Paper 165-2022

Utility Macros to Check for Changes in Macro Variables, Options, or Formats in SAS®

Aaron Brown, South Carolina Department of Education

ABSTRACT

Imagine a scenario where you create a particular macro variable, set your options as needed, or create formats you need early in a program. Later, you accidentally change one of these settings, which breaks assumptions further down in the program. Such a scenario can be difficult to troubleshoot. This paper introduces three sets of macros that use the metadata SAS® creates during a SAS session in order to check if macro variables, options, or formats changed. It utilizes the COMPARE, FORMAT, SQL, and OPTSAVE procedures.

INTRODUCTION

I worked on a project where a master program calls (via %include statements) several sub-programs written by different programmers. Near the start of the main program, certain "settings" are set, such as creating macro variables via %let statements or setting system options via an OPTION statement. Later programs assume these settings are unchanged, but sometimes a setting is accidentally changed in one of the included programs. This leads to errors in subsequent programs, which take a lot of time to debug.

To alleviate this time sink, I wrote a pair of macros: a *Store* macro and a *Compare* macro. The *Store* macro saves the current macro variables to a dataset at the start of the centralized program. After the included programs are run, a *Compare* macro compares the current macro variables to the snapshot saved via the *Store* macro.

After seeing the value of comparing the macro variables, I then wrote macro pairs for to check for changes in options and formats. This idea could be expanded to other settings, such as library references or file references, but this paper will limit itself to just 3 macro pairs: macro variables, options, and formats. See the appendix for the complete code used for this paper.

This code was written and tested in SAS Enterprise Guide. It assumes your SAS session has access to the SASHELP.VMACRO view and the WORK.FORMATS library. If your formats are automatically saved to a different library, update the code accordingly.

MACRO VARIABLES

The first pair of macros checks for changes in macro variables. It works by using the automatically-generated SASHELP.VMACRO view. To help limit extraneous output, we filter on the Scope variable to isolate global macro variables.

```
%MACRO StoreMacro;
      PROC SQL NOPRINT;
            create table _STOREMACRO_ as
            select name, value
            from sashelp.vmacro
            where SCOPE="GLOBAL"
            order by Name;
      QUIT;
%MEND;
```

In most situations, you would invoke this macro after creating the macro variables that you expect to stay static for the remainder of the program. Then, near the end of the program, invoke this comparison macro:

```
%MACRO CompMacro;
      PROC SQL NOPRINT;
            create table _STOREMACRO_COMP as
            select name, value
            from sashelp.vmacro
            where SCOPE="GLOBAL" and
                  Name IN (SELECT Name FROM _STOREMACRO_)
            order by Name;
      QUIT;
      PROC COMPARE DATA=_STOREMACRO_ COMPARE=_STOREMACRO_COMP;
            ID Name;
      RUN;
%MEND;
```

The subquery assures that we only compare macro variables that were "saved" during the invocation of the Store macro. The COMPARE procedure states any changes to the macro variables. For example, let's say there's a macro variable named AdminID that has the value of 0 when StoreMacro is invoked. Later, it is erroneously changed to the word BOGUS. When CompMacro is invoked, PROC COMPARE will give this output:

```
                    The COMPARE Procedure
      Comparison of WORK._STOREMACRO_ with WORK._STOREMACRO_COMP
                          (Method=EXACT)

              Value Comparison Results for Variables

                      ||  Macro Variable Value
                      ||  Base Value          Compare Value
name                  ||  value                value
____________________  ||  _________________+  _________________+
                      ||
ADMINID               ||  0                    BOGUS
```

Output 1. PROC COMPARE: Example of a Flagged Change

If there were no changes, you should expect a printout like the below:

```
Number of Observations in Common: 54.
Total Number of Observations Read from WORK._STOREMACRO_: 54.
Total Number of Observations Read from WORK._STOREMACRO_COMP: 54.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 54.

NOTE: No unequal values were found. All values compared are exactly equal.
```

