PharmaSUG 2014 - Paper CC02

Let Hash SUMINC Count For You

Joseph Hinson, Accenture Life Sciences, Berwyn, PA, USA

ABSTRACT

Counting of events is inevitable in clinical programming and is easily accomplished with SAS® procedures such as FREQ, MEANS, SUMMARY, TABULATE, SQL or even by simple DATA step statements. In certain situations where counting is a bit intricate involving data partitioning and classifying, another convenient and efficient way is by hash programming. Within the DATA step, hash objects make data directly available to summary variables without the need to first save the data into a dataset. A little-known hash utility, SUMINC, was introduced with SAS® version 9.2. SUMINC is an argument tag for the hash object declaration statement in which it designates a numeric variable for counting. Together with other new SAS 9.2 hash methods REF() and SUM(), counting of items becomes feasible in a key-based manner thus allowing any pattern of counting to be easily accomplished and directly available for summary reporting, as demonstrated in this paper for clinical trial protocol deviation analysis. SUMINC is also useful for NWAY summarization of data, but unlike the SUMMARY procedure or PROC SQL with GROUP BY, NWAY summarization with SUMINC can be combined with DATA step logic within the same DATA step.

INTRODUCTION

Clinical trial data analyses routinely involve the counting of events, and many SAS® tools exist for accomplishing that, notably the FREQUENCY, MEANS, TABULATE, and SQL procedures, among others, and various DATA step approaches including BY processing. Often, one needs to capture the resulting count data from the SAS procedures either through ODS or by simply outputting into a dataset. SAS® 9 hash objects are also able to compute counts, for instance with the hash attribute “num_items”. By grouping populations of interest into their own hash tables, the object.num_items functionality can directly provide the counts for each hash table. A new feature of hash objects introduced in SAS® 9.2 was SUMINC[1], an argument tag in the hash declaration statement that assigns a numeric variable for counting. SUMINC instructs the hash object to allocate internal storage for maintaining a count value for each hash key. That count value is incremented by the value of the SUMINC counting variable whenever data is encountered with the matching key(s). The SUMINC-designated counting variable can be negative, positive, zero-valued, or even a non-integer. This paper will demonstrate the use of the SUMINC argument tag, in combination with two other new SAS® 9.2 hash methods: REF() and SUM(), to determine the number of occurrences of protocol violations in a clinical study.

FEATURES OF THE SUMINC METHOD

1. SUMINC

The syntax is: ***declare hash object (SUMINC: counting_variable);*** where “object” is the name of the hash object, and “counting_variable” must be numeric.

2. REF()

When hash objects were first introduced with SAS version 9, two available methods: CHECK() and ADD() could be used to populate a hash object with data. The CHECK() method determines if data exists for a particular hash key. If yes, a return code of 0 is produced, otherwise a non-zero code is issued. The ADD() method, as the name suggests, adds data to a hash table in a key-specific manner, whether the item already exists or not. SAS version 9.2 introduced an additional method, REF(), which combines CHECK() and ADD() into a single method. Thus, ***object.REF()*** first checks if the current key-value already exists in the hash table for “object”. If yes, nothing is done. But if no, the key-value pair is added to the hash table.

3. SUM()

The SUM() method retrieves the value of the count associated with the current key and stores it in a totaling variable designated by a SUM argument tag in the SUM statement: ***rc=object.SUM (SUM: totaling_variable).***

CODING STRATEGY

The entire count processing gets implemented in three stages:

1. ***SUMINC:counting_variable*** is specified in the hash declaration such that each time a key is found during data input, the value of counting_variable is added to an internal accumulator associated with that particular key.
2. The ***object.REF*** method does the key finding and the accumulator value updating. During data input, REF checks to see if the current key is already in the hash table. If not, the key is added and the internal accumulator is initialized to the value of the counting_variable. If the key was found to be already present, no key is added but the counting_variable value is added to the internal accumulator for that key.
3. Once all data have been read, the hash table is read out row-by-row by the hash iterator and the internal accumulator value for each key is captured by the SUM method. The ***object.SUM*** method has an argument tag also called "SUM", with a designated variable,"totaling_variable", to capture the internal accumulator counts for each key:

   object.SUM (SUM:totaling_variable).

AN EXAMPLE WITH PROTOCOL DEVIATION ANALYSIS

Consider a small clinical study situation in which there is the need to determine how many enrolled and treated subjects violated the protocol. Suppose that they were four possible violations:

(a) Chemistry test showing ABNORMAL LIVER ENZYMES.

