory, I see that the CSV file has been generated with the correct name.



If I open it in Notepad, I can see that my data appear as desired.

```
SCHOOL MEMBERSHIP TABLE,13,SCSCHMEMBERSHP05217AB.CSV,SCDE AB 041018,2016-2017
,,,0001,001,,,,,,,3
,,,0001,002,,,,,,,2
,,,0001,001,,,H,,,,1
,,,0001,001,,,W,,,,2
,,,0001,002,,,W,,,,2
,,,0001,001,,,,F,,,1
,,,0001,001,,,,M,,,2
,,,0001,002,,,,F,,,1
,,,0001,002,,,,M,,,1
,,,0001,001,,,H,F,,,1
,,,0001,001,,,W,M,,,2
,,,0001,002,,,W,F,,,1
```

[3] This is the actual put/input statement I use for file 052. I have simplified the data for our example, so this statement contains more variables than actually appear in this paper. Note that SAS renders these as blanks if you run the code in the appendices.

```
,,,0001,002,,,W,M,,,1
```

Output 3. Contents of the Generated School-Level CSV

As a quality control step, I highly recommend always opening files you generate so that you can check that they look as intended.

EXTRA TOPIC: SOME HINTS ON DEBUGGING MACROS

The SAS macro language is powerful and helpful, but it can be difficult to remember what each variable refers to. SAS has two powerful options for this purpose: SYMBOLGEN and MPRINT. MLOGIC can also be helpful, but I find it less informative than MPRINT for most applications.

The SYMBOLGEN option is helpful to see how variables are reconciled upon compilation, although if you are using a lot of macro variables, it can clutter the SAS log. In short, it works somewhat like hard-coding the %PUT statements. The MPRINT option is helpful if you are using %macro statements, as it prints the generated macro code. MLOGIC provides some information whenever a macro is run and when it finishes.

You can see an example of the three options, and how much clutter all three can generate when run together, in the below excerpt from the SAS log.

```
100        %let firstline="SCHOOL MEMBERSHIP TABLE,&obsnum,&SCH,SCDE AB
%sysfunc(putn(&today,mmddyy6.)),&myyear";
SYMBOLGEN:  Macro variable OBSNUM resolves to 13
SYMBOLGEN:  Macro variable SCH resolves to SCSCHMEMBERSHP05217AB.CSV
SYMBOLGEN:  Macro variable TODAY resolves to 21284
SYMBOLGEN:  Macro variable MYYEAR resolves to 2016-2017
101        data _null_;
102
SYMBOLGEN:  Macro variable OUTPUTDIR resolves to O:\EDFacts\1617\052\output\
SYMBOLGEN:  Macro variable SCH resolves to SCSCHMEMBERSHP05217AB.CSV
102      !  file "&outputdir\&SCH" dsd lrecl=500;
103          set mystats;
104
105          if _n_=1 Then
SYMBOLGEN:  Macro variable FIRSTLINE resolves to "SCHOOL MEMBERSHIP
TABLE,13,SCSCHMEMBERSHP05217AB.CSV,SCDE AB 041018,2016-2017"
105      !               Put &firstline;
106
MLOGIC(PUTSTATEMENT):  Beginning execution.
106      !  Put %putstatement;
MPRINT(PUTSTATEMENT):   (ctr)(:8.) (StateCode)(:$2.) (StateNum)(:$2.)
(DistrictCode)(:$4.) (SchoolCode)(:$3.) (TableName)(:$6.)
(Grade)(:$2.) (Race)(:$3.) (Gender)(:$1.) (TotalInd)(:$1.) (Details)(:$100.)
(StudentCount)(:10.)
MLOGIC(PUTSTATEMENT):  Ending execution.
107        run;
```

Output 4. SAS Log Excerpt, showing macro options

Be aware that these options use some additional system resources and thus could slow your programs. For most programs, the change is probably negligible. However, you may want to turn off these options after debugging if you notice your program is running slowly, especially if your program is looping through a macro hundreds or thousands of times.

Also, great advice for writing macro code is to write it first without using macros or macro variables. Then debug it, then add in the macros, and then debug your updated version.

Lastly, be aware that macros can make your program hard for others to read. If you need your code to be easily understood by other programmers in your office, or if your code may be inherited by future programmers, try to balance readability against utility. Add good comments to your code.

QUALITY CONTROL

This paper has assumed that the input data's format remains consistent from year-to-year. When you re-run a program with new input, please check any assumptions you had about the input. Also, it is wise to check your program to make sure it is still working as you intended. Although updating consists of merely updating a couple macro variables, as a programmer we should be responsible and confirm that our code runs as intended.

CONCLUSION

This paper has gone through five examples of how the SAS macro language can be used to streamline updating one's program from year to year. I hope that these examples are helpful for those learning or already using the macro language, so that they may save time and decrease the chance of typos and bugs when updating their programs.

