PhUSE 2015

Paper TS09

Clinical data to information, quick, standardised and validated with knowledge of proc FCMP

Cathal Gallagher, d-Wise, Manchester, UK

ABSTRACT

Proc FCMP is a fairly new feature to base SAS® that allows you to build your own functions. It is similar to building and using your own macros, although once they are built it's just like using the preloaded SAS functions such as substr() or catx(). Building functions is not limited to advanced users and they can be used by anyone as long as they know the syntax. This paper will look at building some clinical data specific functions and their use. Once they are built they can be shared easily and can be used in base SAS as well as other SAS products such as DI Studio. Knowing that a function has been validated removes possibilities of errors in this code and the need for re-work. Free download of clinical functions available

INTRODUCTION

Study programming takes a long time, and a lot of the programming is repetitive, both within a study and between studies. SAS comes with a bunch of useful preloaded functions such as substr() or cats() that are designed to make common programming tasks easier. Now SAS has given us the ability to write and call our own functions with the addition of proc FCMP. This paper will show you how to create your own functions and give examples of real life clinical functions in use today. This paper discusses how validated clinical functions can help address issues with finding problems in code, rework of programs, and reducing programming time.

THE PROBLEM

New data arrives from our electronic data caption system, perhaps in CDASH format but perhaps in another format. We get a specification and we all start programming each SDTM domain from scratch. Within each study there are lots of instances of repetitive coding and across studies there are even more. Even within the same domain you can find yourself writing repetitive code for two or more variables. It may not be exactly the same, obviously the variable name changes and the domain changes and sometimes the inputs change, but the code can be very similar and therefore repetitive. Variables such as USUBJID, VISTNUM all the xxSTDTC and xxENDTC variables as well as xxSTDY and xxENDY variables take up huge amounts of time for what is essentially the same code over and over again.

CURRENT PROGRAMMING METHOD

Those not in the industry may be under the impression that once a programmers gets the specification they start coding from scratch on a blank editor every single line of code required to get the data from its starting place to its end place. Those not in the industry would be wrong. Most programmers have snippets of code that they use often copied somewhere that they can access easily and drop into their program when it is required. Simply edit the variable names or input datasets and that variable is complete. Some companies have a central location where programmers can share snippets of code or full macros with each other. This removes a lot of the pain when programming variables such as those mentioned above. However, the need still exists that a programmer has to inspect the specification, try to understand what the spec writer means, find the code snippets, copy and paste code, change variable names and dataset names. Once they are done with this they have to validate all their code. It's easy to miss a variable name when changing names in the same code 50 times. These things all take time.
Figure 1 below shows a typical code snippet that a programmer would copy and paste into a program and then edit the variable names. Figure 1.2 shows the finished code.

```
if DATYY ne "" then
   date=cat(strip(DATYY),
   ifc(DATMO not in ("","-99"),'-'||put(input(left(DATMO),best2.),z2.),' '),
   ifc(DATDD not in ("","-99"),'-'||put(input(left(DATDD),best2.),z2.),'  ' ),
   ifc(TIMHR not in ("","-99"),'T'||put(input(left(TIMHR),best2.),z2.),'  ' ),
   ifc(TIMMI not in ("","-99"),':'||put(input(left(TIMMI),best2.),z2.),'  ' )  ) ;
else date='' ;
```

Fig1: Code snippet for xxdtc variables

```
Data AE;
 set work.AE_CRF;

 if AESTDATYY ne "" then
       AESTDTC=cat(strip(AESTDATYY),
       ifc(AESTDATMO not in ("","-99"),'-'||put(input(left(AESTDATMO),best2.),z2.),' '),
       ifc(AESTDATDD not in ("","-99"),'-'||put(input(left(AESTDATDD),best2.),z2.),'  ' ),
       ifc(AESTTIMHR not in ("","-99"),'T'||put(input(left(AESTTIMHR),best2.),z2.),'  ' ),
       ifc(AESTTIMMI not in ("","-99"),':'||put(input(left(AESTTIMMI),best2.),z2.),'  ' )  ) ;
    else AESTDTC='' ;

 if AEENDATYY ne "" then
       AEENDTC=cat(strip(DATYY),
       ifc(AEENDATMO not in ("","-99"),'-'||put(input(left(AEENDATMO),best2.),z2.),' '),
       ifc(AEENDATDD not in ("","-99"),'-'||put(input(left(AEENDATDD),best2.),z2.),'  ' ),
       ifc(AEENTIMHR not in ("","-99"),'T'||put(input(left(AEENTIMHR),best2.),z2.),'  ' ),
       ifc(AEENTIMMI not in ("","-99"),':'||put(input(left(AEENTIMMI),best2.),z2.),'  ' )  ) ;
    else AEENDTC='' ;



 run;
```

