Paper 069-2012

Splitting Data Sets on Unique Values: a Macro That Does It All

Hans Sempel, Belastingdienst, Apeldoorn, The Netherlands

ABSTRACT

Sometimes it is necessary to split a data set into multiple data sets, depending on the unique values of a variable. After this task was requested several times at the Belastingdienst in the Netherlands, a macro was developed that could split every data set based on a variable. The results are a data set for every unique value of that variable. And, all it took was two DATA steps!

INTRODUCTION

The Belastingdienst (Dutch Tax and Customs Administration) is responsible for:

- levying, monitoring and collecting state taxes
- supervising the inflow, outflow and transit of goods
- supervising compliance with legislation in the fields of economy, health, environment and safety, economic classification and financial integrity
- levying and collecting national insurance schemes contributions and employee insurance premiums
- allocating and monitoring income-related benefits
- conducting investigations in all of the above areas

To perform these duties the Belastingdienst has databases containing huge tables with billions of records.

About 2008 we started to use SAS® as tooling to extract data and information from these databases.

To facilitate the developers a programming environment was created that contains a toolbox with ready to use macros for all sorts of tasks.

One of these tasks was the splitting of data sets on the value of a variable. It was decided to create a macro that could perform this task just by telling him the data set to split and the variable on which it has to be split.

This paper describes that macro.

THE MACRO

After the decision was made to create the macro we found a paper named '241 – 31: Data Step Hash Objects as Programming Tools' written by Koen Vyverman and Paul Dorfman in which two methods were described. They called these methods the Pre-V9-hash method and the Hash .OUTPUT() method (see Appendix A for the code).

The Pre-V9-hash method consists of a PROC SQL with a distinct select, followed by a DATA step. The result of the PROC SQL is stored in two macro variables, one to be used as list of data sets in the DATA statement and the other to be used as WHEN clauses in a SELECT statement.

The other method is based on the use of the Hash .OUTPUT() Method in combination with the Hash of Hashes. This method consists of one DATA step that declares a hash table (HOH) that contains hash tables (HH) as data. For each unique value a HH-hash table is initiated and populated with observations. At the end of the data set an iterator is used to loop through the table and dump the contents of every HH-hash table into a data set.

We decided to introduce a third method and create our own macro because of problems we experienced with the two mentioned. One does not have an optimal performance and the other has problems with the memory when splitting huge data sets.

The macro consists of three parts. To explain the macro the following data set will be used.

```
data sample;
    input region $ amount number;
    cards;
A 2000 4
C 543 2
B 287 6
```

```
C 764 4
A 1098 3
B 456 8
A 982 9
;
run;
```

PART 1: DETERMINING THE TYPE OF VARIABLE

The macro is called with the keyword parameters dataset (the name of the data set to split), varname (the variable on which to split) and the optional parameter outlib (the library in which the data sets are put). In this example the parameter dataset has the value ‘sample’ and the parameter varname has the value ‘region’.

```
%macro split (dataset=, varname=, outlib=WORK );
    %local sel sets;
    %if %INDEX(&dataset,.) eq 0 %then %do;
        %let libname = WORK ;
        %let setname = &data set ;
    %end;
    %else %do;
        %let libname = %SUBSTR(&dataset,1,%INDEX(&dataset,.)-1);
        %let setname = %SUBSTR(&dataset,%INDEX(&dataset,.)+1) ;
    %end;
```

Because some statements in the next two parts differ for numeric and character variables, a simple PROC SQL determines the type of the variable on which the data set must be split. The type is put into the macro variable vartype. In the case of this example the value is ‘char’.

```
    proc sql noprint;
        select type into :vartype
        from sashelp.vcolumn
        where libname = upcase("&libname")
        and   memname = upcase("&setname")
        and   memtype = "DATA"
        and   upcase(name) = upcase("&varname")
        ;
    quit;
```

PART 2: DETERMINING THE UNIQUE VALUES

Here the data set is read and the values of the variable are put into a hash table. Because this hash-table has the variable as key, the return code is 0 when the variable doesn’t already exist in the hash-table.

If the return code is 0 a new data set name is added to the sets-variable and a new WHEN clause is added to the sel-variable.

At the end of the data step the variables sel and sets are put in the macro variables sel and sets. Using the example database, sets gets the value ‘WORK.setA WORK.setC WORKsetB’and sel the value ‘select (region); when (‘A’) output WORK.setA; when (‘C’) output WORK.setC; when (‘B’) output WORK.setB; otherwise; end;’.

```
    data _null_;
        retain sel sets;
        format sel sets $20000.;
        if _n_=1 then do;
            dcl hash members (ordered: 'a');
            rc = members.definekey("&varname");
            rc = members.definedone();
            sel="select (&varname);";
            sets='';
        end;
        set &data set (keep=&varname) end = eof;
        rc=members.add();
        if rc eq 0 then do;
            flag+1;
            sets=trim(sets)||" &outlib..set"||trim(left(&varname.));
            if "&vartype" eq "num" then
                sel=trim(sel)||" when ("||trim(left(&varname.))||
                    ") output &outlib..set"||trim(left(&varname.))||";";
            else
                sel=trim(sel)||" when ('"||trim(left(&varname.))||
                "') output &outlib..set"||trim(left(&varname.))||";";
        end;
        if eof then call symputx('sel',trim(sel)||' otherwise; end;','L');
        if eof then call symputx('sets',trim(sets),'L');
    run;
```

