Paper 73-2019

Using PROC FCMP to the Fullest: Getting Started and Doing More

Arthur L. Carpenter
California Occidental Consultants, Anchorage, AK

ABSTRACT

The FCMP procedure is used to create user defined functions. Many users have yet to tackle this fairly new procedure, while others have only attempted to use only its simplest options. Like many tools within SAS®, the true value of this procedure is only appreciated after the user has started to learn and use it. The basics can quickly be mastered and this allows the user to move forward to explore some of the more interesting and powerful aspects of the FCMP procedure.

Starting with the basics of the FCMP procedure, this paper also discusses how to store, retrieve, and use user defined compiled functions. Included is the use of these functions with the macro language as well as with user defined formats. The use of PROC FCMP should not be limited to the advanced SAS user; even those fairly new to SAS should be able to appreciate the value of user defined functions.

KEYWORDS

Function, Routine, %SYSFUNC, %SYSCALL, FCMP

The INTRODUCTION

The FCMP procedure allows you to write, compile, and test DATA step functions and CALL routines that you can then use in the DATA step, with the macro language, and within a number of procedures that allow the use of functions.

In the simplest sense creating a function is fairly straightforward, and more complex functions are possible. The FCMP procedure is very powerful and the concepts are not that difficult.

What are Functions and Routines?

Functions are pre-written code tools that perform a specific task. They may or may not require arguments and they may or may not return values. There are two general types of functions and there is a bit of naming ambiguity associated with them. The less commonly used type is known as a 'routine' (or 'call routine'), while the more commonly used type of function is called a 'function'. SAS ships with over 450 predefined functions and routines. In addition you have the ability to define and store your own functions and routines through the use of the FCMP procedure.

The FCMP procedure is fairly new to SAS (SAS9.1) and even experienced SAS programmers have not always been quick to see the potential of user defined functions and routines. In actuality user defined functions can be written to solve complex coding problems. With the added ability to store the functions in libraries that can be shared, it becomes imperative that all SAS programmers know how to create user defined functions and routines.

PROC FCMP Basics

Through the use of PROC FCMP, functions and routines are compiled and stored in a special type of SAS data set. In Example 1 a function is defined that will convert weight in pounds to kilograms.

```
* Example 1: FCMP Basics;
proc fcmp outlib=funcsol.functions.conversions;❶
   function lb2kg(lb); ❷
      kg = lb/2.2; ❸
      return (kg); ❹
   endsub; ❺
   run;

options cmplib=(funcsol.functions); ❻

data kilos;
   set sashelp.class(keep=name age weight);
   Kilos = lb2kg(weight); ❼
   run;
```

Example 1

❶ The OUTLIB= option identifies the data set that is to contain the function definition. This data set is subdivided into packages (CONVERSIONS).
❷ The FUNCTION statement names the function (LB2KG) and identifies its incoming arguments (LB).
❸ Standard DATA step statements are used in the function definition. The variables (LB and KG) are independent of the DATA step's PDV and are maintained in a separate instance of memory.
❹ The RETURN statement identifies the value (here the value of the variable KG) that is to be returned by the function.
❺ The ENDSUB statement closes the function definition. A given PROC FCMP step can contain multiple function definitions.
❻ The CMPLIB system option is used to point to one or more data sets that contain function definitions. This option must be set before the function defined and stored by FCMP can be used.
❼ The function is used just as you would any other SAS function.

Returning a Character Value

By default functions work with numeric values. However it is possible for a function to act on character strings and to return character values.

```
* Example 2: Return a Character Value;
proc fcmp outlib=funcsol.functions.conversions; ❶
   function lb2kgC(lb) $; ❷
      length kg $10; ❸
      kg = catt(put((lb/2.2),6.2),'Kg');
      return (kg);
   endsub;
   run;

options cmplib=(funcsol.functions);

data kilos;
   set sashelp.class(keep=name age weight);
   Kilos = lb2kg(weight);
   Kg_c = lb2kgc(weight); ❹
   run;
```

Example 2

❶ This function will be added to the CONVERSIONS package (along with the function LB2KG) created in Example 1.
❷ The dollar sign ($) is used to indicate that the value to be returned will be character.
❸The LENGTH statement is used to set the length for the new character variable.
❹Based on the length of the returned value, KG_C will be a $10 variable.

