ABSTRACT
The hash object provides an efficient method for quick data storage and data retrieval. Using a common set of lookup keys, hash objects can be used to retrieve data, store data, merge / join tables of data, and split a single table into multiple tables. This paper illustrates what a hash object is, why use hash objects, and basic programming instructions associated the construction and use of hash objects in a DATA step.

INTRODUCTION
Please note: this paper uses hash object and hash table interchangeably.
A hash object is created, used, and automatically removed inside a SAS DATA step. The hash object is created within the local memory and it can be written to a permanent file in which case it will lose all of its “hash” functionalities outside of the DATA step in which it was created.
The DATA step provides the following methods to create and use a hash object:

- Declare a hash object
- Instantiate a hash object
- Store data in a hash object
- Merge/Join datasets using a hash object
- Output / write out a hash object to temporary or permanent file.

This paper covers the following topics:
- What is an Object in Object-oriented programming?
- What is a hash object?
- Why use hash objects?
- How is a hash object constructed and used?
- How a hash object can be used to split a dataset into multiple datasets based on a key?
- What is a Hash Object Iterator?
- SAS Hash Memory Usage – Will not be covered
- Hash of Hashes – Will not be covered
This paper does NOT cover the following topics:

- SAS Hash Memory Usage
- Hash of Hashes

**TOPIC 1: WHAT IS AN OBJECT IN OBJECT-ORIENTED PROGRAMMING?**

In an object oriented programming language, an object is a composition of nouns (such as a number, a string, a variable) and verbs (such as actions, functions).

A class is a “blue print” that conceptually describes an object. In other words, an object is a physical instance (manifestation) of a class.

**TOPIC 2: WHAT IS A HASH OBJECT?**

A hash object is a "table/dataset" that has been key-indexed for faster access and retrieval.

The index allows for direct access; hence, accessing the data becomes faster.

In case of duplicate keys, the hash object is constructed using the first sequential occurrence of the duplicate key. A hash object is dynamically created in memory at run-time. The size of a hash object grows as items are added and it contracts as items are removed.

A hash object consists of key columns, data columns, and methods such as DECLARE, FIND, etc.

A hash object’s scope is limited to the DATA step in which it is created.

A hash object is automatically removed when the DATA step completes.

**TOPIC 3: WHY USE HASH OBJECTS?**

A hash object can perform database merges and joins faster than DATA step MERGE or PROC SQL joins. A hash object can also be used to split a single dataset into multiple datasets based on a key.

In case of duplicate keys, the hash object is constructed using the first sequential occurrence of the duplicate key.

**TOPIC 4: HOW IS A HASH OBJECT CREATED AND USED?**

A hash object is created within a DATA step, used within the same DATA step, and it is automatically removed when the DATA step completes.

A hash object has NO scope outside of the DATA step in which it was created.

Please see Figure 1:
**EXAMPLE: CREATE AND USE A HASH OBJECT**

This example merges two datasets.
The first dataset is called WORK.PERSON_POSSESION_TBL and the second dataset is called WORK.PERSON_OCCUPATION_TBL.

Step 1: The first dataset is created by using a DATA step and DATALINES:

```plaintext
DATA WORK.PERSON_POSSESION_TBL;
  INPUT PERSON $ POSSESSION $;
  DATALINES;
  A       BMW
  B       BMW
  C       HOUSE
  C       MUTUALFUND
;
RUN;
```

The resulting dataset is: Dataset: WORK.PERSON_POSSESION_TBL

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
</tr>
<tr>
<td>C</td>
<td>MUTUALFUND</td>
</tr>
</tbody>
</table>

Step 2: The second dataset is created by using a DATA step and DATALINES:

```plaintext
DATA WORK.PERSON_OCCUPATION_TBL;
  INPUT PERSON $ OCCUPATION $;
  DATALINES;
  A       DOCTOR
  B       ENGINEER
  C       TEACHER
```

**Figure 1. Using a Data step to create/define a hash object.**
The resulting dataset is: WORK.PERSON_OCCUPATION_TBL.

<table>
<thead>
<tr>
<th>PERSON</th>
<th>OCCUPATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>DOCTOR</td>
</tr>
<tr>
<td>B</td>
<td>ENGINEER</td>
</tr>
<tr>
<td>C</td>
<td>TEACHER</td>
</tr>
<tr>
<td>D</td>
<td>SALES</td>
</tr>
</tbody>
</table>

