Paper SA06

Painless Extraction: Options and Macros with PROC PRESENV

Keith Fredlund, MS (candidate) Grand Valley State University, Allendale, Michigan;
Thinzar Wai, MS (candidate) Grand Valley State University, Allendale, Michigan

ABSTRACT

The PRESENV procedure was introduced as part of SAS™ version 9.4, but little has been written about the capabilities of this procedure. The PRESENV procedure works in tandem with the PRESENV system option allowing the user to preserve certain global options, macros, macro variables, formats, and temporary data across SAS sessions. The PRESENV option can be turned on and off anytime within a program and a collection of global statements will be collected accordingly. This paper details procedure syntax, provides example code and output, and describes potential uses and limitations. Emphasis within the paper is given to use of the procedure to debug code written by others, macro value evaluation and the creation of a template for other programs. This procedure would be beneficial for users with an intermediate level of SAS data step programming.

INTRODUCTION

The PRESENV procedure was introduced as part of SAS version 9.4 and gives the user the ability to save most global options (including titles and footnotes), macros, macro variables, formats, and data between SAS sessions. While global statements and macro variables are written to a user-specified file, macros and temporary data sets are saved to a separate auxiliary directory that is stipulated within PROC PRESENV statement. This paper will focus on (a) the use of PROC PRESENV when trying to understand or debug code written by someone else, (b) the use of PROC PRESENV to extract macro values at specific points within a program, and (c) the use of PROC PRESENV to find and group global options used within a program to create a template for other programs. The saved code can then be reintroduced as part of a separate session using a %INC statement.

As a shorthand, throughout this paper PROC PRESENV will be referred to as collecting global statements, but bear in mind that it actually collects a wide set of information including macros, macro variables, formats, and data rather than only collecting global statements.

THE GENERAL FORM AND SIMPLE CODE

In its general form, PROC PRESENV requires an OPTIONS statement and a PROC step surrounding code containing the information to be extracted. The following is the general form of the code:

```
options presenv;
<sas-code-containing-a-macro,-macro-variable,-data-set,-format-or-option>
proc presenv
      permdir='<library-in-which-to-save-temporary-data-set-code-and-macro-
file>'
      sascode='<path-on-which-to-save-generated-code>'
run;
```

While PROC PRESENV functions best when both PERMDIR= and SASCODE= are both stipulated, the code can be run without either the PERMDIR= statement (if there is no need to preserve temporary data sets created by the code) or without the SASCODE= (if there is no need to extract SAS code that could be used to recover options, macro variables and macros). If the SASCODE= option is used, the extracted output code could be read back into a future program with the use of the code %INC '<path-on-which-to-save-generated-code>'.

PROC PRESENV has only a single optional argument: SHOW_COMMENTS. This optional argument prompts the PRESENV procedure to display all global statements in the generated code. It is of somewhat modest utility but one possible use is described in further detail later in this paper in the section describing limitations. (See Limitations: Running PROC PRESENV more than once in the same session.)

A simple sample code employing PROC PRESENV:

```
options presenv;
%let mv_day= Thursday;
proc presenv
sascode= 'C:\PROC PRESENV\Simple_Sample_Output.SAS';
run;
```

Based on this code, SAS would create the SAS file named as "Simple_Sample_Output." The created SAS file will contain the following code as the Simple Generated Code:

```
options presenv;
%let saved_options=options %sysfunc(getoption(source))
%sysfunc(getoption(notes)) %sysfunc(getoption(obs,keyword))
%sysfunc(getoption(firstobs,keyword));
options obs=max firstobs=1;
options nosource nonotes;
%macro dummy_macro_for_presenv; %mend;
data _null_; rc=filename('presenvf',' ','temp'); run;
proc printto log=presenvf new; run;
proc sql;
create table _data_ as select distinct memname
       from dictionary.catalogs
       where libname='PERMDIR' and memtype='CATALOG';
    quit;
data _null_; set;
    call execute(
     cats('proc catalog force c=PERMDIR.',memname,
     '; copy out=work.',memname,'; quit;')
     );
    run;
proc delete; run;
proc copy in=PERMDIR out=work mt=data; quit;
proc printto; run;
data _null_; rc=filename('presenvf'); run;
data _null_;
    call symput('MV_DAY',' Thursday');
    call symput('AFDSID','0' );
    call symput('AFDSNAME',TRIMN(' '));
    call symput('AFLIB',TRIMN(' '));
    call symput('AFSTR1',TRIMN(' '));
    call symput('AFSTR2',TRIMN(' '));
    call symput('FSPBDV',TRIMN(' '));
run;
options _last_=_null_;
&saved_options;
```

Please note that the newly created output code includes the macro variable created by the Simple Sample Code (MV_DAY) within the DATA _NULL_ step towards the end of the file. In addition to the

macro variable created by the Simple Sample Code, six other automatic variables are included in the Simple Generated Code: AFDSIS, AFDSNAME, AFLIB, AFSTR1, AFSTR2, and FSPBDV.

