

# Getting Started with the DATA Step Hash Iterator

Janice Bloom, SAS Institute Inc., Cary, NC

Jason Secosky, SAS Institute Inc., Cary, NC

## ABSTRACT

The DATA step hash iterator is a companion for the DATA step hash object. It enables programs to step through hash object records without performing key lookups. The dynamic nature of the hash object, along with the hash iterator, enables writing simpler and more efficient programs than were possible with techniques available prior to SAS®9. This paper introduces the hash iterator and how it interacts with a hash object. It also demonstrates common uses for the hash iterator and illustrates stumbling blocks when first coding with the hash iterator. This paper expands on topics presented in the authors' "Getting Started with the DATA Step Hash Object" paper [1].

## INTRODUCTION

As previously discussed in "Getting Started with the DATA Step Hash Object", the hash object is an in-memory lookup table accessible from the DATA step. A hash object is loaded with records and is only available from the DATA step that creates it. A hash record consists of two parts: a key part and a data part. The key part consists of one or more character and numeric values (as defined with the DEFINEKEY method) and must be a unique identifier for a hash record. The data part consists of zero or more character and numeric values (as defined with the DEFINEDATA method).

The hash iterator works with a hash object and allows another means to access values in the hash object without using a key lookup. Methods are used with the hash iterator to point to an item in the hash object based upon location, not value. The hash iterator can also be used to output items in the hash object conditionally when you do not want to use the hash OUTPUT method.

The first section of this paper introduces the functionality of the hash iterator by discussing the methods used to define an iterator and methods used to move an iterator across hash object records. The second section presents examples of common data retrieval using the hash iterator. Examples include creating subset data sets of a certain number of observations from one master data set, merging using date ranges, creating a data set from observations found in multiple data sets at least one time and removing items from a hash object. The final section of the paper presents common messages and missteps you may encounter when you first start coding your hash iterator logic.

## HASH ITERATOR METHODS

The hash iterator is designed to work with a hash object. You cannot define an iterator without also defining a hash object. The hash iterator acts as a pointing mechanism for its associated hash object and is used to move backwards and/or forwards over a hash object.

The following program demonstrates how a hash iterator is declared and manipulated. The first DATA step creates a data set that contains student names and test scores. The second DATA step uses a hash object to hold the previously created data set and associates a hash iterator with the hash object. The top and bottom quartiles are calculated and the observations in the top 25% are output to TOP\_QUARTER and the bottom 25% are output to BOTTOM\_QUARTER.

```
data class_scores;
  input name :$9. score @@;
datalines;
Callum 82 Chyou 92
Damien 89 Grace 74
Jorge 99 Kyle 85
Lachlan 83 Lucia 90
Mia 91 Niamh 80
Philipp 89 Tomas 76
;

data top_quarter bottom_quarter;
  length name $9 score 8;
  keep name score;
  if _n =1 then do;
    declare hash h(dataset: 'class_scores', ordered: 'descending');
    h.defineKey('score','name');
    h.defineData('score','name');
    h.defineDone();
    call missing(name, score);
```

```

    declare hiter hi('h');
end;

count=h.num_items;
n=int(count * .25);
hi.first();
do i=1 to n;
    output top_quarter;
    hi.next();
end;

hi.last();
do i=1 to n;
    output bottom_quarter;
    hi.prev();
end;
run;

proc print data=top_quarter;
run;

proc print data=bottom_quarter;
run;

```

#### Output:

Obs	name	score
1	Jorge	99
2	Chyou	92
3	Mia	91

Obs	name	score
1	Grace	74
2	Tomas	76
3	Niamh	80

In the second DATA step, during the first iteration of the implicit loop (`_N_ is 1`) the program executes to create and load the hash object and define the hash iterator. This initialization includes the following:

