Paper 121-2011

A Macro to Modify Attributes for All Character Variables in SAS® Data Sets

Suresh B. Kadaru, Syngenta Seeds, Inc., Stanton, MN, USA

ABSTRACT

Often SAS® programmers modify the attributes of text columns in data tables as part of their SAS routines. Some of these modifications might be necessary to avoid erroneous output results, especially while performing SQL queries, DATA step merges, or while subsetting tables. In such cases, a macro can be used to apply desired SAS character functions across all variables in incoming data tables. Thus, there is no need to repeatedly type the function names in SAS statements, eliminating the risk of forgetting the use of necessary functions in the code, and providing flexibility to incorporate the latest SAS functions. This macro has a high utility value when it is run in a production environment ahead of other user tasks, without altering the existing code. Currently, this macro can handle several of the SAS character functions that modify the attributes of text variables and do not return a value after execution.

INTRODUCTION

While performing queries, merges, and data subsetting steps, it is quite common that we obtain incomplete or inaccurate results if the character variables from incoming tables do not match properly e.g., differ by case. SAS character functions are very handy in such cases and can solve the issue by incorporating them into key IF, WHERE, and ON statements.

However, if we are dealing with multiple tables and multiple columns, it then becomes difficult to keep track of variable differences. An example of this may be a situation where one needs to perform several SQL queries involving multiple text columns from multiple SAS tables. One way to deal with this situation is to include the required SAS functions in all query statements. Soon the SAS code becomes lengthy and there remains the risk that the programmer may forget the use of a few required SAS functions. This situation may have already happened in the existing in-house macros where appropriate SAS functions need to be inserted, which sometimes can be difficult to do.

Another approach would be to harmonize all the variables in the input SAS tables before performing queries, merges or subsetting. The macro described in this paper provides the functionality of modifying all the character variables in the input tables using a given SAS function prior to their use in the queries. The code reads the SAS table either from work or permanent libraries and applies a user-defined SAS function to all text columns present in that SAS table. The user can run this macro as a first step and does not have to worry about the use of the same SAS function multiple times.

During the macro run, the original table is only modified once using a PROC SQL UPDATE statement and hence, incoming table dependencies such as primary keys, indices, and sorting order etc., should not be disturbed. Only the values within text columns will be changed permanently as per the SAS function defined by the user. This SAS macro can be run on both Linux and Windows operating systems and is compatible with several previous SAS base package versions, which support both PROC CONTENTS and PROC SQL procedures.

ADDITIONAL FEATURES

In addition to the merits discussed in the above paragraphs, the concise SAS code used in this macro also provides the following additional functionality:

1) Verification of input table(s) presence at the beginning of the run
2) Verification of user input function name prior to running the main code; else run will abort
3) Ability to display feedback and error messages in log window for easy troubleshooting
4) Ability to abort the run automatically if character variables are not detected in the submitted dataset(s)

DESCRIPTION

The user needs to provide only two input parameters: 1) "datasets" parameter – a list of SAS table name(s) that need to be modified and 2) "function_name" parameter – the SAS function to be used. Both are text inputs and should be of at least one character length. Only the valid SAS names will be processed, else an error note is written to the log. Multiple SAS data tables can be processed through the macro by separating names in the "datasets" input using a “space” character. However, during batch processing of multiple datasets, the same SAS function will be applied

similarly across all SAS tables.

The macro code uses two simple procedures from the base SAS package, namely, PROC CONTENTS and PROC SQL. In addition, the following eleven macro variable names are generated for either checking the input parameters or for execution of code: D, DSNO, DATASETS, EXIT, EXITVARIABLE, FUNCTION_NAME, LASTEXIT, MODIFY_ALL_CHARACTER_VARIABLES, SEL_DATASET, SEL_CVARIABLES, SEL_NVARIABLES. Users need to pay attention to these macro variable names when incorporating this macro code into their own macros.