Fig1.2: finished code in place for AESTDTC & AEENDTC

What is hard to notice is the mistake that the programmer has made. Whilst editing the snippet of code for AEENDTC the programmer has forgotten to add on the AEEN prefix to a DATYY variable. This will cause the code to fail. This is a very common mistake, and although it is easily fixed, it all takes up time. The mistake is highlighted below in Figure 1.3.

```
Data AE;
 set work.AE_CRF;

 if AESTDATYY ne "" then
       AESTDTC=cat(strip(AESTDATYY),
       ifc(AESTDATMO not in ("","-99"),'-'||put(input(left(AESTDATMO),best2.),z2.),' '),
       ifc(AESTDATDD not in ("","-99"),'-'||put(input(left(AESTDATDD),best2.),z2.),'  ' ),
       ifc(AESTTIMHR not in ("","-99"),'T'||put(input(left(AESTTIMHR),best2.),z2.),'  ' ),
       ifc(AESTTIMMI not in ("","-99"),':'||put(input(left(AESTTIMMI),best2.),z2.),'  ' )  ) ;
    else AESTDTC='' ;

 if AEENDATYY ne "" then
       AEENDTC=cat(strip(DATYY),
       ifc(AEENDATMO not in ("","-99"),'-'||put(input(left(AEENDATMO),best2.),z2.),' '),
       ifc(AEENDATDD not in ("","-99"),'-'||put(input(left(AEENDATDD),best2.),z2.),'  ' ),
       ifc(AEENTIMHR not in ("","-99"),'T'||put(input(left(AEENTIMHR),best2.),z2.),'  ' ),
       ifc(AEENTIMMI not in ("","-99"),':'||put(input(left(AEENTIMMI),best2.),z2.),'  ' )  ) ;
    else AEENDTC='' ;



 run;
```

Fig1.3: finished code with mistake highlighted

WHAT COULD WE DO?

If you find yourself either typing repetitive code or copying and pasting snippets of code then you should consider writing your own functions. A set of fully validated functions can eliminate large portions of code from you programs. This will make searching for errors in you code easier as there is a lot less code to look through. Shorter programs as easier to edit, and if everyone is coding common variables in the same way then these programs are much easier to understand.

Figure 2 below shows how the code would look if a function is called instead in order to populate AESTDTC and AEENDTC.

```
Data AE;
  set work.AE_CRF;

  AESTDTC=sdtmxxdtc(AESTDATYY,AESTDATMO,AESTDATDD,AESTTIMHR,AESTTIMMI);
  AEENDTC=sdtmxxdtc(AEENDATYY,AEENDATMO,AEENDATDD,AEENTIMHR,AEENTIMMI);


  run;
```

Fig2: Shortened code using functions to populate AESTDTC and AEENDTC

This is by no means perfect, and of course a programmer can still make mistakes. Variable names can be spelt incorrectly or the order of the variables could be mixed up. The point is that finding the error in just one line of code is much easier than finding it in several.

HOW IS A FUNCTION CREATED?

Basic functions are created very easily in a SAS editor. Below, this paper will demonstrate how to create a function that creates the xxDTC variables for SDTM datasets. Figure 3 shows the full function.

```
proc fcmp outlib=work.funcs.clinical;*This tell us where the functions will be saved;

 *function name with inputs and their datatypes. Dollar at end indicates this returns a Char value;
 function sdtmxxdtc(DATYY$,DATMO$,DATDD$,TIMHR$,TIMMI$)$;
    if DATYY ne "" then
       date=cat(strip(DATYY),
       ifc(DATMO not in ("","-99"),'-'||put(input(left(DATMO),best2.),z2.),' '),
       ifc(DATDD not in ("","-99"),'-'||put(input(left(DATDD),best2.),z2.),'  '),
       ifc(TIMHR not in ("","-99"),'T'||put(input(left(TIMHR),best2.),z2.),'  '),
       ifc(TIMMI not in ("","-99"),':'||put(input(left(TIMMI),best2.),z2.),'  ')  ) ;
    else date='' ;
 return(date);*the value to be returned;
 endsub;
```

