

Why call your macros when SAS can do it for you?

Kenneth Bissett - United Services Automobile Association

Abstract

This paper will present a method for automating repetitive coding using the SAS Macro language. Specifically, the `%macro-%mend` code designed in part to reduce repetitive coding. It's one thing to write macros to automate and shorten your programs, but this method may require you to write a unique call to your macro for each execution. The method presented in this paper goes a step further by making SAS write the macro calls for you.

Introduction

The macro language itself can be intimidating for a new user. It is important that the user have a good understanding of basic `%macro-%mend` blocks before attempting to automate any further. Understanding how the macros work and how macro variables resolve is critical.

Methodology

The example used in this presentation will be a simple PROC FREQ against a list of variables in a SAS dataset. In this case, we want to output each frequency table to a SAS file for further processing. Unfortunately, PROC FREQ will only output the last table in a list using the `OUT=` option. This causes us to have to write a separate table statement for each variable you want to process.

We need to create a macro to process each variable, one at a time to create each SAS dataset. Keep in mind that this particular program will generate a frequency table for every variable in the input

dataset. Some variables such as phone numbers, names and social security numbers are not necessarily good candidates for PROC FREQ.

The use of CALL SYMPUT will be the method for invoking your macros in an automated fashion. This involves writing at least two additional, short macros to support the automation. In most cases, the code is generated from an input data set, which provides the data necessary for iterative unique macro invocations. With a few extra steps, most repetitive macro calls can be automated using this method.

As stated earlier, this example will attempt to automate an otherwise basic PROC FREQ, with the addition of the `OUT=` option to store the tables in SAS datasets. Here is what the code might look like written manually.

```
Proc freq data=work.test;  
Table v1 / out=v1;  
Table v2 / out=v2;  
Table v3 / out=v3;  
Run;
```

Imagine the amount of code you would need to write if there were fifty variables in your dataset!

Below is the SAS program we will be automating. It uses a simple PROC FREQ inside of a `%macro-%mend` block. The macro has one parameter called `var`. The three lines at the bottom are called macro calls, which execute the macro `%freak`. For simplicity, the

program is using a CARDS statement to create the dataset *work.test*.

```
options mprint nocenter;

data test;
  input @1 (v1-v3) (1.);
cards;
111
101
100
001
111
;
run;

%macro freak(var);
  proc freq noprint data=test;
    tables &var / out = &var;
  run;
%mend freak;
%freak(v1);
%freak(v2);
%freak(v3);
```

Let's Automate

Now, we can break down the program to understand the automation process. Let's start with the creation of a small dataset with five records and three variables.

```
data work.test;
  input @1 (v1-v3) (1.);
cards;
111
101
100
001
111
;
run;
```

Below we're adding a global macro variable called *dsn* to hold the name of the input dataset. This makes the code a little more flexible allowing it to be run against any SAS dataset.

```
%let dsn=work.test
```

Below is the actual macro code, which executes the PROC FREQ. Note that it refers to *&dsn* for the input dataset. The tables statement references *&var*, which will be storing each variable name, one at a time from the input dataset. The *out=* statement also references *&var* so that the output SAS table will have the same name as the input variable.

```
%macro freak(var);
  proc freq noprint data=&dsn;
    tables &var / out = &var;
  run;
%mend freak;
```

In order for the code to be automated, we'll need to create a list of the unique variables that are in the input dataset. I am using a PROC CONTENTS with an output statement to create a small SAS table, which contains one record for each variable and one column called name.

```
proc contents data=&dsn noprint
out=varlist(keep=name);
run;
```

The table looks like this:

Table 1

OBS	Name
1	V1
2	V2
3	V3

Now we can read the *varlist* table (our parameter file) to create the individual macro calls with the appropriate variable names. This data step uses the *call symput* statement to create a new macro variable for each input record from

our parameter file. Each record from **Table 1** is read into the data step and the *call symput* statement uses the record to dynamically create a new macro variable. The macro variable names are made unique by using the observation number in the macro name. We also need to obtain the total number of records in the parameter table. For this we use *call symput* to create a global macro variable called *num*.

```
data _null_;
    set varlist nobs=cnt end=last;
    call
    symput('invoke'||left(_n_),'%freak('||name||')');
    if last then call symput('num',cnt);
run;
```

Let's break down the *call symput* syntax:

Call *symput*(macro name, macro value);

The string on the left side of the comma will become the macro variable name:

'invoke'||left(_n_) resolves to
invoke1

The string on the right side of the comma will become the macro variable value:

'%freak('||name||')' resolves to
%freak(var1)

Now we need to create a macro called *%lazy*, which will create each individual call to the *%freak* macro created above. The macro is a simple do loop which executes once for each record in our parameter file. The touch of magic in this macro is the use of double ampersands. On the first pass, the macro variable called *&&invoke&i*

resolves to *&invoke1*, which is a newly created macro variable. This macro variable then resolves to:

```
%freak(v1);
```

This is now a valid SAS macro, which simply executes as if you had typed it in by hand!

```
%macro lazy;
    %do i = 1 %to &num;
        &&invoke&i;
    %end;
%mend lazy;
```

When we put all of the code together, the very last step is to execute the macro *%lazy*. This of course starts the process of executing each individual macro call. Below is the entire program completely automated.

```
options mprint nocenter;
```

```
data test;
    input @1 (v1-v3) (1.);
cards;
111
101
100
001
111
;
run;
```

```
%let dsn=work.test;
```

```
%macro freak(var);
    proc freq noprint data=&dsn;
        tables &var / out = &var;
    run;
%mend freak;
```

```
%macro lazy;
    %do i = 1 %to &num;
        &&invoke&i;
    %end;
%mend lazy;
```

(continued)

```
proc contents data=&dsn noprint
out=varlist(keep=name);
run;
data _null_;
    set varlist nobobs=cnt end=last;
    call
symput('invoke' || left(_n_), '%freak('
|| name || ')');
    if last then call
symput('num', cnt);
run;

%lazy;
```

Additional Tips

Listed below is another approach to creating your parameter file. Depending on the complexity of your particular macro, you may not be able to "read" your parameters from any single source of data. Although this approach is manual, it is usually only done once. This also allows you to maintain highly detailed parameters if needed.

```
Data mylib.params;
    Input parm1 parm2 $;
Cards;
1001 Jones
1002 Wilson
1003 Smith
1004 Robinson
1005 Williams
;
run;
```

A frequency distribution from another SAS dataset is also a great candidate for a parameter file. The output table contains the freq variable and the count variable. You can also consider storing a set of parameters in an excel spreadsheet. This can then be imported and read by SAS as input to your macro.

Many of the SAS procedures can generate output to feed another automation process. With a little creative thinking, you can use the basic concept in this paper to automate some of your repetitive code.

Contact Information:

Kenny Bissett
USAA
San Antonio, TX 78288
Kenneth.bissett@usaa.com