

Need More "Functionality"? Using the SAS Macro Language to Create User-Written Functions

Pete Lund, Washington State Caseload Forecast Council, Olympia, Washington

ABSTRACT

The data step language component of the SAS System has always been rich in the variety functions available to the programmer. There are currently close to 200 functions available in the data step. In fact, the SAS Language Guide lists 14 different categories of functions. Even with all this at your fingertips, there are times when you can't find one to do just what you need.

In many cases, the SAS macro language allows the wily programmer to overcome these occasional deficiencies and write a macro that looks and acts like a function.

This paper demonstrates the use of the macro language to create a number of user-written functions. In addition, the use of macro autocall libraries and stored, compiled macros are also explored as a means for managing a growing library of user-written functions.

INTRODUCTION

If SAS does not provide you with a function that you want or need there are at least 3 ways to handle the situation:

- *A workaround in code.* You can almost always write the code you need in your dataset. This can be cumbersome if the same code needs to be repeated numerous times for different variables or in different datasets.
- *Write your function in C, PowerBuilder or another language which can write DLL's.* These functions can be referenced with MODULEN or MODULEC in a dataset. This is beyond the scope of this talk. See the SAS Language Guide or on-line documentation for more details on referencing external functions.
- *Write a SAS macro.* The SAS macro language is a powerful tool for generating SAS code. If the function you want could be written in SAS it may be possible to generate that code with the

SAS macro language -- preventing you from typing and retying the code.

This paper will focus on using macros to generate function-like statements by examining a number of working examples.

SIMPLE STRING HANDLING FUNCTIONS

The SAS language provides a SUBSTR function for snipping a piece of a character variable. It's syntax is similar to the MID function in many other programming languages, specifying the variable name, start position and length of the sub-string. Languages which use a MID function also have a LEFT and a RIGHT function, to extract the leftmost and rightmost characters, respectively.

We could create our own MID and LEFT functions by simply wrapping a SUBSTR function in a macro:

```
%macro mid(varname,start,len);  
    substr(&varname,&start,&len)  
%mend mid;  
  
%macro left(varname,len);  
    substr(&varname,1,&len)  
%mend mid;
```

These could be used in a dataset as any other function would be used. For example, if BIGVAR = ABC123, the following

```
leftvar = %left(bigvar,3);  
midvar = %mid(bigvar,2,4);
```

would set LEFTVAR to ABC and MIDVAR to BC12. These two functions do not provide too much savings in time or typing over the SUBSTR function.

A RIGHT function, however, is often very handy and will save a bit of coding. Say, for instance, that you have a variable that can be variable length and contains a status code as the last 2 characters. The following function can be called to extract those characters:

```
%macro right(vn,len);
  reverse(substr(reverse(&vn),1,&len))
%mend mid;
```

Calls with the following values would yield these values:

```
status = %right(CatID,2);
```

Initial Value (CatID)	Extracted Result (Status)
B2DN	DN
A123AC	AC
F3CL	CL

A Quick Note on Syntax and Macro Processing

Have you noticed that there are no semicolons on the statements inside the macros above? In this case it's an important, and necessary, omission. Simply put, as the macro processor is parsing the statements inside the macro it is determining, on a "token" by "token" basis, whether to send to a token to the macro compiler or the SAS language compiler. In most cases, tokens from statements that begin with a % are sent to the macro compiler, others are sent to the language compiler. In the function below, the SUBSTR function is sent to the language compiler.

```
%macro left(varname,len);
  substr(&varname,1,&len)
%mend mid;
```

If this function were used in a datastep statement,

```
if %left(x,3) eq 'ABC' then y = 1;
```

the macro would generate the following statement:

```
if substr(x,1,3) eq 'ABC' then y = 1;
```

If the macro included a semicolon,

```
%macro left(varname,len);
  substr(&varname,1,&len);
%mend mid;
```

the same datastep statement would resolve as:

```
if substr(x,1,3); eq 'ABC' then y= 1;
```

Obviously, the semicolon following the (x,1,3) is not what was intended and the code segment eq 'ABC' would generate an error.

A DATE CONVERSION FUNCTION

I currently work with a database in which service months are stored as an integer, in CCYYMM format. For example, June 1998 would be stored as 199806. SAS does not provide functions or formats to directly convert these values to SAS dates. As with the character functions above it is a simple, though not quite as straightforward, process to build a function to do the conversion. We will create a SAS date equivalent to the first day of the month of the CCYYMM integer.

