Inside [Functions and Macros]

Srinivas Vanam Mahipal Vanam Phaneendhar Vanam

Percept Pharma Services, Bridgewater, NJ

ABSTRACT

SAS is rich in various built-in functions, and predefined SAS supplied macros. Similarly, functions and macros can also be developed by user based on his needs and requirements. In this paper, we will walk through the various types of functions, macros, and how the functions and macros are stored and processed internally. In addition, this paper also covers several interesting scenarios like:

1. USING **[FUNCTIONS IN MACROS]** AND **[MACROS IN FUNCTIONS]**.
2. USING **[FORMATS IN FUNCTIONS]** AND **[FUNCTIONS IN FORMATS]**.
3. **SPECIAL SCENARIO**: Demonstrates the Need for **RUN_MACRO()**.

SAS FUNCTION

A SAS Function performs a computation or system manipulation on arguments, and returns a value that can be used in an assignment statement or elsewhere in expressions. In Base SAS software, you can use SAS functions in DATA step programming statements, in a WHERE expression, in macro language statements, in PROC REPORT, and in Structured Query Language (SQL).

CALL ROUTINE

A CALL routine is similar to Function except the difference that, CALL routines cannot be used in assignment statements and expressions.

SAS MACRO FACILITY

The macro facility is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do common tasks. The macro facility enables you to assign a name to character strings or groups of SAS programming statements. You can work with the names you created rather than with the text itself. Macros enable you to substitute text in a program which enhances reusability.

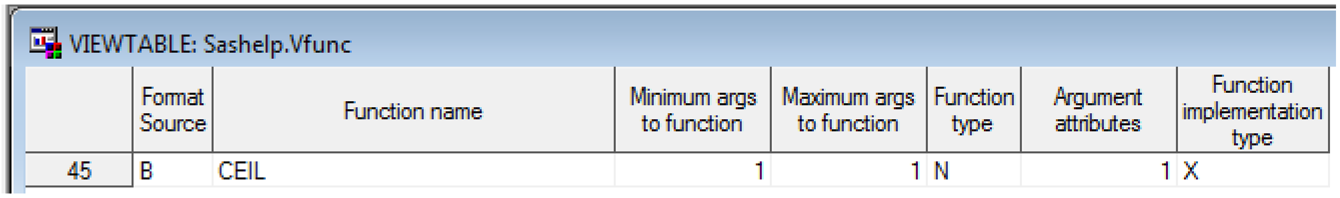
PRE-DEFINED AND USER-DEFINED FUNCTIONS

SAS has a lot of pre-defined functions that perform a variety of computations. They are categorized into Arithmetic, Array, Bitwise Logical Operations, Character string matching, Character, Combinatorial, Date and Time, Descriptive Statistics etc. Similarly, SAS provides the user with the capability to create his own functions that may be used repetitively later on in various tasks. The different ways to create user-defined functions are:

1. **SAS/IML Modules**:
   - The SAS/IML Modules can be used only within PROC IML and cannot be used within a DATA step.
2. **SCL:**
   - SCL provides object-oriented programming concepts for developing classes, attributes, methods, events and event handling mechanism in SAS, but writing SCL programs is not easy for beginners.
   - The methods created using SCL
3. **PROC FCMP:**
   - The SAS FCMP (Function Compiler) procedure enables you to create, test, and store SAS functions, CALL routines and subroutines.
   - These functions and subroutines can be used in a DATA step, the WHERE statement, Output Delivery System (ODS), and procedures like CALIS, COMPILE, COMPUTAB, GA, GENMOD, SQL etc.
4. **PROC PROTO:**
   - This procedure has the ability to make sure of C/C++ functions, but the functions developed using this procedure cannot be used directly in a DATA step.
   - It needs to use PROC FCMP as a bridge for creating a wrapper function.

[**NOTE:** Examples on creating functions using the above 4 ways can be found in the paper, "*Developing User-Defined Functions in SAS: A Summary and Comparison*" by Songfeng Wang and Jiajia Zhang.]

