Paper CC-23

How variable-dependent macros can help you

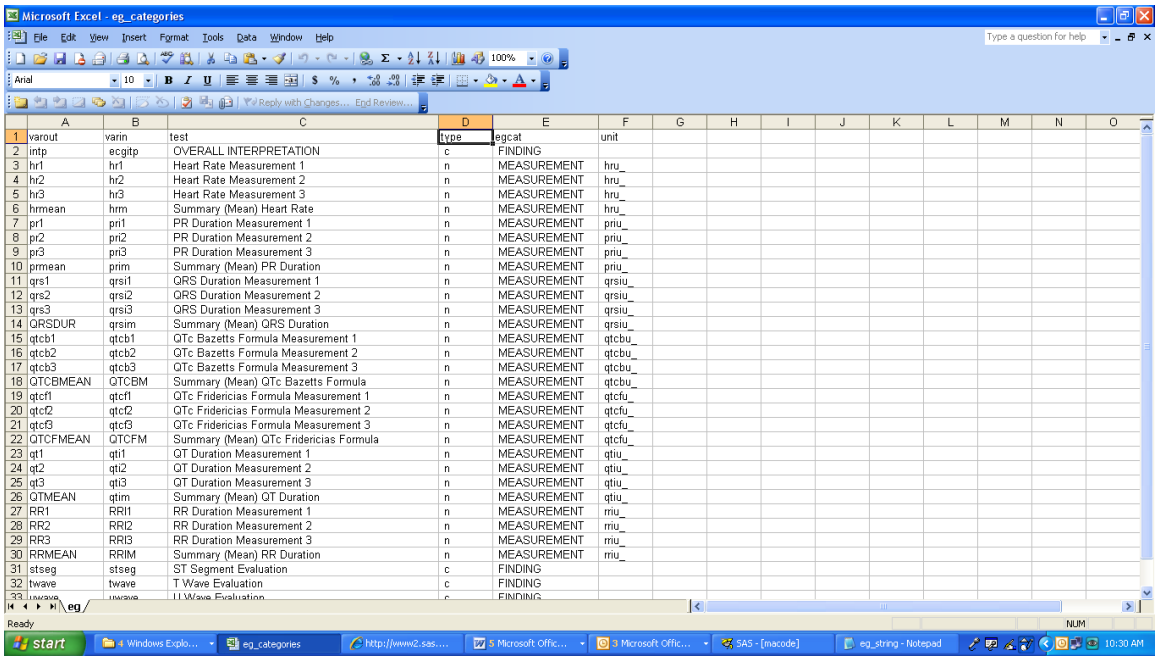
Mindy Wang

ABSTRACT

In clinical trials and many other research fields, repetitive statements happen very often. This paper illustrates how to deal with the same task with three different approaches. The first approach uses simple copy and paste any SAS® beginner can do. The second approach uses a simple macro program to make the task more efficient, and reduce the length of the program. To further improve the program and to make the program more adaptable to other projects, the third approach uses variable-dependent macros which minimize tedious typing, eliminate possible human errors, and ensure the accuracy of data.

INTRODUCTION

This program is based on a study to convert EG data to SDTM formats in a clinical trial. However, the SAS codes can be rewritten easily to handle other fields of study as well. The parameter variables came in as a csv file. What we need to accomplish is to do the repetitive tasks which apply to all the variables specified in the parameter variables data set that are in csv formats. Following is the screen print of the parameter data.