The variables KG ❸ and LB ❷ will not appear in the data set KILOS. The variable names internally to a function are completely independent of any names used in the DATA step's PDV.

Functions with Multiple Arguments

Functions that have multiple arguments require only a slight expansion of the previous examples. In this example the Body Mass Index, BMI, is calculated based on the height and weight.

```
* Example 3: Function with Multiple Arguments;
proc fcmp outlib=funcsol.functions.conversions;
   function BMI(lb,ht) ; ❶
      return((lb*703)/(ht*ht)); ❷
   endsub;
   run;

options cmplib=(funcsol.functions);

data bmi;
   set sashelp.class(keep=name age weight height);
   BMI = bmi(weight,height); ❸
   run;
```

Example 3

❶ The arguments are comma separated, and the order that they are listed is important.
❷ Rather than create an intermediate variable, as was done in the previous examples, the conversion expression is written directly into the RETURN statement.
❸ The order of the arguments is very important.

Creating a Subroutine

Subroutines are created using the SUBROUTINE statement rather than the FUNCTION statement. Like the function BMI in Example 3, the BODYMASSINDEX routine described here calculates the body mass index based on the individual's height and weight (units are assumed to be inches and pounds).

```
Example 4: Creating a subroutine";
proc fcmp outlib=funcsol.functions.conversions;
   subroutine BodyMassIndex(w,h,b) ; ❶
      outargs b; ❷
      b=((w*703)/(h*h));❸
   endsub;
   run;

options cmplib=(funcsol.functions);

data bmi;
   set sashelp.class(keep=name age weight height);
   BMIndex=.; ❺
   call bodymassindex(weight,height,BMIndex); ❹
   run;
```

Example 4

❶ The SUBROUTINE statement defines both the incoming and outgoing arguments.
❷ When needed the OUTARGS statement is used to let the function know which values are to be returned by the subroutine.
❸ The body mass index is calculated and stored in the variable B, where it will be passed back to the calling program.
❹ The BODYMASSINDEX subroutine is executed using the CALL statement.

❺ The variable BMINDEX is initialized on the PDV by assigning it a missing value. The BODYMASSINDEX routine then returns a body mass index value which is placed into this variable. Variables to be returned by a routine MUST be initialized!

Including Logic and Character Arguments

A great many of the statements that you use in the DATA step are also available in FCMP functions. This includes IF-THEN/ELSE processing as is shown in this example. The FROMTO function converts weight from pounds to kilograms or from kilograms to pounds. The first argument is a code that tells the function which way to convert the value of the second argument.