MORE COMPLEX CODE

The following example is more complicated, containing an options statement, a macro, macro variables and a temporary data set (hereafter called as the More Complex Code):

```
options presenv;
option mlogic;

%let mv_day= Tuesday;

data test_data_set;
      day = "The Day is Tuesday";
run;

%macro m_printday;
      %if &mv_day = Tuesday %then %do;
      proc print data = test_data_set;
      run;
      %end;
%mend;
%m_printday;

libname libra 'C:\PROC PRESENV';
filename presfile 'C:\PROC PRESENV\Generated_Code.SAS';
proc presenv
      permdir=libra
      sascode=presfile;
run;
```

The code above generates a folder to permanently hold macro code in the folder specified in the LIBNAME statement, creates permanent SAS data sets of all the temporary data sets in the same folder and generates the following code (the More Complex Generated Code):

```
options presenv;
option mlogic;
libname libra 'C:\PROC PRESENV';

filename presfile 'C:\PROC PRESENV\Generated_Code.SAS';
%let saved_options=options %sysfunc(getoption(source))
%sysfunc(getoption(notes)) %sysfunc(getoption(obs,keyword))
%sysfunc(getoption(firstobs,keyword));
options obs=max firstobs=1;
options nosource nonotes;
%macro dummy_macro_for_presenv; %mend;
data _null_; rc=filename('presenvf',' ','temp'); run;
proc printto log=presenvf new; run;
proc sql;
create table _data_ as select distinct memname
       from dictionary.catalogs
       where libname='LIBRA' and memtype='CATALOG';
    quit;
data _null_; set;
```

```
    call execute(
     cats('proc catalog force c=LIBRA.',memname,
     '; copy out=work.',memname,'; quit;')
     );
    run;
proc delete; run;
proc copy in=libra out=work mt=data; quit;
proc printto; run;
data _null_; rc=filename('presenvf'); run;
data _null_;
    call symput('MV_DAY','Tuesday' );
    call symput('AFDSID','0' );
    call symput('AFDSNAME',TRIMN(' '));
    call symput('AFLIB',TRIMN(' '));
    call symput('AFSTR1',TRIMN(' '));
    call symput('AFSTR2',TRIMN(' '));
    call symput('FSPBDV',TRIMN(' '));
run;
options _last_=WORK.TEST_DATA_SET;
&saved_options;
```

As with the simple example described earlier, this code can be read into a future program with the use of the %INC <path-to-generated-code> statement.

The execution of the More Complex Code also generates a permanent SAS data set with the filename specified in the SASCODE= statement. This permanent SAS data set can be referenced by other SAS programs.

A file (sasmacr.sas7bcat) containing all of the macros captured by PROC PRESENV is also created in the folder specified by the PERMDIR= statement. Note that while options, with one exception, are only added to the generated code when OPTION PRESENV is turned on, macros and macro variables will be saved in the macro catalogue (macros) or the generated code (macro variables) even if the macros or macro variables were created before OPTION PRESENV was first turned on. The exception to the way options are stored is that each time the presenv option is turned on or off throughout the program, this is recorded in the generated code as commented code, except for the last time they appear.

Options, file references and library references included in the More Complex Code appear at the top of the More Complex Generated Code and macro variables appear near the bottom.

LIMITATIONS

RUNNING PROC PRESENV MORE THAN ONCE IN THE SAME SESSION

Among the limitations to be aware of when running PROC PRESENV is the way the contents of the generated code behave if the PROC PRESENV code is run multiple times during the same session. Rather than over-writing options used within the code, this procedure duplicates options each time it is run. Consequently if the More Complex Code above was run only once, the options section would contain the line of code:

```
option mprint;
```

However, if PROC PRESENV was run a second time, the generated code would include OPTION MPRINT twice: Once at the top of the code and then again a few lines later. Subsequent executions of the More Complex Code would result in more and more repetitions of any options that were a part of the program.

The optional argument SHOW_COMMENTS calls attention to each of the duplicated global options by adding three lines of code below each duplicated option:

```
libname libra'<library-in-which-to-save-temporary-data-set-code-and-macro-
file>';
filename presfile '<path-on-which-to-save-generated-code>';
options presenv;
```

CAPTURING OPTIONS, MACROS, AND MACRO VARIABLES FROM OTHER PROGRAMS RUN DURING THE SAME SESSION

Depending on the use to which PROC PRESENV is being put, the user should be aware that after the PRESENV option has been turned on, code being generated will capture only the last iteration of all options, macros and macro variables being employed in other SAS programs within the same session.

