Macros that Emulate Macro Functions

Warren F. Kuhfeld, The University of North Carolina at Chapel Hill

ABSTRACT

Programming in the SAS® macro language, like other computer languages, is more efficient after a library of utilities that perform commonly done tasks is compiled. The macro language does not explicitly give the user the ability to write library subroutines or functions, however it is possible to write macros that can subsequently be used in other macros as if they were functions or subroutines. Such macros should contain no generic SAS code so that they may be "called" from anywhere. This poster will illustrate this capability with six macros that can be %INCLUDE'd and used as if they were macro functions. These "functions" check their arguments, which will typically be the parameters of the calling macro, to see if their value is a valid SAS name, SAS variable list, dataset name, integer, signed integer, or floating point constant, possibly expressed in "E" format. The functions resolve to 1 (true) if the argument is valid and 0 (false) otherwise. In addition, a macro function that gives the macro language a convenient way to change part of the value of a symbolic variable (like a substring pseudovariable operation) will be described.

INTRODUCTION

A useful capability and application of the macro language that has not been explicitly documented in the 1982 SAS User's Guide: Basics, is the ability to write macros that can subsequently be used in other macros as if they were a part of the macro language. This poster will illustrate this capability with seven macros that can be used as if they were macro functions. The macros contain no constant text containing generic SAS code so that they may be called from anywhere, inside or outside a macro.

The SAS macro facility gives the SAS user the ability to write flexible "canned" SAS programs which can be executed specifying options through keyword and positional parameters. A parameter passed to a macro will be a character string that will often have a value of fixed form such as a SAS variable list, dataset name, other SAS name, integer, signed integer, or floating point constant, possibly expressed in an "E" format. It would be desirable for macros that will be repeatedly used, possibly by many users, to have routines at the top that check parameters to see if they are valid. When incorrect parameters are encountered the macro could then end gracefully instead of with cryptic errors. For example, a macro function that checks a parameter to see if it is a SAS name that could be used as follows would be desirable:

```
%IF NOT %NAME(\$PARAM) %THEN %DO;
  %PUT ERROR: PARAM MUST BE A NAME.;
  %PUT VALUE SPECIFIED: \$PARAM;
  %LET ERROR = 1;
%END;
```

The macro facility does not provide such functions, but it is fairly easy to write macros that will check parameters. This poster will present macros that look like macro functions that can be used to check for all of the common forms mentioned above, and can be used like the example above. These "functions" resolve to a 1 (TRUE) if the argument is valid and 0 (FALSE) otherwise. In addition a macro called %REPLACE that looks like a macro function, and gives the macro language a convenient way to change only part of the value of a symbolic variable (since the macro language does not have an operation like the substring pseudovariable) will be described.

%NAME(\$PARAM) resolves to a 1 if the value of \$PARAM is one to eight characters in length beginning with a letter or underscore and continuing with letters, underscores and numerals. Otherwise it resolves to 0.

%DSNAME(\$PARAM) resolves to a 1 if the value of \$PARAM is a SAS name, or if it consists of two SAS names separated by a period, where the first SAS name does not begin with an underscore. Otherwise it resolves to 0.

%INTEGER(\$PARAM) resolves to a 1 if the value of \$PARAM consists of a string of one or more numerals. Otherwise it resolves to 0.

%SIGN(\$PARAM) resolves to a 1 if the value of \$PARAM begins with a minus sign, plus sign or numeral and continues with numerals. Otherwise it resolves to 0. All integers are also signed integers.

%FLOAT(\$PARAM) resolves to a 1 if the value of \$PARAM is a valid floating point number. A valid floating point value includes integers, signed integers, and any string that begins with a numeral, decimal point, plus sign, or minus sign, continues with numerals with the possibility of a decimal point, but no more than a total of one decimal point. In addition, any value consisting of a first part that meets the above definition followed by an "E", followed by a signed integer (as defined above) in the range -73 to 73 inclusive is also valid. Also, a missing value (.) is considered valid. Otherwise it resolves to 0.
VARLIST(STR) resolves to a 1 if the value of STR is a valid variable list. The value may contain SAS names, _CHARACTER_, _NUMERIC_, and lists of the form: name CHARACTER name, name NUMERIC name, name--name, and numbered name abbreviations such as VAR1-VAR10. Otherwise it resolves to 0.

