PhUSE US Connect 2019

Paper CT13

An Antidote for Huge Clinical Trial Data Processing: Enclosing Hash in FCMP

Premanand Kumaravel, Symbiance Inc, New Jersey, USA

ABSTRACT:
In clinical trial handling huge data either in SDTM or ADaM or TLF will consume considerable amount of time and system resources, which will lead to further complications and other constraints. Therefore, going through huge volume of data in a short span of time with SAS and utilizing optimal system processing will be an added boon. Hashing technique coupled with user defined subroutines through FCMP Procedure will enhance a user's capability to handle large data by optimizing system resources and retaining the simplicity of their programs. This paper will elucidate with examples demonstrating operations performed with hashing in Proc FCMP and how it improves performance and organises an existing program while handling huge clinical trial data efficiently.

INTRODUCTION:
FCMP (Function Compiler Procedure) of SAS® enables programmer to create user defined SAS functions which can be used in SAS DATA steps or some procedures. Once function created from this procedure is compiled from their storage location it can be reused in any of the SAS DATA steps or procedures which has access to its location. Moving to DATA step component object, hash and hash iterator objects where introduced in SAS® from SAS 9.0. Hash and hash iterator objects enables swift and effective searching, retrieval and storing data based on indexes (lookup keys) created.

In SAS 9.3, hash and hash iterator objects were available in FCMP procedure. This enabled enclosing hash objects in FCMP which brings in most of hash objects features wrapped in within a single procedure. Here in this paper we explore new ways to implement hash in FCMP and how it can be utilized to streamline performance while handling large datasets and implementing other predefined functions like 'RUN_MACRO' within FCMP.

HASH IN FCMP:
Within FCMP some of the statements and methods supported for hash object are:

DECLARE statement	To declare a hash object by creating instance and initializing it
DEFINEDATA method	Dataset specified to be sorted is defined
DEFINEKEY method	Key variables are defined for the hash object
DEFINEDONE method	Its specified when all data definitions and keys are complete
ADD method	Specified data of the key is added to the hash object
DELETE method	Associated hash or hash iterator object gets deleted
CHECK method	Specified key is checked in hash object if stored
FIND method	Specific key is determined if stored in hash object
CLEAR method	Without deleting hash object instance, all items from it is cleared
REMOVE method	For the specified key, associated data are removed from the hash object
REPLACE method	For the specified key, associated data gets replaced to new data

Similarly, for hash iterator objects following methods and statements are supported:

DECLARE statement	To declare a hash iterator object by creating instance and initializing it
FIRST method	First value is returned for the primary hash object
LAST method	Last value is returned for the primary hash object
NEXT method	Next value is returned for the primary hash object
PREV method	Previous value is returned for the primary hash object

Whenever Hash and Hash Iterator utilized it involves more syntax and steps. But when hash is enclosed within FCMP it provides a strong platform with more simplified and generic approach to coding. For better efficiency hashing in FCMP can be utilized for faster processing of large data and simplification of certain parts of the code. Below example illustrates a simple use of Hash within FCMP.