Microsoft Excel - eg_categories

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	varout	varin	test	type	egcat	unit									
2	intp	ecgitp	OVERALL INTERPRETATION	c	FINDING										
3	hr1	hr1	Heart Rate Measurement 1	n	MEASUREMENT	hru_									
4	hr2	hr2	Heart Rate Measurement 2	n	MEASUREMENT	hru_									
5	hr3	hr3	Heart Rate Measurement 3	n	MEASUREMENT	hru_									
6	hrmean	hrm	Summary (Mean) Heart Rate	n	MEASUREMENT	hru_									
7	pr1	pri1	PR Duration Measurement 1	n	MEASUREMENT	priu_									
8	pr2	pri2	PR Duration Measurement 2	n	MEASUREMENT	priu_									
9	pr3	pri3	PR Duration Measurement 3	n	MEASUREMENT	priu_									
10	prmean	prim	Summary (Mean) PR Duration	n	MEASUREMENT	priu_									
11	qrs1	qrsi1	QRS Duration Measurement 1	n	MEASUREMENT	qrsiu_									
12	qrs2	qrsi2	QRS Duration Measurement 2	n	MEASUREMENT	qrsiu_									
13	qrs3	qrsi3	QRS Duration Measurement 3	n	MEASUREMENT	qrsiu_									
14	QRSDUR	qrsim	Summary (Mean) QRS Duration	n	MEASUREMENT	qrsiu_									
15	qtcb1	qtcb1	QTc Bazetts Formula Measurement 1	n	MEASUREMENT	qtcbu_									
16	qtcb2	qtcb2	QTc Bazetts Formula Measurement 2	n	MEASUREMENT	qtcbu_									
17	qtcb3	qtcb3	QTc Bazetts Formula Measurement 3	n	MEASUREMENT	qtcbu_									
18	QTCBMEAN	QTCBM	Summary (Mean) QTc Bazetts Formula	n	MEASUREMENT	qtcbu_									
19	qtcf1	qtcf1	QTc Fridericias Formula Measurement 1	n	MEASUREMENT	qtcfu_									
20	qtcf2	qtcf2	QTc Fridericias Formula Measurement 2	n	MEASUREMENT	qtcfu_									
21	qtcf3	qtcf3	QTc Fridericias Formula Measurement 3	n	MEASUREMENT	qtcfu_									
22	QTCFMEAN	QTCFM	Summary (Mean) QTc Fridericias Formula	n	MEASUREMENT	qtcfu_									
23	qt1	qti1	QT Duration Measurement 1	n	MEASUREMENT	qtiu_									
24	qt2	qti2	QT Duration Measurement 2	n	MEASUREMENT	qtiu_									
25	qt3	qti3	QT Duration Measurement 3	n	MEASUREMENT	qtiu_									
26	QTMEAN	qtim	Summary (Mean) QT Duration	n	MEASUREMENT	qtiu_									
27	RR1	RRI1	RR Duration Measurement 1	n	MEASUREMENT	rriu_									
28	RR2	RRI2	RR Duration Measurement 2	n	MEASUREMENT	rriu_									
29	RR3	RRI3	RR Duration Measurement 3	n	MEASUREMENT	rriu_									
30	RRMEAN	RRIM	Summary (Mean) RR Duration	n	MEASUREMENT	rriu_									
31	stseg	stseg	ST Segment Evaluation	c	FINDING										
32	twave	twave	T Wave Evaluation	c	FINDING										
33	uwave	uwave	U Wave Evaluation	c	FINDING										

COPY AND PASTE METHOD

Copy and paste method is the easiest way we can do to accomplish repetitive tasks. No extensive knowledge of SAS is needed. Following copy and paste method codes illustrate this easy method. However, the downside of this method is that it takes busy work. If we need to repeat it three times, it won't be too big a deal. However, if we need to repeat it 50 times or 100 times, the task will be very tedious. In addition, as we use more manual manipulation, it is more likely we might make mistakes along the way.

However the copy and paste method is still very important. Even though we can improve the program by macro, it is always a good idea to make the program work in plan text and then try to improve it by incorporating macros. For example we start out with plain text, copy it a few times, and figure out which variables need to be changed, and then incorporate Macro Definition to make it more efficient.