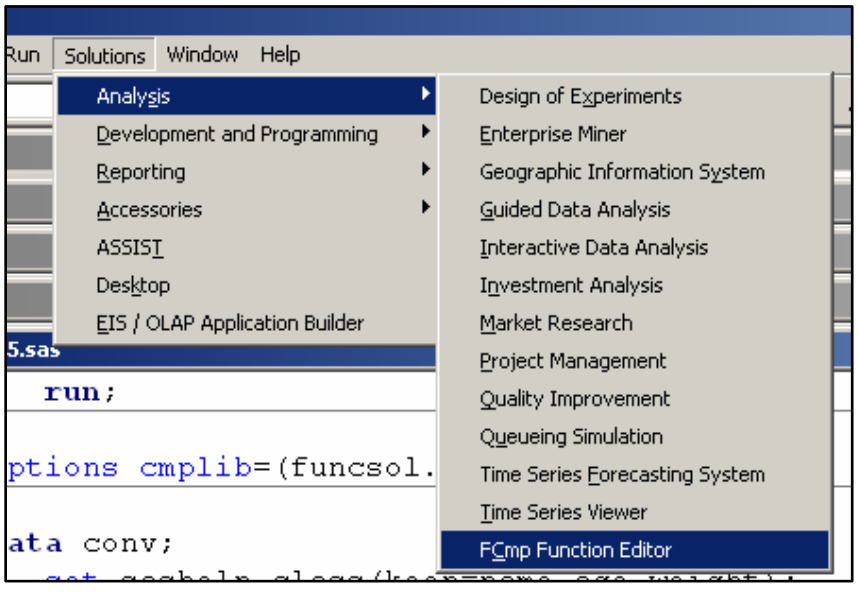
```
* Example 5: Create a simple function;
proc fcmp outlib=funcsol.functions.conversions;
   function fromto(code $, v); ❶
      if upcase(code)='LB2KG' then r = v/2.2;        /* Lb to KG */ ❷
      else if upcase(code)='KG2LB' then r = v*2.2;   /* KG to LB */
      else r=.;                                      /* unknown  */
      return (r);
   endsub;
   run;

options cmplib=(funcsol.functions);

data conv;
   set sashelp.class(keep=name age weight);
   Kilos = fromto('lb2kg',weight); ❸
   Pounds= fromto('kg2lb',kilos);
   run;
```

Example 5

❶ The dollar sign ($) following the first argument (CODE) lets the function know that the argument is character.
❷ The value of CODE is used to determine which type of conversion is to take place.
❸ The code of LB2KG causes the value of the second argument to be converted to kilograms, while KG2LB converts from kilograms to pounds.

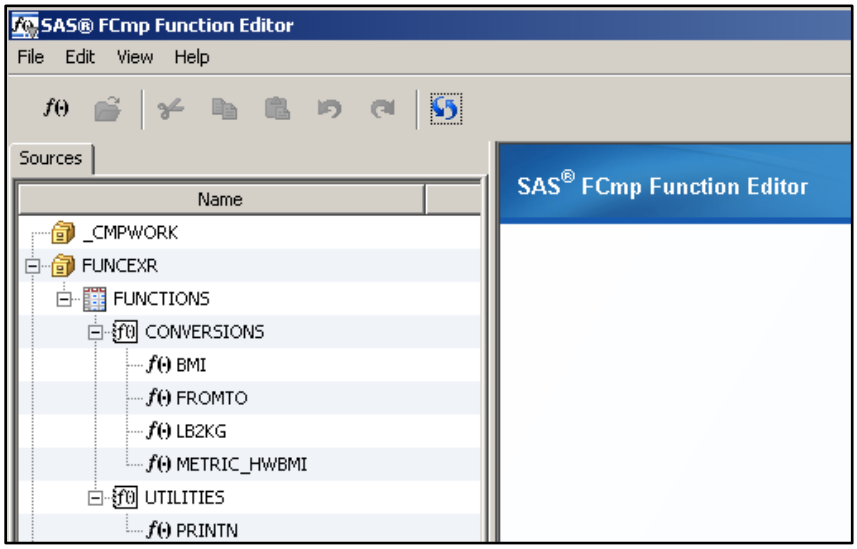
USING THE FCMP FUNCTION EDITOR

The FCMP Function Editor is an interactive tool that is available to you through the display manager. Using the pull down menus as shown in Example 6a select :
Solutions → Analysis → FCmp Function Editor.



Example 6a

Although am not a big fan of tools such as this one, it does allow you to do a number of function maintenance operations fairly easily.

