Paper 060-2012

Standardized Macro Programs for Macro Variable Manipulation

Chris Swenson, UW Health, Madison, WI, USA

ABSTRACT

Creating macro variables for later use can help to create dynamic, powerful SAS® programs. Creating those variables in a standard, consistent manner without causing any conflicts with other macro variables, however, can be repetitive, time consuming, and prone to error. Here we present macro programs that assist with generating macro variables with the intent to use the variables in specific ways later in the program. Each macro meets a set of criteria including allowing the filtration of input data (if possible); deleting conflicting macro variables; creating macro variables in a consistent, predefined manner; and reporting the result of the macro's activity.

INTRODUCTION

The concept of a standardized set of macro programs that consistently generated macro variables came about when I was asked to design a SAS program that accepted on-the-fly user-written specifications and could be run many different ways. The goal was to allow the user to input various parameters in a spreadsheet and then run the SAS code by calling a very large autocall macro with only one argument: the type of specification to run.

Part of the request was the ability to run the macro more than once with different specifications each time. Therefore, it was necessary to purge the generated macro variables in between each run; otherwise, the code may inadvertently include specifications from a previous program. Additionally, the generated macro variables may ultimately populate a list within an IN statement, or the values may be looped through in a %DO loop, or they may simply been used to define a table name or variable name. For these reasons, it was necessary to develop a method of creating macro variables in different ways, but in a consistent, standard manner.

Here I describe the standards, a set of macro programs that adhere to the standards, and how these programs can be used to write dynamic, reusable code.

STANDARDS

The following standards were developed in order to write macro variable manipulation programs that provide flexibility, consistent results, avoid conflicts, and produce accurate results.

1. Delete similar macro variables that already exist to avoid conflicts.
2. Create the specified macro variables in the specified manner.
3. Report the macro variables created in a single list, ideally using SASHELP.VMACRO as the source.
4. Allow for filtering of the input data (if possible).

Deleting similar macro variables is necessary for the purpose of reusability (within a single SAS session), especially when a series of numbered macro variables is created (e.g., Table1, Table2, Table3…). The first series may have 10 items, where the second may have only 3. However, if items 4–10 are not deleted, SAS may loop through the new 3 items and the rest of the old items up to 10. At best, this is wasted processing; at worst, this would generate inaccurate results.

The following code demonstrates the danger of not purging macro variables between projects.

```
data input;
    format project $5. tables $40.;
    infile datalines dsd;
    input project $ tables $;
datalines;
class,sashelp.class
class,sashelp.classfit
cars,sashelp.cars
;
run;

%macro loop;
  %local project1 project2;
  %let project1=class;
  %let project2=cars;
```

```
    %do p=1 %to 2;
      data _null_;
          set input;
          where project="&&PROJECT&P";
          call symputx(compress('table' || put(_n_, 8.)), tables, 'G');
      run;
      %local t;
      %let t=1;
      %do %while(%symexist(TABLE&T));
        %put NOTE: Table &&TABLE&T included.;
        %let t=%eval(&T+1);
      %end;
    %end;
  %mend loop;
  %loop;
```

In the first run, two macro variables are generated: TABLE1 = sashelp.class and TABLE2 = sashelp.classfit. In the second run, one macro variable is *updated* (it is not generated, since it already exists): TABLE1 = sashelp.cars. However, TABLE2 still exists; thus, it is included in the process for the Cars project, even though it is not relevant. The macro programs described below delete conflicting macro variables and generate a count, when sensible, to use in %DO loops. Both features help to avoid the problem in the above code.

Reporting the activity of the macro is important for two reasons: First, in development, the developer can run the macro and see what is generated in order to develop references to the variables in the code. Second, in production, the execution of the macro writes to the log and acts as documentation of what parameters were used in executing the code. Without this information, it may be more difficult to understand or troubleshoot a program.

The ability to filter data input to the programs allows for more flexibility in the structure of the input data sets. It is possible to set up a set of master tables that feed in to any given program, and based on a simple filter applied to all of the master tables, the macros could generate different macro variables, changing the way the program works.