PART 3: THE SPLITTING

```
    data &sets;
        set &dataset;
        &sel.
    run;
%mend split;
```

This code resolves into:

```
    data WORK.setA WORK.setC WORK.setB;
        set sample;
        select (region);
            when ('A') output WORK.setA;
            when ('C') output WORK.setC;
            when ('B') output WORK.setB;
            otherwise;
        end;
    run;
```

PERFORMANCE

So now we have three methods, each with its own advantages and disadvantages. To determine which method is fit for purpose, we did some tests with data sets varying from 10,000 to 10,000,000 observations on a Windows platform and on z/OS (mainframe). We measured the CPU time and the elapsed time.

The results on Windows platform are:

Number of observations	**Pre-V9-Hash method**		**Hash .OUTPUT() method**		**The macro %split**	
	cpu time	**elapsed time**	**cpu time**	**elapsed time**	**cpu time**	**elapsed time**
10,000	0.06	0.28	0.06	0.07	0.04	0.37
100,000	0.34	8.85	0.29	2.04	0.12	2.92
1,000,000	3.12	28.09	2.28	10.67	0.88	25.65
10,000,000	31.68	264.80	30.84	118.28	9.03	250.67
100,000,000	413.83	2236.00	Memory Error		93.67	1539.67

Table 1. Performance Measurements on Windows

On the mainframe we didn't test with 100,000,000 because the Hash .OUTPUT() method here also went into and abend due to memory errors:

Number of observations	**Pre-V9-Hash method**		**Hash .OUTPUT() method**		**The macro %split**	
	cpu time	**elapsed time**	**cpu time**	**elapsed time**	**cpu time**	**elapsed time**
10,000	0.08	0.74	0.06	0.63	0.06	0.60
100,000	0.41	1.77	0.22	1.29	0.16	1.63

1,000,000	3.95	28.67	2.10	13.72	1.23	15.43
10,000,000	41.84	194.81	25.12	153.23	12.44	162.38

Table 2. Performance Measurements on Mainframe

An examination of the results of the measurements shows:

- On both platforms the Hash .OUTPUT() method doesn't work with big data sets (exactly how big depends on the configuration of your system).
- The Pre-V9-Hash method has by far the worst performance on both platforms.
- The Hash .OUTPUT() method uses more cpu time than the %split macro, but the elapsed time is less.

CONCLUSION

There isn't a best method.

The intention of this paper is to give the programmer an alternative and an insight in the advantages and disadvantages of the three methods. It depends on the situation which method fits the needs of the developer and hopefully this paper helps to choose the best one.

REFERENCES

Dorfman, Paul M. and Vyverman, Koen. March 1, 2006 "Data Step Hash Objects as Programming Tools." *SUGI 31*. San Francisco Available at http://www2.sas.com/proceedings/sugi31/241-31.pdf.

CONTACT INFORMATION

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

Name:	Hans Sempel
Enterprise:	Belastingdienst
Address:	Hoofdstraat 21
City, State ZIP:	7311 JT Apeldoorn, The Netherlands
E-mail:	hans@sempel.eu

APPENDIX A

The Pre-V9-hash method and the Hash .OUTPUT() method as they are described in paper 241-31.

In this code a data set ‘sample’ is split on the variable ‘id’.

Pre-V9-hash:

```
proc sql noprint ;
    select distinct 'OUT' || put (id, best.-l)
    into : dslist
    separated by ' '
    from sample
    ;
    select 'WHEN (' || put (id, best.-l) || ') OUTPUT OUT' || put (id, best.-l)
    into : whenlist
    separated by ';'
    from sample
    ;
quit ;

data &dslist ;
    set sample ;
    select ( id ) ;
        &whenlist ;
        otherwise ;
    end ;
run ;
```

Hash .OUTPUT():

```
data _null_ ;
    dcl hash hoh (ordered: 'a') ;
    dcl hiter hih ('hoh' ) ;
    hoh.definekey ('id' ) ;
    hoh.definedata ('id', 'hh' ) ;
    hoh.definedone () ;
    dcl hash hh () ;
    do _n_ = 1 by 1 until ( eof ) ;
        set sample end = eof ;
        if hoh.find () ne 0 then do ;
            hh = _new_ hash (ordered: 'a') ;
            hh.definekey ('id','transid', '_n_') ;
            hh.definedata ('id','transid', 'amt') ;
            hh.definedone () ;
            hoh.replace () ;
        end ;
        hh.replace() ;
    end ;
    do rc = hih.next () by 0 while ( rc = 0 ) ;
        hh.output (dataset: 'out'|| put (id, best.-L)) ;
        rc = hih.next() ;
    end ;
    stop ;
run ;
```