EXAMPLE FOR HASHING IN PROC FCMP (VALUE LOOKUP VIA ENCLOSING HASH IN FCMP):

```
/*** Defining HASH within PROC FCMP to create SAS User Defined Function ***/
/*** User Defined function to populate First Dose Date ***/
%macro idts(); /*** Macro for Sorting Dataset ***/
      %let ids = &ip;
      %let odats = &op;
      data _null_;
            if 0 then set &ids (keep = _all_);
            declare hash srt (dataset:"&ids", ordered:"&ascdsc",hashexp:12) ;
            srt.definekey (&invar);
            srt.definedata (all:"Yes") ;
            srt.definedone ();
            srt.output(dataset:"&odats");
            stop;
      run;
%mend idts;

proc fcmp outlib = work.sfn.bd_srt; /*** function created to Sort Dataset ***/
            function srtds(ids $,odats $);
            rc = run_macro('idts',ids,odats);
            return(rc);
          endsub;
run;

proc fcmp outlib=work.sfn.fnd_fval;
      function fnd_fval(subjid $) $40;
      declare hash fnd(dataset: "work.exdose"); /*** Declaring Hash 'fnd' ***/
      declare hiter fst("fnd"); /*** Declaring Hash Iterator 'fst' ***/
      rc=fnd.definedata("exstdtc"); /*** Defining variable to store ***/
      rc=fnd.definekey("subjid"); /*** Defining key variable ***/
      rc=fnd.definedone(); /*** Completion of defining Hash 'fnd' ***/
      rc = fst.first();
      rc=fnd.find();
      if rc = 0 then return(exstdtc);
      else return('NO FIRST DOSE DATE'); endsub;
quit;

/*** Compiling User Defined Function from its Respective Library ***/
options cmplib = (work.sfn);
/*** Calling User Defined Function within Datastep ****/
%let invar = "SUBJID","EXSEQ","EXSTDTC","EXENDTC";
%let ascdsc = A;
%let ip = exdose_us;  %let op = exdose;

data _null_; ids = "&ip"; odats = "&op"; rc = srtds(ids,odats); run;

data dm_ex; set demog; EXSTDTC = fnd_fval(subjid);  run;
```

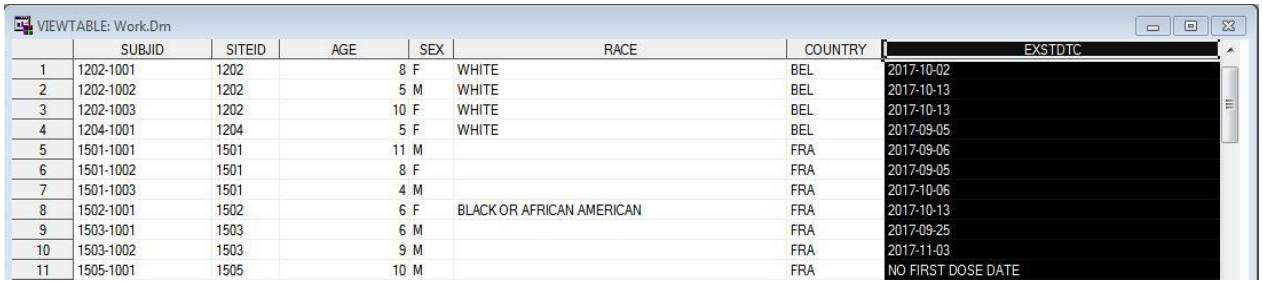
VIEWTABLE: Work.Demog

	SUBJID	SITEID	AGE	SEX	RACE	COUNTRY
1	1202-1001	1202	8	F	WHITE	BEL
2	1202-1002	1202	5	M	WHITE	BEL
3	1202-1003	1202	10	F	WHITE	BEL
4	1204-1001	1204	5	F	WHITE	BEL
5	1501-1001	1501	11	M		FRA
6	1501-1002	1501	8	F		FRA
7	1501-1003	1501	4	M		FRA
8	1502-1001	1502	6	F	BLACK OR AFRICAN AMERICAN	FRA
9	1503-1001	1503	6	M		FRA
10	1503-1002	1503	9	M		FRA
11	1505-1001	1505	10	M		FRA
12	1506-1001	1506	7	F		FRA
13	1506-1002	1506	4	F		FRA
14	1801-1001	1801	10	M	WHITE	HUN
15	1801-1002	1801	7	M	WHITE	HUN
16	1801-1003	1801	8	M	WHITE	HUN

