Getting Connected — The Power of Using Hash Objects

ABSTRACT

The Hash Object and the Hash Iterator Object[1], two data step component objects, offer a simple and powerful solution for many complex tasks. This paper will describe the hash object and show how to merge a number of data sets using hash objects, base SAS, and object dot notation. It will also illustrate how the program data vector (PDV) enables the interaction of data set variables and the data variables within the objects. The hash object is an in-memory solution allowing direct access via keys during execution. It is fast and can contain an unlimited number of key items and data items. The only limitation is available memory. As memory size increases and the constraints of available memory lessen, the usefulness of hash objects can only increase.

INTRODUCTION

Upon her balcony in the privacy of her garden, Shakespeare's Juliet, mused on Romeo's forbidden surname,

> What's in a name? That which we call a rose
> By any other word would smell as sweet.[2]

The hash object is a Data Step Component Interface (DSCI) or, more completely, a Data Step Component Object Interface. This is a descriptive name for a sweet thing. What is in the name Data Step Component Object Interface? Data Step: It can only be used in a data step. It cannot be used in a procedure. Component Object: It is a predefined structure, containing a hash function, map or multi-map, slots or hash buckets, and Adelson-Velsky and Landis balanced binary trees for referencing keys and storing data.

Using object dot notation, the programmer only needs to declare, instantiate, and refine the definition of the hash object by assigning values as key items and, optionally, data items. The hash object is then created as a variable in the program data vector at execution time, rather than compile time, and will only exist for the duration of the data step. This difference in timing creates an issue between the Data Step PDV and the object. Data Step programming must be used to create variables in the program data vector at compile time to interface with the key items and keyed data items in the hash object. Data Step programming creates vaiables in the PDV that are either numeric or character. Object dot notation creates a variable that is an object containing keyed data items. If composite keys are defined, they can be a mixture of character and numeric types. The data definition can also be a mixture of types. The bridge between the hash object and the data step program is the PDV. Variables created with Data Step programing at compile time host the items in the hash object and must match the attributes of the variables with which the hash object is loaded.

	Cust_id	policy_id	First_name	Last_Name	city	Country
1	pe12489	81880	Seth	Pecksniff	London	Eng
2	sh15878	11489		Shakespeare	London	Eng
3	sh15878	98355	Shakespeare		London	Eng
4	ma48798	22389	Thomas	Mann	Princeton	USA
5	ch12489	81889	Martin	Chuzlewit	London	Eng
6	do78521	12587	Amy	Dorrit	London	Eng
7	to12587	36901	Leo	Tolstoy	Yasnaya Polyana	Russia
8	ga78521	38525	Elizabeth	Gaskell	Knutsford	Eng
9	ni33691	66381	Nicholas	Nickleby	London	Eng
10	cl11483	22871	Arthur	Clennam	London	Eng
11	go78224	36999	Oliver	Goldsmith	London	Eng
12	ti78902	65547	Montague	Tigg	London	Eng
13	ti78902	58749	Tigg	Montague	London	Eng
14	ta48963	22158	Mark	Tapley	London	Eng
15	ch78984	44879	Chevy	Slyme	London	Eng
16	ja48548	22487	John	Jarndyce	Bleak House	Eng
17	gu77892	14856	William	Guppy	London	Eng

	Policy_id	vin	Year	Make	A
1	11489	1FA00000030000001	2003	Ford	F3
2	22389	1G100000050000002	2005	Chevrolet	G5
3	81889	JHG00000030000003	2003	Honda	H3
4	12587	1FA00000050000004	2005	Ford	F5
5	36901	1GC00000030000005	2005	Chevrolet	G5
6	38525	JHG00000030000006	2003	Honda	H3
7	66381	1FB00000050000007	2005	Ford	F5
8	22871	1G100000030000008	2003	Chevrolet	G3
9	36999	JHG00000050000009	2005	Honda	H5
10	81889	1FA00000030000010	2003	Ford	F3
11	12587	1G100000050000011	2005	Chevrolet	G5
12	36901	JHG00000030000012	2003	Honda	H3
13	65547	1FA00000030000013	2003	Ford	F3
14	65547	1G100000050000014	2005	Chevrolet	G5
15	58749	JHG00000030000015	2003	Honda	H3
16	58749	1FA00000050000016	2005	Ford	F5
17	22158	1GC00000030000017	2005	Chevrolet	G3
18	98355	JHG00000030000018	2003	Honda	H3
19	22158	JHG00000030000019	2003	Honda	H3
20	44879	1FB00000050000020	2005	Ford	F5
21	44879	1G100000030000021	2003	Chevrolet	G3
22	22487	JHG00000050000022	2005	Honda	H5
23	14856	1FA00000030000023	2003	Ford	F3
24	65547	1G100000050000024	2005	Chevrolet	G5
25	65547	JHG00000030000025	2003	Honda	H3

	Policy_id	claim_no	Vin
1	81889	7141	JHG00000030000003
2	12587	7249	1FA00000050000004
3	22871	7681	1G100000030000008
4	98355	7897	JHG00000030000018
5	36901	8113	JHG00000030000005

Data sets used for illustration. The code for the creation of the data sets is included at the back of this paper.