Step 3: using a DATA step, we will create the hash object out of the POSSESSION table and merge it with the occupation table.

```sas
DATA WORK.PERSON_POSS_OCCUP_TBL (DROP=RC);

/*Compile directive – makes the columns of the source table for the hash object*/
/*visible to the DATA step*/
IF 0 THEN SET WORK.PERSON_POSSESION_TBL;

/*Upon the first iteration of the input dataset, declare and create the hash table*/
IF (_N_ = 1) THEN
   DO;
      /*Declare the hash table*/
      DECLARE HASH HH_POSSESION
      (DATASET: 'WORK.PERSON_POSSESION_TBL', ORDERED: 'A');

      /*DefineKey defines the key column in the hash table*/
      HH_POSSESION.DefineKey('PERSON');

      /*DefineData defines the data columns in the hash table*/
      /*It is not required to include the key column in the hash table.*/
      /*However, if there is a need to know the key; then, then the key needs*/
      /*to be included in the data list*/
      HH_POSSESION.DefineData('PERSON', 'POSSESION');

      /*DefineDone completes the declaration of the hash table*/
   END;
```
/* Output writes the hash table to an external file. This is not required. It is*/
/* done only if the hash table needs to be preserved for subsequent processing*/
/* at which point it will not be a hash object. */

HH_POSSESSION.Output(Dataset: 'WORK.HH_POSSESSION');
END;

/* Set the second dataset as input dataset*/
SET WORK.PERSON_OCCUPATION _TBL;

/* Using the PERSON column from the OCCUPATION table, search the*/
/* hash table for the key. */
RC = HH_POSSESSION.Find(KEY: PERSON);

/* If the key is NOT found, initialize the output column to NotFound. */
IF (RC NE 0) THEN POSSESION = "NotFound";

/* Output outputs the file whether they is found or not. */
OUTPUT;
RUN;

The resulting hash object/dataset: WORK.PERSON_POSSESSION_TBL.

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
</tr>
</tbody>
</table>

The resulting merged dataset: WORK.PERSON_POSSESS_OCCUP_TBL:

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESION</th>
<th>OCCUPATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
<td>DOCTOR</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
<td>ENGINEER</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
<td>TEACHER</td>
</tr>
<tr>
<td>C</td>
<td>NotFound</td>
<td>SALES</td>
</tr>
</tbody>
</table>
TOPIC 5: HOW A HASH OBJECT CAN BE USED TO SPLIT A SINGLE DATASET INTO MULTIPLE DATASETS BASED ON A KEY?

Hash objects can be used to split a single dataset into multiple datasets based on a common key. This concept is demonstrated by the following example:

Step 1: The dataset is created by using a DATA step and DATALINES.

```
DATA WORK.PERSON_POSSESION_TBL;
  INPUT PERSON $ POSSESSION $;
  DATALINES;
  A      BMW
  B      BMW
  C      HOUSE
  C      MUTUALFUND
;
RUN;
```

The resulting dataset is: WORK.PERSON_POSSESION_TBL.

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
</tr>
<tr>
<td>C</td>
<td>MUTUALFUND</td>
</tr>
</tbody>
</table>

Step 2: The input dataset needs to be sorted by the “split” key: POSSESSION.

```
PROC SORT DATA= WORK.PERSON_POSSESION_TBL
   OUT= WORK.PERSON_POSSESION_TBL;
   BY POSSESSION;
RUN;
```

Step 3: By using a hash object, we will split the dataset into multiple datasets by concatenating
the key (POSSESSION) with the current record count (_N_).

```plaintext
DATA _NULL_;  
    /*Declare the hash object.*/  
    DECLARE HASH HH_SPLIT (ORDERED: 'A'); 
    HH_SPLIT.DefineKey('POSSESSION', '_N_');  
    HH_SPLIT.DefineData('POSSESSION', 'PERSON');  
    HH_SPLIT.DefineDone(); 
    /*Iterate thru the input dataset by POSSESSION column and create*/  
    /*the hash table by using the hash object’s ADD method*/  
    DO _N_ = 1 BY 1 UNTIL (LAST. POSSESION);  
        SET WORK.PERSON_POSSESION_TBL;  
        BY POSSESION;  
        HH_SPLIT.ADD();  
    END;  
    /* Output the split datasets by concatenating the word 'WORK.' with the*/  
    /*value of the POSSESSION column and the with the word '_TBL' */  
    HH_SPLIT.Output (Dataset: 'WORK.' || Trim(POSSESION) || '_TBL');  
RUN;```

The three resulting datasets are:
Dataset: WORK.BMW_TBL:

<table>
<thead>
<tr>
<th>POSSESSION</th>
<th>PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>A</td>
</tr>
<tr>
<td>BMW</td>
<td>B</td>
</tr>
</tbody>
</table>

Dataset: WORK.HOUSE_TBL:

<table>
<thead>
<tr>
<th>POSSESSION</th>
<th>PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOUSE</td>
<td>C</td>
</tr>
</tbody>
</table>
Dataset: WORK. MUTUALFUND _TBL;

<table>
<thead>
<tr>
<th>POSSESSION</th>
<th>PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTUALFUND</td>
<td>C</td>
</tr>
</tbody>
</table>

**TOPIC 6: WHAT IS THE HASH OBJECT ITERATOR?**

The hash object iterator enables traversing thru the hash object data in either forward or reverse key order.

This concept is demonstrated by the following examples:

Example 1: Hash object with **forward iterator**.

Step 1: Create the input dataset.