NOTE: The predefined functions are built-in into the SAS system whose source code cannot be seen. But we can see that there is a record for each function in the SASHELP.VFUNC view. SASHELP.VFUNC is a view that gets the data from "select * from dictionary.functions". One such record from SASHELP.VIEW is as follows:



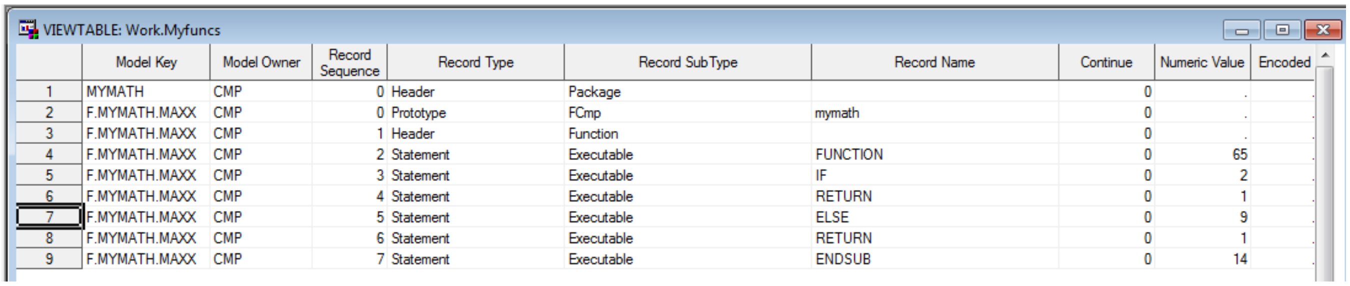
VIEWTABLE: Sashelp.Vfunc

	Format Source	Function name	Minimum args to function	Maximum args to function	Function type	Argument attributes	Function implementation type
45	B	CEIL	1	1	N	1	X

NOTE: The user-defined functions are created within packages. The value for an outlib= option is <libname.dsname.packagename>. For example, the program to create a user-defined function "MAXX(X,Y)" is as follows:

```
proc fcmp outlib = work.myfuncs.mymath ;
      function maxx(x,y);
            if x>=y then return(x);
            else return(y);
      endsub;
run;
```

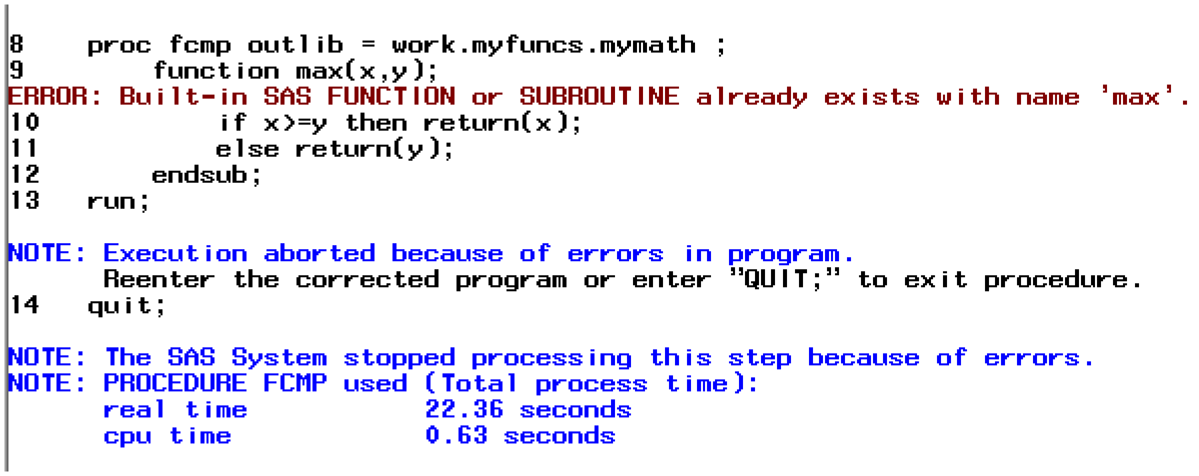
After submitting the above program, here is the MYFUNCS dataset created in WORK library with the records which have function information.