Bevery Johnson was a remarkable woman and an adventuress. She was the first woman to climb the face of El Capitan in Yosemite National Park. It took her 10 days to make the ascent. When she reached the top, a videographer was waiting to interview her. The interviewer asked her what kept her going. She said that she kept asking herself the same question, “How do you eat an elephant?” The answer, “One bite at a time.” The answer is very appropos hash objects in the era of big data. The hash object is capable of containing an infinite number of key items and data items in each slot or bucket; however, since it is a memory resident object, it is constrained by available memory. To handle large amounts of data, it needs to take the data one bite at a time. To work efficiently with hash objects, it is necessary to judiciously use keep options and subsetting where options, or pre-filter data to limit the number of variables and the number of observations to what is required.

```
data _null_;
      *if _n_=0 then set work.cust(keep=policy_id First_name Last_name Country);
      attrib policy_id length= $ 10
            First_name length= $ 15
            Last_Name length= $ 15
            Country length= $ 15
             ;
      if _n_=1 then do;
            dcl hash CMP(dataset:'work.cust(where=(Country="Russia"or Country="USA")');
            CMP.definekey ('policy_id');
            CMP.definedata ("policy_id","First_name","Last_Name","Country");
            CMP.definedone();
            call missing(Cust_id, First_name, Last_Name, city, Country, Cust_dt);
      end;
      set  work.pol;
      rc=CMP.find();
      if rc=0 then CMP.output(dataset:'DumpH (where=(Country="Russia")');
      else CMP.output(dataset:'else1');
run;
```

	policy_id	First_name	Last_Name	Country
1	36901	Leo	Tolstoy	Russia

	policy_id	First_name	Last_Name	Country
1	36901	Leo	Tolstoy	Russia
2	22389	Thomas	Mann	USA

To write a declare statement you can use “declare” or the abbreviated “dcl”. There are two ways to declare and instantiate a hash object. The first offers the convenience of combining the declaration and the instantiation in one statement:

```
dcl hash h_obj ( );
```

The second is more versatile breaking the declaration and instantiation into separate statements. These statments can be separated by code.

```
dcl hash h_obj;
h_obj=_new_ hash ( );
```

The first statement declares the name reference, h_obj, a hash object, and the second statement uses the operator _new_ to assign the object to h_obj. This second method shows more clearly that the hash object is a variable in the PDV.

When a data step set statement is used to input values into the PDV and retrieve values from a hash object, the timing of the flow of data into any variables that share the same name has to be considered. The data step inputs its values first, and any values drawn from the hash object overwrite the values from the data step.

```
data work.cust2;
      set work.cust;
      where policy_id in (“98355”,”65547”);
run;
 data work.share;
      if _n_=0 then set work.cust(keep=cust_id first_name Last_name);
      /*attrib        policy_id length= $ 10
                  First_name length= $ 15
                  Last_Name length= $ 15
                        ; */
      if _n_=1 then do;
            dcl hash SHAR(dataset:’work.cust(where=(last_name=”Shakespeare”or last_name=”Montague”)’);
            SHAR.definekey (‘cust_id’);
            SHAR.definedata (“First_name”,”Last_Name”);
            SHAR.definedone();
            *call missing(First_name, Last_Name);
      end;
      set work.cust2 (keep=cust_id first_name Last_name);
      rc=SHAR.find();
      If rc ne 0 then call missing(first_name, last_name);
      Shar.output(dataset:’H_Dump’);
run;
```

	Cust_id	policy_id	First_name	Last_Name	city	Country
1	sh15878	98355	Shakespeare		London	Eng
2	ti78902	65547	Montague	Tigg	London	Eng

Cust_id	First_name	Last_Name	rc
sh15878		Shakespeare	0
ti78902	Tigg	Montague	0

	First_name	Last_Name
1	Tigg	Montague
2		Shakespeare

Methods used for traversing the hash key items using a hash itereator or do loops.