Following is the illustration of copy and paste method.

```
Data eg2;
     set sdtm.ert;
     length egtest egtestcd egcat egorres egstresc $ 200 egstresu egorresu $ 8;
egtestcd = " " ;
egorres = " " ;
egstresc = " " ;
egtest = " ";
egtestcd = "INTP" ;
egtest = upcase("OVERALL INTERPRETATION");
egorres = left(trim(upcase ( ecgitp_ ))) ;
egstresc = upcase ( egorres ) ;
egcat="FINDING";
if ( egorres ne " " ) then output ;
drop ecgitp: ;

egtestcd = " " ;
egorres = " " ;
egstresc = " " ;
egtest = " ";
egtestcd = "HR1" ;
egtest = upcase("Heart Rate Measurement 1");
if ( hr1 ne . ) then do ;
      egorres = trim ( left ( put ( hr1 , best. ))) ;
      egstresn = hr1 ;
      egorresu=hru_;
      egstresu=hru_ ;
end ;
egstresc = upcase ( egorres ) ;
egcat="MEASUREMENT";
if ( egorres ne " " ) then output ;
drop hr1: hru_;

egtestcd = " " ;
egorres = " " ;
egstresc = " " ;
egtest = " ";
egtestcd = "HR2" ;
egtest = upcase("Heart Rate Measurement 2");
if ( hr2 ne . ) then do ;
      egorres = trim ( left ( put ( hr2 , best. ))) ;
      egstresn = hr2 ;
      egorresu=hru_;
      egstresu=hru_ ;
end ;
egstresc = upcase ( egorres ) ;
egcat="MEASUREMENT";
if ( egorres ne " " ) then output ;
drop hr2: hru_;

egtestcd = " " ;
egorres = " " ;
egstresc = " " ;
egtest = " ";
egtestcd = "HR3" ;
```

```
egtest = upcase("Heart Rate Measurement 3");
if ( hr3 ne . ) then do ;
      egorres = trim ( left ( put ( hr3 , best. ))) ;
      egstresn = hr3 ;
      egorresu=hru_;
      egstresu=hru_ ;
end ;
egstresc = upcase ( egorres ) ;
egcat="MEASUREMENT";
if ( egorres ne " " ) then output ;
drop hr3: hru_;

egtestcd = " " ;
egorres = " " ;
egstresc = " " ;
egtest = " ";
egtestcd = "HRMEAN" ;
egtest = upcase("Summary (Mean) Heart Rate");
if ( hrm ne . ) then do ;
      egorres = trim ( left ( put ( hrm , best. ))) ;
      egstresn = hrm ;
      egorresu=hru_;
      egstresu=hru_ ;
end ;

egstresc = upcase ( egorres ) ;
egcat="MEASUREMENT";
if ( egorres ne " " ) then output ;
drop hrm: hru_;

.....
run;
```

SIMPLE MACRO METHOD

The following method takes a step further and substitutes the variables need to be changed in keyword parameters. We only need to define the macro program once. By invoking the macro with different parameters, we can achieve the same goal with a much shorter and efficient program. However, when the there are many sets of keyword parameters, even writing up the commands to invoke the macro takes substantial time. There is still room for improvement. In the section following this one, the program is further improved by incorporating variable dependent macro.

```
%macro setx (varout=, varin=, test=, type=, egcat=, unit=);
  %if ( &varin eq ) %then
    %do ;
      %let varin = &varout;
    %end ;
  egtestcd = " " ;
  egorres = " " ;
  egstresc = " " ;
  egtest = " ";
  egtestcd = %upcase ( "&varout" ) ;
  egtest = upcase("&test");
  %if ( &type eq c ) %then
    %do ;
      egorres = left(trim(upcase ( &varin._ ))) ;
    %end ;
```

```
  %else %if ( &type eq n ) %then
    %do ;
      if ( &varin ne . ) then
        do ;
          egorres = trim ( left ( put ( &varin , best. ))) ;
          egstresn = &varin ;
          egorresu=&unit ;
          egstresu=&unit ;
        end ;
    %end ;
  egstresc = upcase ( egorres ) ;
  egcat="&egcat";
  if ( egorres ne " " ) then
    output ;
  drop &varin: &unit;
 %mend setx ;

data eg2;
     set sdtm.ert;
     length egtest egtestcd egcat egorres egstresc $ 200 egstresu egorresu $ 8;
      %setx (varout=intp, varin=ecgitp, test=OVERALL INTERPRETATION, type=c,
egcat=FINDING);
      %setx (varout=hr1, varin=hr1, test=Heart Rate Measurement 1, type=n,
egcat=MEASUREMENT, unit=hru_);
.........
Run;
```

