# Using the SAS® Hash Object for Sample Allocation Procedures with Large Datasets/Big Data

Julia Batishev, Michael Yang
NORC at the University of Chicago, 4350 East-West Highway, Bethesda, MD 20814

**Abstract**
Sample allocation procedures for complex sample designs are usually implemented in multiple steps. For example, we may need to allocate the sample by strata to meet precision targets by analysis domains, where the domains may be defined in a hierarchical manner. In addition, higher level domains may not be defined for all the data in the frame, or we may choose not to target them. We may want to repeat the allocation procedure many times to adjust the conditions and simulate the results based on different allocation schemes and sample size levels. To allocate the sample under many restrictions is a challenging task by itself. When we need to apply the allocation procedures to big data, such as U.S. household frames or Medicaid beneficiary files, we also face problems associated with elapsed time, CPU and Memory usage. In this paper, we compare the use of the Hash Object (SAS), the traditional SAS DATA step processing mode, and PROC SQL in SAS for complex sample allocation tasks and present the advantages and tradeoffs of using the Hash Object.

**Key Words:** SAS Hash Object, Sample Allocation, Sample Size, Domains, DOW-Loop SAS, SAS Macro, Big Data

## 1. Introduction

We often encounter issues with computational resources when working with large datasets. In some cases we need to upgrade computer hardware, but sometimes we can simply change the implementation to make better use of existing computational resources.

To study the efficiency of different implementations of a sample allocation task, we carried out the same sample allocation task in SAS in three ways: traditional SAS DATA step, SAS PROC SQL, and SAS Hash Object. We evaluated computational resource use in terms of memory usage, elapsed (real) time, and CPU (Central Processing Unit) time.

We examined each implementation with simulated frame files of different sizes: ten thousand, one million, five million, and ten million records. For each frame size we ran each program fifty times on the same computer and recorded resulting computational resource usage measures.

We then compare the performance of each method based on the observed data and discuss the implications for selecting an implementation depending on data size and computational resources are available.

## 2. Sample Allocation Example

Consider the sample allocation task as a two-steps process where we need to meet precision requirements for the strata and improve representation of the sample for target domains. The domains are defined as collections of strata. When we allocate additional cases to meet the domain level precision, we need to add samples to strata by some rules, considering that the total sample sizes in strata may not exceed the population sizes in these strata.

In this example we consider the following task:

For simplicity, in step one, at the strata level, we allocate the same number of cases to each stratum. This step will allocate about two thirds of the total sample size. In step two we allocate the rest of the sample in such a way that the full sample would be distributed as close to the frame distribution per domain as possible.

We calculate the sample sizes for domain level proportionally based on the full sample and take into account the number of cases already allocated in step one per stratum for each domain. If we exceed the necessary sample size per domain for some domains in the first step we don't add any additional cases in step two for those domains. Finally, we readjust the sample size calculated initially per domain and redistribute samples to strata within each domain to get the final sample sizes for each stratum.

We divided the sample allocation algorithm in our example into the following parts:

1. Calculate totals per strata, domain, and overall total based on the frame file
2. Calculate proportions for domains within frame and strata within domain for each domain
3. Calculate the sample size for each stratum in step one
4. Calculate the sample size for each domain in step two
5. Redistribute the cases allocated in step two to each stratum within each domain

Each part of the algorithm is described in detailed using pseudocode (See Appendix 1.)

We used the same variable names in the examples of the SAS programs (See Appendix 2). We included the SAS programs to show the differences in coding and their impact on the computational resources.

In the programs we used the following features and procedures:

- for traditional SAS DATA step code: PROC MEANS/SUMMARY, PROC SORT, DOW-Loop (SAS), SAS Macro, regular data step with calculations, and merging
- for SAS PROC SQL program: Inline queries/subqueries, used calculated variables in the select statement for expressions/formulas, summarized groups of data, joined multiple tables, performed advanced queries, combined data horizontally and vertically
- for SAS Hash Object sample allocation program: Used lookup functionalities of hash object, updated/modified contents of hash object, used merging techniques with hash object, used iterators/('hiter') object, summarized hierarchically related data and also with the hash iterator object, updated data values and orders observations with hash object, and created data sets from hash objects.

## 3. Simulation and Measures

We obtained computational performance based on simulated frame files. We created frame files with different sizes: ten thousand, one million, five million, and ten million observations. For each frame size we simulated fifty frame files with the same structure. For each implementation and for each frame file we obtained computational performances for memory usage and CPU time usage.
The computational statistics were recorded by using FULLSTIMER option and collected with the LOGPARSE SAS Macro.
For each individual process we recorded maximum values for the Memory and OS Memory as well as totals for the User CPU time, System CPU Time, and Real Time values.

**Table 1:** Description of FULLSTIMER Statistics from SAS® documentations

| **Statistics** | **Description** |
|---|---|
| Real Time | The amount of time spent to process the SAS job. Real time is also referred to as elapsed time |
| User CPU Time | The CPU time spent to execute SAS code |
| System CPU Time | The CPU time spent to perform operating system tasks (system overhead tasks) that support the execution of SAS code |
| Memory | The amount of memory required to run a step |
| OS Memory | The maximum amount of memory that a step requested from the System |

### 3.1 Real Time and CPU time

The User CPU time and the System CPU time are mutually exclusive. When processor executes the user written code – it records the CPU time as a User CPU time, and when processor executes the operating system tasks to support the user written code – it records the CPU time as a System CPU time.