EXPLANATION FOR THE MACRO CODE

PART1

```
%macro Modify_All_Character_Variables;
%let ExitVariable=1;
/*checking whether or not required input parameters are defined*/
%if %length(&datasets)>0 and %upcase(&datasets)^= and %length(&function_name)>0 and
%upcase(&function_name)^=
%then %let ExitVariable=0;
%if %eval(&ExitVariable)>0 %then %do;
%put;%put !!! Not enough parameters defined to run this macro !!!;%put;
%goto LastExit;
%end;

/*checking whether or not input "function_name" parameter is valid*/
%if %length(%sysfunc(&function_name(TEST)))<4 %then %do;
%put; %put !!! Please provide valid SAS character function name !!!;%put;
%goto LastExit;
%end;
```

This portion of the code checks whether or not the user has provided both of the required inputs by using the %SYSFUNC and %LENGTH functions. The “ExitVariable” macro variable acts as an ON/OFF switch with a default value of ‘1’ (OFF condition). Only when both input parameters are defined does its value get reset to ‘0’ and the macro proceeds further. If any of the two parameters are not defined, an error message will be displayed in the log window. The second check validates the applicability of the user-defined SAS function by testing the changes to the length of the word “TEST”. Failure of either check diverts the code to go to the “LastExit” statement at the end of the macro.

PART2

```
options nonotes;
/*counting the number of submitted datasets*/
data _NULL_; call symput('DSNo',count("&datasets",' ')+1); run;
%do D=1 %to &DSNo;
%let sel_dataset=%scan(&datasets,&D,' ');
%put;
%put Applying "&function_name" function to the dataset: &sel_dataset;
%if %sysfunc(exist(&sel_dataset))=0 %then %do;
%put !!! "&sel_dataset" does not exist. Please check. !!!; %put;
%goto Exit;
%end;
```

This portion of code counts the number of “space” characters in the user-defined “datasets” parameter. The best method to count “useful” spaces is to use the CALL SYMPUT function inside a DATA _NULL_ step. Then a %DO loop is set with the limit of as many datasets as needed to process. Within the %DO loop, “sel_dataset” names are assigned sequentially, and the presence of the respective SAS table is also checked using and EXIST functions.

PART3

```
/*identifying the character variables from the SAS data table*/
proc contents data=&sel_dataset out=Tmp noprint; run;
proc sql noprint;
%let sel_Cvariables=; %let sel_Nvariables=;
select compress(left(NAME)),
catt(compress(left(NAME)),"=&function_name(",compress(left(NAME)),')')
into:sel_Cvariables separated by ', ',:sel_Nvariables separated by ', ' from Tmp where
Type=2;
```

```
drop table Tmp;
%if %length(&sel_Nvariables)=0 %then %do;
%put !!! There are no character variables in this dataset !!!;%put;
%goto Exit;
%end;
```

Using PROC CONTENTS, information about character variables within each of the input data tables is generated. If there are no character variables in the given dataset, the code is diverted to the "Exit" statement before the loop %END statement. In addition, a message is written to the log window regarding this issue. Otherwise, SQL syntax for updating character variables with the user-defined SAS function gets generated. This information is stored in SAS memory with the help of the "sel_Nvariable" macro variable.

PART4

```
/*applying the character function specified by the user*/
update &sel_dataset set &sel_Nvariables;
quit;
%put "&function_name" converted variables: &sel_Cvariables;
%Exit : ;
%let sel_dataset=; %let Sel_cvariables=; %let Sel_nvariables=;
%put;

%end;
%put;%put  !!! Done. Thanks for using "Modify_All_Character_Variables" macro !!!;%put;
%LastExit: ;
%symdel datasets function_name;
option notes;
%mend Modify_All_Character_Variables;
```

The first six lines of this code execute the UPDATE statement using the SQL Procedure and also list the variables that are converted into the log window. The remaining code closes the loop, sets all the macro variable names used in the macro to null values, and ends the macro.