APPENDIX A: SAMPLE CODE, WITH THE MACRO LANGUAGE

Here is the code with the macro language. For the sake of conciseness, only the school-level file is generated below. If there is no change in the input data's name, the only line that actually needs to be updated is *%let yr*=.

```
*year and date;
%let yr=1617;
%let today=%sysfunc(today());
%let myyear=20%substr(&yr,1,2)-20%substr(&yr,3,2);
%let lastyear=%sysevalf(&yr-101); *e.g., 1617 - 101 = 1516;
*input location;
%let input=EDFacts.mydata;
*output directory and name of the CSV files to be generated;
%let outputdir=O:\EDFacts\&yr\052\output;
%let SEA=SCSEAMEMBERSHP052%substr(&yr,3,2)AB.CSV;
%let LEA=SCLEAMEMBERSHP052%substr(&yr,3,2)AB.CSV;
%let SCH=SCSCHMEMBERSHP052%substr(&yr,3,2)AB.CSV;

%macro putStatement;
       (ctr)(:8.)
       (StateCode)(:$2.)
       (StateNum)(:$2.)
       (DistrictCode)(:$4.) /*blank for SEA*/
       (SchoolCode)(:$3.) /*blank for SEA and LEA*/
       (TableName)(:$6.) /*MEMBER*/
       (Grade)(:$2.)
       (Race)(:$3.)
       (Gender)(:$1.)
       (TotalInd)(:$1.)
       (Details)(:$100.)
       (StudentCount)(:10.)
%mend;

data rawdata;
       set &input;
       counter=1; *so that PROC MEANS has something to count;
run;
proc means data=rawdata noprint;
       class DistrictCode SchoolCode Gender Race;
       types DistrictCode*SchoolCode
                DistrictCode*SchoolCode*Gender
                DistrictCode*SchoolCode*Race
                DistrictCode*SchoolCode*Gender*Race
                ;
       output out=mystats n(counter)=StudentCount;
run;
data _null_;

       set mystats end=lastobs;
       if lastobs then CALL SYMPUTX("obsnum",_n_);
run;
```

```
%let firstline="SCHOOL MEMBERSHIP TABLE,&obsnum,&SCH,SCDE AB
%sysfunc(putn(&today,mmddyy6.)),&myyear";
data _null_;
      file "&outputdir\&SCH" dsd lrecl=500;
      set mystats;

      if _n_=1 Then Put &firstline;
      Put %putstatement;
run;
```

APPENDIX B: SAMPLE CODE, WITHOUT THE MACRO LANGUAGE

Here is the code without the macro language. Note how many spots contain hard-coded or repetitive values. Also, although this section of code looks shorter, keep in mind that the PUT statement would be repeated thrice if the appendix showed the code for all three files.

```
data rawdata;
      set EDFacts.mydata;
      counter=1; *so that PROC MEANS has something to count;
run;
proc means data=rawdata noprint;
      class DistrictCode SchoolCode Gender Race;
      types DistrictCode*SchoolCode
               DistrictCode*SchoolCode*Gender
               DistrictCode*SchoolCode*Race
               DistrictCode*SchoolCode*Gender*Race
               ;
      output out=mystats n(counter)=StudentCount;
run;

*you have to look at mystats to get the number of observations & hard-code it;

data _null_;
      file "O:\EDFacts\1617\052\output\SCSCHMEMBERSHP05217AB.CSV" dsd lrecl=500;
      set mystats;

      if _n_=1 Then Put "SCHOOL MEMBERSHIP TABLE,13,SCSCHMEMBERSHP05217AB.CSV,SCDE AB
041018,2016-2017";
      Put
      (ctr)(:8.)
      (StateCode)(:$2.)
      (StateNum)(:$2.)
      (DistrictCode)(:$4.) /*blank for SEA*/
      (SchoolCode)(:$3.) /*blank for SEA and LEA*/
      (TableName)(:$6.) /*MEMBER*/
      (Grade)(:$2.)
      (Race)(:$3.)
      (Gender)(:$1.)
      (TotalInd)(:$1.)
      (Details)(:$100.)
      (StudentCount)(:10.);
run;
```

REFERENCES

SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Advanced Programming for SAS®9, Third Edition*. Cary, NC: SAS Institute Inc.

SAS. "Sample 24671: Dynamically determine the number of observations and variables in a SAS® data set." http://support.sas.com/kb/24/671.html. Accessed 8 August 2018.

US Department of Education. "The ED*Facts* Initiative." https://www2.ed.gov/about/inits/ed/edfacts/index.html. Accessed 8 August 2018.

ACKNOWLEDGMENTS

The author thanks the SESUG chair and staff for this conference and the opportunity to present.

CONTACT INFORMATION

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

Aaron R. Brown
South Carolina Department of Education
1429 Senate Street, Columbia, SC 29201
Work Phone: 803-734-8858
E-mail: ARBrown@ed.sc.gov