USAGE

The macros discussed here are available at the following website: http://sas.cswenson.com/. I recommend setting up the macros as autocall macros, which would require the user to modify the SASAUTOS= option and save the source code in a particular directory. For more documentation on autocall macros, see SAS Institute Inc. (2012). For demonstration purposes, the reader may execute the following code to immediately include the macros in their current SAS session:

```
/* Download and compile SiteMacros */
filename code url "http://sas.cswenson.com/downloads/macros/SiteMacros.sas";
%include code / nosource;
filename code clear;

/* Download and compile multiple macros */
%SiteMacros(IntoList ObsMac SetVars TableVars VarMac);
```

Table 1 in the Appendix lists the arguments for each macro program described above. Each macro program has additional arguments not discussed in detail in this paper. Review the table for more information about these arguments.

Each macro program outputs different types of macro variables generated in different ways. Figure 1 shows how each macro can convert values of a data set into macro variables.

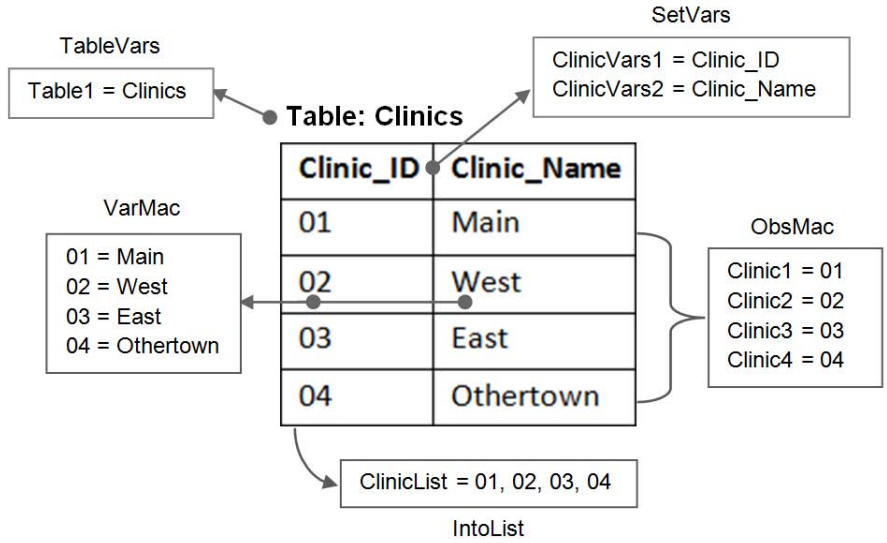

Figure 1. Converting Data Set Values into Macro Variables

In order to demonstrate how these macros can be used, let us assume that the library "Medical" has the following tables, columns, and values, as display in Figure 2.

Table: Clinics

Clinic_ID	Clinic_Name
01	Main
02	West
03	East
04	Othertown

Table: Providers

Provider_ID	Provider_Name
1001	Dr. John A
1002	Dr. Jim B
1003	Dr. Jane C
1004	Dr. Jessica D

Figure 2. Tables, Columns, and Values in the Medical Library

OBSMAC

ObsMac sets observations in a data set to macro variables, using a specified prefix and the observation number to create the macro variable name. For example, to generate multiple macro variables for each clinic, containing the clinic ID:

```
%ObsMac(Medical.Clinics, Clinic_ID, Clinic);
```

The macro will generate 4 macro variables with the values: Clinic1 = 01, Clinic2 = 02, Clinic3 = 03, and Clinic4 = 04.

These macro variables are good for using in macro loops that iterate based on macro variable names. For example:

```
%macro loop;
  %local i;
  /* Note: CLINICCNT is generated by the ObsMac macro program */
  %do i=1 %to &CLINICCNT;
    %let current=&&CLINIC&I;
    proc sql;
      create table Clinic_&CURRENT._Patients as
      select *
      from patients
      where clinic_id="&CURRENT"
      ;
    quit;
  %end;
%mend loop;
%loop;
```

ObsMac creates the macro variables using a null DATA step and the CALL SYMPUTX routine.

```
data _null_;
    set Medical.Clinics end=end;
    call symputx(compress("CLINIC" || put(_n_, 8.)), CLINIC_ID, 'G');
    if end then call symputx("CLINICCNT", put(_n_, 8.), 'G');
run;
```

The extra macro variable CLINICCNT, as shown in the example above, is useful for determining how many iterations to loop through for a particular set of macro variables.

