

How to Implement the SAS® DATA Step Hash Object

Bill Parman, BlueCross-BlueShield of Tennessee, Chattanooga, TN

ABSTRACT

In SAS® Version 9.1, the DATA Step hash object became available. It is safe to say that one of its most valuable features is that of efficient searching of memory-resident data. Because a complete examination of the SAS® DATA Step hash object far exceeds the 20-minute time frame allowed for this presentation, this paper will focus primarily on the syntax required to declare and search a SAS® DATA Step hash table. Two examples are provided. The first example illustrates the implementation of the DATA Step hash object with a simple key and single return value. The second example illustrates the implementation of the DATA Step hash object with a composite key and multiple return values. The objective of this paper is to provide the reader with sufficient information, so that he/she can return to their workstation and implement the SAS® DATA Step hash object within the context of table searching.

INTRODUCTION

Although there are two DATA step objects related to the use of hash tables, HASH and HITER, it is the purpose of this paper to focus on the declaration of the HASH object and its FIND method.

WHAT IS HASHING?

THE DEFINITION

Hashing is a search algorithm based on a mathematical technique to convert a key into an array index. Hashing is not new. SAS® Institute has simply taken the existing complex search algorithm and encapsulated its characteristics and processes into an object accessible from the DATA step.

THE CONCEPT

Hashing incorporates 3 elements: a hash function, a hash table, and buckets. The hash function converts keys into array indices. The hash table is a memory-resident table that stores the data to be searched. Buckets are locations in the hash table, referenced by the indices created by the hash function.

There is the possibility that the hash function may hash two or more key values from its data source into the same array index. This is called a collision. To accommodate collisions, SAS® implements their hash object's hash table as an array of AVL trees. When collisions occur, they are accommodated by placing the key values with the same hash index in separate locations within the AVL tree located at the computed hash index of the hash table. This technique not only accommodates collisions, but the data storage potential for a hash table is limited only by the available memory on one's computer.

Due to the following reasons, searching entries with the hash object is very efficient: 1) the data is memory-resident, 2) the hash function can quickly determine a hash table index, and 3) the AVL tree beginning at each bucket in the hash table makes it optimally fast to find any key values that have resulted in collisions.

A PERFORMANCE COMPARISON OF THE DATA STEP HASH OBJECT

With SAS® running on a server during non-peak hours, a transaction data set of 437,179 observations and a lookup data set of 11,651 observations produced the following results when joining the two tables:

MERGE WITH BY	8.54S	SET WITH KEY=	7.97S
Sort data sets	6.98s	Build Index	0.01s
Search	1.56s	Search	7.96s
FORMAT AND PUT()	1.85S	HASH TABLE	1.62S
Build formats	0.25s	Load and search	1.62s

IMPLEMENTING THE SAS® DATA STEP HASH OBJECT

SAS® Institute makes the implementation of hashing easy by encapsulating the concept into its DATA step hash object. To implement the DATA step hash object, the programmer only needs a data source in the form of a SAS® data set and seven (7) instructions, 5 of which require a simple understanding of “dot” syntax of object oriented coding (i.e. `ObjectName.Method(parameters)`).

To illustrate, suppose that a very large file named “Master” contains a column named “Code” that stores 2-character codes, for which a description must be acquired from another file named “CodeRef.” CodeRef will be used as our data source for a hash object and contains 2 columns. The first column in CodeRef is named “Code” and contains 2-character codes in the same format as the column named, “Code” in the Master file. The second column in CodeRef is named “Description” and contains 25-character descriptions for the corresponding codes. Using CodeRef, let’s look at the instructions required to build a hash object.

STEP 1: LENGTH STATEMENT

```
Length
  Code          $ 2
  Description    $ 25
;
```

- From the data source, each variable used to define the hash table must be defined in a LENGTH statement.
- Without defining the variables from the hash table data source in a LENGTH statement, the following message will be displayed in the SAS® Log:

```
ERROR: Variable <variable name> has been defined as both
character and numeric.
```

STEP 2: DECLARE THE HASH OBJECT

```
Declare Hash MyLkup(HashExp:8,Dataset:'CodeRef');
```

- The “Declare Hash...” statement identifies the name of the hash object created by the programmer, establishes the number of buckets in the hash table, and identifies the SAS® data set from which the hash table will be populated.
- In the example above, MyLkup is the name given to the hash object by the programmer. It can be any valid SAS® variable name.
- The number 8 is the hash exponent. The base for the hash exponent is 2 and it determines the number of buckets in the hash table (i.e. $2^{**}8 = 256$ bucket).
 - Default value is 8
 - Valid values are integers in the range 0 to 16
 - Using zero creates a hash table with one bucket (i.e. $2^{**}0 = 1$) resulting in a binary search, rather than a hash search, of the AVL tree located at that bucket location
 - The hash table size (number of buckets) is not equal to the number of items that can be stored
 - To maximize the efficiency of the hash object lookup routines, you should set the hash table size according to the amount of data in the hash object. Try different hashexp values until you get the best result.
- CodeRef is the name of the SAS® data set from which the hash table, MyLkup, will be populated and must be bounded with single- or double-quotation marks.

