South Central SAS® Users Group 2016

A Time Saver for All: A SAS® Toolbox

Philip Jou, Baylor University, Waco, TX

ABSTRACT

SAS® Macros are a useful tool to any SAS® Programmer. A macro variable can be used to assign a value to a variable that can be called repeatedly. A SAS® Macro Definition serves a similar purpose but is used to repeat a set of instructions. This handy toolbox contains SAS® Macro Functions that will allow a user to check the existence of a variable, get the number of observations in a dataset, and truncate the values of a dataset to name a few. This toolbox is useful to any flavor of SAS® Programmer in any profession.

INTRODUCTION

The SAS® macro facility has been a power tool implemented. There have been many papers written about the fundamentals of the macro language. A google search will yield links to SAS® support pages as well as papers from SAS® Global Forum (formerly SAS® Users Group International or SUGI).

MACRO FACILTY UNDERSTANDING

How to Define, Compile and Call a Macro Definition: A macro definition is defined by encapsulating code between a %MACRO and %MEND statement. After the %MACRO statement, a name must be given to the macro. Though not required, it is recommend to restate the name after the %MEND statement.

```
%macro example;
      /*code goes here*/
%mend example;
```

To compile the macro, highlight all of the include (including the %MACRO and %MEND statements) and execute. If changes are made to the macro definition, SAS® must re-compile the macro in order for the changes to take effect.

A macro definition is called by preceding the macro name with a '%'. In the above example, the macro would be called by executing '**%example**'. Note that a semicolon is not needed as the compiler will execute the code within the macro definition.

Global vs Local Macro Variable: Base SAS® distinguishes the difference primarily by where the macro variable is created.

Global Macro Variable: Within a Macro Definition, declaring the macro variable with %GLOBAL will create a macro variable which can be used at any time within the session.

Local Macro Variable: If a macro variable is defined within a Macro Definition, by default it is considered a Local Variable. A user can also specify this by using the %LOCAL statement. These variables are only available during the execution of the macro.

If a macro definition is called within another macro definition (known as nesting), be sure to note Macro Variable names that are used in each macro definition. If the Macro Variable is modified, it will be modified from the top-level through the last macro definition that utilizes that variable.

To call a Local or Global macro variable, just lead the variable name with an '&'. Macro variables can be combined to form complex variable naming such as enumeration of variables.

Pre-Defined Macro Functions: As a part of the macro facility, SAS® provides a library of macro functions that can be used. These functions are executed in the same manner as executing a macro definition. Many of the statement used within a DATA STEP have a macro function equivalent. For example %LENGTH ⇔ LENGTH, %SCAN ⇔ SCAN, %UPCASE ⇔ UPCASE.

Also built-in are conditional statements such as %IF…%THEN, %DO %UNTIL, and %DO %WHILE.

MACROS INCLUDED IN THIS TOOLBOX

CONVERT_ALLVAR_NUM_TO_CHAR

This macro will convert all numerical variables to character variables. The user submits a space-delimited list of datasets and has optional capability to IGNORE variables and attach a SUFFIX to the modified dataset.

The macro will process each dataset submitted by utilizing the output dataset from PROC CONTENTS. After generating the PROC CONTENTS dataset, a PROC SQL into statement is used to store a series of statements to convert the necessary variables from numeric to character (see SAS NOTE 24590) into a macro variables. Data steps are then used to call the created macro variables.

Data Step Code to create Source Dataset:

```
data DSET_CONVERT_NUM_TO_CHAR_ORIG;
            DSET_CONVERT_NUM_TO_CHAR1 DSET_CONVERT_NUM_TO_CHAR2
            DSET_CONVERT_NUM_TO_CHAR3 DSET_CONVERT_NUM_TO_CHAR4;
      input ID col1 $ col2 $ col3 col4;
      datalines;
      1 1 1 1 1
      ;
run;
```

VIEWTABLE: Work.Dset_convert_num_to_char_orig

	ID	col1	col2	col3	col4
1	1	1	1	1	1

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Num	8

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR_ORIG

The data table 'DSET_CONVERT_NUM_TO_CHAR_ORIG' and subsequent tables are created with 3 numeric columns (ID, col3, and col4).

```
/*convert all numeric variables*/
%CONVERT_ALLVAR_NUM_TO_CHAR(DSET_CONVERT_NUM_TO_CHAR1);
```

