PharmaSUG 2016 – Paper QT08

Trivial Date Tasks? PROC FCMP Can Help

Jueru Fan, PPD, Morrisville, NC

ABSTRACT

Imputing partial date, comparing dates and creating flags are important tasks when working with CDISC Analysis Data Model (ADaM) data sets. If function or CALL routines are available to help us deal with them and therefore free us from repetitive works, we can better focus on tasks requiring more logical thinking. Yes, the SAS Function Compiler Procedure (PROC FCMP) is there for us. This paper will first look at some basic syntax of PROC FCMP to build functions and CALL routines. Then two user-defined functions and a CALL routine will be presented on comparing complete/partial date, imputing partial date and creating pre concomitant flag, concomitant flag, and post concomitant flag, respectively.

INTRODUCTION

Imputing partial date, comparing dates and creating flags are important tasks when working with CDISC Analysis Data Model (ADaM) datasets. It is very common to perform these tasks on domains related to Adverse Event, Concomitant Medication, and more. The logic is not very difficult to understand but it can be programmatically complicated especially when partial dates are involved. Unfortunately partial dates always come with real data. And we need to handle them time and time again. If function or CALL routines are available to help us deal with them and therefore free us from repetitive works, we can better focus on tasks requiring more logical thinking. Yes, the SAS Function Compiler Procedure (PROC FCMP) is there for us.

PROC FCMP gives users ability to create, test, and store SAS functions and CALL routines. Since this feature was expanded to DATA and PROC step in SAS 9.2 a method additional to SAS macro language has been around to break down complicated problems into easier pieces. There are several advantages of user-defined function and CALL routines:

- **Flexibility** - Perhaps SAS function is the first thing most programmers get to know when start learning SAS. It can be embedded into many steps, such as a `WHERE` statement, without much limitation.
- **Independence** - All variables created within the functions or CALL routines blocks are local. They are saved in call stack and will be replaced after the implementation of functions and CALL routines.
- **Maintenance** - PROC FCMP uses DATA step syntax when creating a user-written function and CALL routine and therefore is very familiar to programmers at every level.

This paper will first look at some basic syntax of PROC FCMP to build functions and CALL routines. Then two user-defined functions and a CALL routine will be presented on comparing complete/partial date, imputing partial date and creating pre concomitant flag, concomitant flag, and post concomitant flag, respectively.

PROC FCMP SYNTAX

```
proc fcmp outlib=sasuser.test.date; ← 1
       function NumDate(dt $); ←  2
              n=input(dt,yymmdd10.);
       return (n); ←  3
       endsub; ←  4
       subroutine NumDt(dt $,num_dt2); ←  5
              outargs num_dt2; ←  6
              num_dt2=input(dt,yymmdd10.);
       return; ←  7
       endsub;
```

```
run;
```

The purpose of code above is to create a function called NUMDATE and a CALL routine called NUMDT that convert the below character date variable into numeric.

```
data dts;
       dt  = '2014-10-15';
run;
```

Now let's look into the code.

1. `outlib=sasuser.test.date`
   This statement creates a data set TEST under SAS library SASUSER with package named DATE.

2. `function NumDate(dt $);`
   `function` is the keyword to create functions. `dt` is the argument to be used in this function. `$` means this argument should be a string argument. The value of this argument is copied and passed to the function and therefore ensures the original value would not be altered.

3. `return (n);`
   To return the value of this function use `return` statement followed by an enclosed parentheses. If the value to return is a string then another `$` sign will need to be added outside of the parentheses in function statement.

4. `endsub;`
   `endsub` statement ends the declaration of a function or a CALL routine.

5. `subroutine NumDt(dt $,num_dt2);`
   `subrontine` is the keyword to declare a CALL routine. Similar to `function` statement string arguments should be specified with `$` symbol.

6. `outargs num_dt2;`
   `outargs` statement specifies argument from the argument list that you want subroutine to update.

7. `return;`
   `return` statement for subroutine is optional and does not come with enclosed parentheses.