VIEWTABLE: Work.Exdose

	SUBJID	EXSEQ	EXDOSE	EXSTDTC	EXENDTC
1	1202-1001	1	4	2017-10-02	2017-10-08
2	1202-1001	2	8	2017-10-09	2017-10-15
3	1202-1001	3	12	2017-10-16	2017-11-02
4	1202-1001	4	8	2017-11-03	2017-11-26
5	1202-1001	5	8	2017-11-27	2017-12-17
6	1202-1002	1	4	2017-10-13	2017-10-19
7	1202-1002	2	8	2017-10-20	2017-10-26
8	1202-1002	3	12	2017-10-27	2017-11-09
9	1202-1002	4	16	2017-11-10	2017-11-23
10	1202-1002	5	16	2017-11-24	2017-12-07
11	1202-1003	1	4	2017-10-13	2017-10-19
12	1202-1003	2	8	2017-10-20	2017-10-27
13	1202-1003	3	12	2017-10-28	2017-11-10
14	1202-1003	4	16	2017-11-11	2017-11-19
15	1202-1003	5	16	2017-11-20	2017-11-26
16	1202-1003	6	20	2017-11-27	2017-12-07

Figure 1. Demog and Exdose sample data



VIEWTABLE: Work.Dm

	SUBJID	SITEID	AGE	SEX	RACE	COUNTRY	EXSTDTC
1	1202-1001	1202	8	F	WHITE	BEL	2017-10-02
2	1202-1002	1202	5	M	WHITE	BEL	2017-10-13
3	1202-1003	1202	10	F	WHITE	BEL	2017-10-13
4	1204-1001	1204	5	F	WHITE	BEL	2017-09-05
5	1501-1001	1501	11	M		FRA	2017-09-06
6	1501-1002	1501	8	F		FRA	2017-09-05
7	1501-1003	1501	4	M		FRA	2017-10-06
8	1502-1001	1502	6	F	BLACK OR AFRICAN AMERICAN	FRA	2017-10-13
9	1503-1001	1503	6	M		FRA	2017-09-25
10	1503-1002	1503	9	M		FRA	2017-11-03
11	1505-1001	1505	10	M		FRA	NO FIRST DOSE DATE

Figure 2. DM data with respective first dose date values

Above example shows how hashing is integrated within FCMP procedure to yield a user defined function to find the first given dose date for a subject. By this way one can drastically reduce the steps involved in conventional method with an added advantage of minimal use of system resources. Storing and compiling user defined functions can also be easily referenced to respective library location. This provides more flexibility to a programmer for creating, storing and compiling user defined functions and conserves times in whole. Thus, this method eliminates multiple sort procedure steps, merging datasets through DATA step or using Proc SQL.

Similarly, to find the last dose date for a subject, within the FCMP procedure one needs to change the 'ordered' value to 'Descending' which will return the last value of the given primary hash object.

```
/*** Highlighted parts are changed as per requirement ***/

%macro idts();
      %let ids = &ip;
      %let odats = &op;
      data _null_;
            if 0 then set &ids (keep = _all_);
            declare hash srt (dataset:"&ids", ordered:"&ascdsc",hashexp:12) ;
            srt.definekey (&invar);
            srt.definedata (all:"Yes") ;
            srt.definedone ();
            srt.output(dataset:"&odats");
            stop;
      run;
%mend idts;

proc fcmp outlib = work.sfn.bd_srt;
            function srtds(ids $,odats $);
            rc = run_macro('idts',ids,odats);
            return(rc);
          endsub;
run;

proc fcmp outlib=work.sfn.fnd_lval;
      function fnd_lval(subjid $) $40;
      declare hash fnd(dataset: "work.exdose");
      declare hiter lst("fnd");
      rc=fnd.definedata("exendtc");
      rc=fnd.definekey("subjid");
      rc=fnd.definedone();
      rc = lst.first();
      rc=fnd.find();
      if rc = 0 then return(exendtc);
      else return('NO LAST DOSE DATE'); endsub;
quit;

options cmplib = (work.sfn);

%let ascdsc = D; /*** Macro parameter value for descending order ***/
%let ip = exdose_us;  %let op = exdose;
```

```
data _null_;
      ids = "&ip"; odats = "&op";
      rc = srtds(ids,odats);
run;

data dm_ex; set dm_ex; RFENDTC = fnd_oval(subjid);  run;
```