(b) Chemistry test showing a POSITIVE PREGNANCY RESULT

(c) Immunology test showing ALLERGY TO STUDY DRUG

(d) Medication assessment indicating CONSUMPTION OF PROHIBITED MEDICATION

Consider also that we need the occurrences of deviations in the following manner:

1. Counts per test parameter per treatment
2. Counts per test parameter
3. Counts of all subjects who deviated
4. Counts of all subjects who deviated per treatment
5. Counts for any deviation

The input data has the subject id (USUBJID), the parameter category (PARCAT1), the parameter tested (PARAM), the treatment variable (TRTA), and the test outcome (AVALC).

First, a dataset is created with a counting variable, COUNT=1, added to every observation.

Next, a hash object for counting, ***ppt***, is declared, having the ***suminc*** tag and the numeric counting_variable ***COUNT***.

```
       declare hash ppt(suminc:"COUNT");
```

In order to obtain "counts per parameter per treatment", the hash keys are set up as below:

```
       rc=ppt.defineKey("parcat1","param","trta");*<---[choice of keys determine type
of counts];
       rc=ppt.defineData("parcat1","param","trta");
       rc=ppt.defineDone();
       declare hiter hippt("ppt");
```

The input dataset, “protdev”, is then read in, while the hash method REF checks for keys in order to load the hash table:

```
        do until(done);
        set protdev end=done;
        rc=ppt.ref();
```

Finally, using a DO loop and the hash iterator object ***hippt***, the hash table ***ppt*** is read out row-by-row while using the SUM method to extract the count totals per key into a variable “totcount”:

```
        totcount=0;
        rc=hippt.first();
        do until (hippt.next() ne 0);
                ppt.sum(sum:totcount);
        ---
        ---
        ---
        end;
```

In reality, the count totals are captured and sent to another hash table, ‘***all***’, for summarization.
The ***all*** hash object acts as a data collector, storing all the various counts totals retrieved by the SUM method for all the five kinds of deviation analysis.
This is one advantage afforded by using the hash technique, where intermediate data during analysis in a DATA step can directly be made available for summarization and manipulation without the need to create intermediate datasets.