After examining the code let's test the function and CALL routine. Numeric version of date 2014/10/15 in DATE9 format is expected.

```
option cmplib=sasuser.test; ← 8
data _null_;
       set dts;
       retain num_dt2 .;
       num_dt1=NumDate(dt);
       call NumDt(dt,num_dt2);
       put dt= ;
       put num_dt1= ;
       put num_dt2= ;
       format num_dt1 num_dt2 date9.;
run;
```

SAS log :

dt=2014-10-15
num_dt1=15OCT2014
num_dt2=15OCT2014

NOTE: There were 1 observations read from the data set WORK.DTS.
NOTE: DATA statement used (Total process time):
real time 0.09 seconds
cpu time 0.09 seconds

8. `option cmplib=sasuser.test;`
This statement specifies SAS data set `TEST` under libref `SASUSER` that contain compiler subroutines to include during program compilation.

As we can see, the outcomes of function and CALL routine are matching with expectation.

THREE-DIMENSIONAL CONCOMITANT MEDICATION FLAG CLASSIFICATION

Yam (2004)[1] proposed a thorough discussion of creating concomitant medication flag by three-dimensional classification.

	S=0	S=1	S=2	S=3
E=0	Prior Concomitant Post	Prior Concomitant Post	Concomitant Post	Post
E=1	Prior	Prior	Error	Error
E=2	Prior Concomitant	Prior Concomitant	Concomitant	Error
E=3	Prior Concomitant Post	Prior Concomitant Post	Concomitant Post	Post

S is the comparison outcome among non-study drug onset date, study drug start date and study drug end date. When

- S=0 means non-study drug onset date is missing.
- S=1 means non-study drug onset date is prior to study drug start date.
- S=2 means non-study drug onset date is no earlier than study drug start date and not later than study drug end date.
- S=3 means non-study drug onset date is later than study drug end date.

E is the comparison outcome among non-study drug end date, study drug start date and study drug end date. Its values of 0/1/2/3 have definitions in similar fashion with S.

CREATING FUNCTION `DateComp`

First let's start with building function DateComp. The purpose of this function is to compare dates, including both complete and partial dates. Since we are only to demonstrate the method we assume that both study drug start date and end date are non-missing and complete with year, month and day. This could be a safe assumption since these two dates are most likely to appear in numeric format in ADaM datasets. It is not possible for them to be partial. Another assumption would be the format of incoming date is `ISO8601` and no time part is existing. This is also a reasonable assumption as ISO8601 format is suggested by CDISC Study Data Tabulation Model (SDTM) implementation guide.

Use the code below to create a dummy data set.

```
data dts;
       length ymd ym y trtsdt trtedt $10;
       ymd  = '2014-10-15';
       ym   = '2014-09';
       y    = '2015';
       miss = '';
       trtsdt = '2014-10-01';
       trtedt = '2014-10-30';
run;
```

Now we need to define the comparison rule. Comparison will be made up to the dates' lowest level. That is, if day part is missing then only year and month will be compared, and if day and month parts are missing then only year will be compared. If any of the date is missing then the return of function will be 0 which means missing. Given this rule the expected outcomes of function `DateComp` implemented over variable `YMD`, `YM`, `Y` and miss should be 2, 1, 3, and 0 respectively.

```
options cmplib=_null_;
proc fcmp outlib=sasuser.funcs.date;
       function DateComp(dt $,sdt $,edt $);
              length xxdt xxsdt xxedt $10; ← 9
              if lengthn(dt)=0 then n = 0;
              else do;
                     if lengthn(dt)>= 10 then do;
                            xxdt= substrn(dt,1,10);
                            xxsdt = sdt; ← 10
                            xxedt = edt;
                     end;
                     else if lengthn(dt)=7 then do;
                            xxsdt = substrn(sdt,1,7); ← 11
                            xxedt = substrn(edt,1,7);

                     end;
                     else if lengthn(dt)=4 then do;
                            xxdt = dt;
                            xxsdt = substrn(sdt,1,4);
                            xxedt = substrn(edt,1,4);

                     end;

                     if xxdt<xxsdt then n=1;
                     else if xxsdt<=xxdt<=xxedt then n=2;
                     else if xxdt>xxedt then n=3;
              end;
       return (n);
       endsub;
run;
```