VIEWTABLE: Work.Dm_ex

	SUBJID	SITEID	AGE	SEX	RACE	COUNTRY	EXSTDTC	EXENDTC
1	1202-1001	1202	8	F	WHITE	BEL	2017-10-02	2017-12-17
2	1202-1002	1202	5	M	WHITE	BEL	2017-10-13	2017-12-07
3	1202-1003	1202	10	F	WHITE	BEL	2017-10-13	2017-12-07
4	1204-1001	1204	5	F	WHITE	BEL	2017-09-05	2018-01-25
5	1501-1001	1501	11	M		FRA	2017-09-06	2018-01-25
6	1501-1002	1501	8	F		FRA	2017-09-05	2017-12-21
7	1501-1003	1501	4	M		FRA	2017-10-06	2017-12-11
8	1502-1001	1502	6	F	BLACK OR AFRICAN AMERICAN	FRA	2017-10-13	2017-12-05
9	1503-1001	1503	6	M		FRA	2017-09-25	2017-10-16
10	1503-1002	1503	9	M		FRA	2017-11-03	2017-12-05
11	1505-1001	1505	10	M		FRA	NO FIRST DOSE DATE	NO LAST DOSE DATE

Figure 3. DM_EX data with respective first and Last dose date values

MAKING HASHING IN FCMP DYNAMIC:

Within PROC FCMP by using RUN_MACRO routine, provides a means to call the macro within this procedure. RUN_MACRO function executes the macro and returns a value to DATA step. Creating customised reusable and independent subroutines for a user defined function will enable programmers to maintain complex codes and increases flexibility for the usage of that function for many scenarios and not just for tailored setups.

To make Hash and Hash Iterator object customised within FCMP, RUN_MACRO function can be utilized. If user intends to utilize the function created for multiple scenarios where it requires to change the given input parameters, then this method will be best suited.

When a user defined function of this kind executed, the function creates local macro variable with identical name and value of the variables in the parameter list from RUN_MACRO. Following it macro gets executes and each macro variable values are written back to equivalent variables in the function. The return code (rc) variable specifies whether the method was success or failure. When its zero it denotes success and a nonzero value denotes failure.

SYNTAX OF RUN_MACRO:

rc = RUN_MACRO('macro_name' <, variable_1, ..., variable_n>);

Macro_name – name of the macro to be executed.

Variables – specifies optional PROC FCMP variables, which are set by macro variables of the exact name.

EXAMPLE FOR DYNAMIC HASHING IN PROC FCMP:

```
/*** Creating User Defined Function for splitting a dataset based on given
category variable distinct values ***/
/*** Defining Macro with HASH Objects ***/
%macro data_splt;
      /*** Input parameters ***/
      %let ip = &inpsrt;
      %let asrt = &keyvar;
      %let pfx = &prfxval;

      proc sort data = &inpds out = &ip;
            by &asrt;
      run;

      data _null_;
            if 0 then set &ip;
            declare hash splt(ordered:'Y', hashexp:12);
            splt.definekey(&srtvar);
            splt.definedata(&keepvar);
            splt.definedone();
            do _n_ = 1 by 1 until(last.&asrt);
            set &ip;
```

```
            by &asrt;
            splt.add();
            end;
            /*** Saving Hash Object Data into multiple datasets ***/
            splt.output (dataset: compress("&pfx"||&asrt));
      run;
%mend;


/*** User Defined Function with RUN_MACRO routine***/
proc fcmp outlib = work.sfn.split_ds;
      /*** Function with same macro variable parameters ***/
      function split_ds(ip $,asrt $,pfx $);
      /*** RUN_MACRO function with macro name and macro variable parameters ***/
      rc = run_macro('data_splt',ip,asrt,pfx);
      return(rc);
      endsub;
run;

/*** Assigning Macro parameter variable values ***/
%let inpds = work.lab;
%let inpsrt = labs;
%let srtvar = "subjid","lbseq","_n_";
%let keepvar ="subjid","lbseq","lbtestcd","lbtest","lborres","lborresu","lbstat";
%let keyvar = lbcat;
%let prfxval = CAT_;

/*** Compiling User Defined Function from its Respective Library ***/
options cmplib = (work.sfn);

data _null_;
      ip = "&inpsrt";
      asrt = "&keyvar";
      pfx = "&prfxval";
      rc = split_ds(ip,asrt,pfx); /*** Calling Function ***/
run;
```

Above example shows how macro can be integrated into FCMP using RUN_MACRO function. User defined functions of this kind are dynamic and not created for a specific dataset and its corresponding variable. Here in the given example function, it is created for splitting a dataset based on given categorical variable. And as illustrated in the example laboratory dataset is split based on lab category variable 'LBCAT'. This function takes into consideration of specified data related to the key variable and adds them to the respective hash object. Each LBCAT values is a key and relevant observations are accordingly added under the respective keys using hash add method.