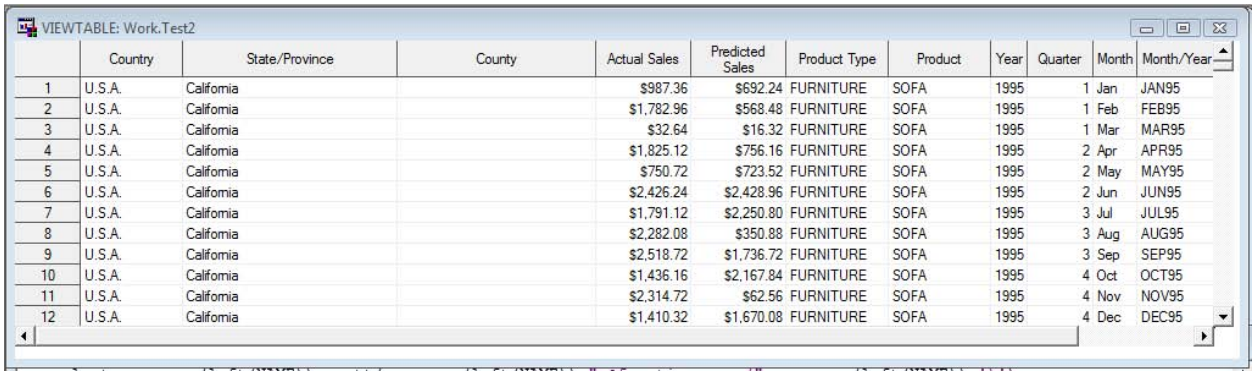
SAMPLE TEST DATASETS

To demonstrate the utility of this macro, the PRDSAL2 table from the SASHELP library was chosen. Using the below SAS code, three additional copies, namely, TEST1 (saved as a permanent table in the CHECK library), TEST2, and TEST3 are created (saved in the work library). Note that table TEST3 was created by dropping all the character columns from the original SASHELP.PRDSAL2 table.

```
%let workingfolder=C:\For SAS Paper\;
libname CHECK "%superq(workingfolder)";

data CHECK.test1 test2 test3 (drop=COUNTRY STATE COUNTY PRODTYPE PRODUCT MONTH);
set sashelp.prdsal2;
run;
```

A portion of original TEST2 table view is shown below in Figure 1.



VIEWTABLE: Work.Test2

	Country	State/Province	County	Actual Sales	Predicted Sales	Product Type	Product	Year	Quarter	Month	Month/Year
1	U.S.A.	California		$987.36	$692.24	FURNITURE	SOFA	1995	1	Jan	JAN95
2	U.S.A.	California		$1,782.96	$568.48	FURNITURE	SOFA	1995	1	Feb	FEB95
3	U.S.A.	California		$32.64	$16.32	FURNITURE	SOFA	1995	1	Mar	MAR95
4	U.S.A.	California		$1,825.12	$756.16	FURNITURE	SOFA	1995	2	Apr	APR95
5	U.S.A.	California		$750.72	$723.52	FURNITURE	SOFA	1995	2	May	MAY95
6	U.S.A.	California		$2,426.24	$2,428.96	FURNITURE	SOFA	1995	2	Jun	JUN95
7	U.S.A.	California		$1,791.12	$2,250.80	FURNITURE	SOFA	1995	3	Jul	JUL95
8	U.S.A.	California		$2,282.08	$350.88	FURNITURE	SOFA	1995	3	Aug	AUG95
9	U.S.A.	California		$2,518.72	$1,736.72	FURNITURE	SOFA	1995	3	Sep	SEP95
10	U.S.A.	California		$1,436.16	$2,167.84	FURNITURE	SOFA	1995	4	Oct	OCT95
11	U.S.A.	California		$2,314.72	$62.56	FURNITURE	SOFA	1995	4	Nov	NOV95
12	U.S.A.	California		$1,410.32	$1,670.08	FURNITURE	SOFA	1995	4	Dec	DEC95