Fig3: Function used to create dates for SDTM tables

Firstly the “proc fcmp” command is given with a library address. This tell SAS where to save the function to. See figure 3.1

```
proc fcmp outlib=work.funcs.clinical;*This tell us where the functions will be saved;

 *function name with inputs and their datatypes. Dollar at end indicates this returns a Char value;
 function sdtmxxdtc(DATYY$,DATMO$,DATDD$,TIMHR$,TIMMI$)$;
    if DATYY ne "" then
       date=cat(strip(DATYY),
       ifc(DATMO not in ("","-99"),'-'||put(input(left(DATMO),best2.),z2.),' '),
       ifc(DATDD not in ("","-99"),'-'||put(input(left(DATDD),best2.),z2.),'  '),
       ifc(TIMHR not in ("","-99"),'T'||put(input(left(TIMHR),best2.),z2.),'  '),
       ifc(TIMMI not in ("","-99"),':'||put(input(left(TIMMI),best2.),z2.),'  ')  ) ;
    else date='' ;
 return(date);*the value to be returned;
 endsub;
```

Fig3.1: proc fcmp command with function library address

Next start programming the function with the command “function” followed by the name of the function. Figure 3.2 below has the function name “sdtmxxdtc”. This function name must then have an opening and closing bracket. Inside the brackets is listed the input variables required for the function. The dollar sign at the end of the variable name indicates that it is a character variable. If there is no dollar sign then a numeric variable is required. After the closing bracket the dollar sign indicates that the function will return a character variable. No Dollar sign would indicate that a numeric variable is returned. The function must be accompanied by an “endsub” command, which tells SAS where your function ends.

```
proc fcmp outlib=work.funcs.clinical;*This tell us where the functions will be saved;

 *function name with inputs and their datatypes. Dollar at end indicates this returns a Char value;
 function sdtmxxdtc(DATYY$,DATMO$,DATDD$,TIMHR$,TIMMI$)$;
    if DATYY ne "" then
       date=cat(strip(DATYY),
       ifc(DATMO not in ("","-99"),'-'||put(input(left(DATMO),best2.),z2.),' '),
       ifc(DATDD not in ("","-99"),'-'||put(input(left(DATDD),best2.),z2.),'  ' ),
       ifc(TIMHR not in ("","-99"),'T'||put(input(left(TIMHR),best2.),z2.),'  ' ),
       ifc(TIMMI not in ("","-99"),':'||put(input(left(TIMMI),best2.),z2.),'  ' )  ) ;
    else date='' ;
 return(date);*the value to be returned;
 endsub;
```

Fig3.2: function command

In between the function command and the endsub, is where the code for your function goes. Figure 3.3 highlights the code that is used to put all the date time parts together into a character variable called date. This is returned using the return command which is highlighted in pink.

```
proc fcmp outlib=work.funcs.clinical;*This tell us where the functions will be saved;

 *function name with inputs and their datatypes. Dollar at end indicates this returns a Char value;
 function sdtmxxdtc(DATYY$,DATMO$,DATDD$,TIMHR$,TIMMI$)$;
    if DATYY ne "" then
       date=cat(strip(DATYY),
       ifc(DATMO not in ("","-99"),'-'||put(input(left(DATMO),best2.),z2.),' '),
       ifc(DATDD not in ("","-99"),'-'||put(input(left(DATDD),best2.),z2.),'  ' ),
       ifc(TIMHR not in ("","-99"),'T'||put(input(left(TIMHR),best2.),z2.),'  ' ),
       ifc(TIMMI not in ("","-99"),':'||put(input(left(TIMMI),best2.),z2.),'  ' )  ) ;
    else date='' ;
 return(date);*the value to be returned;
 endsub;
```

Fig3.3: function code

Once this code has been run, the function is ready for use in other programs. This function has been stored in a work library so it will only be available within the session. If you exit out of SAS and later want to use this function again, it must be compiled again. Alternatively it can be saved in a permanent library. This demonstration will be using an input dataset called sdtmInput1. Figure 3.4 below illustrates its contents.

