PharmaSUG 2021 - Paper AP-151

Using PROC FCMP to Create Custom Functions in SAS

Keith Shusterman and Mario Widel, Reata Pharmaceuticals Inc.

ABSTRACT

PROC FCMP is a powerful but somehow underused tool that can be leveraged for creating custom functions. While macros are of vital importance to SAS programming, there are many situations where creating a custom function can be more desirable than using a macro. Particularly, custom functions created through PROC FCMP can be used in other SAS procedures such as PROC SQL, PROC PRINT, or PROC FREQ. As an alternate approach to SAS macros, PROC FCMP can be used to create custom functions that process character and numeric date and datetime values. In this presentation, we will share two custom functions created using PROC FCMP. The first creates ISO 8601 character values from raw date inputs, and the second creates numeric SAS dates and datetimes from ISO 8601 inputs. We will also show how these newly created custom functions can be called in other SAS procedures.

INTRODUCTION

In SAS Version 5, the SAS/TOOLKIT software was used to create custom SAS functions. Use of this software was limited to programmers experienced in programming in C, PL/I, FORTRAN, or IBM 370 assembler languages. Fortunately, SAS today has a procedure specifically for defining custom functions, and it can be used by programmers who know how to program a SAS data step.

The SAS Function Compiler (FCMP) Procedure is used to create and store custom SAS functions and subroutines. The syntax is very similar to that of the SAS DATA step language, with several small variants. The custom functions and subroutines created by PROC FCMP can be used in other SAS procedures.

SAS FUNCTIONS AND SUBROUTINES

The SAS 9.4 reference defines a function as a block of code that takes arguments and returns a single value, either character or numeric, as output. In contrast, it defines a CALL routine as a block of code that can alter variables or perform other system functions. While it's important to note that while PROC FCMP can compile CALL routines, this paper will specifically focus on functions.

We will show that custom functions created with PROC FCMP are able to alter variables called as input arguments but this is not best practice in the authors' opinion for reasons discussed below.

SYNTAX

PROC FCMP uses the following statements:

```
proc fcmp options;
declaration statements;
program statements;
```

One commonly used option is the OUTLIB option, which allows the user to store the function in a physical location. Once compiled and stored, the function can then be called from any other SAS program using the SAS global option CMPLIB.

For example, consider the following code:

```
proc fcmp outlib=work.funcs.trial;
```

After PROC FCMP compiles the functions, the OUTLIB option here will store the compiled version for further use in work.funcs.trial;

From here, the SAS global option CMPLIB = work.funcs; will give the user access to any functions stored in that library.

In this paper, we output the function to the work library out of convenience. However, these can be output to a permanent library. An advantage to outputting to a permanent library is that the function can then be called by any SAS program at any time. However, having the code to generate the function in your program and output to the work library may aid a reviewer in following the code.

Declaration statements function similarly to their data step counterparts. Familiar statements like ATTRIB and FORMAT are available in PROC FCMP. However, the key declaration statement for creating a function is the FUNCTION statement:

```
function funcname( arg1, arg2, ..., argn );
outargs outarg1, outarg2 ... outargN;
program-statements;
return( expression );
endsub;
```

Here, a function called “funcname” takes n arguments. The OUTARGS option specifies input arguments that can be updated by the function. The program statements contain the actual function code, and the RETURN statement returns the requested value. The ENDSUB statement then ends the definition of the subroutine.

Program statements make up the actual definition of the function. Variables can be defined, and arithmetic operations can be performed like that of a data step.

SIMPLE EXAMPLE

Let’s define a simple function that sums two numeric inputs:

```
proc fcmp outlib=work.funcs.trial;

function test_sum(a,b);
c=a+b;
return(c);
endsub;

run;
```

This code outputs a SAS dataset called FUNCS into the work library that stores information regarding the test_sum function.

We then use the CMPLIB option to read this function in and test it in a data step:

```
options cmplib = work.funcs;

data test;
      m = 1;
      n = 2;
      k = test_sum(m,n);
run;
```

The resulting dataset is as follows:

m	n	k
1	2	3

Table 1: Output dataset with the variable k defined using the custom test_sum() function

As expected, the function took the two arguments n and m, and output their sum in the variable k.