With the default of argument tag Multidata: ‘N’
It is easy to declare a hash iterator to traverse the data in the hash object; for example:

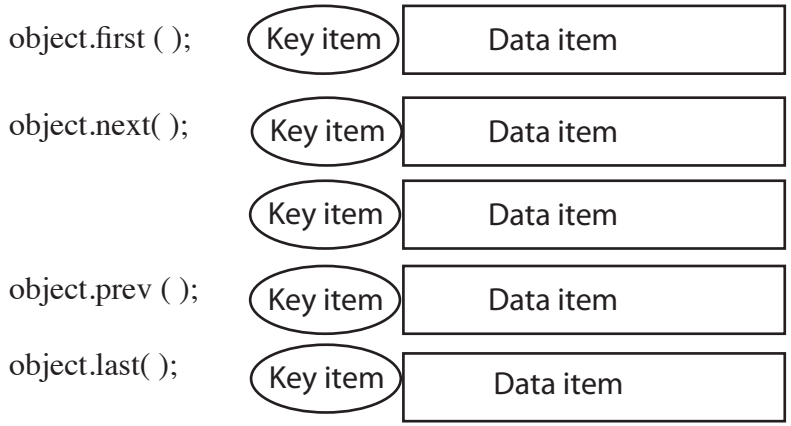
```
dcl Hiter hobji('hobj');
```

The methods used to traverse the data are first, next, prev, last, and setcur(key:‘value’). If the argument tag Ordered: ‘Y’ is used in the hash declaration, the data can be traversed and output in ascending or descending order. The key items are traversed sequentially similar to the data in an SQL table with a unique key.

With the Multidata argument tag “Y”
The data in a hash object needs different methods to traverse the keys when the Multidata argument tag is “Y” because the keys must be accessed sequentially and hierarchically. The hash object has a head node or header node, and non-unique data is attached as an ordered list or data chain, so one must traverse the data across the head nodes and down the linked data.

It seems to me that the actual structure of the keys and data in a hash object most resembles a web, the strands a mesh of references and the nodes the intersections. The following is a simple conceptual representation to aid understanding the methods used with the hash iterator and do loops to extract data from a hash object.

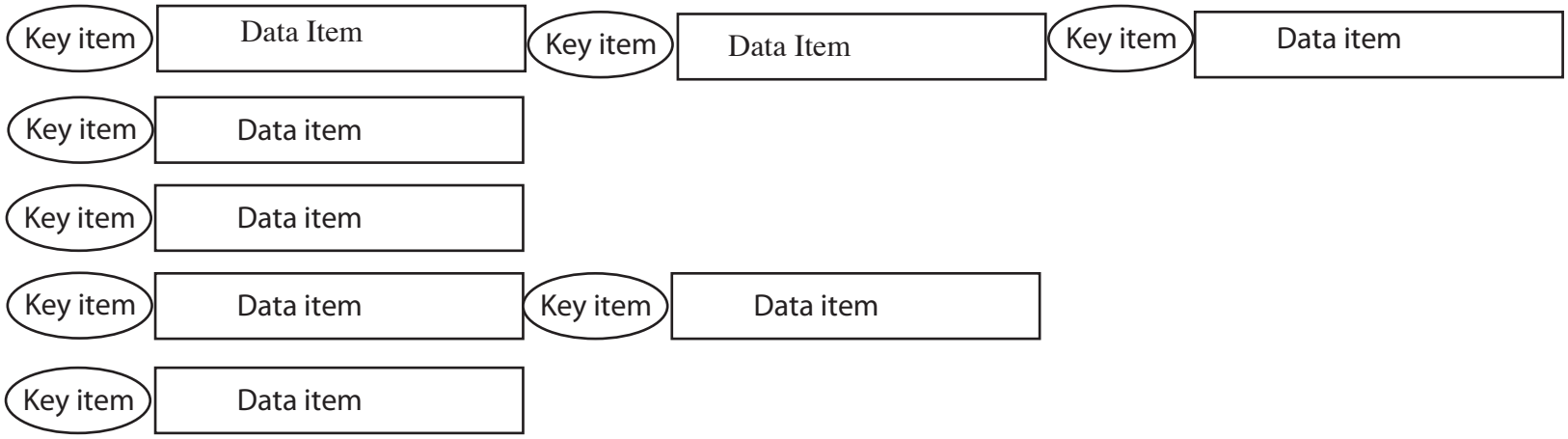
Multidata: ‘N’



There is also a object.setcur(key: ‘value’). It is usually used with a lteral to begin an iteration at a specific point in the data.