We plot the averages of the distribution of the User CPU time and the System CPU time based on fifty replicates for each frame size and for each method. The User CPU time and System CPU time increases with the size of the frame file. The SAS Hash Object sample allocation program uses less System CPU time compared with the other implementations for the same frame size and it allocates more User CPU time.

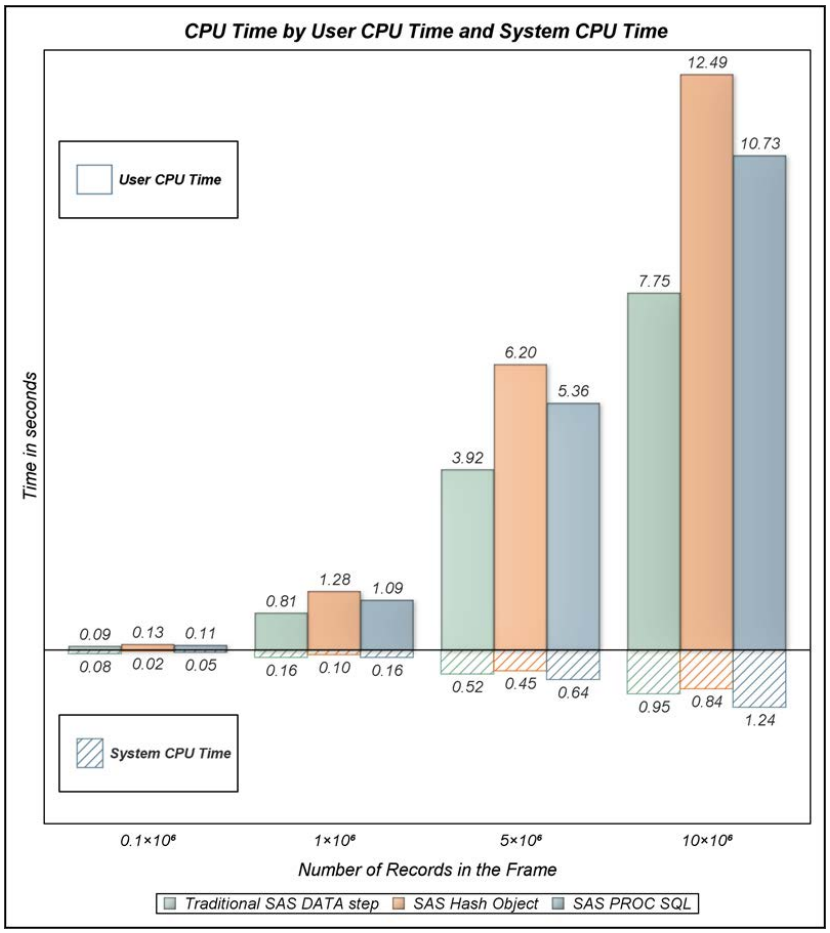


**Figure 1:** Averages of the User and System CPU Times for each implementation by frame size.

We also compared the distribution of the total CPU time, which is the sum of the User CPU and the System CPU, by frame size for each method. The larger the size of the frame file the more visible the difference in the usage of the CPU resources between the methods. The traditional SAS DATA step process uses less CPU time than other methods for larger frame files and there are almost no differences between methods for smaller frame sizes i.e. one million records and less.

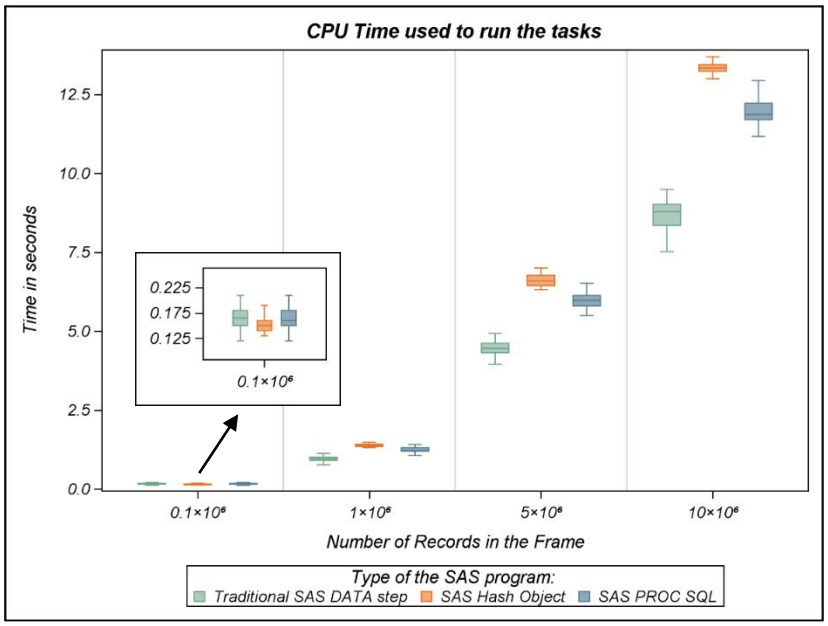


**Figure 2:** Distribution of the CPU Time for each implementation by frame size.

Figure 3 shows the distribution of the elapsed time (Real time) by frame size for each method. There are large difference between the methods by frame size. The larger the size of the frame file the larger the differences between the methods. The traditional SAS DATA step processing mode performs faster for all sizes of the frame files. The distribution of the values of the Real time for the SAS Hash Object implementation method is the same as the distribution of the CPU time for this method.