The Proc Contents on the dataset 'DSET_CONVERT_NUM_TO_CHAR1' shows that all numeric variables have been converted to character variables.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Char	12
2	col1	Char	8
3	col2	Char	8
4	col3	Char	12
5	col4	Char	12

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR1 AFTER MACRO CALL

```
/*convert all numeric variables except ID*/
%CONVERT_ALLVAR_NUM_TO_CHAR(DSET_CONVERT_NUM_TO_CHAR2,IGNORE=ID);
```

The Proc Contents on the dataset 'DSET_CONVERT_NUM_TO_CHAR2' shows that all numeric variables except for ID have been converted to character variables.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Char	12
5	col4	Char	12

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR2 AFTER MACRO CALL

```
/*convert all numeric variables except ID & COL3 and create copy of dataset
with SUFFIX 'NEW'*/
%CONVERT_ALLVAR_NUM_TO_CHAR(DSET_CONVERT_NUM_TO_CHAR3,IGNORE=ID
COL3,SUFFIX=NEW);
```

The PROC CONTENTS of the dataset 'DSET_CONVERT_NUM_TO_CHAR3_NEW' shows that all numeric variables except for ID and COL3. The original dataset 'DSET_CONVERT_NUM_TO_CHAR3' is left untouched, as shown.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Num	8

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR3 AFTER MACRO CALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Char	12

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR3_NEW AFTER MACRO CALL

```
/*process multiple datasets and convert all numeric variables except ID,
COL3, & COL8 and create copy of datasets with SUFFIX 'NEW'*/
%CONVERT_ALLVAR_NUM_TO_CHAR(DSET_CONVERT_NUM_TO_CHAR4
DSET_CONVERT_NUM_TO_CHAR5,IGNORE=ID COL3 COL8,SUFFIX=NEW);
```

This example shows how to submit a macro call for multiple datasets. Note that variable ID exists in all datasets, COL3 only exists in 'DSET_CONVERT_NUM_TO_CHAR4' and COL8 only exists in 'DSET_CONVERT_NUM_TO_CHAR5'. The macro will still process as expected.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Char	12

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR4_NEW AFTER MACRO CALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col5	Char	8
3	col6	Char	8
4	col7	Char	12
5	col8	Num	8

CONTENTS OF DSET_CONVERT_NUM_TO_CHAR5_NEW AFTER MACRO CALL

DELETE EMPTY COLUMNS

This Macro will delete any column from the dataset that contains all missing values. The user submits a space-delimited list of datasets and has optional capability to IGNORE specific variables or variable type and attach a SUFFIX to the modified dataset.

The macro will process through each dataset submitted one at a time. Each observation is processed column by column and if the sum of the missing values is calculated. If the sum of the missing values is equal to the number of observations within the dataset, the column (aka variable) is dropped from the final dataset.

Data Step code to generate sample dataset:

```
data two_empty_columns;
      input ID col1 $ col2 $ col3 col4;
```

```
      datalines;
      1 1 . 1 .
      2 2 . 2 .
      3 3 . 3 .
      4 4 . 4 .
      ;
run;
```

VIEWTABLE: Work.Two_empty_columns

	ID	col1	col2	col3	col4
1	1	1		1	.
2	2	2		2	.
3	3	3		3	.
4	4	4		4	.

The data table 'TWO_EMPTY_COLUMNS' is created with two empty columns. COL2 is a character variable and COL4 is a numeric variable.

```
/*remove all empty columns and create dataset with SUFFIX 'DEL'*/
%delete_emtpy_columns(two_empty_columns,SUFFIX=DEL);
```

VIEWTABLE: Work.Two_empty_columns_del

	ID	col1	col3
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

After submitting the macro statement, the two columns with all missing variables is removed. The new dataset 'TWO_EMPTY_COLUMNS_DEL' now contains only 3 columns.

```
/*remove all empty columns except COL2 and create dataset with SUFFIX
'DEL_IGNORE'*/
%delete_emtpy_columns(two_empty_columns,IGNORE=COL2,SUFFIX=DEL_IGNORE);
```

VIEWTABLE: Work.Two_empty_columns_del_ignore

	ID	col1	col2	col3
1	1	1		1
2	2	2		2
3	3	3		3
4	4	4		4

After submitting the macro statement, COL4 is removed as the user chose to IGNORE COL2. The new dataset 'TWO_EMPTY_COLUMNS_DEL_IGNORE' now contains only 4 columns.

```
/*remove all empty columns except numeric columns and create dataset with
SUFFIX 'DEL_CHAR'*/
%delete_emtpy_columns(two_empty_columns,CHARONLY=Y,SUFFIX=DEL_CHAR);
```

VIEWTABLE: Work.Two_empty_columns_del_char

	ID	col1	col3	col4
1	1	1	1	.
2	2	2	2	.
3	3	3	3	.
4	4	4	4	.