VIEWTABLE: Work.Lab

	SUBJID	LBSEQ	LBTESTCD	LBTEST	LBCAT	LBORRES	LBORRESU	LBSTAT
1	1202-1001	1	LBCSAMP	Laboratory Test Sample	URINALYSIS			NOT DON
2	1202-1001	2	HCG	Choriogonadotropin Beta	URINALYSIS			NOT DON
3	1202-1001	3	HCG	Choriogonadotropin Beta	URINALYSIS			NOT DON
4	1202-1001	4	HCG	Choriogonadotropin Beta	URINALYSIS			NOT DON
5	1202-1001	5	HCG	Choriogonadotropin Beta	URINALYSIS			NOT DON
6	1202-1001	6	CL	Chloride	CHEMISTRY	100	mEq/L	
7	1202-1001	7	CL	Chloride	CHEMISTRY	100	mEq/L	
8	1202-1001	8	CL	Chloride	CHEMISTRY	100	mEq/L	
9	1202-1001	9	CL	Chloride	CHEMISTRY	100	mEq/L	
10	1202-1001	10	K	Potassium	CHEMISTRY	3.3	mEq/L	
11	1202-1001	11	K	Potassium	CHEMISTRY	3.3	mEq/L	
12	1202-1001	12	K	Potassium	CHEMISTRY	3.3	mEq/L	
13	1202-1001	13	K	Potassium	CHEMISTRY	3.3	mEq/L	
14	1202-1001	14	SODIUM	Sodium	CHEMISTRY	143	mEq/L	
15	1202-1001	15	SODIUM	Sodium	CHEMISTRY	143	mEq/L	
16	1202-1001	16	SODIUM	Sodium	CHEMISTRY	143	mEq/L	
17	1202-1001	17	SODIUM	Sodium	CHEMISTRY	143	mEq/L	
18	1202-1001	18	ALP	Alkaline Phosphatase	CHEMISTRY	184	U/L	
19	1202-1001	19	ALP	Alkaline Phosphatase	CHEMISTRY	184	U/L	
20	1202-1001	20	ALP	Alkaline Phosphatase	CHEMISTRY	184	U/L	
21	1202-1001	21	ALP	Alkaline Phosphatase	CHEMISTRY	184	U/L	
22	1202-1001	22	ALT	Alanine Aminotransferase	CHEMISTRY	16	U/L	
23	1202-1001	23	ALT	Alanine Aminotransferase	CHEMISTRY	16	U/L	
24	1202-1001	24	ALT	Alanine Aminotransferase	CHEMISTRY	16	U/L	
25	1202-1001	25	ALT	Alanine Aminotransferase	CHEMISTRY	16	U/L	
26	1202-1001	26	AST	Aspartate Aminotransferase	CHEMISTRY	18	U/L	
27	1202-1001	27	AST	Aspartate Aminotransferase	CHEMISTRY	18	U/L	

Figure 4. LAB sample data

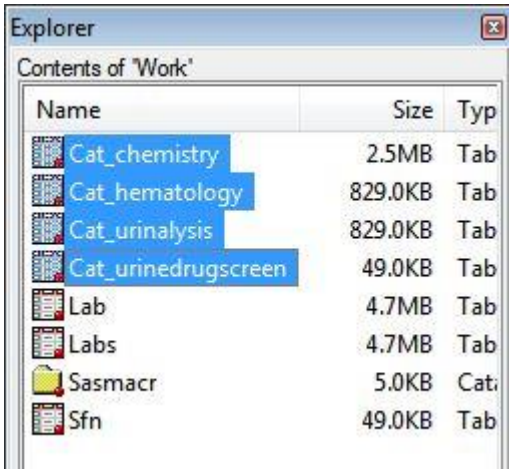


Figure 5. Split datasets with respective Categories as dataset names and corresponding data