VARIABLE DEPENDENT MACRO METHOD

With variable dependent macro, the utmost advantage is that there is no need to type in the parameters for macro manually, since it is already in a csv file. Without the manual manipulation, chances for typing errors are largely reduced. The parameters only need to be imported. In the data step call symputx is used to save the parameters into macro variables with corresponding record number. In this case, since we don't have any do loop in the data step. The number of record would be consistent with the number of iteration. We can keep _n_ as our record ID. Then for each record ID, we use call symputx to save each parameter to macro variable with corresponding record ID.

For example, we save the value of the first parameter named "varout" in the first record, and call the macro variable "varout1". We do the same for the other variables. Therefore for record ID 1, we have saved macro variables "varin1", "test1", "type1", "egcat1", "unit1" as well. After we saved the macro variables for the first record id, the data step saved the macro variables for record ID 2, as "varout2", "varin2", "test2", "type2", "egcat2", and "unit2". Then we saved the macro variables for other record ID until we reach the end of the parameter data set. We can then use %put n, %put varout1, etc. to see if the program is doing what we expected and print out the data set one to verify.

The variable dependent program can be adapted to difference projects very easily. As long as the parameter is in a csv or excel file, we can just import the parameter data in and very little modification is needed to make it work. Also, if we need to make any change, we only need to make change in the csv file once. We don't need to go through all 33 different places in the program to make the change. Unlike in the first two methods which a lot of manual manipulation is needed, this method is very easy for new parameters.