If the Multidata:'Y' argument tag is used in the hash declaration, nested do loops can be used with methods to extract data from the hash object. One loop crosses the head nodes and the other runs down or up the hierarchially arranged chain of non-unique key data.

object.find ( ) object.has_next (result: r) object.find_next( )



There are also methods for operating in the oposite direction.

object.find ( ) object.has_prev (result: r) object.find_prev( )

Object dot notation has two attribute statements. One returns the number of head keys and the other returns the size in bytes of the keys and data in a single hash Item. These can be used in a calculation to get the approximate size of the hash object. When using the Multidata argument tag 'Y' the size of hash items can greatly vary.

```
variable_name=object.Item_size
variable_name=object.Num_items
```

```
data work.loop_out_all
    work.loop_out_unq
    work.loop_out_dup;
      attrib Policy_id length= $ 10
             vin length= $ 17
             Year length= $ 4
             Make length= $ 17
             ;
      If _n_=1 then do;
         dcl hash lp(dataset:'work.pol', ordered:'Y', multidata:'Y');
         lp.definekey('Policy_id');
         lp.definedata('Policy_id','vin','year','make');
         lp.definedone();
         Call missing (Policy_id, Vin, Year, Make);
      end;
      do until(eof1);
         set work.cust (keep=Policy_id Cust_id First_name last_name) end=eof1;
         rc=lp.find();
         *    output work.loop_out;
           If (rc = 0) then do;
              output work.loop_out_unq;
              output work.loop_out_all; /*unique*/
              lp.has_next (result: r);
              do while (r ne 0);
                 rc=lp.find_next();
                     output work.loop_out_all; /*non-unique*/
                    output work.loop_out_dup;
                 lp.has_next (result: r);
              end;
           end;
      end;
   lp.output(dataset: 'H2_dump');
   stop;
run;
```

	Policy_id	vin	Year	Make	Cust_id	First_nam e	Last_Nam e	rc	r
1	11489	1FA00000030000001	2003	Ford	sh15878		Shakespeare	0	.
2	98355	JHG00000030000018	2003	Honda	sh15878	Shakespeare		0	0
3	22389	1G100000050000002	2005	Chevrolet	ma48798	Thomas	Mann	0	0
4	81889	JHG00000030000003	2003	Honda	ch12489	Martin	Chuzlewit	0	0
5	81889	1FA00000030000010	2003	Ford	ch12489	Martin	Chuzlewit	0	1
6	12587	1FA00000050000004	2005	Ford	do78521	Amy	Dorrit	0	0
7	12587	1G100000050000011	2005	Chevrolet	do78521	Amy	Dorrit	0	1
8	36901	1GC00000030000005	2005	Chevrolet	to12587	Leo	Tolstoy	0	0
9	36901	JHG00000030000012	2003	Honda	to12587	Leo	Tolstoy	0	1
10	38525	JHG00000030000006	2003	Honda	ga78521	Elizabeth	Gaskell	0	0
11	66381	1FB00000050000007	2005	Ford	ni33691	Nicholas	Nickleby	0	0
12	22871	1G100000030000008	2003	Chevrolet	cl11483	Arthur	Clennam	0	0
13	36999	JHG00000050000009	2005	Honda	go78224	Oliver	Goldsmith	0	0
14	65547	1FA00000030000013	2003	Ford	ti78902	Montague	Tigg	0	0
15	65547	1G100000050000014	2005	Chevrolet	ti78902	Montague	Tigg	0	1
16	65547	1G100000050000024	2005	Chevrolet	ti78902	Montague	Tigg	0	1
17	65547	JHG00000030000025	2003	Honda	ti78902	Montague	Tigg	0	1
18	58749	JHG00000030000015	2003	Honda	ti78902	Tigg	Montague	0	0
19	58749	1FA00000050000016	2005	Ford	ti78902	Tigg	Montague	0	1
20	22158	1GC00000030000017	2005	Chevrolet	ta48963	Mark	Tapley	0	0
21	22158	JHG00000030000019	2003	Honda	ta48963	Mark	Tapley	0	1
22	44879	1FB00000050000020	2005	Ford	ch78984	Chevy	Slyme	0	0
23	44879	1G100000030000021	2003	Chevrolet	ch78984	Chevy	Slyme	0	1
24	22487	JHG00000050000022	2005	Honda	ja48548	John	Jarndyce	0	0
25	14856	1FA00000030000023	2003	Ford	gu77892	William	Guppy	0	0