REPLACE(STR1,POS,STR2) is used when one would like to have the capabilities of the substring pseudovariable operation which is not available in the macro language. For example, the statement:

%LET STR1 = REPLACE(STR1,POS,STR2);

states that the characters in STR1, starting with the character in the POS position, should be replaced with all of the characters in STR2. This function can be used to insert a character or several characters, or it can replace the last part of STR1 with a longer string resulting in a new STR1 that is longer than the original. POS must be less than or equal to LENGTH(STR1).

LIMITATIONS, NOTES, AND DISCLAIMERS

These functions, to the best of the author's knowledge, work correctly with any "reasonable" parameter value. A reasonable value consists of one or more of the following characters: letters, numerals, underscores, plus signs, minus signs, and decimal points. Lower case letters are translated to upper case and are not considered to be errors. Many strings containing special characters are also reasonable. An unreasonable string contains percent signs, ampersands, unmatched quotes, unmatched parentheses, or anything else that is "special" to the macro language. Since these functions are written in macro and hence arguments will be substituted in the code, etc., it is impossible to write macros that will handle all possible arguments. It would be desirable if these functions were written into the macro system so that these limitations could be overcome. If you find strings that are reasonable that the macros cannot handle, or if they work incorrectly, the author would appreciate notification.

External macros such as these functions, that are called by other macros, contain symbolic variables whose scope is not local. In other words, after one of these macro functions is called, the calling macro has access to all symbolic variables used by the function and the function may change any symbolic variables used in the calling macro that share a name with symbolic variables in the function. Since this is not desirable for this application, a convention to eliminate the chance of confusion was adopted. All symbolic variables in the functions begin with three characters consisting of: 1) an underscore, 2) the first letter of the function name, and 3) an underscore.

To be safe, all names of the form "underscore letter underscore name" where letter is: 'D', 'N', 'S', 'I', 'R', 'F', or 'V' should be avoided since many of the functions use other functions. For example, %DSN uses %NAME; %SINTEGER uses %SINTEG; %FLOAT uses %REPLACE; and %SINTEG; and %VARLIST uses %REPLACE, %NAME, and %SINTEG. This of course also implies that one cannot simply type up one of the functions without checking to see if it needs some of the others.

These macros do not check if the arguments are reasonable, they only check to see if they are of the proper form. For example, %DSN does not check to see if the dataset already exists, it merely checks to see if the name is a valid dataset name, %VARLIST does not see if the variables are in any dataset, %FLOAT does not check if a value is reasonable for a convergence criterion for an iterative algorithm, etc.

SUMMARY

Small "building blocks" such as the %NAME, %REPLACE, and %SINTEGER functions can be written entirely in macro, which can then be used in the macro language as if they were a built-in part of the macro language. The macro language provides us with the tools necessary to enhance and customize the language.

REFERENCES


1 Some of the work presented in this poster, previously appeared in Kuhfeld (1985).

SAS® and SAS Communications® are registered trademarks for SAS Institute Inc., Cary, NC, USA.
OPTIONS NOSOURCE; /* APPENDIX OF MACROS */

THESE MACRO'S WERE WRITTEN BY WARREN F. KIRKFIELD OF THE UNIVERSITY
OF NORTH CAROLINA, DAVID HALL - 013A, CHAPEL HILL NC 27514.

EVERY EFFORT HAS BEEN MADE TO TEST THESE MACRO'S TO MAKE SURE THAT
THEY PERFORM CORRECTLY, HOWEVER THE AUTHOR DOES NOT GUARANTEE
THAT THEY WILL PERFORM CORRECTLY. THEY ARE TO BE USED AT YOUR
OWN RISK. NOTE SOME MACROS USE PRECEDING MACROS.