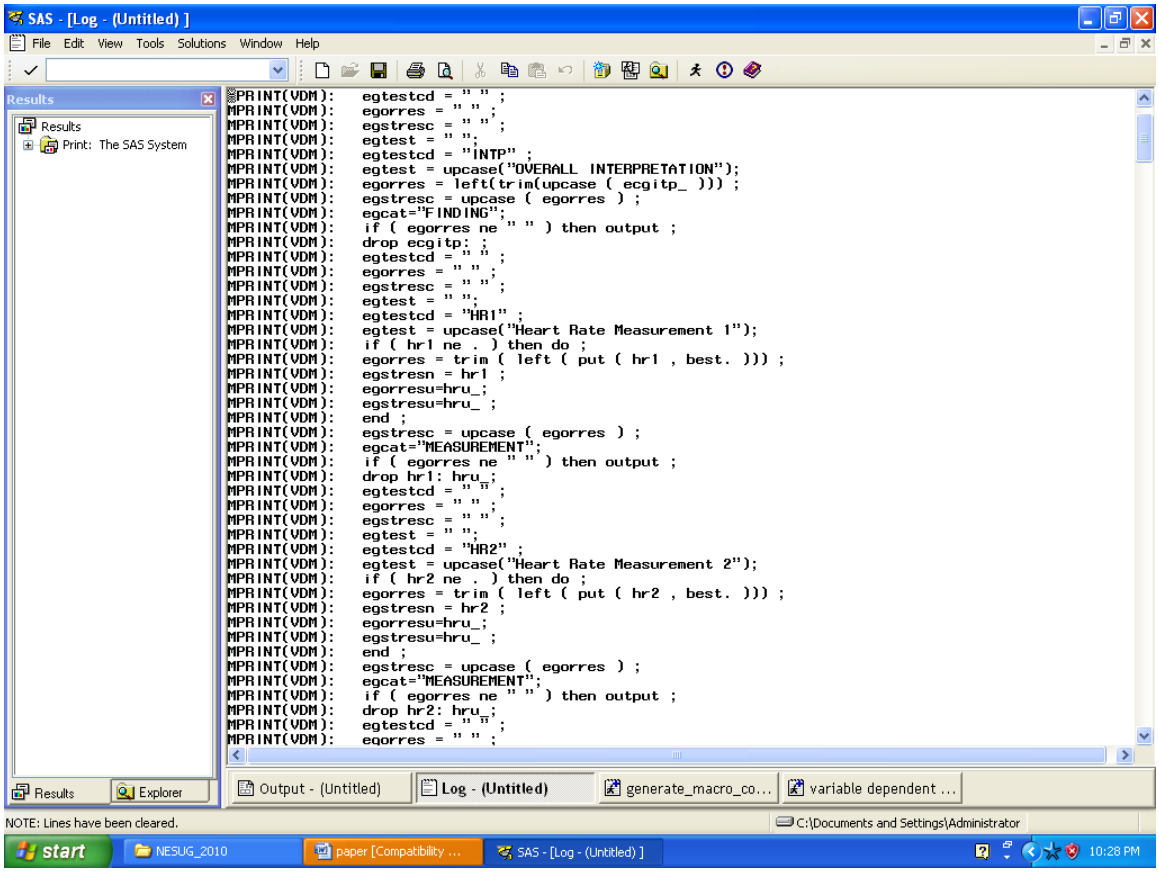
```
proc import datafile = "C:\Documents and
Settings\mindy.wang\Desktop\SAS\NESUG_2010\eg_categories.csv"
 out = work.zero dbms = dlm replace;
 guessingrows=200;
delimiter = ',';
run;
```

```
data one;
  set zero end=eof;
  n=put(_n_,2.);
  call symputx('varout'||left(n), varout);
  call symputx('varin'||left(n), varin);
  call symputx('test'||left(n), test);
  call symputx('type'||left(n), type);
  call symputx('egcat'||left(n), egcat);
  call symputx('unit'||left(n), unit);
  if eof then call symputx('n',n);
run;

/*CHECK THE NUMBER OF CASES*/
%put n=&n;
proc print data=one;
run;

%macro vdm (varout=, varin=, test=, type=, egcat=, unit=);
%do i=1 %to &n;
  %if ( &&varin&i eq ) %then
    %do ;
      %let varin = &&varout&i;
    %end ;
  egtestcd = " " ;
  egorres = " " ;
  egstresc = " " ;
  egtest = " ";
  egtestcd = %upcase ( "&&varout&i" ) ;
  egtest = upcase("&&test&i");
  %if ( &&type&i eq c ) %then
    %do ;
      egorres = left(trim(upcase ( &&varin&i.._ ))) ;
    %end ;
  %else %if ( &&type&i eq n ) %then
    %do ;
      if ( &&varin&i ne . ) then
        do ;
          egorres = trim ( left ( put ( &&varin&i , best. ))) ;
          egstresn = &&varin&i ;
          egorresu=&&unit&i;
          egstresu=&&unit&i ;
        end ;
    %end ;
  egstresc = upcase ( egorres ) ;
  egcat="&&egcat&i";
  if ( egorres ne " " ) then
    output ;
  drop &&varin&i: &&unit&i;
 %end;
 %mend vdm;

data eg1;
  set sdtm.ert;
  length egtest egtestcd egcat egorres egstresc $ 200 egstresu egorresu $ 8;
    %vdm
run;
```

OTHER WAYS TO REDUCE TYPING

Sometimes, it is equally important for our co-workers to understand our SAS programs, as to make our SAS programs robust and easier to adapt to other projects. Suppose your co-workers don't use macros at all, there are still ways that you can use the variable dependent macros to eliminate manually coping and pasting. You can always set "options mprint". That way, there will be a translation of what the macro does in the log, then copy the log to the program editor and replace "MPRINT(VDM): "with blank space.