1. **Create hash object H.** Use the DATASET: argument tag to load the hash object with the data set CLASS\_SCORES. The ORDERED: argument tag specifies the key values from CLASS\_SCORES be loaded into the hash object H in descending order. 'D' or 'd' can be used as an alias for 'descending'.
2. **Specify key variables.** The DEFINEKEY method specifies the character variables SCORE and NAME as keys in the hash object H. Using NAME and SCORE as key variables gives a unique combination, so every observation in CLASS\_SCORES is in the hash object H. If SCORE was the only key variable defined, only the first occurrence of SCORE would be added to the hash object H because duplicate key values are not allowed in a hash object.
3. **Specify data variables.** The DEFINEDATA method is used to specify both SCORE and NAME as data variables. Only variables defined as data variables in the hash object have their values automatically returned to DATA step variables when hash iterator methods like FIRST or LAST are executed.
4. **Complete the definition.** The definition of a hash object is concluded by calling the DEFINEDONE method. When DEFINEDONE is called, DEFINEKEY and DEFINEDATA can no longer be called. If DATASET: was specified on the DECLARE statement, the data set is loaded into the hash object.

The SAS compiler does not see variable assignments in the hash iterator. If there is no explicit assignment statement for the variables specified in the DEFINEKEY and DEFINEDATA methods in the program itself, a NOTE is written to the log that the variables are not initialized. CALL MISSING is used to avoid these notes.

5. **Declare the hash iterator HI.** Specify the associated hash object H for the hash iterator HI within quotes. You must declare the associated hash object first, as its name is used as the argument tag to the hash

iterator.

Once the hash object is created and loaded and a hash iterator is associated with the hash object, we can use the iterator to retrieve data from the hash object. The FIRST method is used to point to the first item in the hash object. The LAST method is used to point the iterator to the last item in the hash object. Once you have the iterator pointing to a terminal item in the hash object, you can use NEXT and PREV (previous) methods to move back and forth over items in the hash object.

In this program, we purposefully load CLASS\_SCORES into the hash object H in descending order. For many tasks, the ordered tag is key to obtaining the correct results with the hash iterator. No other data is read in this step. All processing and output is done by the hash object with the hash iterator.

6. **Determine quartile size.** Use the method NUM\_ITEMS to return the number of items in the hash object H. Multiply this result by .25, take the integer of the product and assign the result into a variable N. N is the number of items we need to retrieve from the 'top' and 'bottom' of the hash iterator, HI.
7. **Output top quartile.** The FIRST method moves the hash iterator to the first item in the hash object. Once we are on the first item, we output the values for the DEFINEDATA variables to TOP\_QUARTER and use the NEXT method to move to the next item in the hash object N times.

The same type of logic is employed to pull the last quartile.

8. **Output bottom quartile.** The LAST method moves the hash iterator to the last item in the hash object and the values for the DEFINEDATA variables are output to BOTTOM\_QUARTER when the PREV method executes N times.

This program demonstrates the steps required to create and load a hash object and associate a hash iterator. It also shows how to move over the hash object with the iterator, rather than a KEY value. A DATA step with a SET statement, FIRSTOBS= option, and OBS= option could have been used to write an equivalent program. The input data set would have required a PROC SORT for a SET with OBS= approach to work. The hash iterator is able to perform the task with less I/O and without modifying the sort order of the data on disk.

## HASH ITERATOR AND DATA RETRIEVAL

The previous section illustrated one application of the hash iterator. With only four iterator methods, you may be surprised to find the hash iterator can fit many purposes. This section presents common data retrieval tasks that can not be accomplished with the FIND method, but can be done with the help of a hash iterator.

### FUZZY MERGE

The iterator is useful for table lookups that can not be based on the exact value of a key. For example, fuzzy merges combine values that are not exact matches. If you want a 'match' that is close to the defined key value, but is not the exact value, the FIND method is not an option. The hash iterator allows retrieval of data values from a hash object when lookup values may not be equal to the value of the defined key variable(s) but may be considered 'within the range' of the target value and be considered a 'match'.