The OUTARGS option allows the user to change input values in the function, as shown below:

```
proc fcmp outlib=work.funcs.trial;

function test_sum2(a,b);
outargs b;
b = b+1;
c = a+b;
return(c);
endsub;

run;

options cmplib = work.funcs;

data test;
      m = 1;
      n = 2;
      k = test_sum2(m,n);
run;
```

In this case, we get the following output:

m	n	k
1	3	4

Table 2: Output dataset with the variable k defined using the custom test_sum() function and the variable n modified using the outargs option

The value n is changed in the output dataset, and the sum k is calculated based on the changed n value. In general, we do not recommend using this OUTARGS functionality, as it is not best practice to change input values as part of the custom function.

RECURSION

Recursion is allowed in these custom function definitions. The example below shows how the factorial function can be defined recursively using PROC FCMP:

```
proc fcmp outlib=work.funcs.trial;

function factorial(n);
if n <= 1 then f=1;
      else f = n*factorial(n-1);
return(f);
endsub;


run;

options cmplib = work.funcs;

data test;
      k = factorial(6);
run;
```

The factorial function is called within its own definition to define itself recursively. The output of this call to the factorial function is shown below:

k
720

Table 3: Output dataset with the variable k defined using the custom factorial() function.

While we have no specific examples of recursion being used in a pharmaceutical context, we decided to point out this functionality because it is really cool. Recursion is generally not trivial to perform in SAS.

For further reading regarding PROC FCMP syntax, please refer to the papers by Art Carpenter and Christina Garcia listed in the References section.

PRACTICAL EXAMPLE FOR PROCESSING DATES – RAW TO ISO 8601

For SDTM datasets, dates are required to be presented as character values in ISO 8601 format (yyyy-mm-ddThh:mm:ss). This is advantageous as records can be sorted chronologically even when represented as character values and having incomplete dates. When one component is missing, it is replaced by a dash unless it's the last component. (e.g. 2017---09 for a record with month missing, or 2018-11 for day missing).

SAS formats can handle ISO dates perfectly well if the original date is complete. However, missing dates can cause issues when relying solely on SAS formats. This is particularly problematic because SDTM does not allow for date imputations, forcing programmers to work with incomplete character dates.

It is common to use a macro to take an entire dataset as input, process the available date information, derive the ISO 8601 date as a new variable, and output the dataset with the new variable appended. We will show that PROC FCMP can be used to create a function that will take available date information as input and output an ISO 8601 date value. We will call this function toiso(). Please refer to the appendix for the full code.

The toiso() function has six character arguments as inputs: Year, month, day, hours, minutes, and seconds. The function will combine all provided information into a valid ISO 8601 character date.

Consider the below dataset named “dates”:

YY	MM	DD	HH	MI	SS
1980	6	2	14	32	33
1981		2	14	32	33
1982	6		14	32	33
1983	6	2		32	33
1984	6	2	14		33
1985	6	2	14	32	
1986			14	32	33
1987	6	2			33
1988	6	2			
1989	6				
1990					
1901				32	33
1902					4
	6	2	14	32	33
2002					4
					5

Table 4: Source “dates” table containing complete and incomplete date/time information.

By simply calling the toiso() function in PROC SQL, we can process all of these rows into valid ISO 8601 character dates:

```
proc sql;
create table test as
select *, toiso(yy,mm,dd,hh,mi,ss) as xxdtc
from dates
;
quit;
```

The output dataset is shown below:

YY	MM	DD	HH	MI	SS	XXDTC
1980	6	2	14	32	33	1980-06-02T14:32:33
1981		2	14	32	33	1981---02T14:32:33
1982	6		14	32	33	1982-06--T14:32:33
1983	6	2		32	33	1983-06-02T-:32:33
1984	6	2	14		33	1984-06-02T14:-:33
1985	6	2	14	32		1985-06-02T14:32
1986			14	32	33	1986----T14:32:33
1987	6	2			33	1987-06-02T-:-:33
1988	6	2				1988-06-02
1989	6					1989-06
1990						1990
1901				32	33	1901----T-:32:33
1902					4	1902----T-:-:04
	6	2	14	32	33	
2002					4	2002----T-:-:04
					5	-----T-:-:05

Table 5: Output "test" table with the variable XXDTC defined using the custom toiso() function.