MACRO NAME CHECKS TO SEE IF THE ARGUMENT IS A VALID SAS NAME.
IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

%MACRO NAME(_N_ARG);
%LET _N_ERR = 1;
%LET _N_LEN = %LENGTH(&_N_ARG);
%IF 1 LE &_N_LEN AND &_N_LEN LE 8 %THEN %DO;
%LET _N_ARG = %UPCASE(&_N_ARG);
%IF %INDEX(0123456789,%SUBSTR(&_N_ARG,1,1)) NE 0
%THEN %LET _N_ERR = 0;
%IF &_N_ERR %THEN %DO;
%DO _NI = 1 %TO &_N_LEN;
%IF %INDEX(ABCDEFGHIJKLMNOPQRSTUVWXYZOI23456789_ ,
%SUBSTR(&_N_ARG,&_NI,1)) EQ 0 %THEN %LET _N_ERR = 0;
%END;
%END;
%ELSE %LET _N_ERR = 0;
%END NAME;

MACRO DSN CHECKS TO SEE IF THE ARGUMENT IS A VALID SAS DATASET NAME. IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

%MACRO DSN(_D_ARG);
%LET _D_ERR = 1;
%IF %LENGTH(&_D_ARG) GT 0 %THEN %DO;
%LET _D_DOT = %INDEX(&_D_ARG, 1);
%IF NOT &_D_DOT %THEN %LET _D_ERR = %NAME(&_D_ARG);
%ELSE %DO;
%LET _D_DDN = %SUBSTR(&_D_ARG, 1, %EVAL(&_D_DOT - 1));
%IF NOT %INDEX(&_D_DDN, .)
%THEN %LET _D_ERR = %NAME&_D_DDN;
%ELSE %LET _D_ERR = 0;
%IF &_D_ERR
%THEN %LET _D_ERR = %NAME%SUBSTR(&_D_ARG,
%EVAL(&_D_DOT+1));
%END;
%END;
%ELSE %LET _D_ERR = 0;
&D_D_ERR
%MEND DSN;

MACRO INTEGER CHECKS TO SEE IF THE ARGUMENT IS A VALID UNSIGNED INTEGER. IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

%MACRO INTEGER(_I_ARG);
%LET _I_ERR = 1;
%LET _I_LEN = %LENGTH(&_I_ARG);
%IF &_I_LEN GT 0 %THEN %DO;
%DO _I_I = 1 %TO &_I_LEN;
%IF %INDEX(0123456789, %SUBSTR(&_I_ARG, _I_I, 1)) EQ 0
%THEN %LET _I_ERR = 0;
%END;
%END;
%ELSE %LET _I_ERR = 0;
&I_ERR
%MEND INTEGER;
MACRO SINTEGER CHECKS TO SEE IF THE ARGUMENT IS A VALID SIGNED OR UNSIGNED INTEGER. IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

MACRO SINTEGER(_S_ARG);
%LET _S_ERR = 1;
%LET _S_LEN = %LENGTH(&_S_ARG);
%IF _S_LEN GT 0 %THEN %DO;
  %IF %INDEX(+,%SUBSTR(&_S_ARG,1,1)) NE 0 AND _S_LEN GT 1 %THEN %LET _S_ARG = %SUBSTR(&_S_ARG,2);
  %IF NOT %INTEGER(&_S_ARG) %THEN %LET _S_ERR = 0;
%END;
%ELSE %LET _S_ERR = 0;
&_S_ERR%MEND SINTEGER;

MACRO REPLACE PROVIDES AN OPERATION LIKE THE SUBSTRING PSUEDO-VARIABLE IN THAT IT IS A CONVENIENT WAY TO INSERT CHARACTERS IN A STRING.