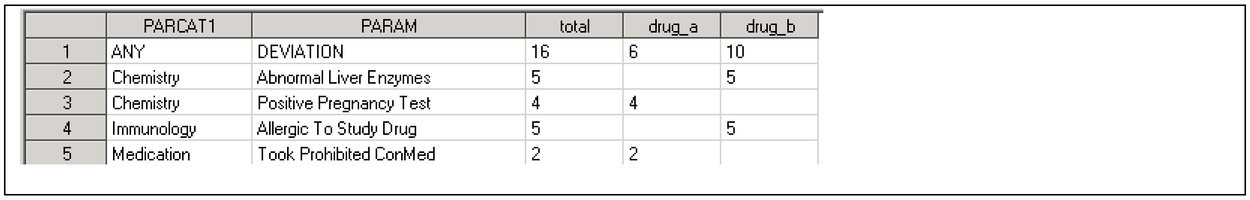
THE FULL CODE *(tested in SAS 9.2):*

```
data protdev;
        infile datalines;
        input USUBJID $ TRTA $ PARCAT1 $12. PARAM &  $23.  AVALC $ ;
        count=1; *<--------[gives every observation a count of 1];
datalines;
P001    Drug_A Medication    Took Prohibited ConMed            Yes
P005    Drug_B Chemistry     Abnormal Liver Enzymes            Yes
P006    Drug_B Immunology    Allergic To Study Drug            Yes
P009    Drug_A Chemistry     Positive Pregnancy Test           Yes
P011    Drug_B Chemistry     Abnormal Liver Enzymes            Yes
P016    Drug_B Immunology    Allergic To Study Drug            Yes
P026    Drug_B Immunology    Allergic To Study Drug            Yes
P029    Drug_A Chemistry     Positive Pregnancy Test           Yes
P031    Drug_A Medication    Took Prohibited ConMed            Yes
P035    Drug_B Chemistry     Abnormal Liver Enzymes            Yes
P045    Drug_B Chemistry     Abnormal Liver Enzymes            Yes
P055    Drug_B Chemistry     Abnormal Liver Enzymes            Yes
P056    Drug_B Immunology    Allergic To Study Drug            Yes
P059    Drug_A Chemistry     Positive Pregnancy Test           Yes
P066    Drug_B Immunology    Allergic To Study Drug            Yes
P079    Drug_A Chemistry     Positive Pregnancy Test           Yes
;
run;

*-----------------------------------------------------------
             [1]  CREATION OF HASH OBJECT FOR GROUP COUNTS
-----------------------------------------------------------;

data _null_;
        length totcount 8 totalcount $8;
        if(1=2) then set protdev;*<-------[a trick to just put all variables into PDV];

****COUNTS PER PARAMETER PER TREATMENT*************;
        declare hash ppt(suminc:"count");*<[suminc counting variable MUST be numeric!];
        rc=ppt.defineKey("parcat1","param","trta");*<[choice of keys determine counts];
```

```
       rc=ppt.defineData("parcat1","param","trta");
       rc=ppt.defineDone();
       declare hiter hippt("ppt");

****COUNTS PER PARAMETER***************************;
       declare hash pp(suminc:"count");
       rc=pp.defineKey("parcat1","param");
       rc=pp.defineData("parcat1","param","trta");
       rc=pp.defineDone();
       declare hiter hipp("pp");

****COUNTS OF ALL SUBJECTS WHO DEVIATED************;
       declare hash sb(suminc:"count");
       rc=sb.defineKey("count");*<-[variable COUNT used also to indicate any subject];
       rc=sb.defineData("count","parcat1","param","trta");
       rc=sb.defineDone();
       declare hiter hisb("sb");

****COUNTS OF ALL SUBJECTS WHO DEVIATED PER TREATMENT*******;
       declare hash ts(suminc:"count");
       rc=ts.defineKey("trta","count");*<[variable COUNT used also for any subject];
       rc=ts.defineData("count","parcat1","param","trta");
       rc=ts.defineDone();
       declare hiter hits("ts");

****FINAL SUMMARY TABLE****************************
--------------------------------------------------------------------------;
       length total drug_a drug_b $8;
       declare hash all(ordered:"Y");
       rc=all.defineKey('parcat1','param');
       rc=all.defineData('parcat1','param','total','drug_a','drug_b');
       rc=all.defineDone();

*-----------------------------------------------------
               [2]   INITIATE COUNTING
------------------------------------------------------;
       call missing(of _all_);
       do until(done);
              set protdev end=done;

*************FOR COUNT PROCESSING************************;
              rc=ppt.ref();*<------(count per parameter per treatment);
              rc=pp.ref();*<------(count per parameter);
              rc=sb.ref();*<------(count per all deviators);
              rc=ts.ref();*<------(count per all deviators per treatment);
       end;

*----------------------------------------------------------------------
[3] HASH READOUT OF COUNT TOTALS USING THE SUM FUNCTION
 (The data-collection hash object ALL always has keys: parcat1, param)
-----------------------------------------------------------------------;
*********counts per param per treatment*****************;
       array cx{6} $23  _temporary_;
       totcount=0;
       rc=hippt.first();
       do until (hippt.next() ne 0);
       rc=all.find(key:parcat1,key:param);

       cx(1)=parcat1;cx(2)=param;cx(3)=trta;cx(4)=total;cx(5)=drug_a;cx(6)=drug_b;
       ppt.sum(sum:totcount);
       x=3+whichc(trta,'TOTAL','Drug_A','Drug_B');*<-[do some data transposition];
              totalcount=strip(compress(put(totcount, 8.)));
              cx(x)=totalcount;
       rc=all.replace(key:cx(1),key:cx(2),data:cx(1),data:cx(2),data:cx(4),data:cx(5),
data:cx(6));
              call missing(of _all_);
       end;

*********counts per parameter only***********************;
       totcount=0;
       rc=hipp.first();

       do until (hipp.next() ne 0);
       TRTA="TOTAL";
              rc=all.find(key:parcat1,key:param);

       cx(1)=parcat1;cx(2)=param;cx(3)=trta;cx(4)=total;cx(5)=drug_a;cx(6)=drug_b;
       pp.sum(sum:totcount);
       x=3+whichc(trta,'TOTAL','Drug_A','Drug_B');*<-[does some data transposition];
       totalcount=strip(compress(put(totcount, 8.)));
       cx(x)=totalcount;

       rc=all.replace(key:cx(1),key:cx(2),data:cx(1),data:cx(2),data:cx(4),data:cx(5),
data:cx(6));
              call missing(of _all_);
       end;

*********counts per subjects only************************;
       totcount=0;
       rc=hisb.first();

       do until (hisb.next() ne 0);
       TRTA="TOTAL";
       PARCAT1="ANY";
       PARAM="DEVIATION";
           rc=all.find(key:parcat1,key:param);

       cx(1)=parcat1;cx(2)=param;cx(3)=trta;cx(4)=total;cx(5)=drug_a;cx(6)=drug_b;
       sb.sum(sum:totcount);
       x=3+whichc(trta,'TOTAL','Drug_A','Drug_B');*<--[does some data transposition];
       totalcount=strip(compress(put(totcount, 8.)));
       cx(x)=totalcount;

       rc=all.replace(key:cx(1),key:cx(2),data:cx(1),data:cx(2),data:cx(4),data:cx(5),
data:cx(6));
       call missing(of _all_);
       end;

********counts per treatments only***********************;
       totcount=0;
       rc=hits.first();

       do until (hits.next() ne 0);
       PARCAT1="ANY";
       PARAM="DEVIATION";
       rc=all.find(key:parcat1,key:param);

       cx(1)=parcat1;cx(2)=param;cx(3)=trta;cx(4)=total;cx(5)=drug_a;cx(6)=drug_b;
       ts.sum(sum:totcount);
       x=3+whichc(trta,'TOTAL','Drug_A','Drug_B');*<--[performs some transposition];
       totalcount=strip(compress(put(totcount, 8.)));
       cx(x)=totalcount;

       rc=all.replace(key:cx(1),key:cx(2),data:cx(1),data:cx(2),data:cx(4),data:cx(5),
data:cx(6));
```

```
       call missing(of _all_);
       end;

**********overall output*********************************;
       all.output(dataset:'alltable');
       stop;
run;


 *********convert missing values to zeroes*************;
data finaltable;
       set alltable;
       drug_a=ifc(missing(drug_a),'0',drug_a,'0');
       drug_b=ifc(missing(drug_b),'0',drug_b,'0');
run;
```