User doesn't require to update the FCMP procedure statements based on the data. Rather user is only required the pass the necessary parameter values for the macro variables in the RUN_MACRO routine. This enables flexibility and conserves time for a programmer and utilizing hash object further reduces the processing time and streamlines coding. If the initial example is taken comparatively with current one, the differences can be noticed on how the function '*fnd_fval(subjid)*' created with no macro integration is rigid and couldn't be made as common. Whereas the later example with macro integration makes it more dynamic.

When large datasets are processed by conventional methods it will take up substantial amount of system resources. Incorporating user defined functions based on hash objects will yield optimised system resource usage with well streamlined programming.

WHEN TO USE HASH IN FCMP:

One of the major factors affecting the efficiencies of the methods used in SAS is due to dataset size. And the other factor to be considered will be number of variables in a dataset. When data size is small and contains many variable or an extremely large dataset with few variables will be more well suited for the method discussed here.

But considerably if a small dataset with many variables can be handled in ease by conventional DATA steps or SQL procedures. The consideration for Hash objects is preferred with larger datasets with lesser variables and they are more efficient. Hash objects does works for datasets with extremely large data and many variables yet the efficiency which we get won't be like larger datasets with lesser variables.

HOW SYSTEM RESOURCE UTILIZED CAN BE OPTIMISED BY HASHING IN FCMP:

Using conventional method for a large dataset with few variables will utilize all of system resources and number of steps involved will be more which directly increases processing time as the dataset will parse through every step which will keep on prolonging the time taken. Whereas when a hash object is integrated into a user defined function via FCMP procedure, it drastically shortens the use of conventional steps and procedures instead a simple function will be called.

The steps involved in creating the function with hash objects is complex but when its stored and complied in a library it can be accessed with ease and complied accordingly when needed to be used. Thus, when its used by a programmer the steps created are short and simple with only reference to the user defined function by avoiding multiple DATA steps, sort procedures, SQL procedures or use of redundant steps which will drastically streamline the coding of a programmer and keeping it simple and clean. Since hash object entries are placed in system memory, searching or finding data that linking to its key will occur much quicker when compared to data read from disk. The major benefit is increase in efficiency while utilizing this method which directly reduces system resource and processing time taken.

MEMORY MANAGEMENT:

In utilizing hash objects memory allocation is basically based on the need. Hence system allocated memory for SAS will be provided for hash object processing and this is also one of the deciding factors to how much data will it process. When the given memory is sufficient SAS will load hash objects with its respective data into it.

If the given memory is not sufficient as data corresponding to the hash object loaded is extremely large, SAS will try to load data until the available memory gets exhausted and will stop executing DATA step and error message will be written log concerning memory failure. This also to be considered while using hash objects because if system memory is very low or not capable to handle large data then merits of hash and hash iterator objects can't be seen.

EXAMPLE MEMORY ERROR NOTE:

```
ERROR: Hash object added 73943213 items when memory failure occurred.
FATAL: Insufficient memory to execute DATA step program. Aborted during the EXECUTION phase.
ERROR: The SAS System stopped processing this step because of insufficient memory.
NOTE: DATA statement used (Total process time):
real time           1:30.2
cpu time            45.89 seconds
```

FUNCTION OR MACRO:

When a macro is created and referenced it should be first included its location to be compiled and later that macro is called. But when a user defined function is created it will be stored as package in the provided location and later while calling the function only needed to do is compile and its ready to use.

As for FCMP procedure since it already supports some of hash objects methods and statements it will be a better option to create user defined functions which can be tailored for specific usage in analysing large data. Based on the requirement choosing between user defined function or macro is one's preference.