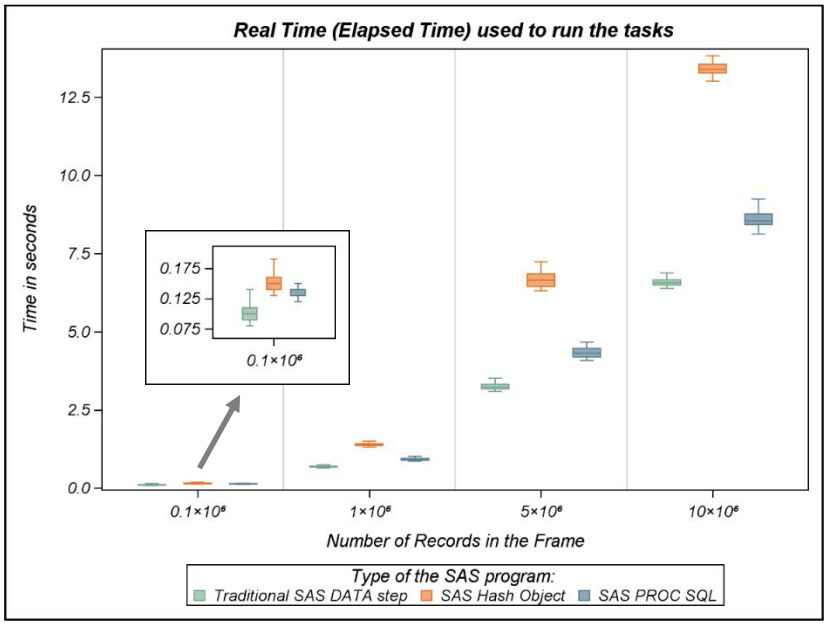


**Figure 3:** Distribution of the Real Time for each implementation by frame size.

Some SAS procedures automatically use available parallel processing resources and redistribute the job over available CPU cores (We used PC with four cores for the simulations). The SAS Hash Object program folds up to one data step, therefore it uses

only one CPU core, while the traditional SAS DATA step program and the SAS PROC SQL program use all available CPU cores for some steps.

We plot the median values of the distributions of the CPU time and Real time for frame files with sizes of one million, five million, and ten million by each implementation method side by side.

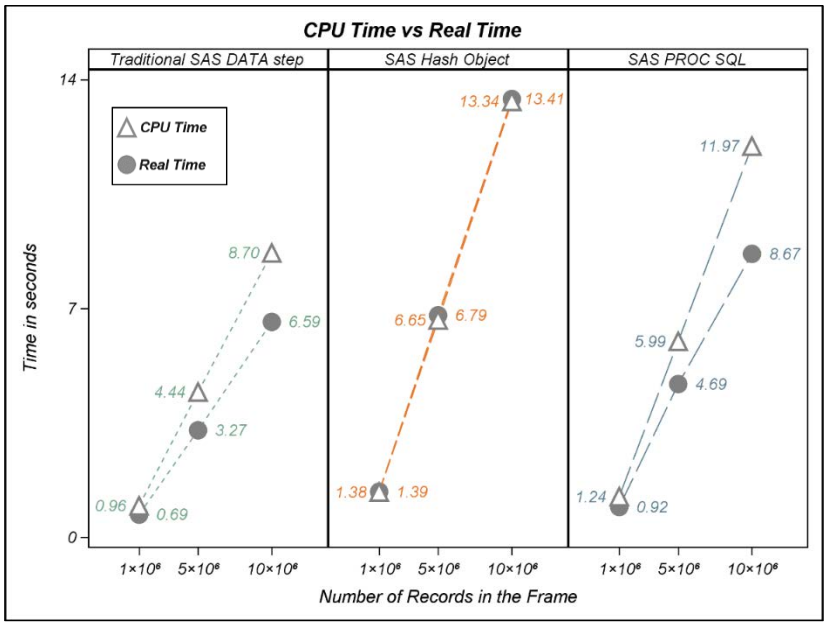


**Figure 4:** Median values of the distributions of the CPU time and Real time for sizes one, five, and ten million by implementation method.