As we can see the program above is using DATA step syntax familiar to a SAS user. One thing worth paying attention is that as mentioned before, function or subroutine will not make any modification to original value. This means the input variables are "read-only" unless they have been listed in `outargs` statement within subroutine. By looking at arrow 9, 10 and 11 above we will notice that new local variables have been created and values of input variables have been passed over to them. These values can be either changed or unchanged. Now let's apply this function to dummy data we just created.

```
options cmplib=sasuser.funcs;
```

```
data _null_;
       set dts;
       n1 = DateComp(ymd,trtsdt,trtedt);
       n2 = DateComp(ym,trtsdt,trtedt);
       n3 = DateComp(y,trtsdt,trtedt);
       n4 = DateComp(miss,trtsdt,trtedt);

       put (ymd trtsdt trtedt n1 ) (=);
       put (ym trtsdt trtedt n2 ) (=);
       put (y trtsdt trtedt n3 ) (=);
       put (miss trtsdt trtedt n4) (=);
run;
```

SAS log :

ymd=2014-10-15 trtsdt=2014-10-01 trtedt=2014-10-30 n1=2
ym=2014-09 trtsdt=2014-10-01 trtedt=2014-10-30 n2=1
y=2015 trtsdt=2014-10-01 trtedt=2014-10-30 n3=3
miss= trtsdt=2014-10-01 trtedt=2014-10-30 n4=0

NOTE: There were 1 observations read from the data set WORK.DTS.
NOTE: DATA statement used (Total process time):
real time 0.14 seconds
cpu time 0.10 seconds

As we can see the results of function `DateComp` over variables `YMD`, `YM`, `Y` and `MISS` are just exactly as what we desired. Creation of this function is done. Now what if we want to impute the partial dates?

CREATING FUNCTION `DateImpute`

Different with function `DateComp`, the purpose of this function is to impute single partial date under pre-defined imputation rule. In most clinical trials this rule should be well written in Statistical Analysis Plan (SAP). Now to demonstrate this example let's use the easiest and most conservative rule. That is:

For drug start date,

- If day is missing and year and moth are present then impute this date as the first day of the month
- If day and month are missing and year is present then impute the date as the first day of the year.
- If all day and month and year are missing then this date will not be imputed.

For drug end date,

- If day is missing and year and moth are present then impute this date as the last day of the month
- If day and month are missing and year is present then impute the date as the last day of the year.
- If all day and month and year are missing then this date will not be imputed.

As what we did previously, a dummy data set will be created.

```
data dts;
       length id $1 cmstdtc cmendtc $10;

       id = '1';
       cmstdtc = '2014-10-15';
```

```
        cmendtc = '2014-11-15';
        output;

        id = '2';
        cmstdtc = '2014-09';
        cmendtc = '2014-10';
        output;

        id = '3';
        cmstdtc = '2015';
        cmendtc = '2014';
        output;
run;
```

Variable CMSTDTC is the concomitant medication onset date, and variable CMENDTC is the concomitant medication end date. After the imputation, below result is expected.

- For ID 1, CMSTDTC = 2014-10-15 and CMENDTC = 2014-11-15
- For ID 2, CMSTDTC = 2014-09-01 and CMENDTC = 2014-10-31
- For ID 3, CMSTDTC = 2015-01-01 and CMENDTC = 2014-12-31

Now let's see how the program is built.

```
proc fcmp outlib=sasuser.funcs.date;
    function dateImpute(dt $,type) $ 10; ← 12
              length mdt $10 xxy $4 xxm xxd $2;
              xxy=substrn(compress(dt),1,4);
              xxm=substrn(compress(dt),6,2);
              xxd=substrn(compress(dt),9,2);

if type = 1 then do;
       if (cmiss(xxy,xxm,xxd)=3) or notdigit(compress(xxy))>0 then n=.;
       else do;
              if missing(xxm) or notdigit(compress(xxm))>0 then do;
                     xxm='1';
                     xxd='1';
              end;
              else if missing(xxd) or notdigit(compress(xxd))>0 then do;
                     xxd='1';
              end;
              n=mdy(input(xxm,best.),input(xxd,best.),input(xxy,best.));
       end;
end;
else if type = 2 then do;
       if (cmiss(xxy,xxm,xxd)=3) or notdigit(compress(xxy))>0 then n=.;
       else do;
              if missing(xxm) or notdigit(compress(xxm))>0 then do;
                     xxm='12';
                     xxd='31';
              end;
              else if missing(xxd) or notdigit(compress(xxd))>0 then do;
                     xxd=substrn(strip(put(intnx( ← 13
                                   'month',input(catx('-
                                   ',compress(dt),'15'),yymmdd10.),0,'e'
                                   ),date9.)),1,2);
```

```
                end;
                n=mdy(input(xxm,best.),input(xxd,best.),input(xxy,best.));
        end;
end;
mdt = if not missing(n) then strip(put(n,yymmdd10.)) else ''; ← 14
return (mdt);
endsub;
run;
```