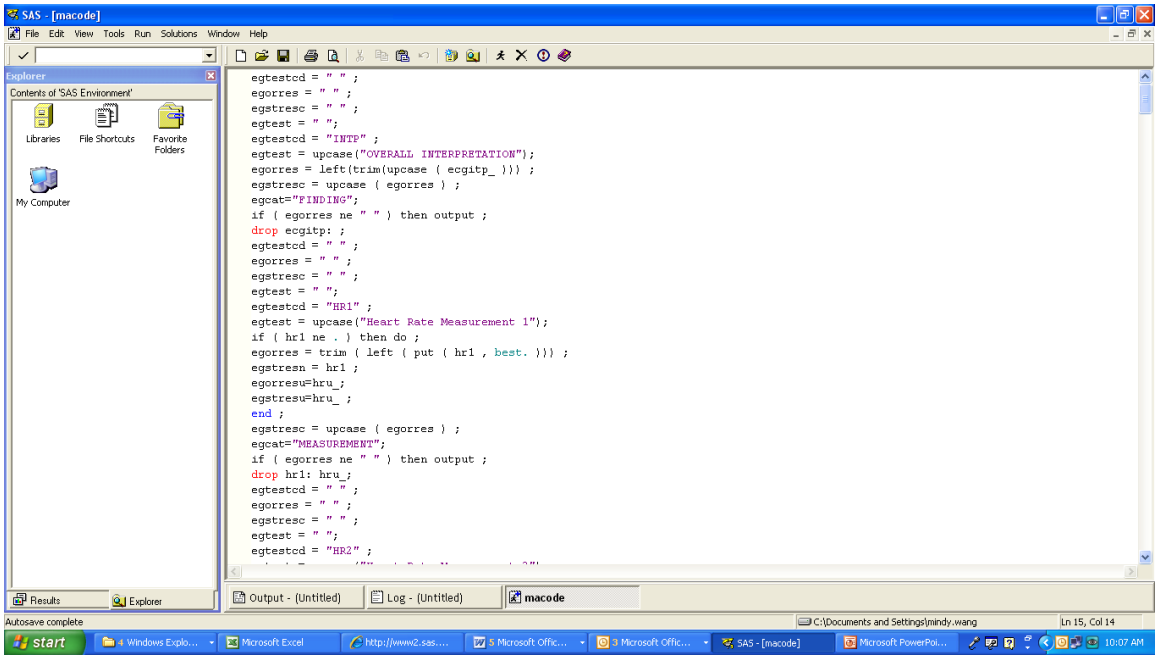
Following is the screen print of the codes that mprint generated in the log.



An even better alternative is using mprint and mfile options together. Following codes (before invoking the macro) would save the plain text generated by mprint to a SAS file without doing the copy and paste. SAS automatically generates the plain text in the file called macode.sas for you.

```
filename mprint 'desktop\macode.sas';
options mprint mfile;
```