MACRO REPLACE(_R-STG,_R_POS,_R_NEW);
%LET _R_LEN = %LENGTH(&_R_STG);
%LET _R_LN = %LENGTH(&_R_NEW);
%LET _R_ERR = 0;
%IF _R_POS GT &_R_LEN %THEN %LET _R_ERR 1;
%ELSE %IF _R_POS = 1 AND _R_LN GE _R_LEN %THEN %LET _R_STG = &_R_NEW;
%ELSE %IF _R_POS = 1 %THEN %LET _R_STG = _R_NEW .%SUBSTR(&_R_STG,_R_POS+_R_LN);
%ELSE %IF %EVAL(_R_POS +&_R_LN - 1) GE &_R_LEN %THEN %LET _R_STG = %SUBSTR(_R_STG,1,_R_POS-1)%R_NEW.%SUBSTR(&_R_STG,_R_POS-&_R_LN);
%IF _R_ERR %THEN %PUT ERROR: POSITION OUT OF RANGE;
%ELSE &_R_STG%MEND REPLACE;

MACRO FLOAT CHECKS TO SEE IF THE ARGUMENT IS A VALID FLOATING POINT CONSTANT. IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

MACRO FLOAT(_F_ARG);
%LET _F_ERR = 2;
%LET _F_LEN = %LENGTH(&_F_ARG);
%LET _F_ARG = %UPCASE(&_F_ARG);
%IF &_F_LEN GT 0 AND %INDEX(&_F_ARG,E) = 0 %THEN %DO;
  %LET _F_DOT = %INDEX(&_F_ARG,.)
  %IF _F_DOT %THEN %LET _F_ARG = %REPLACE(&_F_ARG,_F_DOT,O);
  %IF NOT %SINTEGER(&_F_ARG) %THEN %LET _F_ERR = 0;
%END;
%ELSE %IF &_F_LEN GT 0 AND %INDEX(&_F_ARG,E) NE 0 %THEN %DO;
  %LET _F_DOT = %INDEX(&_F_ARG,.)
  %IF _F_DOT NE 1 AND &_F_DOT LT &_F_LEN %THEN %DO;
  %LET _F_PT1 = %SUBSTR(&_F_ARG,1,&_F_DOT-1);
  %LET _F_PT2 = %SUBSTR(&_F_ARG,&_F_DOT+1);
  %LET _F_DOT = %INDEX(&_F_PT1,.)
  %IF _F_DOT %THEN %LET _F_PT1 = %REPLACE(&_F_PT1,_F_DOT,O);
  %IF NOT %INTEGER(&_F_PT1) %THEN %LET _F_ERR = 0;
  %IF %INTEGER(&_F_PT2) %THEN %DO;
  %IF NOT (-73 LE&_F_PT2 AND &_F_PT2 LE 73) %THEN %LET _F_ERR = 0;
%END;
%ELSE %LET _F_ERR = 0;
%END;
%ELSE %LET _F_ERR = 0;
&_F_ERR%MEND FLOAT;
MACRO VARLIST CHECKS TO SEE IF THE ARGUMENT IS A VALID SAS VARIABLE LIST. IF SO IT "RETURNS" A 1, OTHERWISE IT RETURNS A 0.

%MACRO VARLIST(_V_ARG);
%LET _V_ERR = 1; %LET _V_LEN = %LENGTH(_V_ARG);
%IF &_V_LEN NE 0 %THEN %DO;
  %LET _V_ARG = %UPCASE(_V_ARG); %DO _V_I::: 1 ;;TO &_V_LEN;
  /* FOR CONVENIENCE, BLANKS ARE REPLACED WITH A $ SIGN SINCE */
  /* BLANKS ARE DIFFICULT TO DEAL WITH IN THE MACRO LANGUAGE */
  &._V_ARG = %SCAN(_V_ARG, &._V_I, $) %DO %WHILE(&._V_PTR);
  %LET _V_ARG = %REPLACE(&._V_ARG, &._V_PTR, $); %LET _V_PTR = %INDEX(&._V_ARG, $)
%END;
%IF %INDEX(%SUBSTR(&_V_ARG,1,1),-) GT 0 OR %INDEX(%SUBSTR(&_V_ARG,_V_LEN),-) GT 0 %THEN %LET _V_ERR = 0;
%ELSE %LET _V_ERR = 1;
%END;
%END;
%IF %INDEX(%SUBSTR(&_V_ARG,1,1),-) GT 0 OR %INDEX(%SUBSTR(&_V_ARG,_V_LEN),-) GT 0 %THEN %LET _V_ERR = 0;
%END