VIEWTABLE: Work.Myfuncs

	Model Key	Model Owner	Record Sequence	Record Type	Record SubType	Record Name	Continue	Numeric Value	Encoded
1	MYMATH	CMP	0	Header	Package		0	.	
2	F.MYMATH.MAXX	CMP	0	Prototype	FCmp	mymath	0	.	
3	F.MYMATH.MAXX	CMP	1	Header	Function		0	.	
4	F.MYMATH.MAXX	CMP	2	Statement	Executable	FUNCTION	0	65	
5	F.MYMATH.MAXX	CMP	3	Statement	Executable	IF	0	2	
6	F.MYMATH.MAXX	CMP	4	Statement	Executable	RETURN	0	1	
7	F.MYMATH.MAXX	CMP	5	Statement	Executable	ELSE	0	9	
8	F.MYMATH.MAXX	CMP	6	Statement	Executable	RETURN	0	1	
9	F.MYMATH.MAXX	CMP	7	Statement	Executable	ENDSUB	0	14	

NOTE: You cannot create a user-defined function with the same name as a pre-defined function. For example, in the previous example, if you try to create the function name as MAX(X,Y) instead of MAXX(X,Y), it results in the following ERROR.



```
8     proc fcmp outlib = work.myfuncs.mymath ;
9          function max(x,y);
ERROR: Built-in SAS FUNCTION or SUBROUTINE already exists with name 'max'.
10              if x>=y then return(x);
11              else return(y);
12         endsub;
13    run;

NOTE: Execution aborted because of errors in program.
      Reenter the corrected program or enter "QUIT;" to exit procedure.
14    quit;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE FCMP used (Total process time):
      real time           22.36 seconds
      cpu time            0.63 seconds
```

PRE-DEFINED AND USER-DEFINED MACROS

SAS has some built-in macro variables (automatic macro variables) and macro definitions. In this paper, our interest is on macro definitions rather than macro variables. Some examples of macro definitions include %left(), %lowcase(), %verify() etc. Similarly, SAS provides the user with the ability to write his own macros which can be used for repetitive tasks. The different ways of creating user-defined macros are:

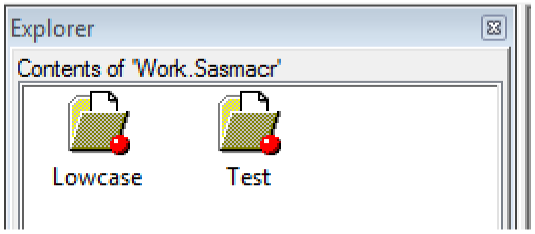
1. **Session-compiled macros:**

- The macros written in our programs when submitted will be compiled and the compiled version of our macros will be stored in the WORK library SASMACR catalog.
- These session- compiled macros will be deleted at the end of the SAS session.
- Example: Consider the following code snippet,

```
%macro test;
      %put This is a session-compiled macro;
%mend;

%test;
```

When you submit these statements, a session-compiled test macro will be created in WORK.SASMACR catalog as follows:



2. **Autocall facility:**
   - SAS provides some built-in macros which will be stored in the SAS installation folder "…\9.2\core\sasmacro".
   - An autocall macro is a .sas program file that contains only the macro definition and no other code in it.
   - By default, autocall facility is enabled in the SAS system. You can check this by submitting the below code. Proc options option = mautosource ; run;
   - You can write your own autocall macros, save them in a particular location and add that location to SASAUTOS system option, so that SAS will be able to locate them.
   - When you invoke the autocall macro, SAS will create a session-compiled version of it in the WORK.SASMACR catalog and executes that compiled version of program.

3. **Stored compiled macro facility:**
   - Macros that are compiled can be saved in particular catalog by using stored compiled macro facility instead of losing them at the end of SAS session.
   - This way, the macros need not be compiled each and every time they are needed. They can be compiled once, and used as many times as needed.
   - By default, stored compiled macro facility is disabled. You can check this by submitting the following code. Proc options option = mstored ; run;
   - You can let SAS know the catalog where you want to save/refer to the compiled macros by setting them with SASMSTORE= option.

[NOTE:] You can create a user-defined macro with the same name as predefined macro. For example, consider the following program,

```
%let value = %lowcase(Hello World!);%put &value. ; /* hello world! */

%macro lowcase(input);
     %sysfunc(upcase(&input.));
%mend;

%let value = %lowcase(Hello World!);%put &value. ; /* HELLO WORLD! */
```

The first occurrence of %lowcase() macro calls the predefined lowcase() macro and returns the result as "hello world!" where as the second occurrence of %lowcase() macro calls the user-defined lowcase() macro and returns the result as "HELLO WORLD!".