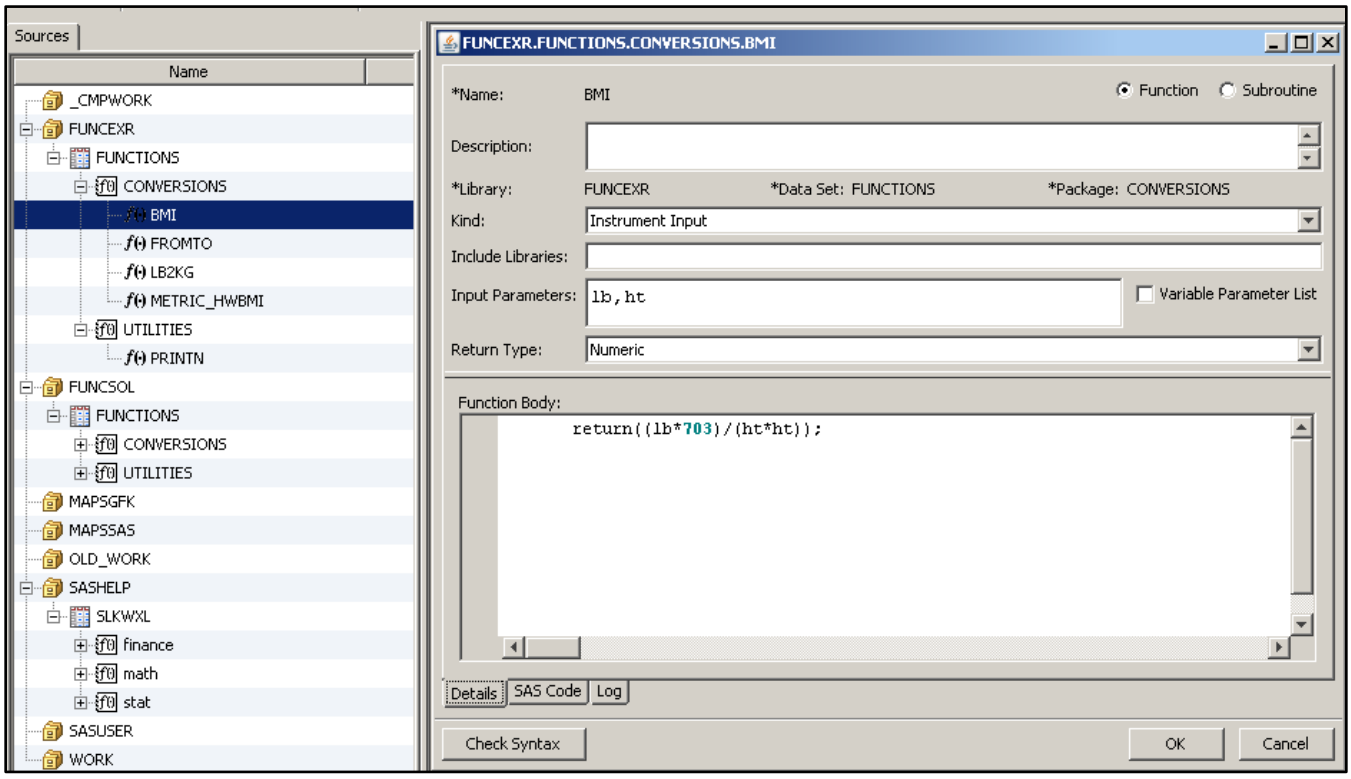


Once the function editor has started you will discover that the left panel (Example 6b) shows each of the function libraries and the functions that they contain.

These function libraries include a number of compiled functions that ship with SAS, and are worth additional exploration. Look in the SASHELP library for the data set SLKWXL.

Example 6b

By double clicking on a function in the left panel, you can bring up a dialogue box that contains the function definition. This definition includes the arguments (parameters) and coding. Tabs at the bottom of this dialogue box allow you to see the full PROC FCMP step that created this function or subroutine.



Example 6c

While you can use the FCMP Function Editor to edit or modify function definitions, my recommendation would be to not do so. Most programmers are used to working with source code and changes made interactively will not be reflected in the original source code.

DOING MORE WITH FUNCTIONS

By necessity this paper cannot cover all the capabilities of this procedure. The examples that follow should give you a good start on working with user defined functions and routines.