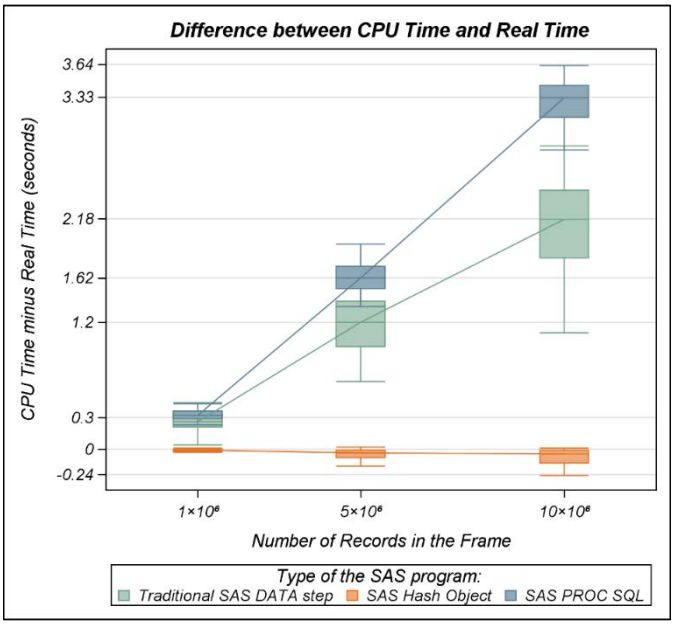


**Figure 5:** The distribution of the differences between CPU time and Real time.

We also analyzed the distribution of the differences between CPU time and Real time for each implementation to understand which method takes more advantage of distributed computing.
The largest difference between CPU time and Real time is for SAS PROC SQL implementation, which means that SAS PROC SQL takes better advantage of multiple cores. The difference between CPU time and Real time is close to zero and negative for SAS Hash Object implementation. It means that SAS Hash Object implementation uses only one CPU with the default settings within the data step. The traditional SAS DATA step processing mode uses multiple cores with the default settings, but not as much as SAS PROC SQL.

### 3.2 Memory Usage

We collected two measures for memory usage for each process:

- Memory - the amount of memory that is allocated for the sample allocation process which does not include the memory that needed for SAS as overhead to manage and execute the task.
- OS Memory – the total amount of memory that includes the amount of memory allocated for the sample allocation process and overhead.

We plot the averages over fifty simulations of the amount of memory per implementation and frame size in Log scale. The results show that the SAS Hash Object is more memory-efficient for large datasets than the traditional SAS DATA step and the SAS PROC SQL techniques. The amount of memory needed by the SAS Hash Object program does not appreciably change for different sizes of input data. The traditional SAS DATA step program needs more memory, but has the same pattern. For the SAS PROC SQL implementations the usage of the memory is increasing as the size of the frame file is increasing.

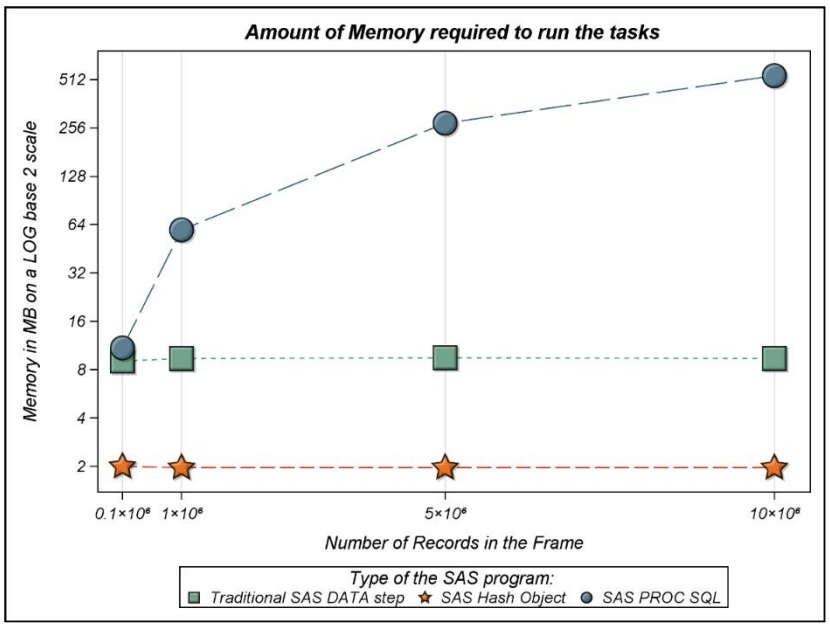


**Figure 6:** The average amount of memory per implementation and frame size in Log scale.

The overhead of the memory that needed for SAS to manage and execute the sample allocation processes does not change considerably for different sizes of frame files and different methods. It is in the 9MB -12MB range for all three implementation methods and for different sizes of the input data.