By concatenating a '01' to the end of the integer value (19980601, for example) we can use the YYMMDD8. informat to convert the value to a SAS date. Wrapping this in a macro yields the following function:

```
%macro ccyyymm(vn);
  input(put(&vn,6.)||'01',yyymmdd8.)
%mend;
```

Assuming the variable ccdat contained the value 199806, the following call,

```
thedata = %ccyyymm(ccdat);
```

would result in the value 14031 (6/1/98) being set to thedate.

USING THE MACRO LANGUAGE

The functions we've seen up to this point have simply wrapped a macro around a SAS code segment and replaced a variable name or parameter. We'll now look at an example of using the macro language to build the SAS statement for us.

The Problem

I worked once on a project, which required extensive analysis of medical procedure codes. These codes are 4-character values, usually composed of numbers, and we were often interested in groups of ranges of values. For example, we may have wanted to select 4201, 4203, all the 4300's and the 4520's, 4530's and 4540's. The SAS IN function comes to mind, but those 4300's cause a bit of a problem. The colon ":" operator can restrict comparison to a portion of a string, but IN does not support and operators inside the value list. (Can you see where we're going with this?)

Applications and Publishing

The Solution

The solution was to create a function, using the macro language, which had a similar look-and-feel to the IN function, but allowed operators, like the colon operator, and embedded functions, like SUBSTR.

What the macro does is to generate a set of variable eq value OR code segments which can be placed in an IF...THEN or WHERE statement. For example, suppose we were interested in extracting the procedure values listed above from variable CPT. What the macro will generate is:

```
CPT eq '4201' or CPT eq '4203' or
CPT eq ':43' or CPT eq ':452' or
CPT eq ':453' or CPT eq ':454'
```

The Call

The syntax of the function call is as follows:

```
%in(varname,value1,<value2>,...,<valuen>)
```

where varname is the name of the variable to search and value1 - valuen are the values to check. The variable name and at least one value are required. Commas separate the parameters.

The call to the macro, in the example above, would be as follows:

```
if %in(CPT,'4201','4203':43',
      ':452',:453',:454');
```

The Code

The macro code itself is found in Appendix A of this paper and is actually quite simple. Each line of the macro will be discussed here, in light of the example above.

❶ The macro statement contains the PARMBUFF option. This option causes the entire parameter list to be passed as one string, placed in a macro variable called SYSPBUFF. This allows a variable number of parameters to be passed. The macro will parse the variable names and values out of that string.

❷ The SYSPBUFF variable contains the entire parameter list, including the surrounding parentheses. The parentheses are stripped off here, leaving:

```
CPT,'4201','4203':43',
      ':452',:453',:454'
```

❸ The variable name is always the first parameter in the list. Using the macro %SCAN function the macro variable VAR is created, with value CPT.

❹ The first value to be compared is always in the second parameter. In the current example that value is '4201'. This statement is the first step in building our SAS code. It parses out the second parameter and combines it with the variable name to begin to build our statement. So,

```
infunc = &var eq %scan(&parms,2,%str(),)
```

resolves to:

```
infunc = CPT eq '4201'
```

Can you see where we're going with this?

❺ Now, we need to see how many values are passed to the function so we can add them to our statement. A simple way to do this is to count the number of commas in the parameter list. This will always be one less than the number of parameters. Since the first parameter is the variable name, the number of commas equals the number of values.

There is no simple function to count the number of occurrences of a character. It is easy to do however. If we take the length of the incoming parameter list and subtract the length of the list after compressing out the commas, the result is the number of commas.

In our example,

```
numparms = %eval(%length(&parms) -
%length(%sysfunc(compress(&parms,%str(),))))
```

resolves to:

```
numparms = %eval(44 - 38)
```

or 6. There are indeed 6 values passed to the function.

❻ Now, the macro loops through the remaining parameters - beginning with the third (we've already processed the first and second) and ending with the last. Notice that we calculated the number of values in the statement above. Since the variable name is one of the parameters, there is one more parameter than the number of values. Hence, the loop runs from 3 to number of values + 1.