After submitting the macro statement, COL2 is removed as the user chose to only process character variables. The new dataset 'TWO_EMPTY_COLUMNS_DEL_CHAR' now contains only 4 columns.

```
/*remove all empty columns except character columns and create dataset with
SUFFIX 'DEL_NUM'*/
%delete_emtpy_columns(two_empty_columns,NUMONLY=Y,SUFFIX=DEL_NUM);
```

VIEWTABLE: Work.Two_empty_columns_del_num

	ID	col1	col2	col3
1	1	1		1
2	2	2		2
3	3	3		3
4	4	4		4

After submitting the macro statement, COL4 is removed as the user chose to only process numeric variables. The new dataset 'TWO_EMPTY_COLUMNS_DEL_NUM' now contains only 4 columns.

DELETE_MACRO_VARIABLES

This macro will delete macro variables from the space-delimited list submitted in the current SAS session if they exist. The macro utilizes the %symexist and %symdel macro functions to determine if a macro exists, and delete the macro if it does exist.

Code to create macro variables (Note: these macros are created with a global scope)

```
%let testMacro1=1;
%let testMacro2=2;
%let testMacro3=3;
```

Code to create the dataset from SASHELP.VMACRO (Note: the sashelp.vmacro table will automatically uppercase the name of the macro variable)

```
data vmacro;
      set sashelp.vmacro;
      where upcase(name) in ('TESTMACRO1' 'TESTMACRO2' 'TESTMACRO3');
run;
```

VIEWTABLE: Work.Vmacro

	scope	name	offset	value
1	GLOBAL	TESTMACRO1	0	1
2	GLOBAL	TESTMACRO2	0	2
3	GLOBAL	TESTMACRO3	0	3

The dataset 'VMACRO' created from DICTIONARY.MACROS (aka SASHELP.VMACRO) shows the newly created macros.

```
/*delete macro variables testMacro1 and testMacro3 and create data set to
show changes*/
%delete_macro_variables(testMacro1 testMacro3);
data vmacro_del;
      set sashelp.vmacro;
```

```
      where upcase(name) in ('TESTMACRO1' 'TESTMACRO2' 'TESTMACRO3');
run;
```

VIEWTABLE: Work.Vmacro_del

	scope	name	offset	value
1	GLOBAL	TESTMACRO2	0	2

The dataset 'VMACRO_DEL' created from DICTIONARY.MACROS (aka SASHELP.VMACRO) shows only one macro as the other two were deleted.

DIR_EXIST

This macro will determine if a File Directory exists under a Windows Operating System. Using the functions 'filename', 'fileref' and 'fexist', the macro will return 1 if the directory exists and 0 if the directory does not exist.

Example Code and Log Output

```
%put Does the Directory Exist: 1=Yes, 0=No;
%put Does the Directory 'C:\WINDOWS\SYSTEM' Exist:
%DIR_EXIST(C:\WINDOWS\SYSTEM);
%put Does the Directory 'C:\NOT_REAL_DIRCTORY' Exist:
%DIR_EXIST(C:\NOT_REAL_DIRCTORY);
```

```
Does the Directory Exist: 1=Yes, 0=No
Does the Directory 'C:\WINDOWS\SYSTEM' Exist: 1
Does the Directory 'C:\NOT_REAL_DIRCTORY' Exist: 0
```

DSET_LENGTH_MAX

This macro calls DSET_VALUELENGTH_MAX and DSET_VARLENGTH_MAX in the order stated. The options that exist for this macro are the same as the macros called.

When calling the macro, the user must provide a space-delimited list of datasets. If the dataset is in the WORK library, it is not necessary to submit the dataset as WORK.<DATASET> but it is suggested to avoid confusion if datasets from other libraries are submitted.