There are several points to note:

- Arrow 12: as mentioned before, by adding the `$` sign outside of parentheses SAS will know that string value will be returned at the end of this function. In this case we want to directly return the imputed date in `ISO8601` format. If numeric value is expected, which could be even more likely as in most cases imputed dates are saved as analysis date (`ASTDT`), just remove the `$` sign and the trailing length. Also you may have noticed another argument `TYPE` has been added. By using this argument we will be able to apply different imputation algorithm to start date and end date.

- Arrow 13: a SAS default function `INTNX` is used here. By using the modifier `e` the ending of this interval will be returned. Since a complete numeric date is expected as argument of function `INTNX` day '15' is added temporarily to pad the partial date as 'XXXX-XX-15'.
- Arrow 14: `IF` expression is used here. This type of expression is supported by PROC FCMP but not DATA step. It has the same function as traditional two pieces `IF-THEN` statements but not as complex.

Now let's test the function.

```
options cmplib=sasuser.funcs;
data _null_;
       set dts;
       length m_cmstdtc m_cmendtc $10;
       m_cmstdtc = DateImpute(cmstdtc,1);
       m_cmendtc = DateImpute(cmendtc,2);
       put (id cmstdtc m_cmstdtc cmendtc m_cmendtc) (=);
run;
```

SAS log :

id=1 cmstdtc=2014-10-15 m_cmstdtc=2014-10-15 cmendtc=2014-11-15 m_cmendtc=2014-11-15
id=2 cmstdtc=2014-09 m_cmstdtc=2014-09-01 cmendtc=2014-10 m_cmendtc=2014-10-31
id=3 cmstdtc=2015 m_cmstdtc=2015-01-01 cmendtc=2014 m_cmendtc=2014-12-31

NOTE: There were 3 observations read from the data set WORK.DTS.
NOTE: DATA statement used (Total process time):
real time 0.12 seconds
cpu time 0.10 seconds

Just as what we expected, partial dates have been imputed in the way they should be and meanwhile complete dates remain the same. Building of function `DateImpute` is done. Now we can move forward to the final step that is to create a CALL routine for concomitant medication flags.

CREATING CALL ROUTINE `CMFlag`

With the existing two functions `DateComp` and `DateImpute` the task of creating CALL routine `CMFlag` becomes straightforward. As usual let's first get the mock data.

```
data dts;
	length id $1 cmstdtc cmendtc trtsdt trtedt $10;

	id = '1';
	cmstdtc = '2014-10-15';
	cmendtc = '2014-11-15';
	trtsdt = '2014-10-01';
	trtedt = '2014-10-30';
	output;

	id = '2';
	cmstdtc = '2014-09';
	cmendtc = '2014-10';
	trtsdt = '2014-10-15';
	trtedt = '2014-11-30';
	output;

	id = '3';
	cmstdtc = '2015';
	cmendtc = '2013';
	trtsdt = '2014-08-01';
	trtedt = '2014-09-15';
	output;
run;
```

What concomitant medication flags should we get? By referring to the previous classification chart,

- For ID 1, S equals to 2 and E equals to 3. So concomitant and post concomitant flag should be marked.
- For ID 2, S equals to 1 and E equals to 2. So pre-concomitant and concomitant flags should be marked.
- For ID 3, S equals to 3 and E equals to 1. So this is a data error and an error flag should be marked.