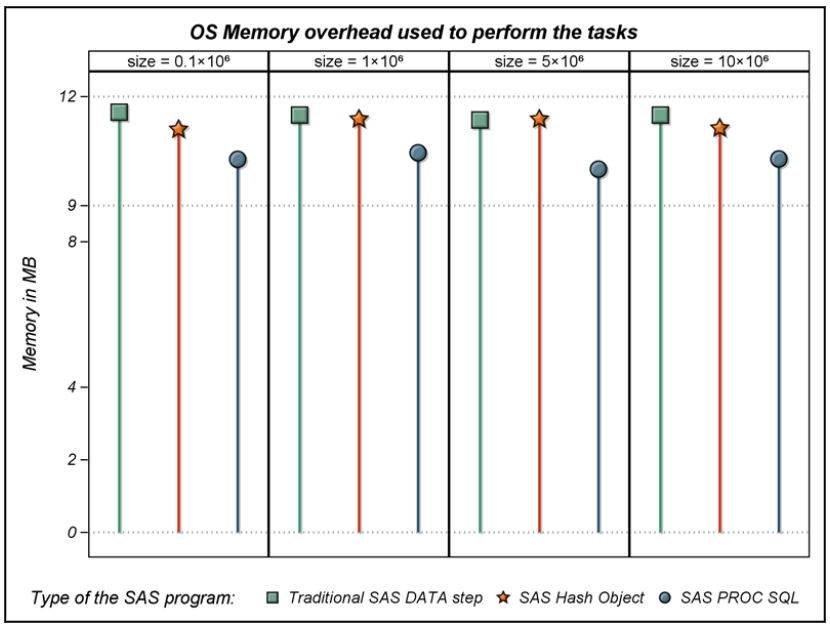


**Figure 7:** Median values of the distributions of the OS memory overhead used to perform tasks.

## 4. Conclusions

Each implementation approach has their advantages and tradeoffs. The SAS Hash Object technique is very useful when we need to work with very large data and may face a memory overflow issue.
The SAS PROC SQL approach is the easiest way to program this task and it also takes better advantage of multiple cores to reduce elapsed time with large datasets.
The traditional SAS DATA step style program executes the task faster and also does not use much OS memory.

## Acknowledgements

We would like to thank Y. Sverchkov for his diligent proofreading of this paper. Special thanks to Edward Mulrow, Nola du Toit, and B. Boex for their help with the preparation of the JSM poster.

## Appendix 1

**Table 1 Appendix 1:** Detailed algorithm (pseudocode) for the example of the sample allocation program

| **1.** Calculate totals per strata, domain, and overall total based on the frame file |
|---|
| UNT – number of observations in the frame |
| UD1_K – number of observations in domain K (calculate for each domain) |

UNT_S – number of observations in stratum S (calculate for each strata)

**2.** Calculate proportions for domains within frame and strata within domain

UPD_K = UD1_K/UNT – proportion of domain K in the frame (for each domain)
UPS_wD1_S = UNT_S /UD1_K – proportion of stratum S in Domain K (for each stratum)

**3.** Calculate the sample size for each stratum in step one

LN – Sample size
WS1_LN_S = (LN×2/3)/Str – calculated sample size for stratum S
ACTS1_LN_S = min (WS1_LN_S, UNT_S) – actual sample size for stratum S, which is minimum of calculated sample size for stratum S and frame size for stratum S (for each stratum)

**4.** Calculate the sample size for each domain in step two

WS2_LD_K = LN×UPD_K – calculated sample size for domain K
ACTS1_LD_K = sum (ACTS1_LN_S) by S within domain K – allocated sample size for domain K in step one
ACTS1_LD = sum (ACTS1_LN_S) by S – sample size allocated in step one
NEEDS2_LD_K =max (0, WS2_LD_K - ACTS1_LD_K) – need to allocate for domain K
NEEDS2_LD = sum (NEEDS2_LD_K) by K – additional cases that need to allocate in step two
ADJS2_LD_K= max (0, LN- ACTS1_LD)* NEEDS2_LD_K/ NEEDS2_LD – adjusted sample for domain K (calculate for each domain)

**5.** Redistribute the cases allocated in step two to each stratum within each domain

ADDS2_LN_S = ADJS2_LD_K × UPS_wD1_S – cases that we need to add to stratum S
CALCS2_LN_S = ACTS1_LN_S + ADDS2_LN_S – sum of allocated cases in step one and two for stratum S
R_CALCS2_LN_S = round (min (UNT_S, CALCS2_LN_S)) - adjusted and rounded sample size for stratum S (calculate for each stratum)

## Appendix 2

### A2.1 Example of the sample allocation implementation using traditional SAS DATA step programming techniques

The code in traditional SAS DATA step techniques is divided into five parts that corresponds Table 1 Appendix 1.

1. Calculation of totals per strata, domain, and overall total based on the frame file:

```
proc means data=S noprint missing;
class STRATA DOMAIN_1;
var WT1;
output out=SUMMARY n=UN sum=UNT;
run;
data FRAME_TOT(keep =UNT)
     Domain1_TOT(keep=DOMAIN_1 UNT rename=(UNT=UD1_K))
     Strata_TOT(keep=STRATA UNT rename=(UNT=UNT_S))
     STRATA_DOMAIN_TOT(keep =STRATA DOMAIN_1 UNT
                       rename=(UNT=UNT_S_D1_K));
        set SUMMARY;
        if _TYPE_=0 then output FRAME_TOT;
        else if _TYPE_=1 then output Domain1_TOT;
        else if _TYPE_=2 then output Strata_TOT;
     else if _TYPE_=3 then output STRATA_DOMAIN_TOT;
run;
```

2. Calculation of proportions for domains within frame and strata within domain for each domain:

```
data Domain1_TOT;
    if _N_=1 then set FRAME_TOT;
    set Domain1_TOT end=last;
    UPD_K=UD1_K/UNT;
    if last then call symput('K_DOMAIN',_N_);
run;
proc sort data = STRATA_DOMAIN_TOT; by DOMAIN_1 STRATA;run;
proc sort data = Domain1_TOT; by DOMAIN_1;run;
/*DOW Loop *//* data with all totals and proportions */
data STRATA_DOMAIN_TOT;
    do _N_=1 by 1 until (LAST.Domain_1);
        set Domain1_TOT; by DOMAIN_1;
    end;
    do _N_=1 by 1 until(LAST.Domain_1);
       do _N_=1 by 1 until(LAST.STRATA);
         set STRATA_DOMAIN_TOT ; by DOMAIN_1 STRATA;
         UPS_wD1_S=UNT_S_D1_K/UD1_K;
         output;
       end;
    end;
run;
```