The optional parameters a user can enter are a space-delimited list of variables to IGNORE, setting the system to only process character or numeric variables, and add a SUFFIX to the modified dataset.

DSET_VALUELENGTH_MAX

This macro will process through each variable within the dataset and apply the maximum necessary length to store the variable. The default option of this macro is to only process character variables.

The macro will first check that the dataset(s) submitted exists and if non-existing dataset(s) are submitted will remove that dataset from the list and print a warning to the log. The revised list of dataset(s) are processed one at a time. The order of the variables is maintained.

For each dataset, the macro will use PROC CONTENTS to generate the properties of the dataset. From there, macro variables are created using PROC SQL into that will find the maximum length

necessary to store each variable. Once the maximum length is discovered, the length, format and informat statement are submitted to apply the new properties.

DSET_VARLENGTH_MAX

This macro will apply the maximum length of matched variables from the comparison of datasets. The default option of this macro is to only process character variables. Overall, this macro functions similarly to 'DSET_VALUELENGTH_MAX' except it requires at least two datasets be submitted and there is a matched variable between the submitted datasets.

This macro is especially useful prior to performing processes similar to 'PROC COMPARE' or 'DATA MERGE'. Some of these processes require that similar variables have the same length or will print a warning that truncation may occur. Using this macro removes any errors or warnings that may occur.

Example code for DSET_LENGTH_MAX, DSET_VALUELENGTH_MAX and DSET_VARLENGTHMAX

```
data TEST_DSET_1;
      input ID col1 $ col2 $ col3 col4;
      datalines;
      1 1 1 1 1
      2 2 . 2 2
      3 3 3 3 .
      4 4 . 4 .
      ;
run;
data TEST_DSET_2;
      input ID col1 $ col2 $ col3 col4;
      datalines;
      1 111 1 11 1
      2 2 . 2 2
      3 33 3 33 .
      4 4 . 4 .
      ;
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Num	8

CONTENTS OF TEST_DSET_1

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	8
2	col1	Char	8
3	col2	Char	8
4	col3	Num	8
5	col4	Num	8

CONTENTS OF TEST_DSET_2

```
%DSET_VALUELENGTH_MAX(TEST_DSET_1 TEST_DSET_2,SUFFIX=VALUE_DEFAULT);
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	ID	Num	8		
2	col1	Char	1	$1.	$1.
3	col2	Char	1	$1.	$1.
4	col3	Num	8		
5	col4	Num	8		

CONTENTS OF TEST_DSET_1_VALUE_DEFAULT

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	ID	Num	8		
2	col1	Char	3	$3.	$3.
3	col2	Char	1	$1.	$1.
4	col3	Num	8		
5	col4	Num	8		

CONTENTS OF TEST_DSET_2_VALUE_DEFAULT

```
%DSET_VALUELENGTH_MAX(TEST_DSET_1
TEST_DSET_2,CHARONLY=Y,SUFFIX=VALUE_CHARONLY);
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	ID	Num	3		
2	col1	Char	1	$1.	$1.
3	col2	Char	1	$1.	$1.
4	col3	Num	3		
5	col4	Num	3		

CONTENTS OF TEST_DSET_1_VALUE_ALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	ID	Num	3		
2	col1	Char	3	$3.	$3.
3	col2	Char	1	$1.	$1.
4	col3	Num	3		
5	col4	Num	3		

CONTENTS OF TEST_DSET_2_VALUE_ALL

```
%DSET_VALUELENGTH_MAX(TEST_DSET_1
TEST_DSET_2,CHARONLY=N,NUMONLY=Y,SUFFIX=VALUE_NUMONLY);
/*Notice that CHARONLY=N since the default setting within the macro function
is CHARONLY=Y*/
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	3
2	col1	Char	8
3	col2	Char	8
4	col3	Num	3
5	col4	Num	3

CONTENTS OF TEST_DSET_1_VALUE_NUMONLY

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	ID	Num	3
2	col1	Char	8
3	col2	Char	8
4	col3	Num	3
5	col4	Num	3

CONTENTS OF TEST_DSET_2_VALUE_NUMONLY

```
%DSET_VARLENGTH_MAX(TEST_DSET_1 TEST_DSET_2,SUFFIX=VAR_DEFAULT);
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	8	$8.	$8.
3	COL2	Char	8	$8.	$8.
4	COL3	Num	8		
5	COL4	Num	8		
1	ID	Num	8		

CONTENTS OF TEST_DSET_1_VAR_DEFAULT

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	8	$8.	$8.
3	COL2	Char	8	$8.	$8.
4	COL3	Num	8		
5	COL4	Num	8		
1	ID	Num	8		

CONTENTS OF TEST_DSET_2_VAR_DEFAULT

