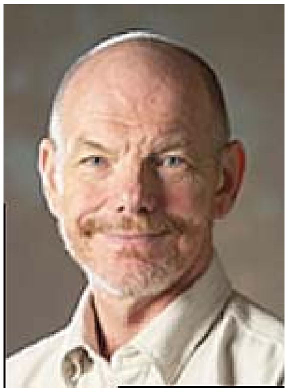

Peter Eberhardt is a long time SAS consultant, a certified SAS professional, and a SAS Alliance Partner. Although he enjoys working with SAS and teaching his EG classes, he does take time to speak and generally hob-nob with friends and strangers alike at SESUG, NESUG, and the SAS Global Forum. He is the Co-Chair of the upcoming SESUG 2008. And when he is not working, Peter partakes in the great outdoors, either paddling his home-made cedar canoe, or snowshoeing and camping with his home-made ash snowshoes.

An Intro to PROC FCMP

How many times have you tried to simplify your code with LINK/RETURN statements?

How much grief have you put yourself through trying to create macro functions to encapsulate business logic?

How many times have you uttered "If only I could call this DATA Step as a function"?

If any of these statements describe you, then the new features of PROC FCMP are for you.

If <u>none</u> of these statements describe you, then you <u>really</u> need the new features of PROC FCMP!

What I am about to describe is not so much a techie-tip, but more a brief introduction to the SAS PROC FCMP and its new ability to create your own DATA Step functions and CALL routines using DATA Step syntax. If you are a statistician you may be familiar with PROC FCMP; a number of SAS/Stat and SAS/ETS procedure already had the ability to call functions created with PROC FCMP. This ability now extends to the DATA Step programmer. For a concise overview of PROC FCMP see the paper by Jason Secosky, http://www2.sas.com/proceedings/forum2007/008-2007.pdf

You can create functions, as my example shows. You can also create CALL routines. You can pass in ARRAYS. And your functions can even be recursive. All in all, PROC FCMP will allow for creative encapsulation of your programming and business logic.

Getting started:

You need a library to store the functions. In my example I will be using a WORK. Although this is fine for developing your functions, you will need to keep them in a permanent location where all users can access them. You also have to tell SAS where to find the functions; the OPTIONS CMPLIB= statement does this.

Example:

I commonly have to do some simple division of two values in a data set. This is simple until we discover that some of the variables have unexpected missing values, or a value of 0. The following code shows the 'oldway' by explicitly checking values before each division; even with only 2 divisions my code is becoming repetitive. The 'newway' uses two functions I created to do the same divisions; note how cleaner the new code is.

```
* create a test data set;

DATA wonkyData;
    drop i;
    do i = 1 to 100;
      if mod(i,3) = 0
        then divisor = 0;
        else if mod(i,13) = 0
                      then divisor = .M;
             else divisor = i;
      dividend1 = (int(ranuni(1) * 10) * i ) + (int(ranuni(2) * 10) * i );
           dividend2 = dividend1 * 2;
      output;
    end;

RUN;

* process the data;
```

(Continued on page 2)

(Continued from page 1)

```
* divByMissingResult set to .X missing so it is easy to see;
* a common problem is it use IF divisor = . ;
DATA oldWay;
    set wonkyData;
    retain divByZeroResult      0
           divByMissingResult1  .X
           divByMissingResult2  .F ;
     drop  divByZeroResult divByMissingResult1 divByMissingResult2;
* only 2 tests, but it could be more;
    if divisor = 0          then answer1 = divByZeroResult;
    else if missing(divisor) then answer1 = divByMissingResult1;
    else                          answer1 = dividend1 / divisor;

    if divisor = 0          then answer2 = divByZeroResult;
    else if missing(divisor) then answer2 = divByMissingResult2;
    else                          answer2 = dividend2 / divisor;
run;

*----------------------------------------------;
* repeat, but use a function                   ;
*----------------------------------------------;

* outlib= tells FCMP where to store the functions;
PROC FCMP outlib=work.funcs.math;

* this function takes explict values for divByZeroResult and divByMissingResult;
FUNCTION divX(dividend, divisor, divByZeroResult,  divByMissingResult);
    if divisor = 0          then answer = divByZeroResult;
    else if missing(divisor) then answer = divByMissingResult;
    else                          answer = dividend / divisor;
    return(answer);
ENDSUB;

* this function uses fixed values for divByZeroResult and divByMissingResult;
FUNCTION div(dividend, divisor);
    if divisor = 0          then answer = 0;
    else if missing(divisor) then answer = .F;
    else                          answer = dividend / divisor;
    return(answer);
ENDSUB;
RUN;

* tell SAS where to look for functions;
options cmplib= work.funcs;
* much simpler code;
DATA newWay;
    set wonkyData;
     answer1 = divX(dividend1, divisor, 0, .X);
     answer2 = div(dividend2, divisor);
RUN;

* compare the results;
PROC COMPARE BASE=oldWay COMPARE=newWay;
RUN;
```