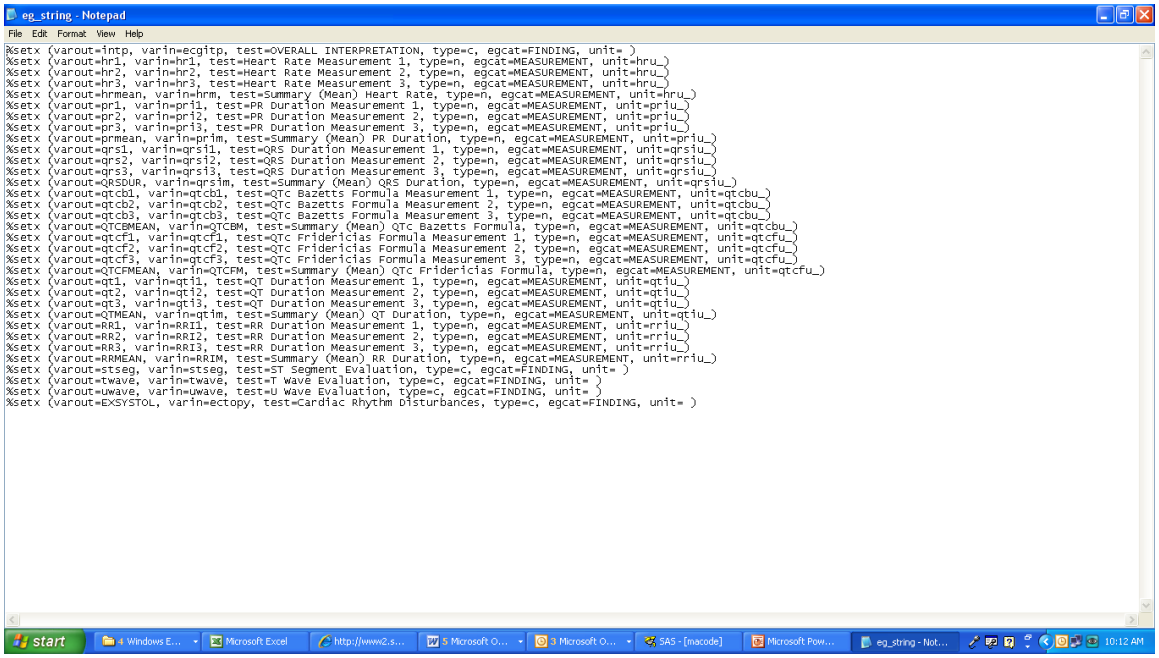
Following is a screen print of the codes that are generated and saved by mprint and mfile.



Suppose your co-workers just started to use macro, they understand the simple macro, but the variable dependent macro is way too abstract for them. You can still use part of the variable dependent macro to make your life easier. The following codes will generate the command codes in a text file. Then you can copy and paste the command lines to the simple macro method program to invoke the macro.

```
data one;
  set zero end=eof;
  n=put(_n_,2.);
  call symputx('varout'||left(n), varout);
  call symputx('varin'||left(n), varin);
  call symputx('test'||left(n), test);
  call symputx('type'||left(n), type);
  call symputx('egcat'||left(n), egcat);
  call symputx('unit'||left(n), unit);
  if eof then call symputx('n',n);
run;

%macro string;
      %do i=1 %to &n;
        stri='%setx (varout='||"&&varout&i"||', varin='||"&&varin&i"||', test='||
             "&&test&i"||', type='||"&&type&i"||', egcat='||"&&egcat&i"||
             ', unit='||"&&unit&i"||')';
      put stri;
      %end;
%mend string;

data _null_;
 length stri $250;
file 'desktop\eg_string.txt';
%string
run;
```

The following are the command codes generated by the above SAS codes.



CONCLUSIONS

The variable dependent macro method is certainly the best method to use among all three methods discussed above. Not only it has minimized typing, but it is also most adaptable to other projects. Oftentimes, we have time constraint to get thing done at certainly time, copy and paste method is a sure thing we can get it done. However, we should strive to rewrite the programs so it is easier to adapt to new situation. Even though at first it takes time to develop; In the long run, the variable dependent macro really saves time. Even if we need to increase the readability of Macro, the knowledge of variable dependent macro still makes our life easier by generating the codes that can be easily understood without a lot of manual manipulation as illustrated in the above two examples.

REFERENCES:

SAS Macro Language 1: Essentials Course Notes
Carpenter, Art (2004) Carpenter's Guide to SAS Macro Language Second Edition
Burlew, Michele M. (2006) SAS Macro Programming Made Easy

ACKNOWLEDGMENTS

Many thanks to Kevin Chung in providing the valuable suggestions for the variable-dependent macro.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the author at:
Mindy Wang
14412 Stonebridge View Dr. North Potomac, MD 20878
Phone: (240) 855-3479
Email: ymindywang@yahoo.com