STEP 3: DEFINE THE KEY FOR THE HASH OBJECT

```
Rc = MyLkup.DefineKey('Code'); or MyLkup.DefineKey('Code');
```

- DefineKey is a method used by the hash object to define the hash table's key.
- MyLkup is the name of the hash object coded in the Declare Hash ... statement.
- Rc is a numeric variable into which the return code value, from executing the DefineKey method, is stored.
- "Code" is the column name from the data set, CodeRef that is used as the hash table key. Enclose it with single- or double-quotation marks.

STEP 4: DEFINE THE DATA VALUES FOR THE HASH OBJECT

```
Rc = MyLkup.DefineData('Description'); or MyLkup.DefineData('Description');
```

- DefineData is a method used by the hash object to define the values returned when the hash table is searched.
 - If it is not necessary to return a data value(s) when searching a hash table, it is not necessary to code this method.
- MyLkup is the name of the hash object coded in the Declare Hash ... statement.
- Rc is a numeric variable into which the return code value, from executing the DefineData method, is stored.
- "Description" is the column from the data set, CodeRef that is used as the hash table's data value. Enclose it with single- or double-quotation marks.

STEP 5: CONCLUDE THE HASH OBJECT DECLARATION

```
Rc = MyLkup.DefineDone(); or MyLkup.DefineDone();
```

- DefineDone is a method used by the hash object to conclude the declaration of the hash table.
- MyLkup is the name of the hash object coded in the Declare Hash ... statement.
- Rc is a numeric variable into which the return code value, from executing the DefineDone method, is stored.

STEP 6: INITIALIZE THE VARIABLES USED IN THE HASH OBJECT TO MISSING VALUES

```
Call Missing(Code, Description);
```

- This statement is used to suppress the following messages from the SAS® Log:

NOTE: Variable Code is uninitialized.

NOTE: Variable Description is uninitialized.

STEP 7: SEARCHING THE HASH OBJECT

```
Rc = MyLkup.Find(Key:Code);
```

- 'Find' is a method used by the hash object to search the hash table
- MyLkup is the name of the hash object coded in the Declare Hash ... statement.
- Rc is a numeric variable into which the return code value, from executing the Find method, is stored.

- A return code value of zero indicates that the execution of the Find method was successful.
- Code is the column name from the SAS® data set, Master, used to search the hash table.

NOTE: Steps 2 through 6 are coded in a DO/END block on an IF statement checking _N_ for a value of 1.

NOTE: Only variables listed in the DefineKey and DefineData methods will be stored in the hash table.

EXAMPLES

ESAMPLE 1: SIMPLE KEY WITH SINGLE DATA VALUE

```

/* Create the data set to be used as the data source for */
/* the hash table */
Data CodeRef;
  Infile Cards Dlm=', ' DSD;
  Input
    Code          $
    Description   : $25.
  ;
  Output CodeRef;
Cards;
01,Bolts
02,Screws
03,Taps
04,Nails
05,Tacks
;
Run;

/* Create the data set (transaction file) from which the */
/* values in its column named, Code, will be used to */
/* search the hash table */
Data Master;
  Infile Cards Dlm=', ' DSD;
  Input
    Region      $
    State       $
    Store       : $15.
    Code        $
  ;
  Output Master;
Cards;
North,MI,King Hardware,01
North,MI,King Hardware,03
North,MI,King Hardware,04
North,WI,Home Crafts,02
North,WI,Home Crafts,05
South,IN,King Hardware,01
South,IN,King Hardware,02
South,IN,King Hardware,03
South,IN,King Hardware,04
South,IN,King Hardward,05
South,IL,Home Crafts,01
South,IL,Home Crafts,03
South,IL,Home Crafts,05
;
Run;

/* Illustrate the creation of the hash table, the processing of */
/* the Master data set, and the searching of the hash table */
Data Results(Drop=Rc);

  /* Define the variables to be used in the hash table declaration */
  Length
    Code          $ 2
    Description   $ 25
  ;