In the following example, MASTER contains the variable SEGMENT and a date range, BEGIN\_DATE and END\_DATE. CHECKPOINTS contains ID and DATE. The goal is to identify observations in CHECKPOINTS that fall within the date range from MASTER.

```
data master;
  input begin_date :mmddy10. end_date :mmddy10. segment $1.;
datalines;
01/15/2006 02/28/2006 A
03/07/2006 05/20/2006 B
06/27/2006 08/03/2006 C
09/07/2006 10/16/2006 D
11/01/2006 12/05/2006 E
;

data checkpoints;
  input id date mmddy10.;
datalines;
1 01/31/2006
1 07/04/2006
2 08/15/2006
3 10/05/2006
```

```

3 11/28/2006
3 12/26/2006
;

data captured;
  length begin_date end_date 8 segment $1;
  format begin_date end_date date mmddyy8.;
  if _n_ = 1 then do;
    declare hash h(dataset: 'master', ordered: 'ascending');
    h.defineKey('begin_date');
    h.defineData('begin_date', 'end_date', 'segment');
    h.defineDone();
    call missing(begin_date, end_date, segment);

    declare hiter iter('h');
  end;

  set checkpoints;

  /* Point iterator to first item in the hash object H */
  iter.first();

  /* Keep executing the loop, looking for a 'matching range' until the NEXT */
  /* method has no other item to point to in the hash object, and fails. */
  do until (rc ne 0);
    if begin_date <= date <= end_date then do;
      output;
      /* Found a matching range, no need to continue moving through the hash object */
      leave;
    end;

    /* Since the data is sorted in the hash object by BEGIN_DATE, once you */
    /* encounter a BEGIN_DATE that is greater than DATE, no remaining unchecked */
    /* items can meet the IF condition, so do not use the resources to check them. */
    else if begin_date > date then leave;

    /* Move to the next item in the hash object */
    rc = iter.next();
  end;
run;

proc print;
  var id date segment begin_date end_date;
run;

```

#### Output:

Obs	id	date	segment	begin_date	end_date
1	1	01/31/06	A	01/15/06	02/28/06
2	1	07/04/06	C	06/27/06	08/03/06
3	3	10/05/06	D	09/07/06	10/16/06
4	3	11/28/06	E	11/01/06	12/05/06

In the third DATA step, the hash object H and its associated hash iterator ITER are defined. The value of DATE is checked to see if it falls between BEGIN\_DATE and END\_DATE.

1. **Set up hash object and hash iterator.** On the first iteration of the DATA step, MASTER is loaded in ascending order into the hash object H. 'A', 'a', or 'yes' can be used as an alias for 'ascending'. Loading in ascending order allows for 'short circuiting' logic in the step to save checking hash object items unnecessarily. BEGIN\_DATE is defined as the key variable in H. BEGIN\_DATE, END\_DATE, and SEGMENT are defined as associated data. A hash iterator, ITER, is associated with H.
2. **Read data from CHECKPOINTS.** When the SET statement executes, an observation is read from CHECKPOINTS. The values of ID and DATE are loaded into DATA step variables.
3. **Move to the first item in the hash object.** The FIRST method moves the iterator pointer to the first item in H.

4. **Iterate over the hash object.** The DO UNTIL loop contains two logic checks based upon the value of DATE. If DATE falls between BEGIN\_DATE and END\_DATE, it is considered a 'match' and is output to CAPTURED. In this example, there are no overlapping date ranges, so once a match is found, no further checking is needed. The LEAVE statement is used to 'short circuit' further searching with the hash iterator. The LEAVE statement moves execution out of the DO UNTIL loop, where there are no additional executable statements, so execution resumes at the top of the DATA step.

If the date condition is not met and items still remain in the hash object, the NEXT method continues to execute until each item in the hash object has been checked. Because the hash object was loaded in ascending order, once you detect a BEGIN\_DATE in the hash object that is greater than DATE, no other items in the hash object can meet the date range condition. Again, a LEAVE statement is used to 'short circuit' further searching of the hash object. Short circuiting is a good way to save resources when using the hash iterator.

If neither scenario above is met, every item in the hash object is checked. After the last item in the hash object is checked, the NEXT method is not successful and RC is set to a non-zero value. The DO UNTIL loop stops, the DATA step iterates, and another record is read from CHECKPOINTS.