PRACTICAL EXAMPLE FOR PROCESSING DATES – ISO 8601 TO SAS NUMERIC

When programming SDTM, it is necessary to convert source date values into an ISO 8601 character date. When programming ADaM, however, it is necessary to transform ISO 8601 character dates into SAS numeric dates or datetimes. Imputation may be needed if any portion of the ISO value is missing.

To this end, we will implement a function we named "uniso". The function has seven input arguments. The first argument will be the character date variable XXDTC we created in the previous section. The next five arguments involve imputation rules for months, days, hours, minutes, and seconds, respectively. The final input specifies whether the output variable will be a date or a datetime variable. The full code for the uniso() function is available in the appendix. We will show this function call twice in the below datastep:

```
data test2;
      set test;
      xdtm=uniso(xxdtc,'07','01','00','00','00','dt');
      xdt=uniso(xxdtc,'07','01',' ',' ',' ','da');
      format xdt date9. xdtm datetime20.;
run;
```

In our first use of the uniso() function to define the variable XDTM, we have the last argument 'dt' to specify that the output variable will be a datetime. The second argument of '07' will impute a missing month as July, and the third argument of '01' will impute a missing day as the 1st of the month. The next three inputs will impute the hours as '00', the minutes as '00', and the seconds as '00', effectively imputing a missing time as midnight exactly.

In our second use of the uniso() function to define the variable XDT, we have the last argument 'da' to specify that the output variable will be a date. The month is again being imputed as July, and the day is again being imputed as the first of the month. Since this is a date and not a datetime, the input arguments for time imputation are left as spaces.

The output dataset is shown below:

YY	MM	DD	HH	MI	SS	XXDTC	XDTM	XDT
1980	6	2	14	32	33	1980-06-02T14:32:33	02JUN1980:14:32:33	02JUN1980
1981		2	14	32	33	1981---02T14:32:33	02JUL1981:14:32:33	02JUL1981
1982	6		14	32	33	1982-06--T14:32:33	01JUN1982:14:32:33	01JUN1982
1983	6	2		32	33	1983-06-02T-:32:33	02JUN1983:00:32:33	02JUN1983
1984	6	2	14		33	1984-06-02T14:-:33	02JUN1984:14:00:33	02JUN1984
1985	6	2	14	32		1985-06-02T14:32	02JUN1985:14:32:00	02JUN1985
1986			14	32	33	1986----T14:32:33	01JUL1986:14:32:33	01JUL1986
1987	6	2			33	1987-06-02T-:-:33	02JUN1987:00:00:33	02JUN1987
1988	6	2				1988-06-02	02JUN1988:00:00:00	02JUN1988
1989	6					1989-06	01JUN1989:00:00:00	01JUN1989
1990						1990	01JUL1990:00:00:00	01JUL1990
1901				32	33	1901----T-:32:33	01JUL1901:00:32:33	01JUL1901
1902					4	1902----T-:-:04	01JUL1902:00:00:04	01JUL1902
	6	2	14	32	33			
2002					4	2002----T-:-:04	01JUL2002:00:00:04	01JUL2002
					5	-----T-:-:05		

Table 6: Output "test2" dataset with the variables XDTM and XDT defined using the custom uniso() function.

Of course, more complicated and intricate rules for date and time imputation can be built into a custom function like this as needed.

USE IN OTHER SAS PROCEDURES

We have already shown that custom functions created through PROC FCMP can be used in a data step and in PROC SQL. We will now demonstrate that these functions can be used in PROC FREQ and PROC PRINT as well.