NOTE: Searching order: When you invoke a macro "%<macro-name>", SAS searches for that macro in the following order:

1. First of all, it searches if there is any Session-compiled macro with the same name. If a match is found, then it executes that macro. If not, it goes on to the next step.
2. Then, it goes for the "Stored compiled macro facility". If this facility is enabled (MSTORED options is on), then SAS searches for any macros with the same name in the libraries specified by SASMSTORE= system option. If a match is found, that macro will be executed. If not, it goes on to the next step.
3. Then it goes for "Autocall facility". If this facility is enabled (MAUTOSOURCE option is on), then SAS searches for any macros with the same name in the directories specified by SASAUTOS= system option. If a match is found, then that macro will be compiled and a corresponding session-compiled macro will be generated in WORK.SASMACR and executed. If not, then it goes on to the next step.
4. If SAS could not find a macro in session-compiled, stored-compiled and autocall facility level, then it produces an ERROR message saying [Apparent invocation of <macro-name> not resolved].

USING [FUNCTIONS IN MACROS] AND [MACROS IN FUNCTIONS]

1. **Using function within a macro:**
   - SAS functions can be used in a macro context by enclosing them within %SYSFUNC() or %QSYSFUNC() macro call. Ex: Report titles etc.
     Example:

     ```
     %let name = %sysfunc(upcase(SAS Programming));
     %put &name.; /* SAS PROGRAMMING */
     ```

   - Call routines can be invoked with the use of %SYSCALL statement.
     Example:

     ```
     proc fcmp outlib = work.myfuncs.mycall;
          subroutine square(num,sqr);
               outargs sqr;
     ```

```
            sqr = num * num;
        endsub;
    run;

    %let ip = 5 ;
    %let op = . ;
    %syscall square(ip,op);
    %put &op.; /* 25 */
```

2. **Using macro within a function:**
   - Refer to **SPECIAL SCENARIO** section below for example on this topic.

USING [FORMATS IN FUNCTIONS] AND [FUNCTIONS IN FORMATS]

1. **Using Format in Function:**
   - Formats can be used within the function/subroutine definition as if in any other DATA step.
   - Example: Writing a IS8601DT() function that converts partial/full SAS dates in DATE9. format into full SAS dates in IS8601DT. format with imputation.

```
/* Using Formats in Functions */
proc fcmp outlib = work.myfuncs.myform;
    function is8601dt(ipdt $) $;
        if length(ipdt)=9 then do;
            opdt = put(input(ipdt,date9.),yymmdd10.);
        end;
        else if length(ipdt)=7 then do;
            opdt = put(input("01"||strip(ipdt),date9.),yymmdd10.);
        end;
        else if length(ipdt)=4 then do;
            opdt = put(input("01JAN"||strip(ipdt),date9.),yymmdd10.);
        end;
        return(opdt);
    endsub;
run;

options cmplib = work.myfuncs;

data _null_ ;
    ip = "17APR2015" ;
    op = is8601dt(ip);
    put ip= op= ;

    ip = "APR2015" ;
    op = is8601dt(ip);
    put ip= op= ;

    ip = "2015" ;
    op = is8601dt(ip);
    put ip= op= ;
run;
```

Output in the SAS log:

```
ip=17APR2015 op=2015-04-17
ip=APR2015   op=2015-04-01
ip=2015      op=2015-01-01
```

2. Using Function in a Format:

- Starting from SAS v9.3, functions can be used within SAS format declarations.
- **Example:** Creating a format named “enamefmt” which uses the user-defined function “get_ename()”.

```
options cmplib=work.myfuncs;

/* Formats that call functions => Feature available in SAS9.3 */
/* This is not available in SAS version 9.2 */
proc fcmp outlib = work.myfuncs.myname;
   function get_ename(eid) $;
         length ename $12;
         if eid eq 1 then ename="Ram";
         else if eid eq 2 then ename="Venkat";
         else if eid eq 3 then ename="Srinivas";
         else ename="Krishna";
         return(ename);
   endsub;
run;

proc format;
   value enamefmt
         other = [get_ename()]
   ;
run;

data emps; do eid = 1 to 5 ; output; end; run;

proc print data = emps noobs ;
   var eid;
   format eid enamefmt.;
run;
```

Output:

```
The SAS System          14:35 Friday, April 17, 2015   1

   eid

   Ram
   Venkat
   Srinivas
   Krishna
   Krishna
```

SPECIAL SCENARIO

Consider a scenario where we want to create a dataset STUDENT with STU_ID(Num) and STU_NAME(Char) variables. Values for STU_ID are generated as list of 1 to 5 using a DATA step's DO loop. In order to get the values for STU_NAME, we would like to invoke a macro "%get_name(sid=)". Let us walk through three different approaches.