Figure 1. A portion of original WORK.TEST2 table view

MACRO TEST RUN & OUTPUT

```
/*calling in the SAS file containing "Modify_All_Character_Variables" macro*/
%include "%superq(workingfolder)\Modify_All_Character_Variables.sas";

%let datasets=CHECK.test1 test2 test3; /*separate multiple dataset names using space
character*/
%let function_name=lowcase; /*must be a valid SAS character function*/

%Modify_All_Character_Variables;
```

With the help of an INCLUDE statement, the entire code (described in Parts 1-4) for the "Modify_All_Character_Variables" macro is imported into the current SAS session. Then, using the "datasets" parameter, all three tables created from the "SAMPLE TEST DATASETS" step are submitted for processing. The character function used in this example is LOWCASE. Execution of the above four lines of code will result in two updated tables. The following messages are written to the log:

```
7
8
9     /*calling in the sas file containing "Modify_All_Character_Variables" macro*/
10    %include "%superq(workingfolder)\Modify_All_Character_Variables.sas";
64
65    %let datasets=CHECK.test1 test2 test3; /*separate multiple dataset names using space
65 !  character*/
66    %let function_name=lowcase; /*must be a valid sas function*/
67
68    %Modify_All_Character_Variables;

Applying "lowcase" function to the dataset: CHECK.test1
"lowcase" converted variables: COUNTRY, COUNTY, PRODTYPE, PRODUCT, STATE


Applying "lowcase" function to the dataset: test2
"lowcase" converted variables: COUNTRY, COUNTY, PRODTYPE, PRODUCT, STATE


Applying "lowcase" function to the dataset: test3
!!! There are no charater variables in this dataset !!!



!!! Done. Thanks for using "Modify_All_Character_Variables" macro !!!
```

Display 1. Messages displayed in the log window while processing three sample test datasets

As noted from the log messages, table TEST1 from the permanent SAS library CHECK and table TEST3 from the work library are processed successfully. However, table TEST3 was not updated as there are no character variables in this dataset. The table below depicts the updated changes for the TEST2 table columns (please note that "Month" and "Month/Year" variables are not altered as they are numeric).

VIEWTABLE: Work.Test2

	Country	State/Province	County	Actual Sales	Predicted Sales	Product Type	Product	Year	Quarter	Month	Month/Year
1	u.s.a.	california		$987.36	$692.24	furniture	sofa	1995	1	Jan	JAN95
2	u.s.a.	california		$1,782.96	$568.48	furniture	sofa	1995	1	Feb	FEB95
3	u.s.a.	california		$32.64	$16.32	furniture	sofa	1995	1	Mar	MAR95
4	u.s.a.	california		$1,825.12	$756.16	furniture	sofa	1995	2	Apr	APR95
5	u.s.a.	california		$750.72	$723.52	furniture	sofa	1995	2	May	MAY95
6	u.s.a.	california		$2,426.24	$2,428.96	furniture	sofa	1995	2	Jun	JUN95
7	u.s.a.	california		$1,791.12	$2,250.80	furniture	sofa	1995	3	Jul	JUL95
8	u.s.a.	california		$2,282.08	$350.88	furniture	sofa	1995	3	Aug	AUG95
9	u.s.a.	california		$2,518.72	$1,736.72	furniture	sofa	1995	3	Sep	SEP95
10	u.s.a.	california		$1,436.16	$2,167.84	furniture	sofa	1995	4	Oct	OCT95
11	u.s.a.	california		$2,314.72	$62.56	furniture	sofa	1995	4	Nov	NOV95
12	u.s.a.	california		$1,410.32	$1,670.08	furniture	sofa	1995	4	Dec	DEC95