VIEWTABLE: Work.Sdtminput1

	Subjid	RFSTDTC	AESTDATYY	AESTDATMO	AESTDATDD	AESTTIMHR	AESTTIMMI
1	1000	2015-07-14T18:16	2015	08	10	17	32
2	1001	2014-07-10	2014	8	1	07	02
3	1002	2013-07-14	2013	10	11	8	32
4	1003	2015-07-14	2015	1	9	16	2
5	1004	2014-07-14	2014	6	17		
6	1005	2013-07-14	2013	8	14		

Fig3.4: sdtmInput1 Contents

The data step in figure 3.5 illustrates how to call the function. Note the options statement highlighted in yellow that tells SAS where user written functions are stored. The function call is highlighted in pink.

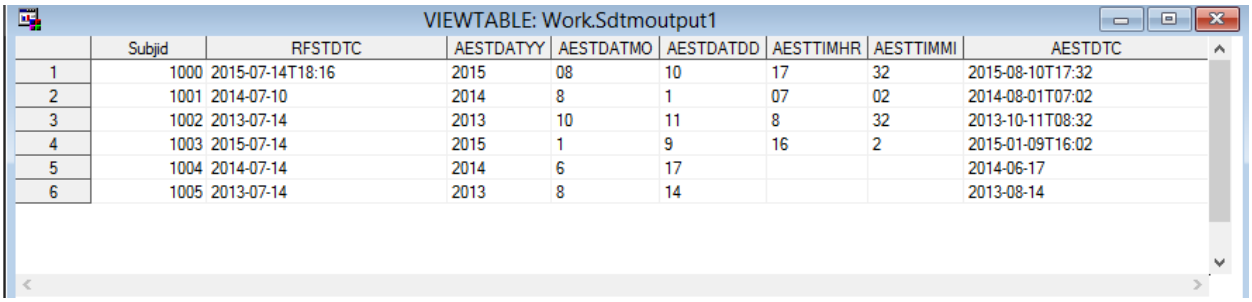
```
options cmplib=work.funcs; *tell sas to get user written functions from here;

data sdtmoutput1;
 set sdtminput1;
 length AESTDTC $20.;

 AESTDTC=sdtmxxdtc(AESTDATYY,AESTDATMO,AESTDATDD,AESTTIMHR,AESTTIMMI);
 run;
```

Fig3.5: calling a function

Figure 3.5 shows the contents of the output dataset sdtmOutput1 Where AESTDTC is now visible.



VIEWTABLE: Work.Sdtmoutput1

	Subjid	RFSTDTC	AESTDATYY	AESTDATMO	AESTDATDD	AESTTIMHR	AESTTIMMI	AESTDTC
1	1000	2015-07-14T18:16	2015	08	10	17	32	2015-08-10T17:32
2	1001	2014-07-10	2014	8	1	07	02	2014-08-01T07:02
3	1002	2013-07-14	2013	10	11	8	32	2013-10-11T08:32
4	1003	2015-07-14	2015	1	9	16	2	2015-01-09T16:02
5	1004	2014-07-14	2014	6	17			2014-06-17
6	1005	2013-07-14	2013	8	14			2013-08-14

Fig3.5: sdtmOutput1 Contents

Figure 3.6 illustrates the coding of a different function that will return a numeric value. Note there is no dollar sign at the end of the function statement. This function is used to calculate study day values. The function is called sdtmxxdy

```
*second function declared, no dollar sign at the end indicates that this returns a numeric value;
function sdtmxxdy(rfstdtc$,dtc$);
   if (rfstdtc in('.','') or dtc in('.','')) then
        stdy=.;
   else if input(substr(dtc,1,10),is8601da.)< input(substr(rfstdtc,1,10),is8601da.)then
        stdy=input(substr(dtc,1,10),is8601da.)- input(substr(rfstdtc,1,10),is8601da.);
   else if input(substr(dtc,1,10),is8601da.)>= input(substr(rfstdtc,1,10),is8601da.)  then
        stdy=input(substr(dtc,1,10),is8601da.)- input(substr(rfstdtc,1,10),is8601da.)+1;
   else
        stdy=.;
return(stdy);

endsub;
```