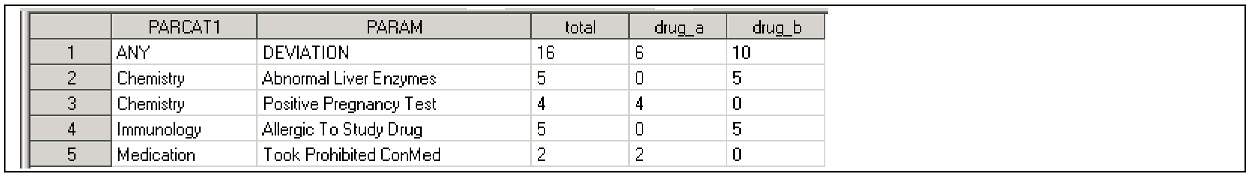
Output 1



	PARCAT1	PARAM	total	drug_a	drug_b
1	ANY	DEVIATION	16	6	10
2	Chemistry	Abnormal Liver Enzymes	5		5
3	Chemistry	Positive Pregnancy Test	4	4	
4	Immunology	Allergic To Study Drug	5		5
5	Medication	Took Prohibited ConMed	2	2	

Table 1. Hash table “ALLTABLE” (from hash object “all”) showing protocol deviation counts

Output 2



	PARCAT1	PARAM	total	drug_a	drug_b
1	ANY	DEVIATION	16	6	10
2	Chemistry	Abnormal Liver Enzymes	5	0	5
3	Chemistry	Positive Pregnancy Test	4	4	0
4	Immunology	Allergic To Study Drug	5	0	5
5	Medication	Took Prohibited ConMed	2	2	0

Table 2. Hash table (“FINALTABLE”) showing protocol deviation counts with blanks replaced by zeroes

CONCLUSION

The novel counting approach demonstrated in this paper involves the hash SUMINC argument tag, the hash REF method, and the hash SUM method. SUMINC defines the counting variable, REF triggers the accumulation of counts, and SUM retrieves the accumulated counts. All the steps are hash-key based, similar to the “TABLES” statement of the FREQUENCY procedure.

Even though the FREQUENCY procedure and the count functionality of the SQL procedure in most cases are quite adequate for counting clinical events, the SUMINC approach provides much more flexibility and convenient data gathering steps. The power of the approach is even more evident when NWAY summarization is required.

REFERENCES

1. R. Ray and J. Secosky, “Better Hashing in SAS® 9.2 ”,

   SAS Global Forum 2008.

   http://support.sas.com/resources/papers/sgf2008/hashing92.pdf

CONTACT INFORMATION

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

Joseph W. Hinson, PhD
Accenture Life Sciences
1160 W. Swedesford Rd
Berwyn, PA 19312
1-610-407-7580
joehinson@outlook.com

sas® Certified Advanced Programmer *for* SAS®9

sas® Certified Clinical Trials Programmer Using SAS®9