The SAS manual on the FCMP procedure lists all of the SAS statistical procedures where functions defined by PROC FCMP can be used. However, these custom functions can be used in other SAS procedures as well any time a function can be used. For example, the uniso() function can be used in a WHERE statement within a PROC FREQ or PROC PRINT as needed.

A PROC FREQ example with output:

```
proc freq data=test2;
where xdt > uniso('1985','07','01',' ',' ',' ','da') ;
tables xdt;
run;
```

The FREQ Procedure

xdt	Frequency	Percent	Cumulative Frequency	Cumulative Percent
01JUL1986	1	20.00	1	20.00
02JUN1987	1	20.00	2	40.00
02JUN1988	1	20.00	3	60.00
01JUN1989	1	20.00	4	80.00
01JUL1990	1	20.00	5	100.00

Table 7: Output from PROC FREQ using the custom uniso() function in a where statement.

Granted, this is equivalent to writing "`where xdt > '1985-07-01'd;`". The purpose of this is simply to demonstrate that a custom PROC FCMP function can be used inside of a PROC FREQ.

A PROC PRINT example with output:

```
proc print data=test2;
where xdt > uniso('1985','07','01',' ',' ',' ','da');
run;
```

Obs	yy	mm	dd	hh	mi	ss	xxdtc	xdtm	xdt
7	1986			14	32	33	1986----T14:-:33	01JUL1986:14:00:33	01JUL1986
8	1987	6	2			33	1987-06-02T-:06:33	02JUN1987:00:06:33	02JUN1987
9	1988	6	2				1988-06-02T-:06	02JUN1988:00:06:00	02JUN1988
10	1989	6					1989-06--T-:06	01JUN1989:00:06:00	01JUN1989
11	1990						1990	01JUL1990:00:00:00	01JUL1990

Table 8: Output from PROC PRINT using the custom uniso() function in a where statement.

ADVANTAGES AND DISADVANTAGES

The primary advantage to using custom functions created with PROC FCMP over SAS macros is that the functions can be used in many SAS procedures without separate macro calls. The versatility of being able to be called from other procedures can provide more convenience than having to make a separate macro call in various situations.

The primary disadvantage to using PROC FCMP is that the custom functions need to be compiled before they can be used, just as macros need to be compiled before they can be used in a session. However, these functions can be stored in a central location and compiled from there with a single line of code. Of course, macros can be compiled as well, but we have found that the process for compiling PROC FCMP functions is much more convenient and straightforward.

Another disadvantage is that, like macros, the actual code for the function is hidden in a location separate from the program itself. This can add an extra layer to documentation and review. Like macros, PROC FCMP can be used to define a function within the current program, but we have found this to be less convenient than including macros in the current program.

Lastly, PROC FCMP functions use positional inputs. This differs from macros, where keyword parameters are an option. When the number of arguments to a function gets large, remembering the order can be a challenge, especially if the user is not the author of the function.

CONCLUSION

PROC FCMP is a powerful tool that is sadly underused by many programmers. Like SAS macros, PROC FCMP can be used to create custom code. If wielded as an additional tool in a programmer's toolbelt alongside macro programming, one can become a more efficient programmer.

REFERENCES

SAS Institute Inc. 2003. The FCMP Procedure. Cary, NC: SAS Institute Inc.

SAS 9.4 Functions and CALL Routines: Reference, Fifth Edition
https://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p0h6cun31ihdqtn1io0sv70zh5l8.htm&docsetVersion=9.4&locale=en

Arthur L. Carpenter. 2013. Using PROC FCMP to the Fullest: Getting Started and Doing More. *SAS Global Forum 2013 - Paper 139-2019*

SAS Institute Inc., SAS Technical Report P-245, SAS/TOOLKIT Software: Changes and Enhancements, Releases 6.08 and 6.09, Cary, NC: SAS Institute Inc., 1992. 149 pp.