Deleting Function Definitions

As was mentioned earlier, function and subroutine definitions are stored in SAS data sets that have a special form. These data sets store the instructions for the generation of the functions. Within the data set there can be one or more packages and the functions are stored within these packages. This means that each function has a four level name: *library.dataset.package.function_name*. It also means that multiple functions with the same name, and with different definitions, can coexist at the same time in different packages in the same data set. Usually this is not wise, but it implies that we need the ability to delete functions as well as to create them.

```
proc fcmp outlib=funcsol.functions.conversions;
   deletefunc lb2kgc;
   deletefunc bodymassindex;
   run;
```

Here the DELETEFUNC statement is used to delete functions from the CONVERSIONS package which is in the FUNCTIONS data set.

Example 7

Executing a Macro

Macro language elements can be used with compiled functions, however like with other compiled elements (views, compiled DATA steps, SCL programs, and such) one must be careful to understand the interaction of the macro language with the compilation process.

Consider the following subroutine (PRINTN). We would like to pass it three arguments and then in turn pass those arguments as parameters to the macro %PRINTIT. This macro is a simple PROC PRINT with some TITLE statements, and is defined below (see Example 8c). Unfortunately THIS DOES NOT WORK!! We must remember that the function is first compiled and later (perhaps even next week) it is executed. A macro call such as the one shown here is executed during the compilation of the function and NOT during its execution.

```
proc fcmp outlib=funcsol.functions.utilities;
   subroutine printN(lib $, dsn $,num);
      * This will NOT work! The macro is
      * executed when the function is compiled;
      %printit(lib,dsn,num)
   endsub;
   run;
```

Example 8a

It is conceivable that you actually would want to execute a macro during the compilation of the function. Remember the macro language is primarily a code generator. So if your macro is used to write the code used by the function, then a macro call within the function definition would be appropriate.

When you want a function or subroutine to execute a macro, ***when the function executes***, you will need to use the special RUN_MACRO routine. This special routine allows you to name a macro to be executed and to list its parameters. The RUN_MACRO routine and the way it interfaces with the macro itself does have some limitations, but they are not too severe. First the parameter values passed into the macro from the function will be surrounded by quotes, which will almost certainly need to be removed. Secondly the macro itself is defined without parameters – as these are supplied by the function. As a result macros written to be called by functions will tend to have limited utility elsewhere.

Here the PRINTN subroutine has been rewritten to use the RUN_MACRO routine. The macro name is quoted as it is a constant in this example. And the macro name is followed by the parameter values.

```
proc fcmp outlib=funcsol.functions.utilities;
   subroutine printN(lib $, dsn $,num);
      rc=run_macro('printit',lib,dsn,num);
   endsub;
   run;
```

Example 8b

```
%macro printit(); ❶
%put &lib &dsn; ❷
   %let lib  = %sysfunc(dequote(&lib)); ❸
   %let dsn  = %sysfunc(dequote(&dsn));
   %let num  = %sysfunc(dequote(&num));
   %if &num= %then %let num=max;
   title2 "&lib..&dsn";
   title3 "First &num Observations";
   proc print data=&lib..&dsn(obs=&num);
      run;
%mend printit;
```

Example 8c

❶ The macro itself is defined with parentheses, but without parameters. The names of the arguments in the subroutine are the same as the parameters in the macro, and they are passed into the macro directly.
❷ This %PUT statement is included here merely to demonstrate that the parameter values are surrounded by quotes.
❸ Because the process of calling a macro through a function inserts quotes around parameter values, these will need to be removed. Here the DEQUOTE function is used to remove the quotes from the parameters.

Formats that Call Functions

Starting in SAS 9.3 it is possible to call a function through a format. Although limited to functions that have at most a single argument, this can still be very advantageous. Functions can now be used wherever you can use a format; including in procedure steps. Because, through the use of formats, you can now execute functions from within procedure steps, your functions are no longer limited to DATA steps or procedures that execute SAS Language elements, such as PROC REPORT's compute block.