This task could be accomplished using DATA step SET statements and array logic or PROC SQL. PROC SQL generates a Cartesian product between the data sets before paring down to the observations that meet the date range criteria. Depending on the size of the data sets involved, this can consume significant resources to generate the full combination just to eliminate much of the output that does not meet the date range condition. Compared to a DATA step with array logic, the hash can be more convenient to load in sort order, lending itself to 'short circuiting' logic.

#### CONDITIONAL OUTPUT

The hash iterator can also be used to output items in a hash object conditionally. In the example below, four attendance data sets are created. The goal is to identify the ID values that appear in three data sets at least one time.

Note ABC123's ID was accidentally entered twice in ATTENDANCE1. ABC123 is not in ATTENDANCE2, so simply checking for three occurrences of an ID does not accurately reflect attendance in *each* of the data sets. Additional logic using the IN= option ensures we count only one occurrence from each data set.

```
data attendance1;
  input id $ @@;
datalines;
ABC123 DEF789 GHI345 JKL567 ABC123
;

data attendance2;
  input id $ @@;
datalines;
ADE798 GLH890 GHI345 TSH234 JKL567
;

data attendance3;
  input id $ @@;
datalines;
GHI345 ABC123 JKL567 GLH890
;

data attendance4;
  input id $ @@;
datalines;
GHI777 DEF789 GLH890
;

data completed;
  length DSN $2;

  if _n_=1 then do;
    declare hash h(ordered:'ascending');
```

```

    h.definekey('id');
    h.definedata('id','cnt','ds');
    h.definedone();
    declare hiter hi('h');
end;

set attendance1(in=a1) attendance2(in=a2) attendance3(in=a3)
    attendance4(in=a4) end=last;
if a1 then dsn='a1';
else if a2 then dsn='a2';
else if a3 then dsn='a3';
else if a4 then dsn='a4';

rc=h.find();
if rc=0 then do;
    if dsn ne ds then cnt +1;
    h.replace();
end;

else do;
    ds=dsn;
    cnt=1;
    h.add();
end;

if last then do;
    rc=hi.first();
    do while(rc=0);
        if cnt=3 then output;
        rc=hi.next();
    end;
end;
run;

proc print data=completed;
    var id;
run;

```

#### Output:

Obs	id
1	GHI345
2	GLH890
3	JKL567

So far, all the examples in this paper have used the DATASET: argument tag to load a data set into the defined hash object. In this example, the hash object is loaded one item at a time with the ADD method. A counter is incremented to track each occurrence of ID, excluding duplicates in the same data set. The hash iterator is used to conditionally output items in the hash object that have a counter value of three.

1. **Set up the hash object and hash iterator.** On the first iteration of the DATA step, the hash object H is declared. The ORDERED: argument tag ensures the items are added to the hash object in ascending order. ('A', 'a', or 'yes' could be used as an alias for 'ascending'.) ID is defined as a key variable. ID, CNT, and DS are defined as the associated data for each key value. ID is defined as a data value so it appears on the output data set. The hash iterator HI is associated with the hash object H.
2. **Add items.** The SET statement reads in the values from ATTENDANCE1, ATTENDANCE2, ATTENDANCE3, and ATTENDANCE4. The END= option is specified to help determine when the last observation has been read from the final data set. The IN= option is used to create Boolean variables that are 'true' when the current value comes from the specified data set. If the IN= value is true, assign its name to DSN.

A FIND method is executed to determine if the current value of ID has already been added to the hash

object. If the current value of ID is already in the hash object (rc=0) then compare the value of DS from the hash object to DSN. If DS=DSN, then we know the current value of ID is a duplicate for the same data set and should not be counted again. If DSN is different from DS, then CNT is incremented and the new value of CNT is updated in the hash object via the REPLACE method. If the current value of ID is not found in the hash object (rc ne 0) then DS is assigned the value from DSN and CNT is set to 1. The values of ID, CNT, and DS are added to the hash object using the ADD method.

3. **Conditional output.** After all the observations have been read from the four data sets, the FIRST method is executed to move the hash iterator HI to the first item in the hash object. While the hash iterator can successfully move to the next item in the hash object, (while rc=0), check the value of CNT. If CNT=3 then output the current item in the hash object to WORK.COMPLETED.