ADVANTAGES:

1. Simplification and streamlining of programmer's code.
2. Multiple DATA steps or procedures can be integrated into FCMP procedure.
3. SAS® Hash and Hash Iterator objects readily available to be used within FCMP procedure.
4. Easy to store and compile functions from desired location.
5. Can also be used within a macro by using %SYSFUNC or %SYSCALL.
6. Functions created based on hash objects can handle large volume of data by reducing processing time and system memory used.
7. User defined functions created by integrating macros via RUN_MACRO routine allows the function to operate more flexible and dynamic.

DISADVANTAGES:

1. When Hash and Hash Iterator used complexity of the process increases.
2. System memory might cause issues if insufficient memory is allocated for hash object processing within a user defined function.
3. Not a game changer in processing speed for small or moderate sized data when utilizing hash object-based functions.
4. Hashing in FCMP can't be used for all scenarios as like macros hence it is based on necessity.

CONCLUSION:

User defined function created using FCMP procedures are some of the most versatile methods available in SAS®. In this paper we have mainly explored on how user defined functions with hash objects work and how it can be implemented for handling large datasets. The examples and methods discussed here are mainly to emphasise that there are other ways to address a scenario and it can be widened based on the requirement. Hash object methods and statements within FCMP does make programmers code void of multiple program steps. It increases the efficiency of the code and processing time. But it should also be noted that it is not suited for all situations. Depending upon the size of data and number of variables in a dataset it should be used accordingly.

REFERENCES:

1. SAS Institute Inc. 2012. Base SAS® 9.3 Procedures Guide, Second Edition. Cary, NC: SAS Institute Inc.
2. SAS Institute Inc. 2011. SAS® 9.3 Component Objects: Reference. Cary, NC: SAS Institute Inc.

3. PROC FCMP and DATA Step Component Objects: http://documentation.sas.com/?docsetId=proc&docsetVersion=9.4&docsetTarget=n03uc8c8fkguxqn1i5iapv1auqrz.htm&locale=en

4. Qing Liu (2017), "Streamline Table Lookup by Embedding HASH in FCMP", Proceedings of PharmaSUG China 2017: https://www.pharmasug.org/proceedings/china2017/AD/PharmaSUG-China-2017-AD02.pdf

5. Paul M. Dorfman, Don Henderson (2017), "Beyond Table Lookup: The Versatile SAS® Hash Object", Proceedings of the SAS Global 2017: http://support.sas.com/resources/papers/proceedings17/0821-2017.pdf

6. Schacherer, Chris (2015), "Introduction to SAS® Hash Objects", Proceedings of the SAS Global 2015: https://support.sas.com/resources/papers/proceedings15/3024-2015.pdf

7. Qixia Shi, Guanyu, su (2016), "Custom useful SAS functions using the PROC FCMP procedure", Proceedings of PharmaSUG China 2016: https://www.lexjansen.com/pharmasug-cn/2016/PS/PharmaSUG-China-2016-PS06.pdf

8. Richann Watson, Lynn Mullins (2016), "Exploring HASH Tables vs. SORT/DATA Step vs. PROC SQL", Proceedings of PharmaSUG 2016: https://www.pharmasug.org/proceedings/2016/TT/PharmaSUG-2016-TT11.pdf

9. Dylan Ellis, Mathematica Policy Research, Washington, DC, (2013), "RUN_MACRO Run! With PROC FCMP and the RUN_MACRO Function from SAS® 9.2, Your SAS® Programs Are All Grown Up", Proceedings of the SAS Global Forum 2013: https://support.sas.com/resources/papers/proceedings13/033-2013.pdf

10. Paul M. Dorfman, Lessia S. Shajenko, Koen Vyverman (2008), "Hash Crash and Beyond", Proceedings of the SAS Global Forum 2008: http://www2.sas.com/proceedings/forum2008/037-2008.pdf

CONTACT INFORMATION:

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

Premanand Kumaravel
Symbiance Inc. 51 Everette Dr,
Suite B-20, Princeton Junction
New Jersey 08550, USA
Email: pkumaravel@symbiance.com
Web: www.symbiance.com