Output 2. PROC COMPARE: Example with Changes to User-Defined Macro Variables

FORMATS

Similarly, you can use this macro to store the current formats to a dataset named _STOREFORMAT_. This code assumes your formats are saved to the WORK library; if you are using an alternate library to store and utilize formats, alter the code appropriately.[1]

```
%MACRO storeformat;
      proc format library = work.formats
            cntlout = _STOREFORMAT_;
      run;
%MEND;
```

Then this macro can be used to compare the current formats to those from when the formats were saved. Note that the filter in the SQL code limits the output to the formats that existed when formats were saved; any formats created afterwards are excluded from the comparison. We COMPARE on format name, format type[2], starting value, and ending value.

```
%MACRO compformat;
      proc format library = work.formats
            cntlout = _STOREFORMAT_COMPx;
      run;
      proc sql NOPRINT;
            create table _STOREFORMAT_COMP as
            select *
            from _STOREFORMAT_COMPx
            where FmtName||Type IN
                  (SELECT FmtName||Type FROM _STOREFORMAT_)
            order by FmtName, Type, Start, End
            ;
            drop table _STOREFORMAT_COMPx;
      quit;
      proc sort data=_STOREFORMAT_; BY FmtName Type Start End; run;
      proc compare data=_STOREFORMAT_ compare=_STOREFORMAT_COMP
            LISTALL;
            ID FmtName Type Start End;
      run;
%MEND;
```

As an example, we can start with these formats and invoke the “save macro”.

```
proc format;
      value test 1='Null' 2='okay';
      value static 5='no change';
      invalue inTest 'original'=1;
      picture picTest 1-5='picture test original';
run;
%storeformat;
```

[1] For further details on the specific utilities of the FORMAT procedure, see the official SAS documentation or Jack Shoemaker’s paper *Eight PROC FORMAT Gems* (https://support.sas.com/resources/papers/proceedings/proceedings/sugi26/p062-26.pdf).
[2] The *Type* column contains metadata about if the format is character or numeric and whether it is a value, invalue, or picture. C=character format. N=numeric format. J=character informat. I=numeric informat. P=picture format.

Then we change some formats and add a character format named *test* by another use of PROC FORMAT, then run the comparison.

```
proc format;
      value test 1='breaking' 2='okay' 3='new' 5='new';
      value $test '1'='Null' '2'='okay';
      invalue inTest 'breaking'=1;
      picture picTest 1-5='picture test change';
run;
%compformat;
```

The PROC COMPARE output has two spots where it gives particularly useful information. The first is

```
                Comparison Results for Observations

Observation 1 in WORK._STOREFORMAT_COMP not found in WORK._STOREFORMAT_:
FMTNAME=INTEST TYPE=I START=breaking END=breaking.

Observation 1 in WORK._STOREFORMAT_ not found in WORK._STOREFORMAT_COMP:
FMTNAME=INTEST TYPE=I START=original END=original.

Observation 8 in WORK._STOREFORMAT_COMP not found in WORK._STOREFORMAT_:
FMTNAME=TEST TYPE=N START=               3 END=               3.

Observation 9 in WORK._STOREFORMAT_COMP not found in WORK._STOREFORMAT_:
FMTNAME=TEST TYPE=N START=               5 END=               5.
```

Output 3. PROC COMPARE: Comparison Results for Observations, for Formats

This section shows the change to the *inTest* format, showing how we changed the character string from “original” to “breaking.” Likewise, the new values we added for the numeric *Test* format show up here, since they are additions to what was originally there.