In Example 1 the function LB2KG() is used to convert pounds to kilograms. This function takes a single argument (pounds) and returns the converted value. In the format POUNDS2KG. shown here, all the data values are mapped to the function using the OTHER keyword of the VALUE statement. Notice that the function is called inside of square brackets which are not quoted. In this example the PROC PRINT will show the weight in KiloGrams, although the data are still stored in pounds.

This gives us the ability to form formats that convert continuous values to continuous values.

```
* Exercise 9: Execute a Function
*              Using a Format';
proc format;
   value pounds2kg
      other=[lb2kg()];
   run;

options cmplib=(funcsol.functions);

title2 'Weight in Kg';
proc print data=sashelp.class;
   var name age weight;
   format weight pounds2kg.;
   run;
```

Example 9

Returning Multiple Values

Functions return a single value and generally subroutines do not return any values. However the subroutine in Example 4 used the OUTARGS statement to return a value. An expansion on that example allows subroutines to return multiple values.

The subroutine METRIC_HWBMI accepts height and weight in English units (inches and pounds) and returns the height in meters, the weight in kilograms, as well as the Body Mass Index.

```
proc fcmp outlib=funcsol.functions.conversions;
   subroutine metric_HWBMI(h,w,mh,mw,bmi); ❶
      outargs mh, mw, bmi; ❷
      mh = h*.0254;
      mw = w*.4536;
      bmi= mw /(mh*mh);
   endsub;
   run;

options cmplib=(funcsol.functions);

data multiple;
   set sashelp.class(keep=name age height weight);
   HeightMeters=.; ❸
   WeightKilos=.;
   BMI=.;
   call metric_hwbmi(height, weight, HeightMeters, WeightKilos, BMI); ❹
   run;
```

Example 10

❶ The subroutine arguments are listed (both incoming and those that are calculated).
❷ The OUTARGS statement lists those values that are to be returned. Variables that are to receive values from a routine must be initialized before the routine is called.
❸ The variables that will be calculated by the call routine are initialized.
❹ The three calculated values are returned from the subroutine back to the DATA step's PDV.

Using FCMP Functions with %SYSFUNC and %SYSCALL

The SAS macro language has the ability to access most of the DATA step functions and call routines indirectly through the use of the macro language function %SYSFUNC and the macro language statement %SYSCALL. Through these same 'bridging' tools, functions created through the use of PROC FCMP are also available to the macro language - you do not need to do anything special to make your FCMP functions and routines available to the macro language.

In Example 3 the BMI function calculates the Body Mass Index based on a height and weight. This same function can be used in the macro language by enclosing the BMI function in a call to the %SYSFUNC function.

```
* Exercise 11: Using Functions with %SYSFUNC;
options cmplib=(funcsol.functions);

%let ht = 69;
%let wt = 112.5;
%let bmi= %sysfunc(bmi(&wt,&ht));
%put &=bmi;
```

The value returned by the BMI function is stored in the macro variable %BMI.

Example 11a

In Example 10 the HWBMI routine returns not only the Body Mass Index, but also returns the height and weight in metric values. Using the %SYSCALL statement (%SYSFUNC is a macro function, while %SYSCALL is a macro statement), we can call this routine from within the macro language. As was the case when returning values with a routine in a DATA step (Exercise 10), the values returned by the HWBMI routine (in this case macro variables) must be initialized prior to the execution of the routine. Notice also that routine arguments associated with a %SYSCALL are assumed to be macro variables and must be specified without the ampersand (&).

```
* Exercise 11: Using a subroutine with %SYSCALL;
options cmplib=(funcsol.functions);

%let ht = 69;
%let wt = 112.5;
%let mh=.;
%let mw=.;
%let bmi=.;
%syscall metric_hwbmi( ht,  wt,  mh,  mw,  bmi);
%put &ht, &wt, &mh, &mw, &bmi;
```

Example 11b

There were some memory issues with the use of %SYSCALL with FCMP routines in early versions of FCMP, however these seem to have been solved with the release of SAS9.4. FCMP routines cannot currently be used within PROC DS2 (use a method instead).