TABLEVARS

TableVars generates macro variables for tables identified from SASHELP.VTABLE, naming the variables using the specified prefix and the observation number of the table. This is useful when processing multiple data sets. It is similar to ObsMac, except it has a static source table (i.e., SASHELP.VTABLE). For example, to generate a macro variable for each data set in the MEDICAL library:

```
%TableVars(Medical, table);
```

The macro will generate 2 macro variables with the values: table1 = Clinics and table2 = Providers

Just like the example for ObsMac, the values can be looped through for processing.

```
%macro loop;
  %local i;
  /* Note: TABLECNT is generated by the TableVars macro program */
  %do i=1 %to &TABLECNT;
    %let current=&&TABLE&I;
    proc append base=master data=&CURRENT;
    run;
  %end;
%mend loop;
%loop;
```

TableVars creates the macro variables in the same manner as ObsMac.

INTOLIST

IntoList creates a macro variable that contains a list of values from a column in a data set. It can optionally define the delimiter between values (e.g., comma). For example, to generate a macro variable that contains a list of all Clinic_ID values:

```
%IntoList(Medical.Clinics, Clinic_ID, ClinicList, sepby=COMMA);
```

The macro will generate 1 macro variable: ClinicList = 01, 02, 03, 04.

To generate a macro variable with a list of all Provider_ID values, with double quotes and commas in between:

```
%IntoList(Medical.Providers, Provider_ID, ProviderList, sepby=QQC);
```

The macro will generate 1 macro variable: ProviderList = "1001", "1002", "1003", "1004".

These macro variables are useful for use in IN lists. Additional delimiters are available and can be found by leaving the SEPBY argument blank:

```
%IntoList(Medical.Providers, Provider_ID, ProviderList, sepby=);
```

Intolist generates the macro variables using PROC SQL with the INTO … SEPARATED BY statement.

```
proc sql noprint;
    select distinct Provider_ID
    into :ProviderList separated by '", "'
    from Medical.Providers
    ;
quit;
```

The SEPBY= argument populates the value following the word “BY”, switching between single and double quotes as necessary to generated the intended result.

SETVARS

SetVars creates one or more macro variables from the variable names in a data set. The generated macro variables can either be a list within one macro variable (like IntoList) or multiple macro variables named with the specified prefix and appended observation number (like ObsMac). For example, to generate macro variables for each column name in the Clinics table:

```
%SetVars(Medical.Clinics, ClinicVars);
```

The macro will generate 2 macro variables with the values: ClinicVars1 = Clinic_ID and ClinicVars2 = Clinic_Name

To generate one macro variable with all columns listed in the value:

```
%SetVars(Medical.Clinics, ClinicVars, type=LIST);
```

The macro will generate 1 macro variable: ClinicsVars = Clinic_ID Clinic_Name.

The TYPE argument is set to MULTI by default; thus, it is not necessary to specify it in the first SetVars example. The user can set an additional argument (VTYPE=) to select only character or numeric variables.

SetVars generates the macro variables by outputting the contents of the input table using PROC CONTENTS, then running a null DATA step (see ObsMac example) or a PROC SQL with the INTO statement (see IntoList example), depending on what the user specified. Similar to IntoList, the user can specify a delimiter using the SEPBY= argument.

VARMAC

VarMac creates macro variables from two columns, where one column supplies the name for the macro variables and another column supplies the values. For example, to create macro variables named after the clinics with the IDs as the values:

```
%VarMac(Medical.Clinics, Clinic_ID, Clinic_Name);
```

The macro will generate 4 macro variables with the values: 01 = Main, 02 = West, 03 = East, and 04 = Othertown.

These macro variables could be used like formats to apply a name to an ID, or to set data and columns sources so that they can be dynamically changed (e.g., a macro variable named "source" with the value "Clinics").

VarMac generates macro variables much like ObsMac, with one key difference. Instead of using a specified prefix for the generated macro variable names, it uses another variable in the input data set:

```
data _null_;
    set Medical.Clinics end=end;
    call symputx(Clinic_ID, Clinic_Name, 'G');
    call symputx(compress('NAMEVAR' || put(_n_, 8.)), upcase(CLINIC_ID), 'L');
    if end then call symputx("CLINIC_IDCNT", put(_n_, 8.), 'G');
run;
```

As with ObsMac, an additional variable is output with the suffix “CNT” to use in %DO loops.