With "options", for instance, if in a program executed earlier in a session, LINESIZE were set to 30, but then in a subsequent program LINESIZE were set to 40, only LINESIZE 40 would be recorded in the code generated by PROC PRESENV. Some of the hurdles of this limitation can be managed by turning on the PRESENV option (OPTION PRESENV) at the beginning of the target program and then turning it off (OPTION NO PRESENV) at the end of the target program.

Similarly macros and macro variables from other programs run during the same session will be recorded into the code generated by PROC PRESENV unless they are overwritten by a new macro or macro variable. Care should be taken in regard to the order that programs are run within a session to make sure that the desired macros and macro variables are included in the macro library: PROC CATALOGUE can be used to delete macros from the macro catalogue and macro variables can be deleted using %SYMDEL.

The user should also consider that sequential executions of PROC PRESENV within a single session will overwrite previously generated code. Consequently, attention should be given to the order and types of programs that are being run while the PRESENV option is active.

GLOBAL OPTIONS NOT SAVED BY PROC PRESENV

As might be expected, certain global options are not appropriate for inclusion in the code generated by PROC PRESENV. Table 1 is a comprehensive list.

Table 1: Global Statements Not Included in Code Generated by PROC PRESENV

Global Statement	Category
Null	Action
Comment	Log Control
Page	Log Control
Resetline	Log Control
Skip	Log Control
X Statement	Operating Environment

CHECKPOINT	Program Control
DM	Program Control
END SAS	Program Control
%INCLUDE	Program Control
EXEC-UTE_ALWAYS	Program Control
%LIST	Program Control
RUN	Program Control
%RUN	Program Control

While the SAS support documentation indicates that PROC PRESENV saves all global statements between one session and another, there are many global statements that are not saved. The omission of specific global statements is understandable: most of the omitted global options come from the Log Control category (COMMENT, PAGE, RESET LINE, SKIP), the Operating Environment category (X Statement) and the Program Control category (CHECKPOINT, DM, END SAS, %INCLUDE, EXECUTE_ALWAYS, %LIST, RUN, %RUN. The only omitted global statement that does not fall into a particular category is the Null statement.

While most of the options are understandably not preserved between sessions (i.e. there would be little use in including the Null statement), there is, however, an argument for the inclusion of the %INCLUDE option, perhaps only as a comment, to allow the user to have a list of the external programs included in a session.

USES OF PROC PRESENV: OVERVIEW

While one of the primarily uses of PROC PRESENV is in the context of a SAS Enterprise Guide grid environment, this paper examines some of the other ways that PROC PRESENV can be used.

USES OF PROC PRESENV: UNDERSTANDING OR DEBUGGING CODE WRITTEN BY SOMEONE ELSE

When encountering a program for the first time, it can be helpful to understand the structure of the code being examined. While some coders provide ample commentary to allow the program to be easily understood, this is not always the case. PROC PRESENV can be used as part of an investigatory process to extract all global statements from within the source program, creating a high-level outline of the program including titles, footnotes, file names, library references and the like that can help a user understand some of the original program structure.

It can be helpful during this process to use the SHOW_COMMENTS option with PROC PRESENV as it will clearly reveal in the output whether the option was changed later on within the session. This is only an aid during the process as global statements can be over-written throughout a program.

USES OF PROC PRESENV: EXTRACTING MACROS FROM SPECIFIC POINTS IN A PROGRAM

A key benefit of PROC PRESENV is that the PRESENV option can be turned on and off repeatedly throughout the program, this provides a method for accumulating the values of different macros and

macro variables at a series of specific points throughout the program. This has a somewhat similar utility to the SYMBOLGEN option except that PROC PRESENV allows a somewhat more targeted approach.

USES OF PROC PRESENV: FINDING GLOBAL OPTIONS WITHIN A PROGRAM FOR USE ELSEWHERE

Programmers sometimes disperse global options throughout a program rather than grouping all of the options, as much as possible, at the beginning of a program. Using PROC PRESENV, a subsequent editor of the program can gather all of these options together into the generated code. This not only allows the programmer to better understand the program, but also would allow this generated code to be easily copied into another program en masse, rather than having to copy multiple sections of code one at a time.

CONCLUSION

Though not a simple procedure, PROC PRESENV provides a new approach for the creative SAS programmer interested in extracting specific elements of a SAS program. This paper proposes a few specific uses for the procedure and describes in detail its implementation and outputs.

REFERENCES

Squillace, Jan. “Flexibility by Design: A Look at New and Updated System Options in SAS® 9.4” http://www.lexjansen.com/wuss/2013/146_Paper.pdf (8/30/2016).

CONTACT INFORMATION

Questions and comments can be directed to the authors at:
Keith Fredlund
(616) 309 - 5857
fredlunk@mail.gvsu.edu

Thinzar Wai
(616) 643 - 7060
wait@mail.gvsu.edu