	Policy_id	vin	Year	Make	Cust_id	First_name	Last_Name	rc	r
1	81889	1FA00000030000010	2003	Ford	ch12489	Martin	Chuzlewit	0	1
2	12587	1G100000050000011	2005	Chevrolet	do78521	Amy	Dorrit	0	1
3	36901	JHG00000030000012	2003	Honda	to12587	Leo	Tolstoy	0	1
4	65547	1G100000050000014	2005	Chevrolet	ti78902	Montague	Tigg	0	1
5	65547	1G100000050000024	2005	Chevrolet	ti78902	Montague	Tigg	0	1
6	65547	JHG00000030000025	2003	Honda	ti78902	Montague	Tigg	0	1
7	58749	1FA00000050000016	2005	Ford	ti78902	Tigg	Montague	0	1
8	22158	JHG00000030000019	2003	Honda	ta48963	Mark	Tapley	0	1
9	44879	1G100000030000021	2003	Chevrolet	ch78984	Chevy	Slyme	0	1

	Policy_id	vin	Year	Make	Cust_id	First_na me	Last_Name	rc	r
1	11489	1FA00000030000001	2003	Ford	sh15878		Shakespeare	0	.
2	98355	JHG00000030000018	2003	Honda	sh15878	Shakespeare		0	0
3	22389	1G100000050000002	2005	Chevrolet	ma48798	Thomas	Mann	0	0
4	81889	JHG00000030000003	2003	Honda	ch12489	Martin	Chuzlewit	0	0
5	12587	1FA00000050000004	2005	Ford	do78521	Amy	Dorrit	0	0
6	36901	1GC00000030000005	2005	Chevrolet	to12587	Leo	Tolstoy	0	0
7	38525	JHG00000030000006	2003	Honda	ga78521	Elizabeth	Gaskell	0	0
8	66381	1FB00000050000007	2005	Ford	ni33691	Nicholas	Nickleby	0	0
9	22871	1G100000030000008	2003	Chevrolet	cl11483	Arthur	Clennam	0	0
10	36999	JHG00000050000009	2005	Honda	go78224	Oliver	Goldsmith	0	0
11	65547	1FA00000030000013	2003	Ford	ti78902	Montague	Tigg	0	0
12	58749	JHG00000030000015	2003	Honda	ti78902	Tigg	Montague	0	0
13	22158	1GC00000030000017	2005	Chevrolet	ta48963	Mark	Tapley	0	0
14	44879	1FB00000050000020	2005	Ford	ch78984	Chevy	Slyme	0	0
15	22487	JHG00000050000022	2005	Honda	ja48548	John	Jarndyce	0	0
16	14856	1FA00000030000023	2003	Ford	gu77892	William	Guppy	0	0

```
data work.mg (drop=Abbr_vin);
   attrib
      policy_id length= $ 10 First_name length= $ 15 Last_Name length= $ 15 city length= $ 15
      vin length= $ 17 Year length= $ 4 Make length= $ 17 Abbr_Vin length= $ 2 claim_no length= $ 10;
   If _n_=1 then do;
      dcl hash a(dataset:'Cust(where=(city="London")',ordered:'Y');
      a.definekey('policy_id');
      a.definedata('first_name','last_name','city');
      a.definedone();

      dcl hash b(dataset:'pol');
      b.definekey('Abbr_vin');
      b.definedata('year','make');
      b.definedone();

      dcl hash c(dataset:'claim');
      c.definekey('policy_id','vin');
      c.definedata('claim_no');
      c.definedone();

   call missing(policy_id, First_name, Last_name, city, vin, Year,
                Make, claim_no);

   end;
   do until (eof2);
      set work.Pol (keep=Policy_id vin) end=eof2;
      a_rc=a.find();
      if a_rc ne 0 then call missing(First_name, last_name, city);
      b_rc=b.find(key: cats((substr(vin, 2,1)),substr(vin,10,1)));
      if b_rc ne 0 then call missing (Year, make);
      c_rc=c.find(key: policy_id, key: vin);
      if c_rc ne 0 then call missing(claim_no);
      c.output(dataset:'cl');
      output work.mg;
      /*If (C_rc=0) then output work.mg;*/
   end;
   Stop;
run;
```

_N_	_ERROR_	Policy_id	First_name	Last_Name	City	Vin	Year	Make	Abbr_vin	Claim_No

The do loop enters the first values into the PDV: Policy_id, Vin, and Abbr_vin. the a.find method uses the value in the PDV to retrieve First_name, Last_name, and city from hash object a. The b.find method uses a concatenation of values in the vin number to match the values of Abbr_vin the key variable in the b object and the value of year and make are retrived. Finally the c.find method uses the keys policy_id and vin to retrive the claim_no from the c object.