```
%DSET_VARLENGTH_MAX(TEST_DSET_1
TEST_DSET_2,CHARONLY=N,NUMONLY=N,SUFFIX=VAR_ALL);
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	8	$8.	$8.
3	COL2	Char	8	$8.	$8.
4	COL3	Num	3		
5	COL4	Num	3		
1	ID	Num	3		

CONTENTS OF TEST_DSET_1_VAR_ALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	8	$8.	$8.
3	COL2	Char	8	$8.	$8.
4	COL3	Num	3		
5	COL4	Num	3		
1	ID	Num	3		

CONTENTS OF TEST_DSET_2_VAR_ALL

```
%DSET_VARLENGTH_MAX(TEST_DSET_1
TEST_DSET_2,CHARONLY=N,NUMONLY=Y,SUFFIX=VAR_NUMONLY);
/*Notice that CHARONLY=N since the default setting within the macro function
is CHARONLY=Y*/
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	COL1	Char	8
3	COL2	Char	8
4	COL3	Num	3
5	COL4	Num	3
1	ID	Num	3

CONTENTS OF TEST_DSET_1_VAR_NUMONLY

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	COL1	Char	8
3	COL2	Char	8
4	COL3	Num	3
5	COL4	Num	3
1	ID	Num	3

CONTENTS OF TEST_DSET_2_VAR_NUMONLY

%***DSET_LENGTH_MAX***(TEST_DSET_1 TEST_DSET_2,SUFFIX=LENGTH_DEFAULT);

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	3	$3.	$3.
3	COL2	Char	1	$1.	$1.
4	COL3	Num	8		
5	COL4	Num	8		
1	ID	Num	8		

CONTENTS OF TEST_DSET_1_LENGTH_DEFAULT

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	3	$3.	$3.
3	COL2	Char	1	$1.	$1.
4	COL3	Num	8		
5	COL4	Num	8		
1	ID	Num	8		

CONTENTS OF TEST_DSET_2_LENGTH_DEFAULT

%***DSET_LENGTH_MAX***(TEST_DSET_1
TEST_DSET_2,CHARONLY=N,NUMONLY=N,SUFFIX=LENGTH_ALL);

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	3	$3.	$3.
3	COL2	Char	1	$1.	$1.
4	COL3	Num	3		
5	COL4	Num	3		
1	ID	Num	3		

CONTENTS OF TEST_DSET_1_LENGTH_ALL

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
2	COL1	Char	3	$3.	$3.
3	COL2	Char	1	$1.	$1.
4	COL3	Num	3		
5	COL4	Num	3		
1	ID	Num	3		

CONTENTS OF TEST_DSET_2_LENGTH_ALL

```
%DSET_LENGTH_MAX(TEST_DSET_1
TEST_DSET_2,CHARONLY=Y,NUMONLY=Y,SUFFIX=LENGTH_NUMONLY);
/*Notice that CHARONLY=N since the default setting within the macro function
is CHARONLY=Y*/
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	COL1	Char	8
3	COL2	Char	8
4	COL3	Num	3
5	COL4	Num	3
1	ID	Num	3

CONTENTS OF TEST_DSET_1_LENGTH_NUMONLY

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	COL1	Char	8
3	COL2	Char	8
4	COL3	Num	3
5	COL4	Num	3
1	ID	Num	3

CONTENTS OF TEST_DSET_2_LENGTH_NUMONLY

GET_FILEPATH

This macro returns the file path of the program the macro is run in and stores the value into a macro variable. Within the macro call, the user has the option to store the returned value in the default macro variable name 'filepath' or pass in a name.

The macro utilizes macro functions %length, %qsubstr, %quote, %sysget, and %sysfunc. It also uses SAS environment variables SAS_EXECFILEPATH and SAS_EXECFILENAME. The returned file path will be missing the ending '\'.