⑦ Each time through the loop, we parse out the next value from the parameter list. In our example, the macro variable THISPARM would contain the following values:

Loop Variable Value	THISPARM Value
3	'4203'
4	:'43'
5	:'452'
6	:'453'
7	:'454'

⑧ . These values are added, one at a time, with an OR, to the previously built SAS statement. In our example, when processing with loop value 3, the value of INFUNC is CPT eq '4201'. We parse '4303' out of the parameter list and recompute INFUNC:

```
CPT eq '4201' or CPT eq '4203'
```

The loop continues through all the values and continues to add to INFUNC until its final value is:

```
CPT eq '4201' or CPT eq '4203' or
CPT eq : '43' or CPT eq : '452' or
CPT eq : '453' or CPT eq : '454'
```

⑨ Up until now all the processing has been macro processing. It's now time to write the SAS statement we've created to the language compiler. We can simply place our macro variable THISPARM in open code. It will resolve and be sent to the language compiler. Let's see how this might look:

```
data mycpt;
  set allcpt;

  if %in(CPT,'4201','4203': '43',
        ':452',':453',':454') then output;
run;
```

would resolve to:

```
data mycpt;
  set allcpt;

  if CPT eq '4201' or CPT eq '4203' or
     CPT eq : '43' or CPT eq : '452' or
     CPT eq : '453' or CPT eq : '454'
  then output;

run;
```

This is just what we wanted. We saved a lot of typing and were able to build a function that had a similar look and feel to the original.

One final note: when INFUNC is written out in the last statement of the macro it is wrapped in parentheses. This allows a leading NOT in the datastep to function correctly. For example,

```
if not %in(cpt,'4001',:'43',:'45') then
  delete;
```

Without the parentheses the NOT would only apply to the first value.

A MIXED CASE FUNCTION

Until now, all the functions we've looked at contained or created a single SAS code segment. This allowed the macros to be treated as functions and called in a line of SAS code. There are often times when function-like capabilities require more than one line of SAS code. These statements can still be wrapped in a macro and thought of as a function. However, the call to the macro will be a stand-alone statement rather than part of an IF or WHERE statement.

An example of this type of function is found in Appendix B. It is a macro that accepts a character variable as an argument and places a mixed case version of that variable in a return variable. For example, JOHN Q PUBLIC would be returned as John Q Public.

The Call

The call to this macro must be a statement in and of itself, as it will generate multiple SAS statements.

```
%mixed(instrstring,outstring)
```

where instrstring is the name of the variable to be converted and outstring is the name of the variable to be assigned the mixed case value.

The following statements:

```
Name = 'JOHN Q PUBLIC';
%mixed(name,newname);
```

would set the value of newname to John Q Public. Notice that the call to this "function" is a "statement" of its own. The macro, as we will see below, actually contains a number of SAS statements. Note also that the semicolon on the %mixed call is optional. I usually include it for consistency in the look of the code.

The Code

The object of this function is to take each word passed to the function and make the first character upper case and every other letter lower case.

① We need to determine how many words were passed to the function so that we can loop through and change the case of each word. As with the parameter list in the %IN function we can find the length of the incoming string and subtract the length of the string after compressing out the spaces. For JOHN Q PUBLIC, we have a length of 14. Compressing out the spaces leaves 12. So, there are 14-12=2 spaces. We'll add 1 since there will always be one more word than space and we get 3 words.

② Here we set the value of a temporary variable, _OUTSTR, to the value of the incoming variable and then set _OUTSTR to missing. This seems a little redundant, but what it accomplishes is to set the length and type of the _OUTSTR to that of the incoming variable. This variable will hold the string as it is converted word by word.

③ Now the length and type of the temporary outgoing variable have been set and we know the number of words passed to the function. We loop through the incoming string and parse each word with the SCAN function. For each word, concatenate an uppercased first letter with the rest of the word, after lowercasing. The only exception is if the current word is a single character. Then simply uppercase the character.

Loop	WORD	NEWWORD	OUTSTR
1	JOHN	John	John
2	Q	Q	John Q
3	PUBLIC	Public	John Q Public

④ Now that we're done, set the designated outgoing variable, &OUTSTR, to the value of _OUTSTR.

There is a reason for using a temporary variable rather than simply using &OUTSTR in the loop logic. If the outgoing variable name was the same as the incoming variable the incoming value would be lost on the first iteration of the loop.

MANAGING MACRO FUNCTIONS

There are a number of ways to make your functions available to a SAS session. Keep in mind that these are macros we created because they used repeatedly.