	claim_no
1	7681
2	7249
3	8113
4	7141
5	7897

	policy_id	First_name	Last_Name	city	vin	Year	Make	claim_no	a_rc	b_rc	c_rc
1	11489		Shakespeare	London	1FA0000003000001	2003	Ford		0	0	160038
2	22389				1G1000005000002	2005	Chevrolet		160038	0	160038
3	81889	Martin	Chuzlewit	London	JHG0000003000003	2003	Honda	7141	0	0	0
4	12587	Amy	Dorrit	London	1FA0000005000004	2005	Ford	7249	0	0	0
5	36901				1GC0000003000005	2003	Chevrolet		160038	0	160038
6	38525				JHG0000003000006	2003	Honda		160038	0	160038
7	66381	Nicholas	Nickleby	London	1FB0000005000007	2005	Ford		0	0	160038
8	22871	Arthur	Clennam	London	1G1000003000008	2003	Chevrolet	7681	0	0	0
9	36899	Oliver	Goldsmith	London	JHG0000005000009	2005	Honda		0	0	160038
10	81889	Martin	Chuzlewit	London	1FA0000003000010	2003	Ford		0	0	160038
11	12587	Amy	Dorrit	London	1G1000005000011	2005	Chevrolet		0	0	160038
12	36901				JHG0000003000012	2003	Honda		160038	0	160038
13	65547	Montague	Tigg	London	1FA0000003000013	2003	Ford		0	0	160038
14	65547	Montague	Tigg	London	1G1000005000014	2005	Chevrolet		0	0	160038
15	58749	Tigg	Montague	London	JHG0000003000015	2003	Honda		0	0	160038
16	58749	Tigg	Montague	London	1FA0000005000016	2005	Ford		0	0	160038
17	22158	Mark	Tapley	London	1GC0000003000017	2003	Chevrolet		0	0	160038
18	98355	Shakespeare		London	JHG0000003000018	2003	Honda	7897	0	0	0
19	22158	Mark	Tapley	London	JHG0000003000019	2003	Honda		0	0	160038
20	44879	Chevy	Slyme	London	1FB0000005000020	2005	Ford		0	0	160038
21	44879	Chevy	Slyme	London	1G1000003000021	2003	Chevrolet		0	0	160038
22	22487				JHG0000005000022	2005	Honda		160038	0	160038
23	14856	William	Guppy	London	1FA0000003000023	2003	Ford		0	0	160038
24	65547	Montague	Tigg	London	1G1000005000024	2005	Chevrolet		0	0	160038
25	65547	Montague	Tigg	London	JHG0000003000025	2003	Honda		0	0	160038

ARGUMENT TAGS
(Dataset: dataset: name<(options)>')
Is a very convenient way to load a hash object. A do loop is another alternative.

(Duplicate: 'R' | 'E')
Determines how duplicates are delt with. The 'R' replaces the first data item and the 'E' stops the program when a duplicate is detected.

(Multidata: 'Y' | 'Yes')
Maps multiple data items to a key item

(Ordered: 'A' | 'Y' | 'D' | 'N')
Sorts the data using the key items like the by in a proc sort. An efficient feature if one may need sorted data later. It can make a proc sort unnessary.

(hashexp: n)
Sets the number of hash slots or buckets. n is an exponent of 2. For 9.3 the default for the number of hash buckets is 8 or 256 slots. To work efficiently the key items must not be distributed too densely or too thinly in the slots. 9 (512 slots) or 10 (1024 slorts) is recommended for one million items.

Calculates the number of slots:

```
data hashexp_arg;
        format exponent 4.;
                buckets comma10.;
        do exponent = 0 to 20;
                buckets = 2**exponent;
                output;
        end;
run;
```

Check size of memory

```
data _null_;
        amt=getoption('xmrlmem')
        put amt=;
run;
```

(Suminc: 'variable-name')
Use as a counter. The variable-name keeps a count in the PDV.

The definedata method has one argument tag: ALL:'Y'. It will define all the data loaded into the hash object as data items. When you use the definedata method if you want the key data ouput it must be defined as a key and as a data item.

The object.definedone is used to finalize the intialization of the hash object. It has one agrument tag: (Memrc: 'y'). It is used to evaluate whether the method was successful or not. A return code of 0 indicates that the hash object was loaded and a non-zero indicates failure, usually because of insuffiecient memory. The non-zero code can be used with the object.delete method to remove the hash object.