Example Code and Log Output

```
%get_filepath;
%put &=filepath;
%get_filepath(myMacroVar);
%put &=myMacroVar;
```

```
FILEPATH=E:\SAS Programs\Sample Code
MYMACROVAR=E:\SAS Programs\Sample Code
```

The Log output from running the example code shows the just running the macro, the value returned is stored in the macro variable 'filepath'. In the second call, the user has submitted the name 'myMacroVar' to store the value in.

NUM_OBS

This macro will return the number of observations in the submitted dataset. Using the macro language, the macro function first tests to see if the dataset exists. If the dataset exists, the function 'open', 'attrn', and 'close' are used to open the dataset, get the number of observations and close the dataset. If the dataset does not exist the macro will return 0.

Data Step Code to generate sample dataset with 10 observations

```
data dset_num_obs;
      input ID ;
      datalines;
      1
      2
      3
      4
      5
      6
      7
      8
      9
      0
      ;
run;
```

Macro call and Log Output

```
%put DSET_NUM_OBS has %num_obs(dset_num_obs) observations;
```

```
DSET_NUM_OBS has 10 observations
```

As expected, the macro function returns 10 as the number of observations

SYSTEM_OPTIONS

This macro will disable/reset and set default system options that specifically relate to printing to the log. The system options SYMBOLGEN, MPRINT, MLOGIC, MERROR, SERROR, QUOTELENMAX, SOURCE, SOURCE2, NOTES, and VARLENCHK are switched on/off dependent on the mode selected. The user calls the macro and can submit 3 modes (DEFAULT, DISABLE, RESET).

DEFAULT will set the SAS session to the default state when a SAS session is launched.
DISABLE will turn off all the system options previously mentioned
RESET will set the system options to the state found prior to using the DISABLE option

This macro utilizes the macro functions %cmpres, %let, %sysfunc, %symexist, %sysmexecdepth. It also uses the functions getoption to attain the current setting for the system option when the macro is called as well as setting the default system option value.

When MODE=DISABLE the macro will read the current state of the system options and save that list to a local macro variable. Using %SYSMEXECDEPTH the depth at which the macro is called is utilized to create a unique global macro variable that will store the current state of the system options.

When MODE=RESET the macro will test to see if there is a global macro variable at the depth of the macro call. If such a global macro variable exists, the macro will set the system options to the previous state and delete the global macro variable.

It is vital that if a call using MODE=DISABLE is made, MODE=RESET should be called at some other point at the same execution depth. Otherwise the SAS session will have the system options turned off until either MODE=DEFAULT is used or the SAS session is closed and reopened.

Example Code:

```
OPTIONS MLOGIC MPRINT SYMBOLGEN; /*Enable system options*/
%macro example_sys_options;
      %let test1=TEST1;
      %put &test1;
      %SYSTEM_OPTIONS(DISABLE);
      %let test2=TEST2;
      %put &test2;
      /*Notice that SYMBOLGEN was not printed for resolving macro variable
test2*/
      %SYSTEM_OPTIONS(RESET);
      %let test3=TEST3;
      %put &test3;
%mend example_sys_options;
%example_sys_options;
OPTIONS NOMLOGIC NOMPRINT NOSYMBOLGEN;
```

```
12   OPTIONS MLOGIC MPRINT SYMBOLGEN;

13   %example_sys_options;
MLOGIC(EXAMPLE_SYS_OPTIONS):  Beginning execution.
MLOGIC(EXAMPLE_SYS_OPTIONS):  %LET (variable name is TEST1)
MLOGIC(EXAMPLE_SYS_OPTIONS):  %PUT &test1
SYMBOLGEN:  Macro variable TEST1 resolves to TEST1
TEST1
MLOGIC(EXAMPLE_SYS_OPTIONS):  Beginning compilation of SYSTEM_OPTIONS using the autocall
      file [redacted]
MLOGIC(EXAMPLE_SYS_OPTIONS):  Ending compilation of SYSTEM_OPTIONS.
MLOGIC(SYSTEM_OPTIONS):  Beginning execution.
MLOGIC(SYSTEM_OPTIONS):  This macro was compiled from the autocall file
      [redacted]
MLOGIC(SYSTEM_OPTIONS):  Parameter MODE has value DISABLE
MLOGIC(SYSTEM_OPTIONS):  %LET (variable name is OPTIONS_LIST_VARLENGTH)
MPRINT(SYSTEM_OPTIONS):   options NOSYMBOLGEN NOMPRINT NOMLOGIC
TEST2
MPRINT(SYSTEM_OPTIONS):   MERROR SERROR QUOTELENMAX SOURCE NOSOURCE2 NOTES VARLENCHK=WARN;
MLOGIC(SYSTEM_OPTIONS):  Ending execution.
MPRINT(EXAMPLE_SYS_OPTIONS):  ;
MLOGIC(EXAMPLE_SYS_OPTIONS):  %LET (variable name is TEST3)
MLOGIC(EXAMPLE_SYS_OPTIONS):  %PUT &test3
SYMBOLGEN:  Macro variable TEST3 resolves to TEST3
TEST3
MLOGIC(EXAMPLE_SYS_OPTIONS):  Ending execution.
14   OPTIONS NOMLOGIC NOMPRINT NOSYMBOLGEN;
```