The second spot to investigate is the comparison for the LABEL column:

```
                                 Value Comparison Results for Variables

__________________________________________________________________________________________________
                                                                ||  Base Value           Compare Value
FMTNAME             TYPE  START                END              ||  LABEL                 LABEL
_________________   ____  _____________  _____________          ||  ________________  +   ________________
                                                                ||
PICTEST             P                  1              5         ||  picture test origina  picture test change
TEST                N                  1              1         ||  Null                  breaking
__________________________________________________________________________________________________
```

Output 4. PROC COMPARE: Comparison Results for LABEL, for Formats

We can see how both the picture format *PicTest* and the numeric format *Test* had a label change. For the picture format, we changed the label for values 1-5 (the Start and End). For the numeric *Test*, we only changed the label corresponding to the value of 1 (both Start and End are equal to 1.)

An important additional note is that two formats can have the same name, but will not be compared if they are of a different Type. Our example starts with just a numeric format *Test*, but we have both a numeric and character format named *Test* when we run the comparison.   Due to the filter in the comparison macro, we do not compare the character *Test* format because it did not exist earlier; this is done on purpose to mirror how SAS treats “test.” and “$test.” as different formats. (If you remove the filter, it would show up among the others shown in Output 3.)

OPTIONS

Now we will check for changes to options. The storage macro saves the current options to a dataset named _STOREOPTION_ via the OPTSAVE procedure. We drop the option called ***_Last_*** from our comparison, because ***_Last_*** is the name of the most-recently created (e.g., last created) data set. As one would expect that to change while running a program, including it would only clutter up the output and be potentially misleading.

```
%MACRO StoreOption;
      PROC OPTSAVE OUT=_STOREOPTION_(WHERE=(OptName NE '_LAST_')); RUN;
      PROC SORT DATA=_STOREOPTION_(RENAME=(OptName=Name OptValue=Value));
            BY Name;
      RUN;
%MEND;
```

Our comparison macro uses the same method to save current macros, then compares it via PROC COMPARE.

```
%MACRO compoption;
      PROC OPTSAVE OUT=_STOREOPTION_COMP(WHERE=(OptName NE '_LAST_')); RUN;
      PROC SORT DATA=_STOREOPTION_COMP(RENAME=(OptName=Name OptValue=Value));
            BY Name;
      RUN;
      PROC COMPARE data=_STOREOPTION_ compare=_STOREOPTION_COMP;
            ID Name;
      RUN;
%MEND;
```

If we changed the default option values of *nomlogic* and *label* to *logic* and *nolabel*, we receive this output:

Name	Base Value	Compare Value
LABEL	1	0
MLOGIC	0	1

Output 5. PROC COMPARE: Comparison Results for LABEL, for Options

Note that the data set generated via PROC OPTSAVE may store the option settings differently than we normally think about them. The LABEL option was 1 for being set as “label” and 0 once changed to “nolabel”; that is, 1 means OPTION LABEL is active and 0 means OPTION NOLABEL is active. We can see the same binary encoding is used for the MLOGIC option.

ONE MACRO PAIR TO CHECK ALL THREE

Finally, let's create a pair of macros that run all these macros for you. These two macros let you quickly run all the store macros before you think changes occurred, then run all the comparisons after the spot where you fear changes might have occurred.

```
%macro QC_Store;
      %storemacro;
      %storeoption;
      %storeformat;
%mend;
%macro QC_Compare (titlelev=2);
      title&titlelev 'Macro Compare'; %compmacro;
      title&titlelev 'Option Compare'; %compoption;
      title&titlelev 'Format Compare'; %compformat;
%mend;
```

Simply invoke %QC_Store early in your program, and %QC_Compare at the spot where you want to check for any changes to macro variables, options, or formats. The *titlelev* variables in the comparison macro lets you choose what Title you want. It defaults to Title2, so as to not overwrite whatever primary Title you are currently using.

My hope is that these macros will be a useful utility for programmers to check for unintended changes that might be causing programs to crash or output to give erroneous results. As noted earlier, this concept could be expanded to checking things like library references or other mutable elements of a SAS session that could potentially change but one usually does not want to change.