```

```

/* Build the hash table */
If _N_ = 1 Then
  Do;
    Declare Hash MyLkup(HashExp:8,Dataset:'CodeRef');
    MyLkup.DefineKey('Code');
    MyLkup.DefineData('Description');
    MyLkup.DefineDone();
    Call Missing(Code, Description);
  End;

/* Read an observation from the Master (transaction) file */
Set Master;

/* Using the value of Code from the current Master file observation, */
/* search the hash table defined above */
Rc = MyLkup.Find(Key:Code);

/* Upon a successful search, write an observation to the output data set */
If Rc = 0 Then
  Do;
    Output Results;
  End;
Run;

```

EXAMPLE 2: COMPOSITE KEY WITH MULTIPLE DATA VALUES

```

/* Create the data set to be used as the data source for */
/* the hash table */
Data MemberRef;
  Infile Cards Dlm=', ' DSD;
  Input
    MemberId      $
    MemberSufx    $
    MemberName    : $20.
    DiscountRt
  ;
  Output MemberRef;
Cards;
100001,00,Samuel Smith,.03
100001,01,Jane Smith,.05
100001,02,Kimberly Smith,.08
100001,03,Raymond Smith,.07
100002,00,Vernon Jones,.03
100003,00,Jerry Rodrigez,.03
100003,01,Maria Rodrigez,.05
100003,02,Juan Rodrigez,.07
100004,00,William Phelps,.03
100004,01,Jennifer Phelps,.05
100004,02,Roger Phelps,.07
;
Run;

/* Create the data set (transaction file) from which the */
/* values in its columns named, MemberId and MemberSufx, */
/* will be used to search the hash table */
Data Claims;
  Infile Cards Dlm=', ' DSD;
  Input
    ClaimNo      $
    MemberId      $
    MemberSufx    $
    BilledAmt
  ;
  Output Claims;
Cards;
A0001,100001,00,200
A0013,100003,01,100
A0013,100003,01,200

```

```

A0215,100004,00,500
A0220,100004,01,400
A0220,100004,01,200
A0220,100004,01,100
A0350,100004,02,200
;
Run;

/* Illustrate the creation of the hash table, the processing of */
/* the transaction data set, and the searching of the hash table */
Data Results(Drop=Rc);

  /* Define the variables to be used in the hash table declaration */
  Length
    MemberId      $6
    MemberSufx    $2
    MemberName    $20
    DiscountRt    8
  ;

  /* Build the hash table */
  If _N_ = 1 Then
    Do;
      Declare Hash MbrLkup(HashExp:8,Dataset:'MemberRef');
      MbrLkup.DefineKey('Memberid','MemberSufx');
      MbrLkup.DefineData('MemberName','DiscountRt');
      MbrLkup.DefineDone();
      Call Missing(MemberId,MemberSufx,MemberName,DiscountRt);
    End;

  /* Read an observation from the Claims (transaction) file */
  Set Claims;

  /* Using the values of MemberId and MemberSufx from the current */
  /* Claims file observation, search the hash table defined above */
  Rc = MbrLkup.Find(Key: MemberId, Key: MemberSufx);

  /* Upon a successful search, compute the member's charged amount */
  /* and write an observation to the output data set */
  If Rc = 0 Then
    Do;
      ChargedAmt = (1 - DiscountRt) * BilledAmt;
      Output Results;
    End;
  ;

Run;

```

CONCLUSION

When joining two data sets, the DATA step hash object is used best when the data set for the hash table is relatively small when compared to the transaction data set. The hash object has the advantage of speed. Since it is wholly implemented in the DATA step, it does not require additional preparation steps, such as, SORT procedures, creation of dynamic FORMAT procedures, and the creation of data set indexes.

Compared to joining two data sets by using a dynamically created PROC FORMAT, the hash object can take advantage of composite keys and multiple return values without the overhead of data conversions, string concatenation, and string parsing. Whereas, PROC FORMAT only return character results, the hash object can return numeric and /or character results without the requirement of any data conversions, specifically, character-to-numeric.

REFERENCES

SAS® Online Help

SAS® Institute Technical Support

http://www.saswin.com/offices/europe/slovakia/uni/resource/SAS_Data_Step_vo_V9.pdf

<http://www2.sas.com/proceedings/sugi31/241-31.pdf>

CONTACT INFORMATION

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

Bill Parman
BlueCross-BlueShield of Tennessee, 3E
801 Pine St
Chattanooga, TN 37402

Business: 423-535-7853
Email: Bill_Parman@BCBST.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.