Notice that there is no 'SYMBOLGEN: …' statement as there is for TEST1 & TEST3

As stated in the example code comments, in the %put call to the macro variable 'test2' the 'SYMBOLGEN' comment is not printed.

VAR_EXIST

This macro determines if a variable exists in the submitted dataset. The user submits a dataset and variable to search for. A '1' is returned if the variable exists and '0' if it is not. This macro is based on the SASCommunity.org posting 'Tips:Check if a variable exists in a dataset'

Using the macro language, the macro function first tests to see if the dataset exists. If the dataset exists, the function 'open', 'varnum', and 'close' are used to open the dataset, get the column number of the variable and close the dataset. If the dataset does not exist, the macro will return 0.

Data Step Code to generate sample dataset

```
data VAR_EXIST_DSET;
      input ID col1 $ col2 $ col3 col4;
      datalines;
      1 1 1 1 1
      ;
run;
```

Example Code and Log Output

```
%put Does the variable exist? 1=Yes, 0=No;
%put COL2: %var_exist(var_exist_dset,col2);
%put COL5: %var_exist(var_exist_dset,col5);
%put COL2: %var_exist(var_exist_dset2,col2);/*dataset does not exist, warning
will print to log and macro will return 0*/
```

```
Does the variable exist? 1=Yes, 0=No
COL2: 1
COL5: 0
COL2: 0
```

In the first call, the log output shows that col2 exists in the submitted dataset 'var_exist_dset'
In the second call, the log output shows that col5 does not exist in the submitted dataset 'var_exist_dset'
In the third call, the log output shows that col2 does not exist in the submitted dataset 'var_exist_dset2'. Since the dataset 'var_exist_dset2' does not exist within the SAS session, a 0 is returned

Source code and additional documentation can be found here:
https://www.dropbox.com/sh/03ha1su31spmk83/AAAsDWd07aBX2uHh7r_bjF8Pa?dl=0

CONCLUSION

The SAS® Macro Facility is a useful tool to replicate values and code as needed. If used properly, it can save a user time and effort.

REFERENCES

CONTACT INFORMATION

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

Philip Jou
Baylor University
Office of Institutional Research
One Bear Place #97032 (254) 710-8340
Waco, TX 76798-7032 philip_jou@baylor.edu

REFERENCES:

SAS® 9.4 Macro Language: Reference, Fourth Edition
https://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#titlepage.htm

SAS® Sample 24590: Convert variable values from character to numeric or from numeric to character
http://support.sas.com/kb/24/590.html

SAS® Sample 24804: %SQUEEZE-ing Before Compressing Data, Redux
http://support.sas.com/kb/24/804.html

SAS® Sample 24671: Dynamically determine the number of observations and variables in a SAS® data set
http://support.sas.com/kb/24/671.html

Return value from SAS macro http://www.ryslander.com/return-value-from-sas-macro/

Art Carpenter, SGF 2008 (023-2008) The Path, The Whole Path, And Nothing But the Path, So Help Me Windows http://www2.sas.com/proceedings/forum2008/023-2008.pdf

How to determine the executing program name and path programmatically
http://studysas.blogspot.com/2009/04/how-to-determine-executing-program-name.html

SASCommunity.org: Determining the number of observations in a SAS data set efficiently
http://www.sascommunity.org/wiki/Determining_the_number_of_observations_in_a_SAS_data_set_efficiently

SASCommunity.org: Tips:Check if a variable exists in a dataset
http://www.sascommunity.org/wiki/Tips:Check_if_a_variable_exists_in_a_dataset