Macros for variables for a hash object

```
%let dsn1=work.cust;
%let out_dsn1=Mro_varlist;
%let dsn_cmp=work.pol;

proc contents data=&dsn1
      out=work.&out_dsn1 (keep=libname memname name type length nobs varnum)
      noprint;
run;
proc sort data=work.&out_dsn1;
      by varnum;
run;

/*dataset for the hash object*/
proc sql noprint inobs=1;
      select libname, memname
      into :H_lib, :H_dsn
      from &out_dsn1;
quit;
```

```
39          %put &H_lib &H_dsn;
WORK     CUST
```

```
%put &H_lib &H_dsn;

/* &Attr_mv Attributes ** &Key_mv Key variables **** &Data_mv Data variables *** &Miss_mv Call missing variables ******/
/*Attrib macro variable*/

data H_mac_v(keep=name Key_Item attr varnum R_name re_name);
      set work.&out_dsn1;
            if type = 2 then A_type="$";
            else if type = 1 then A_type=" ";
            Attr=Catx(" ",name,"length=",A_type,length);
            Key_Item=quote(compress(name));
            R_name=cats("R_",name);
            Re_name=catx(" ",name,"=",R_name);
            where varnum in (2 3 4 5 6);
run;
```

```
/**********attrib and keep statements**************/
proc sql noprint;
      select attr, name, Re_name
            into :Attr_mv separated by " ",
                :Keep1_mv separated by " ",
                  :Re_name separated by " "
            from H_mac_v;
            *where varnum in (1 2 3 4 5 6);
            where varnum between 2 and 6; /*select varnum for attrib*/
quit;

%put &Attr_mv;
 %put &Keep1_mv;
%put &Re_name;
```

```
64          %put &Attr_mv;
policy_id length= $ 10 First_name length= $ 15 Last_Name length= $ 15 city length= $ 15 Country length= $ 15
65          %put &Keep1_mv;
policy_id First_name Last_Name city Country
66          %put &Re_name;
policy_id = R_policy_id First_name = R_First_name Last_Name = R_Last_Name city = R_city Country = R_Country
```

```
/***********KEY Macro variable*****************/
proc sql noprint;
      select Key_Item
            into :Key_mv separated by ","
            from H_mac_v
            where varnum in (2); /*select varnum for key item*/
quit;

%put &Key_mv;
```

```
71          %put &Key_mv;
"policy_id"
```

```
/***************Data Macro variable and Call missing ******************/
proc sql noprint;
      select Key_Item as D_Item,name
            into :Data_mv separated by “,”,
                :Miss_mv separated by “,”
            from H_mac_v
            /*where varnum in (1); /*select varnum for data item(s) and call missing*/
            where varnum between 2 and 7;*/
quit;
```

```
85          %put &Data_mv;
"policy_id","First_name","Last_Name","city","Country"
86          %put &Miss_mv;
policy_id,First_name,Last_Name,city,Country
```

%put &Data_mv;

%put &Miss_mv;

```
data work.examp&H_dsn;
      *if _n_=0 then set work.&H_dsn(keep=&keep1_mv);
            attrib &Attr_mv;
      if _n_=1 then do;
            dcl hash exp(dataset:”work.&H_dsn”);
            exp.definekey (&Key_mv);
            exp.definedata (&Data_mv);
            exp.definedone();
            call missing(&miss_mv);
      end;
      set  &dsn_cmp;
      rc=exp.find(key: policy_id);
      if rc=0 then exp.output(dataset:’DumpH’);
      if rc=0 then output work.examp&H_dsn;
run;
```

REFERENCES

Bloom, Janice and Secosky, Jason, Getting Started with the DATA Step Hash Iterator, Paper 271-2007, SAS Global Forum 2007.

Burlew, Michele M. SAS® Hash Object Programming Made Easy, Cary, NC: SAS Institute, 2012.

Carpenter, Art, Carpenter's Guide to Innovative SAS Techniques, Cary, N: SAS Institute, Inc., 2012.

Dorfman, Paul and Shajenko, Lessia S. , Hash Crash and Beyond Paper 037-2008 SAS Global Forum 2008.

Dorfman, Paul and Vyverman, Koen, The SAS Hash Object in Action, Paper 153-2009 SAS Global Forum 2009.

Dorfman, Paul and Vyverman, Koen, Data Step Hash Objects as Programming Tools, Paper 241-31, SUGI 31.

Dorfman, Paul, Vymerman, Koen, and Dorfman, Victor, Black Belt Hashigana, Paper 023-2010, SAS Global Forum 2010.