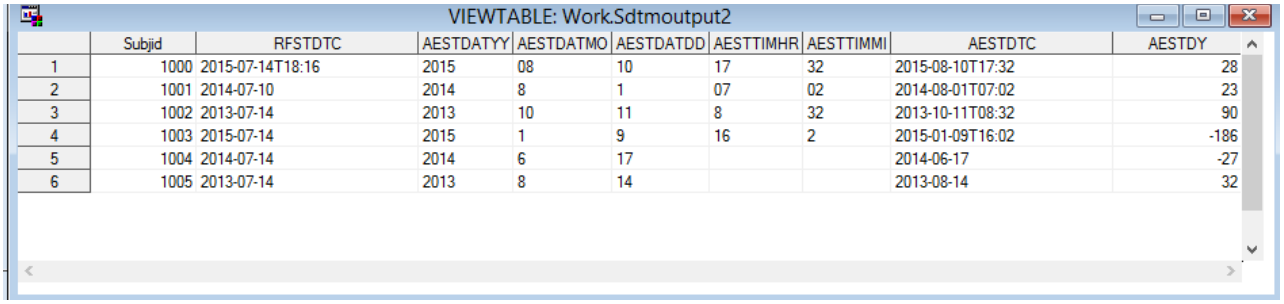
Fig3.6: sdtmxxdy Function code

This numeric function is called in exactly the same way as the character function, and any other SAS function. The datastep in figure 3.7 shows the calling of both functions and the output dataset.

```
options cmplib=work.funcs; *tell sas to get unknown functions from here;

data sdtmoutput2;
set sdtminput1;
length AESTDTC $20.;

AESTDTC=sdtmxxdtc(AESTDATYY,AESTDATMO,AESTDATDD,AESTTIMHR,AESTTIMMI);
AESTDY=sdtmxxdy(RFSTDTC,AESTDTC);
run;
```



VIEWTABLE: Work.Sdtmoutput2

	Subjid	RFSTDTC	AESTDATYY	AESTDATMO	AESTDATDD	AESTTIMHR	AESTTIMMI	AESTDTC	AESTDY
1	1000	2015-07-14T18:16	2015	08	10	17	32	2015-08-10T17:32	28
2	1001	2014-07-10	2014	8	1	07	02	2014-08-01T07:02	23
3	1002	2013-07-14	2013	10	11	8	32	2013-10-11T08:32	90
4	1003	2015-07-14	2015	1	9	16	2	2015-01-09T16:02	-186
5	1004	2014-07-14	2014	6	17			2014-06-17	-27
6	1005	2013-07-14	2013	8	14			2013-08-14	32

Fig3.7: datastep calling two function with output dataset

When validated functions like those shown so far have been used in programs, the amount of code in each program is vastly reduced. Due to the fact that these functions have already been validated the code within the functions does not need to be examined when a program fails. This results in a lot less code that needs to be examined in order to locate and resolve errors. The use of validated clinical functions not only reduces programming and validation time of datasets but it reduces difficulty in locating and fixing errors.

FUNCTIONS VS MACROS

Functions have not been designed to replace macros. All of the things you can do with functions can of course be performed with macros. The aim of using validated functions within clinical programs is to make code easier to read and understand. Code becomes difficult to interpret when datastep syntax and macro syntax are mixed together. This often slows down the error detection and validation process whilst coding. There are performance benefits to functions too, standard Do loops should be used instead of %DO loops for example. Macro code cannot be inserted into most of the transformations within DI Studio whilst user written functions can be used through the SAS product range.

THE FUTURE

Industry standard clinical functions. Pharmaceutical companies have starting standardizing their tables using CDASH SDTM and ADaM. The next step is to start standardizing the programming with industry wide standardized clinical functions.

CONCLUSION

Using validated functions in clinical programming will save time. Repetitive coding can be minimized with the use of functions. The size of programs can be vastly reduced with the use of functions and therefore the validation process can also be reduced. When there is less code to examine in order to locate errors, there will be less rework that needs to happen. All these things added together mean less programming time is required for each study that comes along.

CONTACT INFORMATION

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

Cathal Gallagher
d-Wise
Peter House
1 Oxford Street
Manchester / M1 5AN
Work Phone: +44777 335 1236
Fax:
Email:Cathal.gallagher@d-wise.com
Web: www.d-wise.com

Brand and product names are trademarks of their respective companies. ©®