This task could be accomplished using MERGE with a BY statement and IN= logic but the data sets would need to be sorted or indexed. A PROC FREQ could be run on each individual data set and the results combined to determine attendance rates, but multiple steps would be needed.

Note, beginning in SAS 9.2, it is possible to use data set options with input data sets loaded into a hash object and output data sets created from a hash object. This enhancement simplifies conditional output from a hash object. For example, in SAS 9.2, the iterator logic in the sample above can be replaced with the line

```
if last then h.output(dataset: 'completed(where = (cnt=3))');
```

Another 9.2 addition that simplifies this code sample is the INDSNAME= option for the SET statement. This option stores the two level name of the data set being read in a new variable. This new option could be used to create DSN instead of the IN= data set option and assignment logic above.

### REMOVING ITEMS FROM A HASH OBJECT

For some tasks, items need to be removed from a hash object. The iterator can be used to remove items in a hash object based upon location, rather than KEY value. The next example uses the iterator to remove all of the items in a hash object. The hash object could be deleted, but using the iterator and REMOVE method saves repeatedly rebuilding the hash object.

```
data work.sixitems;
  do a=1 to 6;
    output;
  end;
run;

data _null_ ;
  declare hash h(dataset: 'work.sixitems') ;
  h.defineKey('a') ;
  h.defineDone() ;
  declare hiter hi('h') ;
  call missing(a);
  num=h.num_items;
  put 'Start with ' num= 'items in the hash object';

  hi.last() ;
  temp = a ;

  /* As long as the PREV method successfully executes, execute a REMOVE */
  /* using the previous value of A. */
  do until(prc ne 0);
    prc=hi.prev();
    h.remove(key: temp) ;
    /* Reassign the current value of A into TEMP. */
    temp = a ;
    /* Grab the number of items remaining in the hash object */
    /* and write it to the log. */
    num=h.num_items;
    put num=;
  end ;
run;
```

This sample only executes one time. It is meant to illustrate a method of removing all items from a hash object (in lieu

of deleting the hash object and rebuilding it in the same step.) This example would not be used 'as is', but the removal logic could be used in tasks that require clearing all the values from a hash object without a key value coming from another data set, or clearing the hash when a BY-Group changes, etc.

1. **Create hash object H and hash iterator HI.** Use the DATASET: argument tag to load WORK.SIXITEMS into H. Define A as the key variable. Associate hash iterator HI with the hash object H.
2. **For illustration purposes, create NUM from the NUM\_ITEMS attribute.** Write out a line to the log to confirm the number of items in the hash object H.
3. **Use the LAST method to move iterator HI to the last item in the hash object H.** Successful execution of the LAST method sets the DATA step variable A. Assign the value of A into TEMP.
4. **Use a DO UNTIL loop to move backwards through the items in the hash object with the PREV method.** A non-zero return code for PREV stops the loop when PRC is checked at the bottom of the loop.

When the PREV method executes, the item in the hash object currently being pointed to loads its value for A into the DATA step variable A. At this point A has been updated, but TEMP still holds the value from the last assignment. In essence, we'll be removing the last item and then assign the current item's A value into TEMP for the next REMOVE. Since the DO UNTIL condition is checked at the bottom of the loop, when PREV fails due to having only one item left in the hash object the final REMOVE executes using the last assigned value in TEMP. NUM is written out to the log each iteration to show items are being successfully removed from H.

## MESSAGES AND MISSTEPS

The samples presented in this paper were chosen to illustrate common scenarios for exploiting a hash object with the hash iterator. As you begin creating sample hash iterator code on your own, unexpected ERRORS may appear in the SAS log. This section is meant to present a few of the more common messages you may see in the SAS log as you get started.