3. Calculation of the sample size for each stratum in step one:

```
data _NULL_ ;
set STRATA_DOMAIN_TOT end=last;
if last then do;call symput('S_STRATA',_N_);
   ws1_Ln_s=(&sample_size/_N_)*(&prop_of_samp_s1*1);
   ws1_Ln=(&sample_size)*(&prop_of_samp_s1*1);
   call symput('WS1_LN_S',ws1_Ln_s);
   call symput('WS1_LN',ws1_Ln);
end;
data STRATA_DOMAIN_TOT;/* S1 actual */
   set STRATA_DOMAIN_TOT ;by DOMAIN_1 STRATA;
   acts1_LN_S=min((1*&WS1_LN_S),UNT_S_D1_K);
run;
```

4. Calculation of the sample size for each domain in step two:

```
proc means data = STRATA_DOMAIN_TOT noprint;
class DOMAIN_1;
var acts1_LN_S;
output out=Domain1_tot1 (drop= _freq_ _TYPE_) sum=acts1_LD_K;
run;
data Domain1_tot1;
    if _N_=1 then set Domain1_tot1(where=(DOMAIN_1=.)
    rename=(acts1_LD_K=acts1_LD));
    do _n_ = 1 by 1 until (last.DOMAIN_1);
      set Domain1_tot1 (where=(DOMAIN_1 ^=.));by DOMAIN_1 ;
    end;
    do _n_ = 1 by 1 until (last.DOMAIN_1);
      set Domain1_tot ;by DOMAIN_1 ;
      Ws2_LD_k=(1*&sample_size)*UPD_K;
      Ws2_LD=1*&sample_size;
      needs2_LD_k=max(0,(Ws2_LD_k-acts1_LD_K));
      remained_after_s1=Ws2_LD - acts1_LD;
    end;
run;
proc means data =Domain1_tot1 sum min max n noprint;
var  needs2_LD_k;
output out =needs2_LD(keep=needs2_LD) sum=needs2_LD;
run;
/* adjust count- calculate the number per domain per */
data Domain1_tot1;
    if _N_=1 then set needs2_LD;
    set Domain1_tot1;
    adjS2_LD_k=max(0,(Ws2_LD-acts1_LD))* needs2_LD_k/needs2_LD;
run;
```

5. Calculation of the cases allocated in step two to each stratum within each domain:

```
proc means data=S noprint missing;
class STRATA DOMAIN_1;
var WT1;
output out=SUMMARY n=UN sum=UNT;
run;
data FRAME_TOT(keep =UNT)
     Domain1_TOT(keep=DOMAIN_1 UNT rename=(UNT=UD1_K))
     Strata_TOT(keep=STRATA UNT rename=(UNT=UNT_S))
     STRATA_DOMAIN_TOT(keep =STRATA DOMAIN_1 UNT
                       rename=(UNT=UNT_S_D1_K));
        set SUMMARY;
        if _TYPE_=0 then output FRAME_TOT;
        else if _TYPE_=1 then output Domain1_TOT;
        else if _TYPE_=2 then output Strata_TOT;
     else if _TYPE_=3 then output STRATA_DOMAIN_TOT;
run;
```

**A2.2 Example of the sample allocation implementation using SAS PROC SQL programming techniques**

The PROC SQL (SAS) program is divided into five parts that corresponds Table 1 Appendix 1.

1. Calculation of totals per strata, domain, and overall total based on the frame file:

```
proc SQL ;

create table STRATA_DOMAIN_TOT1 as
select DOMAIN_1,STRATA,sum(WT1) as UNT_S_D1_K, count(WT1) as UN
from s
group by  STRATA, DOMAIN_1
order by DOMAIN_1,STRATA
;
create table Domain1_TOT as
select DOMAIN_1,sum(UNT_S_D1_K) as UD1_K
from STRATA_DOMAIN_TOT1
group by  DOMAIN_1
order by DOMAIN_1
;
```

2. Calculation of proportions for domains within frame and strata within domain for each domain:

```
create table STRATA_DOMAIN_TOT as
select s.DOMAIN_1,s.STRATA,
   f.UNT, d.UD1_K, s.UNT_S_D1_K,
   UD1_K/UNT as UPD_K,
   UNT_S_D1_K/UD1_K as UPS_wD1_S,
```

3. Calculation of the sample size for each stratum in step one:

```
   /* step #1 */
   count(*) as S_STRATA,
   (&sample_size/ calculated S_STRATA)*(&prop_of_samp_s1*1) as
      WS1_LN_S,
   (&sample_size)*(&prop_of_samp_s1*1) as ws1_Ln,
   min((calculated WS1_LN_S),UNT_S_D1_K) as acts1_LN_S
from
(select sum(UD1_K) as UNT from Domain1_TOT)as f ,
Domain1_TOT as d ,STRATA_DOMAIN_TOT1 as s
where d.DOMAIN_1=s.DOMAIN_1
order by s.DOMAIN_1,s.STRATA
;
```