- Include the actual source code in the program that will use them. This means that you have multiple versions of the macros to maintain. You've also added "bulk" to all your programs.
- Keep the macros in a separate program, or programs, and %INCLUDE the code into the program or session that uses them. This eliminates the multiple copies and bulk but still requires that you remember to %INCLUDE all the macros you are going to use.
- Place the code, or %INCLUDE, in your AUTOEXEC.SAS file. This has the advantages of the previous example and also moves all the %INCLUDEs to a single location. However, you still need to remember to %INCLUDE all the macros you'll need.
- Use the SASAUTOS and MAUTOSOURCE options. This solution provides almost transparent access to your SAS macros. We'll look at these simple options in detail below.

The SASAUTOS and MAUTOSOURCE Options

These two options allow you to store your macros in text files. Simply give each text file the same name as the macro that is in it. The SASAUTOS option tells SAS the name of the location, or locations, in which the text files are stored. SAS calls these locations AUTOCALL Libraries. In the Windows environment these are simply directories.

The syntax of the options is:

```
SASAUTOS=(location-name1  
          location-name2...location-namen)
```

The parentheses are not required if only one directory name is specified. This option can be set in an OPTIONS statement, the OPTIONS window, a configuration file or as a SAS invocation option. The syntax of the option may be slightly different in each of these situations and across host systems. The syntax of the library name specification may also be host specific. See the SAS companion or online help for your host system for details.

The MAUTOSOURCE option makes the Autocall facility available. NOMAUTOSOURCE is the default.

For example, I have a directory on my system, k:\sas\macros, which contains the following:

- right.sas
- ccyyymm.sas
- in.sas
- mixed.sas

My AUTOEXEC.SAS file contains the following statement:

```
OPTIONS SASAUTOS=k:\sas\macros MAUTOSOURCE;
```

Whenever a macro is encountered in code for the first time SAS will search the directory listed in the SASAUTOS option for a file with the same name as the macro. In the Windows environment, the file names must have a .sas extension. If a file is found the code inside is executed and the macro is available to be used.

Once a macro is referenced the SASAUTOS directory is not searched again for that macro. So, if you are in an interactive SAS session and change a macro file after it has been loaded in the session the changed macro will not be loaded. During testing and development it is a good idea to change and execute the macro in your session and write it out to a file after it functions properly.

TRADEMARKS

The SAS System is a registered trademark of SAS Institute, Inc., Cary, NC.

REFERENCES

SAS Institute Inc., *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1990.

SAS Institute Inc., *SAS Macro Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc., 1997.

AUTHOR CONTACT INFORMATION

An electronic version of this paper and all the macros can be obtained, via e-mail, from the author.

Pete Lund
Washington State Forecast Council
515 E 15th Ave SE
Olympia, WA 98504-0962
(360) 902-0086 voice
(360) 902-0084 FAX
peter.lund@cfc.wa.gov

Applications and Publishing

Appendix A

```
/*-----*/
/* macro:  IN                                     */
/* programmer: Pete Lund                         */
/* purpose:  replaces the IN function, allowing embedded operators and */
/*           functions                            */
/*-----*/

%macro in() / parmbuff; ❶

❷ %let parms = %qsubstr(&syspbuff,2,%length(&syspbuff)-2);

❸ %let var = %scan(&parms,1,%str(,));

❹ %let infunc = &var eq %scan(&parms,2,%str(,));

❺ %let numparms = %eval(%length(&parms) -
    %length(%sysfunc(compress(&parms,%str(,)))));

❻ %do i = 3 %to (&numparms + 1);
❼ %let thisparm = %scan(&parms,&i,%str(,));
❽ %let infunc = &infunc or &var eq &thisparm;
%end;

❾ (&infunc)
%mend;
```

Appendix B

```

/*-----*/
/* macro: MIXED */
/* programmer: Pete Lund */
/* purpose: creates a mixed case character variable - the */
/* first letter of each word is uppercased and */
/* subsequent letters in a word are lowercased. */
/*-----*/

%macro mixed(InString,OutStr);
  drop _i _NumWrds _Word _NewWord _OutStr;

  ① _NumWrds = (length(&InString) - length(compress(&InString,' '))) + 1;

  ② _OutStr = &InString;
  _OutStr = '';

  ③ do _i = 1 to _NumWrds;
    _Word = scan(&InString,_i);
    if length(_Word) gt 1 then
      _NewWord = upcase(substr(_Word,1,1))||lowercase(substr(_Word,2));
    else _NewWord = upcase(_Word);
    _OutStr = trim(left(_OutStr))||' '||_NewWord;
  end;

  ④ &OutStr = _OutStr;
%mend;

```