SUMMARY

The FCMP procedure allows us to write our own functions and routines. It is very flexible and provides us with a tool that can be applied in any number of ways. When writing functions we are no longer constrained to writing macro functions. We can now write functions and store them in libraries where everyone in our group can have access to them.

ABOUT THE AUTHOR

Art Carpenter's publications list includes five books, and numerous papers and posters presented at SUGI, SAS Global Forum, and other user group conferences. Art has been using SAS® since 1977 and has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Advanced Professional programmer, and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

Certified Advanced Programmer *for* SAS®9

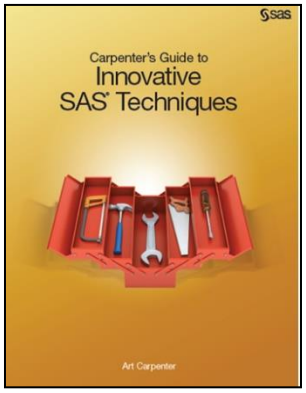
Certified Clinical Trials Programmer Using SAS®9

View Art Carpenter's paper presentations page at:
http://www.sascommunity.org/wiki/Presentations:ArtCarpenter_Papers_and_Presentations

ACKNOWLEDGEMENTS

Thank you to Radha Chebolu of Ad hoc Reporting and Data Delivery, for help with text corrections.

REFERENCES



A further discussion of FCMP functions can be found in the book Carpenter's Guide to Innovative SAS® Techniques by Art Carpenter (SAS Press, 2012).

The classic introduction to the FCMP procedure was written in 2007 by Jason Secosky. Although an older paper, this is still a great place to start when first learning about the FCMP procedure.
Secosky, Jason, 2007, "User-Written DATA Step Functions", published in the *Proceedings of the SAS Global Forum 2007 Conference*, Cary, NC: SAS Institute Inc., paper 008-2007.
http://www2.sas.com/proceedings/forum2007/008-2007.pdf

Two other good introductory papers include:
Adams, John H., 2010, "The new SAS 9.2 FCMP Procedure, what functions are in your future?", *Proceedings of the Pharmaceutical SAS User Group Conference* (PharmaSUG), 2010, Cary, NC: SAS Institute Inc., paper AD02.
http://www.lexjansen.com/pharmasug/2010/ad/ad02.pdf

Eberhardt, Peter, 2011, "A Cup of Coffee and Proc FCMP: I Cannot Function Without Them", published in the *Proceedings of the Pharmaceutical SAS Users Group Conference* (PharmaSUG), 2011,Cary, NC: SAS Institute Inc., paper TU07. http://www.pharmasug.org/proceedings/2011/TU/PharmaSUG-2011-TU07.pdf

The primary documentation for PROC FCMP is:
SAS Institute Inc. *The Base SAS 9.4 Procedures Guide, Seventh Edition*. Cary, NC: SAS Institute Inc. Available at:
http://documentation.sas.com/api/docsets/proc/9.4/content/proc.pdf?locale=en#nameddest=bookinfo

An advanced topic paper on FCMP for SAS 9.3 can be found at:
Secosky, Jason, "Executing a PROC from a DATA Step", published in the *Proceedings of the 2012 SAS Global Forum Conference*, Cary, NC, SAS Institute Inc, Paper 227-2012.
http://support.sas.com/resources/papers/proceedings12/227-2012.pdf

The relative merits of using FCMP functions to store hash tables (as well as how to do it) are discussed in:
Carpenter, Arthur L., 2018, "Using Memory Resident Hash Tables to Manage Your Lookup Control Files", published in the *Proceedings of the 2018 SAS Global Forum Conference*, Cary, NC, SAS Institute Inc, Paper 2399-2018.

Trademark Information