Figure 2. A portion of TEST2 table view after processing

ERROR REPORTING

As described in the "Additional Features" section and Part1 discussion, prior to executing the main code, the macro

checks whether or not the user has defined both “datasets” and “function_name”. It will also check whether or not the user-defined character function is suitable for running the macro. Sets of sample SAS codes below demonstrate these features:

CASE 1

The user has not defined the “function_name” parameter required for macro.

```
69
70
71    %let datasets=test2;
72    %let function_name=; /*function name is not defined*/
73
74    %Modify_All_Character_Variables;

!!! Not enough parameters defined to run this macro !!!
```

Display 2. Error message displayed in the log window when user has not defined “function_name” parameter

CASE 2

The user has not defined the “datasets” parameter required for the macro.

```
75
76
77    %let datasets=; /*input table name is missing*/
78    %let function_name=upcase;
79
80    %Modify_All_Character_Variables;

!!! Not enough parameters defined to run this macro !!!
```

Display 3. Error message displayed in the log window when user has not defined “datasets” parameter

CASE 3

The user has provided a numeric function that is not valid.

```
81
82    %let datasets=test2;
83    %let function_name=mean; /*a numeric function*/
84
85    %Modify_All_Character_Variables;
ERROR: Argument 1 to function MEAN referenced by the %SYSFUNC or %QSYSFUNC macro function is
       not a number.
ERROR: Invalid arguments detected in %SYSCALL, %SYSFUNC, or %QSYSFUNC argument list.  Execution
       of %SYSCALL statement or %SYSFUNC or %QSYSFUNC function reference is terminated.

!!! Please provide valid SAS character function name !!!
```

Display 4. Error message displayed in the log window when user has defined non-valid “function_name” input

Similar to the above example results, other tests indicated that updated values for the character variables were accurate and did not alter the incoming table attributes. As with other macros, prior to running the code, the user needs to carefully consider the implications of applying a SAS function across all character variables. Currently, this macro can handle quite a few of the SAS character functions which modify the attributes of text variables but do not return a value after execution. These functions include UPCASE, LOWCASE, PROPCASE, COMPBL, STRIP, TRIM, SUBPAD, LEFT, RIGHT, REVERSE, QUOTE, and DEQUOTE. Care must be taken while using the SAS functions QUOTE and DEQUOTE, as the result can change the length of the values by two characters. Nesting of SAS functions is not allowed; instead, the user can run this macro multiple times.

CONCLUSION

The macro described in this paper can provide a solution for harmonizing all the character variables in the input SAS tables for a given SAS function. It is indeed a feasible approach and has worked successfully in many real-life

situations. The most utility out of this macro is found when the UPCASE, LEFT and TRIM functions are applied across multiple datasets prior to performing SQL queries. This macro can also serve as a quick way to fix the data-entry errors in SAS tables.

Due to the simplicity of the code, either the entire code or portions of it (as described in the main text) can be easily incorporated into other user-written SAS routines. There is much value for the code (Part1) that checks user input parameters and it can be executed independently. Similarly, the remaining parts of the code can also be modified as per user-specific requirements.

Finally, the SAS code used in this paper demonstrates one working example of how macro variables can be skillfully applied and managed within macros.

ACKNOWLEDGMENTS

I want to extend my sincere thanks to Syngenta for its commitment toward employee learning and development. I express my heartfelt gratitude to my supervisor, Todd L. Warner, Genetic Project Lead, Stanton, MN for his continuous support, encouragement and guidance. Many thanks to Kari Kust, my colleague, for her invaluable word-smith help during preparation of this manuscript.

CONTACT INFORMATION

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

Suresh Babu Kadaru
B.Sc. (Ag), M.Sc. Biotech, Ph.D Agronomy (minor - Applied Statistics)
Scientist I
Syngenta Seeds, Inc.,
317 330th Street
Stanton, MN 55018

Phone: 225 202 0409
Email: suresh.kadaru@syngenta.com, suresh_kadaru@yahoo.com