APPENDIX: FULL PROGRAM CODE

Some spacing may be odd due to copying this from SAS Enterprise Guide into Word. It is recommended that you copy this into SAS or a text editor for readability. This contains some comments that were omitted in the code excerpts above.

```
*MACRO VARIABLES;

%let AdminID=0;
%let macro2=1;
%let macro3=2;
%MACRO StoreMacro;
	PROC SQL NOPRINT;
		create table _STOREMACRO_ as
		select name, value
		from sashelp.vmacro
		where SCOPE="GLOBAL"
		order by Name;
	QUIT;
%MEND;
%MACRO CompMacro;
	PROC SQL NOPRINT;
		create table _STOREMACRO_COMP as
		select name, value
		from sashelp.vmacro
		where SCOPE="GLOBAL" and Name IN (SELECT Name FROM _STOREMACRO_)
		order by Name;
	QUIT;
	PROC COMPARE DATA=_STOREMACRO_ COMPARE=_STOREMACRO_COMP;
		ID Name;
	RUN;
%MEND;
*save initial values;
%StoreMacro;
*do a comparison when no changes;
%CompMacro;
*change a macro variable then check for changes;
%let adminid=BOGUS;
%CompMacro;

*OPTIONS;

%MACRO StoreOption;
	PROC OPTSAVE OUT=_STOREOPTION_(WHERE=(OptName NE '_LAST_')); RUN;
	PROC SORT DATA=_STOREOPTION_(RENAME=(OptName=Name OptValue=Value));
		BY Name;
	RUN;
%MEND;
%MACRO compoption;
	PROC OPTSAVE out=_STOREOPTION_COMP(WHERE=(OptName NE '_LAST_')); run;
	proc sort data=_STOREOPTION_COMP(RENAME=(OptName=Name OptValue=Value));
		BY Name;
	run;
	proc compare data=_STOREOPTION_ compare=_STOREOPTION_COMP;
		ID Name;
	run;
%MEND;
*store options;
%StoreOption;
*change options from default of nomlogic and label;
options mlogic nolabel;
*_LAST_ Specifies the most recently created data set, so drop it from comparison since it is
expected to change;
proc options option=_LAST_; run;
*check for differences;
%compoption;


*FORMATS;
```

```
%macro storeformat;
	proc format library = work.formats
		cntlout = _STOREFORMAT_;
	run;
%mend;
%macro compformat;
	proc format library = work.formats
		cntlout = _STOREFORMAT_COMPx;
	run;
	proc sql NOPRINT;
		*only retain formats/informats with the same name and type as in _STOREFORMAT_.
It is assumed okay if new formats/informats made;
		create table _STOREFORMAT_COMP as
		select *
		from _STOREFORMAT_COMPx
		where FmtName||Type IN (SELECT FmtName||Type FROM _STOREFORMAT_)
		order by FmtName, Type, Start, End
		;
		drop table _STOREFORMAT_COMPx;
	quit;
	proc sort data=_STOREFORMAT_; BY FmtName Type Start End; run;
	proc compare data=_STOREFORMAT_ compare=_STOREFORMAT_COMP LISTALL;
		ID FmtName Type Start End;
	run;
%mend;

proc format;
	value test 1='Null' 2='okay';
	value static 5='no change';
	invalue inTest 'original'=1;
	picture picTest 1-5='picture test original';
run;
%storeformat;

proc format;
	value test 1='breaking' 2='okay' 3='new' 5='new';
	*a character format TEST does not overwrite the numeric format;
	value $test '1'='Null' '2'='okay';

	invalue inTest 'breaking'=1;
	picture picTest 1-5='picture test change';
run;
%compformat;


*MACRO TO RUN ALL THREE AT ONCE;

%macro QC_Store;
	%storemacro;
	%storeoption;
	%storeformat;
%mend;
%macro QC_Compare (titlelev=2);
	title&titlelev 'Macro Compare'; %compmacro;
	title&titlelev 'Option Compare'; %compoption;
	title&titlelev 'Format Compare'; %compformat;
%mend;
```

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
E-mail: arbrown@ed.sc.gov