4. Calculation of the sample size for each domain in step two:

```
create table Domain1_tot1 as
select  distinct s.DOMAIN_1,sum(acts1_LN_S) as acts1_LD_K,
   ns1cal.acts1_LD, s.UD1_K,
   (1*&sample_size)*s.UPD_K as Ws2_LD_k,
   (1*&sample_size) as Ws2_LD,
   max(0,(calculated Ws2_LD_k- calculated acts1_LD_K)) as
needs2_LD_k,
   calculated Ws2_LD - ns1cal.acts1_LD as remained_after_s1
from STRATA_DOMAIN_TOT as s ,
(select sum(acts1_LN_S) as acts1_LD from STRATA_DOMAIN_TOT) as
ns1cal
group by s.DOMAIN_1
order by s.DOMAIN_1
;
create table Domain1_tot as
select d1.* ,d.needs2_LD,
max(0,(d1.Ws2_LD - d1.acts1_LD))* d1.needs2_LD_k/d.needs2_LD as
adjS2_LD_k
from Domain1_tot1 as d1 ,
(select sum(needs2_LD_k) as needs2_LD  from Domain1_tot1) as d
;
```

5. Calculation of the cases allocated in step two to each stratum within each domain:

```
create table Strata_domain_tot1 as
select s.*, d.acts1_LD, d.acts1_LD_k, d.Ws2_LD_k, d.Ws2_LD,
d.needs2_LD_k, d.needs2_LD, d.remained_after_s1, d.adjS2_LD_k,
d.adjS2_LD_k * s.UPS_wD1_S as addS2_Ln_s,
calculated addS2_Ln_s + s.acts1_LN_S as calcS2_Ln_s,
round( (min(s.UNT_S_D1_K,calculated calcS2_Ln_s)),1) as
r_calcS2_ln_s
from STRATA_DOMAIN_TOT as s,
     Domain1_tot as d
where s.DOMAIN_1=d.DOMAIN_1
;
quit;
```

**A2.3 Example of the sample allocation implementation using Hash Object SAS programming techniques**

Below is the Hash Object SAS Code for the full program divided into six parts.
First we initialized the Hash and Hiter (Iterator) objests and then follow the algorithm that described in the the Table 1 Appendix 1.

Initialization of the Hash Objects and Iterator Objects

```
data _NULL_;
retain UNT_key 1;
if _n_=1 then do;
  declare hash popTOTAL ();
  popTOTAL.definekey('UNT_key');
  popTOTAL.definedata('UNT_key','UNT','acts1_LD','Ws2_LD',
                      'remained_after_s1','needs2_LD');
  popTOTAL.definedone();
  declare hash Domain_tmp ();
  Domain_tmp.definekey('UNT_key','DOMAIN_1');
  Domain_tmp.definedata('UNT_key','DOMAIN_1','UD1_K');
  Domain_tmp.definedone();
  declare hash Domain1 (ordered: 'yes');
  declare hiter iterD('Domain1');
  Domain1.definekey('UNT_key','DOMAIN_1');
  Domain1.definedata('UNT_key','DOMAIN_1','UD1_K','UNT','UPD_K',
    'acts1_LD_K','Ws2_LD_k','needs2_LD_k','adjS2_LD_k');
  Domain1.definedone();
  declare hash STRATUM_Domain_tmp ();
  STRATUM_Domain_tmp.definekey('UNT_key','DOMAIN_1','STRATA');
  STRATUM_Domain_tmp.definedata('UNT_key','DOMAIN_1','STRATA',
                                'UNT_S_D1_K');
  STRATUM_Domain_tmp.definedone();
  declare hash STRATUM_Domain_tot (ordered: 'yes' );
  declare hiter iterST_D('STRATUM_Domain_tot');
  STRATUM_Domain_tot.definekey('UNT_key','DOMAIN_1','STRATA');
  STRATUM_Domain_tot.definedata('UNT_key','DOMAIN_1','STRATA',
   'UNT_S_D1_K','UPS_wD1_S', 'UD1_K','UNT','UPD_K','acts1_LN_S',
   'addS2_Ln_s', 'calcS2_Ln_s', 'r_calcS2_ln_s');
  STRATUM_Domain_tot.definedone();
end;
```

1. Calculation of totals per strata, domain, and overall total based on the frame file:

```
do until (EOF) ;
  set S end=EOF;
  if popTOTAL.find() ne 0 then call missing(UNT) ;
  UNT=sum(UNT,WT1);
  popTOTAL.replace() ;
  if Domain_tmp.find() ne 0 then call missing(UD1_K) ;
  UD1_K=sum(UD1_K,WT1);
  Domain_tmp.replace() ;
  Domain1.replace() ;
  if STRATUM_Domain_tmp.find() ne 0 then
    call missing  (UNT_S_D1_K) ;
  UNT_S_D1_K=sum(UNT_S_D1_K,WT1);
  STRATUM_Domain_tmp.replace() ;
  STRATUM_Domain_tot.replace() ;
end ;
```