CONCLUSION

It is possible to use all of the above macro programs within one piece of code. I came up with the following scenario as a demonstration of the power and ease-of use these macros can provide. Imagine that a nurse wishes to review the outcome for a set of quality measures for a list of patients, only including results from 2009 and 2010. The nurse does not want to review counts, amounts, or any other numerical field: only categorical output. Further, the nurse may change any of these parameters at any time. Just to make this even more complicated, the organization is in the middle of restructuring its files, and the locations of all data sets may abruptly change. Currently, each measure is stored in a separate library with one data set for each year the measure was run.

I wrote up the initial steps in compiling the data in less than an hour. It took a little longer than expected because I had to come up with a scenario in which I could use all of the programs (and generate some dummy data). The code is available online at the following URL. Within it, each of the programs above is used with a specific purpose.

http://sas.cswenson.com/sgf/paper-sample

ObsMac is used to convert the current list of measures into macro variables in order to loop through each library. VarMac is used to dynamically set the libraries to different physical locations, since we know the drives and paths may change. IntoList is used to convert a list of patient IDs into a macro variable that can be used in an IN list. TableVars is used to set the tables in each library to loop through and append to a master data set. SetVars is used to select only character variables in each table.

The code does not completely fulfill the request, but it's a start. And best of all, it's completely dynamic. In fact, if the analyst used an input spreadsheet instead of the data sets generated by the code, the nurse could input the parameters him/herself. Using similar processes, I have been able to write dynamic code in which all I had to do to update the code was edit a spreadsheet, and then the next time it was run (as a scheduled task), the code and output changed based on the new input parameters.

Using these macro programs, SAS users can quickly create dynamic programs while maintaining flexibility, transparency, and accuracy. These macro programs can be used for on-the-fly analyses, adhoc requests, or production code, with little need for code modification in between these phases. Best of all, it frees up the analysts' time for working on more complex processes.

REFERENCES

SAS Institute Inc. (2012). SAS 9.2 Macro Language: Reference: Saving Macros in an Autocall Library [URL]. http://support.sas.com/documentation/cdl/en/mcrolref/61885/HTML/default/viewer.htm#a001328769.htm

ACKNOWLEDGMENTS

Special thanks to Lauren Lake, my SAS Global Forum mentor.

CONTACT INFORMATION

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

Chris Swenson
E-mail: chris@cswenson.com
Web: http://www.cswenson.com/

APPENDIX

Macro	Arguments	Type	Values	Description
IntoList	ds	Required	data set	Input data set
IntoList	var	Required	variable(s)	Variable(s) to set to a macro variable list
IntoList	listname	Required	m. variable	Name(s) of macro variable list
IntoList	sepby=	Defaulted	set value to blank for a list of valid values	Separator, defaulted to BLANK
IntoList	where=	Optional	criteria	Criteria to filter the input data set
IntoList	distinct=	Defaulted	Y/N	Y/N flag to indicated whether to pull a distinct list, defaulted to Y
ObsMac	ds	Required	data set	Input data set
ObsMac	invar	Required	variable	Variable(s) to set to multiple macro variables
ObsMac	mvar	Required	m. variable	Prefix used to name macro variables
ObsMac	where=	Optional	criteria	Criteria to filter the input data set
SetVars	ds	Required	data set	Input data set
SetVars	mvar	Required	m. variable	Prefix used to name macro variables
SetVars	type=	Defaulted	MULTI/LIST	Specifies whether to create multiple macro variables, appending a number to the specified macro variable name (MULTI, default), or a list within the one macro variable name (LIST)
SetVars	vtype=	Optional	C/N	Variable type to set as macro variables, either C for character or N for numeric
SetVars	sepby=	Defaulted	set value to blank for a list of valid values	Separator, defaulted to BLANK
SetVars	where=	Optional	criteria	Criteria to filter the input data set
TableVars	libname	Required	library	Input LIBREF
TableVars	tablevar	Required	m. variable	Prefix used to name macro variables
TableVars	name=	Optional	name	Criteria used to filter tables that match the specified prefix
TableVars	where=	Optional	criteria	Criteria to filter the input data set (Note: Review SASHELP.VTABLE for columns)
TableVars	test=	Defaulted	Y/N	Y/N flag to indicate whether to test the macro
VarMac	ds	Required	data set	Input data set
VarMac	namevar	Required	variable	Variable that supplies the name for the macro variables
VarMac	valuevar	Required	variable	Variable that supplies the values for the macro variables
VarMac	where=	Optional	criteria	Criteria to filter the input data set

Table 1. Arguments for the Macro Variable Manipulation Macro Programs