Qin Lin, Tim Kelly. Standard SAS Macros for Standard Date/Time Processing
http://philasug.org/0706-PO17.pdf

Christina Garcia. The Power of the Function Compiler: PROC FCMP.
https://www.lexjansen.com/wuss/2016/118_Final_Paper_PDF.pdf

Peter Eberhardt. A Cup of Coffee and Proc FCMP: I Cannot Function Without Them. SAS Global Forum 2009,
https://support.sas.com/resources/papers/proceedings09/147-2009.pdf

Qixia Shi, Guanyu Su. Custom useful SAS functions using the PROC FCMP procedure. PharmaSUG China 2016.
https://www.lexjansen.com/pharmasug-cn/2016/PS/PharmaSUG-China-2016-PS06.pdf

ACKNOWLEDGMENTS

The authors would like to thank Reata Pharmaceuticals and Angie Goldsberry for facilitating and supporting this paper. We would specifically like to acknowledge Steve Kirby for his input and assistance.

CONTACT INFORMATION

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

Keith Shusterman
Reata Pharmaceuticals Inc.
keith.shusterman@reatapharma.com

Mario Widel
Reata Pharmaceuticals Inc.
mario.widel@reatapharma.com

APPENDIX

```
proc fcmp outlib=work.funcs.trial;

  function toiso(yy$, mm$, dd$, hh$, mi$, ss$) $;

  length outdtc tim $22 ;
  array x{5} $2   mm dd hh mi  ss ;
  array y{5} $2   _m _d _h _i  _s ;

  if  cmiss(yy, mm, dd, hh, mi, ss ) < 6 then do;

   if yy = '' then outdtc='-';
   else outdtc=yy;

   do i=1 to 5;
     if x{i} in ('-99','-') then y{i}=' ';
     else if '0' <= x{i} <= '9' then y{i}=put(input(x{i},best.),z2.);
   end;

   /*** fill inner missing values if something is not missing further
right;*/
  do i=5 to 2 by -1;
      if y{i} ne '' and y{i-1} = '' then y{i-1}='-';
    end;

   do i=1 to 2;
      if y{i} ne '' then outdtc=trim(outdtc)||'-'||strip(y{i});
   end;
   do i=3 to 5;
      if y{i} ne '' then do;
            if i=3 then tim='T'||strip(y{i});
              else tim=trim(tim)||':'||strip(y{i});
       end;
  end;
   outdtc=trim(outdtc)||strip(tim);
end;

   return (outdtc);
endsub;


   function uniso(src$,mm$,dd$,hh$,mi$,ss$,fmt$);
   dtc=src;
   ofmt=fmt;
   if dtc ='' then return(.);
   else do;
     do i=1 to 3 while (index(dtc,'--') > 0);
        p=index(dtc,'--');
          if p=1 then return(.);
          if p=5 then dtc=substr(dtc,1,5)||MM||substr(dtc,7);
          if p=8 then dtc=substr(dtc,1,8)||DD||substr(dtc,10);
```

```
     end;

      if hh ne '' and index(dtc,'T-') then
dtc=substr(dtc,1,11)||HH||substr(dtc,13);
      if mi ne '' and index(dtc,':-:') then
dtc=substr(dtc,1,14)||MI||substr(dtc,16);
      len=length(dtc);
      if len=4 then dtc=substr(dtc,1,4)||'-'||MM||'-'||DD;
      if len=7 then dtc=substr(dtc,1,7)||'-'||DD;
      if mi ne '' and ss ne '' and index(dtc,'T') then do;
          if len=13 then dtc=trim(dtc)||':'||MI||':'||SS;
          if len=16 then dtc=substr(dtc,1,16)||':'||SS;
      end;
      len1=length(dtc);
       if ofmt='dt' then
                do;
                     if len1=10 then dt =
input(substr(dtc,1,10)||'T00:00:00',is8601dt.);
                     else dt = input(dtc,is8601dt.);
                end;
       else if ofmt = 'da' then dt = input(dtc,is8601da.);
       else do;
           if len1=10 then dt=input(dtc,is8601da.);
           else dt=input(dtc,is8601dt.);
       end;
      return(dt);
   end;
   endsub;

run;
```