2. Calculation of proportions for domains within frame and strata within domain for each domain:

```
rcd = iterD.first();
do while (rcd = 0);
    if popTOTAL.find()=0 and Domain_tmp.find()=0 then do;
        UPD_K=UD1_K/UNT;
        rc2=Domain1.replace() ;
    end;
    rcd = iterD.next();
end;
rcST_d = iterST_D.first();
do while (rcST_d = 0);
    if popTOTAL.find()=0 and Domain1.find()=0 and
       STRATUM_Domain_tmp.find()=0 then do;
           UPS_wD1_S=UNT_S_D1_K/UD1_K;
           STRATUM_Domain_tot.replace() ;
    end;
    rcST_d = iterST_D.next();
end;
```

3. Calculation of the sample size for each stratum in step one:

```
S_STRATA=STRATUM_Domain_tot.NUM_ITEMS;
ws1_Ln_s=(&sample_size/S_STRATA)*(&prop_of_samp_s1*1);
ws1_Ln=(&sample_size)*(&prop_of_samp_s1*1);
```

4. Calculation of the sample size for each domain in step two:

```
rcST_d = iterST_D.first();
    do while (rcST_d = 0);
        if popTOTAL.find()=0 then do;
            if Domain1.find()=0 then do;
                if STRATUM_Domain_tmp.find()=0 then do;
                    acts1_LN_S=min((1*WS1_LN_S),UNT_S_D1_K);
                    STRATUM_Domain_tot.replace() ;
                end;
                acts1_LD_K= sum(acts1_LD_K,acts1_LN_S);
                Ws2_LD_k=(1*&sample_size)*UPD_K;
                needs2_LD_k=max(0,(Ws2_LD_k-acts1_LD_K));
                rcd2=Domain1.replace() ;
            end;
            acts1_LD= sum(acts1_LD,acts1_LN_S);
            Ws2_LD=1*&sample_size;
            remained_after_s1=Ws2_LD - acts1_LD;
            rcp2=popTOTAL.replace() ;
        end;
        rcST_d = iterST_D.next();
    end;
    rcd = iterD.first();
    do while (rcd = 0);
        if popTOTAL.find()=0 and Domain1.find()=0 then do;
            needs2_LD=sum(needs2_LD,needs2_LD_k);
        end;
        rcd = iterD.next();
        rc2=popTOTAL.replace() ;
    end;
    rcd = iterD.first();
    do while (rcd = 0);
        if popTOTAL.find()=0 and Domain1.find()=0 then do;
            adjS2_LD_k=
             max(0,(Ws2_LD-acts1_LD))*  needs2_LD_k/needs2_LD;
            rc2=Domain1.replace() ;
        end;
        rcd = iterD.next();
    end;
```

5. Calculation of the cases allocated in step two to each stratum within each domain:

```
 rcST_d = iterST_D.first();
     do while (rcST_d = 0);
         if popTOTAL.find()=0 and Domain1.find()=0 and
             STRATUM_Domain_tmp.find()=0 then do;
                 addS2_Ln_s=adjS2_LD_k*UPS_wD1_S;
                 calcS2_Ln_s= addS2_Ln_s + acts1_LN_S;
                 r_calcS2_ln_s=
                   round( (min(UNT_S_D1_K,calcS2_Ln_s)),1);
                 STRATUM_Domain_tot.replace() ;
         end;
         rcST_d = iterST_D.next();
     end;
 popTOTAL.output (dataset: 'popTOTAL');
 Domain1.output (dataset: "&Domain_tot");
 STRATUM_Domain_tot.output (dataset: "&Strata_domain_tot");
stop;
run;
```

**References**

SAS Institute Inc. 2016. SAS® 9.4 Companion for Windows, Fifth Edition. Cary, NC: SAS Institute Inc

SAS Institute Inc. 2017. Base SAS® 9.4 Procedures Guide, Seventh Edition. Cary, NC: SAS Institute Inc.

P.Dorfman, D.Henderson. Data Aggregation Using the SAS Hash Object. SAS Global Forum, Dallas, TX, 2015. http://support.sas.com/resources/papers/proceedings15/2000-2015.pdf

P.Dorfman, D.Henderson. Beyond Table Lookup: The Versatile SAS® Hash Object. SAS Global Forum, Orlando, FL, 2017. http://beoptimized.be/pdf/Paper_821-2017.pdf

P.Dorfman. Table Lookup by Direct Addressing: Key-Indexing, Bitmapping, Hashing. SUGI 26, Long Beach, CA, 2001. SUGI 26: Table Look-Up by Direct Addressing: Key-Indexing

Li, Arthur. 2014. Understanding and Applying the Logic of the DOW-Loop. Proceedings of the PharmaSUG 2014 Conference. San Diego, CA.

Allen, Richard Read. 2010. Practical Uses of the DOW Loop. Proceedings of the WUSS 2010 Conference. San Diego, CA.

Keintz, Mark. 2016. From Stocks to Flows: Using SAS® HASH objects for FIFO, LIFO, and other FO's. MWSUG 2016. https://www.mwsug.org/proceedings/2016/BB/MWSUG-2016-BB22.pdf

Eberhardt, Peter. 2014. The SAS® Hash Object: It's Time To .find() Your Way Around. SAS Global Forum, Washington, DC, 2014. http://support.sas.com/resources/papers/proceedings14/1482-2014.pdf

Raithel, Michael A. 2005. Programmatically Measure SAS® Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro, SUGI 30, Philadelphia, Pennsylvania, 2005. http://www2.sas.com/proceedings/sugi30/219-30.pdf.