%MACRO VARLIST(_V_ARG);
%LET _V_ERR = 1; %LET _V_LEN = %LENGTH(_V_ARG);
%IF &_V_LEN NE 0 %THEN %DO;
  %LET _V_ARG = %UPCASE(_V_ARG); %DO _V_I::: 1 ;;TO &_V_LEN;
  /* FOR CONVENIENCE, BLANKS ARE REPLACED WITH A $ SIGN SINCE */
  /* BLANKS ARE DIFFICULT TO DEAL WITH IN THE MACRO LANGUAGE */
  &._V_ARG = %SCAN(_V_ARG, &._V_I, $) %DO %WHILE(&._V_PTR);
  %LET _V_ARG = %REPLACE(_V_ARG, &._V_PTR, $); %LET _V_PTR = %INDEX(_V_ARG, $)
%END;
%IF %INDEX(%SUBSTR(&_V_ARG,1,1),-) GT 0 OR %INDEX(%SUBSTR(&_V_ARG,_V_LEN),-) GT 0 %THEN %LET _V_ERR = 0;
%ELSE %LET _V_ERR = 1;
%END;
%END;
%END
%ELSE %IF %INDEX(_V_WORD,-) %THEN %DO;
%IF %NAME(%SCAN(_V_WORD,1,-)) AND %NAME(%SCAN(_V_WORD,2,-)) %THEN %DO;
%LET _V_WORD1 = %SCAN(_V_WORD,1,-);
%LET _V_WORD2 = %SCAN(_V_WORD,2,-);
%LET _V_J = 1;
%LET _V_S1 = %SUBSTR(_V_WORD1,%LENGTH(_V_WORD1));
%LET _V_S2 = %SUBSTR(_V_WORD2,%LENGTH(_V_WORD2));
%LET _V_COND = %EVAL(%INTEGER(_V_S1)
OR %INTEGER(_V_S2));
%DO %WHILE(_V_COND);
%LET _V_J = %EVAL(_V_J+1);
%LET _V_S1 = %SUBSTR(_V_WORD1,_V_J);
%LET _V_S2 = %SUBSTR(_V_WORD2,_V_J);
%LET _V_COND = %EVAL(%NOT(%INTEGER(_V_S1)
OR %INTEGER(_V_S2)));
%END; /* DO WHILE */
%IF _V_J NE 1 %THEN %DO;
%LET _V_NUM1 = %SUBSTR(_V_WORD1,_V_J);
%LET _V_NUM2 = %SUBSTR(_V_WORD2,_V_J);
%LET _V_ZERO = (%SUBSTR(_V_NUM1,1,1) EQ 0)
+ (%SUBSTR(_V_NUM2,1,1) EQ 0);
%LET _V_S1 = %SUBSTR(_V_WORD1,1,_V_J-1);
%LET _V_S2 = %SUBSTR(_V_WORD2,1,_V_J-1);
%IF (_V_NUM1 GT _V_NUM2) OR (_V_ZERO EQ 1) OR (_V_S1 NE _V_S2) %THEN %LET _V_ERR = 0;
%END; /* IF _V_J */
%ELSE %LET _V_ERR = 0;
%END; /* IF NAME */
%ELSE %LET _V_ERR = 0;
%END; /* ELSE IF NAME */
%ELSE %LET _V_ERR = 0;
%END; /* ELSE IF INDEX */
%IF _V_ERR = 0 %THEN %DO;
%LET _V_ARG = %EVAL(_V_I+1);
%LET _V_END = %SCAN(_V_ARG, _V_I,xcf);
%END; /* IF NO ERROR THEN DO */
%V_ERR
%MEND VARLIST;
OPTIONS SOURCE;
773