Method 1: Calling %GET_NAME() in the DATA step.

```
%macro get_name(sid=);
      %if &sid. eq 1 %then RAM ;
      %else %if &sid. eq 2 %then DAVID ;
      %else %if &sid. eq 3 %then KRISHNA ;
      %else %if &sid. eq 4 %then TODD ;
      %else %if &sid. eq 5 %then MODI ;
%mend;

data student;
      do stu_id = 1 to 5 ;
            stu_name = "%get_name(sid=stu_id)" ;
      end;
run;
```

Unfortunately, this code does not work because all the macro triggers will be resolved during the compile time of the DATA step and not during the execution. At the compile time, the variable "stu_id" is passed as text to SID macro parameter rather than the value of "stu_id" variable during program execution.
In order to overcome this problem, there is a feature called RUN_MACRO() in PROC FCMP which encapsulates the macro call within a user-defined SAS function which can be called in the DATA Step that creates STUDENT Dataset.

Method 2: Using PROC FCMP's RUN_MACRO() feature to call %GET_NAME() at run time.

```
%macro get_name;
      /* Dequoting the values */
      %let sid   = %sysfunc(dequote(&sid.));
      %let sname = %sysfunc(dequote(&sname.));

      %if &sid. eq 1 %then %let sname=RAM ;
      %else %if &sid. eq 2 %then %let sname=DAVID ;
```

```
        %else %if &sid. eq 3 %then %let sname=KRISHNA ;
        %else %if &sid. eq 4 %then %let sname=TODD ;
        %else %if &sid. eq 5 %then %let sname=MODI ;
%mend;

proc fcmp outlib = work.myfuncs.myname;
        function sname(sid) $;
                length sname $15;
                rc = run_macro('get_name',sid,sname);
                return(sname);
        endsub;
run;

options cmplib = work.myfuncs;

data student;
        length stu_name $ 15;
        do stu_id = 1 to 5 ;
                stu_name = sname(stu_id);
                output;
        end;
run;
```

This method works fine and produces the STUDENT dataset with 5 observations and 2 variables.

Method 3: Using CALL EXECUTE() routine to call the macro conditionally at run-time.

```
%macro get_name(sid=);
        %if &sid. eq 1 %then RAM ;
        %else %if &sid. eq 2 %then DAVID ;
        %else %if &sid. eq 3 %then KRISHNA ;
        %else %if &sid. eq 4 %then TODD ;
        %else %if &sid. eq 5 %then MODI ;
%mend;

data student;
        length stu_name $15;
        do stu_id = 1 to 5 ;
                call execute('%get_name(sid='||stu_id||')');
                output;
        end;
run;
```

The problem with this approach is that, even if the macro is called at run-time with the appropriate value passed, we are not able to store the value returned by macro in a DATA step variable. In this example, for STU_ID=1, the macro will be called and the text "RAM" will be returned. But this text is saved onto input stack my macro processor, which will be executed at the end of the current DATA step. Hence, this approach does not work with our scenario.

REFERENCES

1. “*Developing User-Defined Functions in SAS: A Summary and Comparison*” by Songfeng Wang and Jiajia Zhang.
2. “*Using PROC FCMP to the Fullest: Getting Started and Doing More*” by Arthur L. Carpenter.

CONTACT INFORMATION

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

Srinivas Vanam (srinivas.vanam@gmail.com)

Mahipal Vanam (mahipal.vanam@perceptservices.com)

Phaneendhar Vanam (phaneendhar.v@gmail.com)

Certified Advanced Programmer for SAS 9

Certified Clinical Trials Programmer Using SAS 9

ACKNOWLEDGEMENTS

Percept Pharma Services (www.perceptservices.com)
1031, US Highway 22W, Suite 304,
Bridgewater, NJ. 08807

I would like to take this opportunity to thank my Technical Lead, **Mr. Karthik Darbha** for his constant support and encouragement during my Project as Bristol-Myers Squibb. – Srinivas Vanam.

I would like to take this opportunity to thank my Manager at Amgen, **Mr. Aakar Shah** for his constant support and encouragement during my Project – Phaneendhar Vanam.

TRADEMARK INFORMATION