Now let's look at the actual code.

```
proc fcmp outlib=sasuser.funcs.date;
	subroutine CMFlag(cmsdt $,cmedt $,sdt $,edt $,prefl $,confl $,postfl
$,errfl $);
		outargs prefl,confl,postfl,errfl;

		length xxcmsdt xxcmedt $10;
		xxcmsdt = DateImpute(cmsdt);
		xxcmedt = DateImpute(cmedt);

		S = DateComp(xxcmsdt,sdt,edt);
		E = DateComp(xxcmedt,sdt,edt);

		call missing(prefl,confl,postfl,errfl);

		if S*E=0 then do;
			if S in (0,1) then prefl='Y';
			if S ^= 3 and E ^= 1 then confl='Y';
			if E not in (1,2) then postfl='Y';
		end;
		else do;
			if S = 1 then prefl='Y';
```

```
                        if S in (1,2) and E in (2,3) then confl='Y';
                        if E = 3 then postfl='Y';
                        if S > E then errfl='Y';
                end;

        return;
        endsub;
run;
```

As we can see with pre-define function `DateComp` and `DateImpute` the subroutine has become relatively short. With several lines of `IF-THEN` logic the cooking is done. Now let's test.

```
options cmplib=sasuser.funcs;
data _null_;
        set dts;
        retain prefl confl postfl errfl '';
        call CMFlag(cmstdtc,cmendtc,trtsdt,trtedt,prefl,confl,postfl,errfl);
        put (id cmstdtc cmendtc trtsdt trtedt prefl confl postfl errfl) (=);
run;
```

SAS log :

id=1 cmstdtc=2014-10-15 cmendtc=2014-11-15 trtsdt=2014-10-01 trtedt=2014-10-30 prefl= confl=Y postfl=Y errfl=

id=2 cmstdtc=2014-09 cmendtc=2014-10 trtsdt=2014-10-15 trtedt=2014-11-30 prefl=Y confl=Y postfl= errfl=

id=3 cmstdtc=2015 cmendtc=2013 trtsdt=2014-08-01 trtedt=2014-09-15 prefl= confl= postfl= errfl=Y

NOTE: There were 3 observations read from the data set WORK.DTS.
NOTE: DATA statement used (Total process time):
real time 0.26 seconds
cpu time 0.21 seconds

From the SAS log we can see all the results are matching with what we expected above. The declaration of CALL routine `CMFlag` is completed.

Debugging

Debugging is always a topic for tool development. As one of the most powerful weapons in debugging arsenal, `PUT` statement plays an important role in debugging. The good news is the `PUT` statement is supported by PROC FCMP. However `PUT` statement operates slightly different in PROC FCMP. First, `PUT` statement in PROC FCMP evaluates expressions by enclosing the expression in parentheses. For example, (X/100) will generate error in a DATA step `PUT` statement but it works perfectly fine in PROC FCMP. As a tradeoff, parentheses cannot be used for variables or format list as in DATA steps since it has been reserved for expression evaluation.

Second, several items cannot be used in PROC FCMP `PUT` statement, such as line pointers, format modifiers, column output, factored lists, iteration factors, overprinting, the `_INFILE_` option, or the special character `$`. Also It does not support features that are provided by the `FILE` statement options, such as `DLM=` and `DSD`.

More details of PROC FCMP debugging can be found in SAS online documents.

CONCLUSION

In this paper a series of application of PROC FCMP on date comparison, date imputation and creating concomitant medication flags have been introduced. These applications shrink the size of main body of program. And also they are portable and can be used as nails whenever, wherever you need them. Author hopes this paper can be trigger that inspires readers to think which of your daily routine jobs can be further simplified with PROC FCMP.

REFERENCES

Aileen L. Yam (2004), “A Simplified Way to Create Flags for the Three-Dimensional Prior, Concomitant, and Post Medication Classification”.

http://www.lexjansen.com/pharmasug/2004/CodersCorner/CC03.pdf

Kunal Agnihotri and Kenneth W. Borowiak (2013), “Extending the PRX Functions with PROC FCMP”

http://www.lexjansen.com/pharmasug/2013/CC/PharmaSUG-2013-CC23.pdf

SAS Online Documents

ACKNOWLEDGEMENT

The author would like to thank Ken Borowiak for his review and precious comments. Also he would like thank his wife Xinming Zhang for her encouragement to submit this paper.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of PPD.

Contact information

Comments, questions and additions are welcome. Contact the author at:

Jueru Fan

PPD

3900 Paramount Parkway

Morrisville, NC 27560

Phone: (919)-456-6450

Email: Jueru.Fan2@ppdi.com