```plaintext
DATA WORK.PERSON_POSSESION_TBL;

   INPUT PERSON $ POSSESION $;

DATALINES;
   A       BMW
   B       BMW
   C       HOUSE
   C       MUTUALFUND
;
RUN;
```

The resulting dataset is: WORK.PERSON_POSSESION_TBL:

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
</tr>
<tr>
<td>C</td>
<td>MUTUALFUND</td>
</tr>
</tbody>
</table>

Step 2: Create and use a hash object **forward iterator**.

```plaintext
DATA _NULL_;```
IF 0 THEN SET WORK.PERSON_POSSESION_TBL;
DECLARE HASH H_POSSESION
   (DATASET: 'WORK.PERSON_POSSESION_TBL', ORDERED: 'A');

DECLARE HITER HH_POSSESION('H_POSSESION');

DECLARE HASH HH_SPLIT (ORDERED: 'A');
H_POSSESION.DefineKey('PERSON');
H_POSSESION.DefineData('PERSON', 'POSSESION');
H_POSSESION.DefineDone();
H_POSSESION.Output(Dataset: 'WORK.H_POSSESION');

/*Traverse the hash object in forward or ascending order of its key
/*Retrieve the first record*/
RC = HH_POSSESION.FIRST();
DO WHILE (RC = 0);
    PUT PERSON    +1    POSSESION;
    RC = HH_POSSESION.NEXT();
END;
RUN;

The log will display:
A       BMW
B       BMW
C       HOUSE

Example 2: Hash object with **forward iterator**.

Step 1: Create the input dataset.

DATA WORK.PERSON_POSSESION_TBL;

   INPUT PERSON $ POSSESION $;

   DATALINES;
   A       BMW

9
The resulting dataset is: WORK.PERSON_POSSESION_TBL

<table>
<thead>
<tr>
<th>PERSON</th>
<th>POSSESION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BMW</td>
</tr>
<tr>
<td>B</td>
<td>BMW</td>
</tr>
<tr>
<td>C</td>
<td>HOUSE</td>
</tr>
<tr>
<td>C</td>
<td>MUTUALFUND</td>
</tr>
</tbody>
</table>

Step 2: Create and use a hash object reverse iterator.

```plaintext
DATA _NULL_;  
IF 0 THEN SET WORK.PERSON_POSSESION_TBL;  
DECLARE HASH H_POSSESION  
   (DATASET: 'WORK.PERSON_POSSESION_TBL', ORDERED: 'A');  
DECLARE HITER HH_POSSESION('H_POSSESION');  
DECLARE HASH HH_SPLIT (ORDERED: 'A');  
H_POSSESION.DefineKey('PERSON');  
H_POSSESION.DefineData('PERSON', 'POSSESION');  
H_POSSESION.DefineDone();  
H_POSSESION.Output(Dataset: 'WORK.H_POSSESION');  
/*Traverse the hash object in forward or ascending order of its key*/  
/*Retrieve the first record*/  
RC = HH_POSSESION.LAST();  
DO WHILE (RC = 0);  
   PUT PERSON +1 POSSESION;  
   RC = HH_POSSESION.PREV();  
END;```
RUN;

The log will display:
C  HOUSE
B  BMW
A  BMW

CONCLUSION
The hash object provides a powerful set of techniques to merge and split datasets. This paper provided an introduction to hash objects.

REFERENCES
An Introduction to SAS® Hash Programming Techniques
Kirk Paul Ladler, proceedings from SESUG 2011

ABOUT THE AUTHOR
Dari Mazloom is a lead business intelligence advisor with over thirty years of practical experience in a number of programming languages such as SAS, C#, C++, JAVA, VB, VBA, Smalltalk, COBOL, Easytrieve, JCL.