Hinson, Joseph The Hash-of-Hashes as a "Russian Doll" Structure: An Example with XML Creation, Paper 021-2013, SAS Global Forum 2013,

Loren, Judy How Do I Love Hash Tables? Let Me Count the Ways!, Paper 029-2008 SAS Global Forum 2008.

Ray, Robert and Secosky, Jason, Better Hashing in SAS® 9.2, Paper 306 2008.

SAS Community Hash_Object_Resources

SAS Institute Inc., SAS® 9.3 Component Objects Reference, 2011.

Secosky, Jason and Janice Bloom, Getting Started with the DATA Step Hash Object, Paper 271-2007, SAS Global Forum 2007.

Endnotes

[1] The Hash Object and the Hash Iterator Object first appeared in SAS 9.0. In later versions of SAS, the Java, Logger, and Appender Objects were added and the hash and hash iterator objects were greatly improved.

[2] Romeo and Juliet, Act II, Scene 2, line 43.

```
data match ;
[3]     set small point = _n_ ; * get key/data attributes for parameter type matching ;
        * set small (obs = 1) ; * this will work, too :-)! ;
        * if 0 then set small ; * and so will this :-)! ;
        * set small (obs = 0) ; * but for some reason, this will not :-( ;
```


Contact information:

Philip Burtch burtch@mac.com or philip.burtch@usaa.com

Data

```
Data work.cust;
        infile datalines dsd;
        input  Cust_id: $10.
                policy_id: $10.
                First_name: $15.
                Last_Name: $15.
                city: $15.
                Country : $15.;

        datalines;
pe12489,81880,Seth,Pecksniff,London,Eng
sh15878,11489,,Shakespeare,London,Eng
sh15878,98355,Shakespeare,,London,Eng
ma48798,22389,Thomas,Mann,Princeton,USA
ch12489,81889,Martin,Chuzlewit,London,Eng
do78521,12587,Amy,Dorrit,London,Eng
to12587,36901,Leo,Tolstoy,Yasnaya Polyana,Russia
ga78521,38525,Elizabeth,Gaskell,Knutsford,Eng
ni33691,66381,Nicholas,Nickleby,London,Eng
cl11483,22871,Arthur,Clennam,London,Eng
go78224,36999,Oliver,Goldsmith,London,Eng
ti78902,65547,Montague,Tigg,London,Eng
ti78902,58749,Tigg,Montague,London,Eng
ta48963,22158,Mark,Tapley,London,Eng
ch78984,44879,Chevy,Slyme,London,Eng
ja48548,22487,John,Jarndyce,Bleak House,Eng
gu77892,14856,William,Guppy,London,Eng
 ;
run;
```

```
Data work.Pol;
        infile datalines dsd;
        input Policy_id:$10.
                vin: $17.
                Year: $4.
                Make: $17.
                Abbr_Vin $2.;
                datalines;
11489,1FA00000030000001,2003,Ford,F3
22389,1G100000050000002,2005,Chevrolet,G5
81889,JHG00000030000003,2003,Honda,H3
12587,1FA00000050000004,2005,Ford,F5
36901,1GC00000030000005,2005,Chevrolet,G5
38525,JHG00000030000006,2003,Honda,H3
66381,1FB00000050000007,2005,Ford,F5
22871,1G100000030000008,2003,Chevrolet,G3
36999,JHG00000050000009,2005,Honda,H5
81889,1FA00000030000010,2003,Ford,F3
12587,1G100000050000011,2005,Chevrolet,G5
36901,JHG00000030000012,2003,Honda,H3
65547,1FA00000030000013,2003,Ford,F3
65547,1G100000050000014,2005,Chevrolet,G5
58749,JHG00000030000015,2003,Honda,H3
58749,1FA00000050000016,2005,Ford,F5
22158,1GC00000030000017,2005,Chevrolet,G3
98355,JHG00000030000018,2003,Honda,H3
22158,JHG00000030000019,2003,Honda,H3
44879,1FB00000050000020,2005,Ford,F5
44879,1G100000030000021,2003,Chevrolet,G3
22487,JHG00000050000022,2005,Honda,H5
14856,1FA00000030000023,2003,Ford,F3
65547,1G100000050000024,2005,Chevrolet,G5
65547,JHG00000030000025,2003,Honda,H3
 ;
run;
```

```
data work.claim;
	infile datalines dsd;
	input Policy_id:$10.
		claim_no: $10.
		Vin: $17.;
		datalines;
81889,7141,JHG00000030000003
12587,7249,1FA00000050000004
22871,7681,1G100000030000008
98355,7897,JHG00000030000018
36901,8113,JHG00000030000005
;
run;
```