1. **ERROR: Invalid hash object specified for Hash iterator at line nn column nn.**  
**ERROR: DATA STEP Component Object failure. Aborted during the EXECUTION phase.**

The hash iterator must specify a hash object that has already been declared. This error occurs if you try to associate an iterator with a hash object that has not been declared yet. To replicate this error, take the first sample presented in this paper and switch the order of the DECLARE statements so the DECLARE hiter hi('h') statement is first.

```
data top_quarter bottom_quarter;
  length name $9 score 8;
  keep name score;
  if _n_=1 then do;
    declare hiter hi('h');
    declare hash h(dataset: 'class_scores', ordered:'descending');
    h.definekey('score','name');
    h.definedata('name', 'score');
    h.definedone();
    call missing(name, score);
  end;

  count=h.num_items;
  n=int(count*.25);
  rc=hi.first();
  do i=1 to n;
    output top_quarter;
    rc=hi.next();
  end;

  rc=hi.last();
  do i=1 to n;
    output bottom_quarter;
    rc=hi.prev();
  end;
run;
```

2. **ERROR: Incorrect number of parameters at line nnn column nn.**  
**ERROR: Hash iterator constructor requires character string argument at line nnn**

```
column nn.  
ERROR: Invalid hash object specified for Hash iterator at line nnn column nn.
```

Failing to specify the name of the hash object in the DECLARE method for the hash iterator generates the first error. Forgetting to quote the correct hash object name when declaring the hash iterator generates the second error. Misspelling the hash object's name when declaring the hash iterator generates the third error.

```
data top_quarter bottom_quarter;  
  length name $9 score 8;  
  keep name score;  
  if _n =1 then do;  
    declare hash h(dataset: 'class_scores', ordered:'descending');  
    declare hiter hi();  
    *declare hiter hi(h);  
    *declare hiter hi('hhh');  
    h.definekey('score','name');  
    h.definedata('name','score');  
    h.definedone();  
    call missing(name, score);  
  end;  
  
  count=h.num_items;  
  n=int(count * .25);  
  rc=hi.first();  
  do i=1 to n;  
    output top_quarter;  
    rc=hi.next();  
  end;  
  
  rc=hi.last();  
  do i=1 to n;  
    output bottom_quarter;  
    rc=hi.prev();  
  end;  
run;
```

3. **ERROR: Cannot remove record from a hash object which has been locked by an iterator at line nnn column nn.**

The iterator can be used to remove items from a hash object. You can start the iterator at the first or last item in the hash object using FIRST or LAST methods and execute the REMOVE method. After deleting the item, use NEXT to move to the next item. This approach removes all of the items in the hash object except the final item. You can not remove an item the iterator is pointing to. (See the example "Removing Items from a Hash Object" for a working solution to this limitation.) Beginning in SAS 9.2, the CLEAR method can be used to clear an entire hash object without deleting the table.

## CONCLUSIONS

The hash iterator is a helpful companion to the hash object. It is useful for data retrieval when an exact key value is not available. The iterator can also be used to conditionally output items from the hash object when the output method is not ideal. Items can also be removed from a hash object with assistance from an iterator.

Using the hash object and hash iterator can help reduce CPU time and I/O when used in appropriate scenarios. There are many ways to accomplish a task in SAS. We hope that by introducing you to the hash object and hash iterator, you are open to trying a new technique for new programs and your existing tasks!

## ACKNOWLEDGMENTS

Bill Heffner and Al Kulik extended the DATA step to include the DECLARE statement and dot syntax. Al Kulik implemented the hash object and hash iterator.

## REFERENCES

[1] Secosky, J., Bloom, J. 2006. *Getting Started with the DATA Step Hash Object*. SAS Institute, Inc. <http://support.sas.com/rnd/base/topics/datastep/dot/hash-getting-started.pdf> (accessed April 2, 2007).

## CONTACT INFORMATION

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

Janice Bloom  
SAS Institute Inc.  
SAS Campus Drive  
Cary, NC 27513  
919-677-8000  
[Janice.Bloom@sas.com](mailto:Janice.Bloom@sas.com)

Jason Secosky  
SAS Institute Inc.  
SAS Campus Drive  
Cary, NC 27513  
919-677-8000  
[Jason.Secosky@sas.com](mailto:Jason.Secosky@sas.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.  
Other brand and product names are trademarks of their respective companies.