ABSTRACT
SAS® users are always interested in learning techniques that will help them improve the performance of table lookup, search, and sort operations. SAS software supports a DATA step programming technique known as a hash object to associate a key with one or more values. This presentation introduces what a hash object is, how it works, the syntax required, and simple applications of it use. Essential programming techniques will be illustrated to sort data and search memory-resident data using a simple key to find a single value.

INTRODUCTION
One of the more exciting and relevant programming techniques available to SAS users today is the Hash object. Available as a DATA step construct, users are able to construct relatively simple code to perform match-merge and/or join operations. The purpose of this paper and presentation is to introduce the basics of what a hash table is and to illustrate practical applications so SAS users everywhere can begin to take advantage of this powerful SAS Base programming feature.

EXAMPLE TABLES
The data used in all the examples in this paper consists of a Movies data set containing six columns: title, length, category, year, studio, and rating. Title, category, studio, and rating are defined as character columns with length and year being defined as numeric columns. The data stored in the Movies data set appears below.

<table>
<thead>
<tr>
<th>Title</th>
<th>Length</th>
<th>Category</th>
<th>Year</th>
<th>Studio</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braveheart</td>
<td>117</td>
<td>Action/Adventure</td>
<td>1995</td>
<td>Paramount Pictures</td>
<td>PG</td>
</tr>
<tr>
<td>Cavalcade</td>
<td>103</td>
<td>Drama</td>
<td>1942</td>
<td>MGM / UA</td>
<td>PG</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>97</td>
<td>Comedy</td>
<td>1989</td>
<td>Warner Brothers</td>
<td>PG-13</td>
</tr>
<tr>
<td>Coming to America</td>
<td>116</td>
<td>Comedy</td>
<td>1980</td>
<td>Paramount Pictures</td>
<td>PG</td>
</tr>
<tr>
<td>Dances</td>
<td>130</td>
<td>Romance</td>
<td>1993</td>
<td>Columbia TriStar</td>
<td>R</td>
</tr>
<tr>
<td>Directed to Kill</td>
<td>95</td>
<td>Drama</td>
<td>1960</td>
<td>Warner Brothers</td>
<td>PG-13</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>142</td>
<td>Drama</td>
<td>1984</td>
<td>Paramount Pictures</td>
<td>PG-13</td>
</tr>
<tr>
<td>Schindler's List</td>
<td>137</td>
<td>Drama</td>
<td>1993</td>
<td>Paramount Pictures</td>
<td>PG-13</td>
</tr>
<tr>
<td>Studio</td>
<td>125</td>
<td>Action/Adventure</td>
<td>1975</td>
<td>Universal Studios</td>
<td>PG</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>110</td>
<td>Action/Cop &amp; Robber</td>
<td>1987</td>
<td>Warner Brothers</td>
<td>PG</td>
</tr>
<tr>
<td>Michael</td>
<td>134</td>
<td>Drama</td>
<td>1987</td>
<td>Warner Brothers</td>
<td>PG-13</td>
</tr>
<tr>
<td>National Lampoon's Vacation</td>
<td>98</td>
<td>Comedy</td>
<td>1983</td>
<td>Warner Brothers</td>
<td>PG-13</td>
</tr>
<tr>
<td>Planes</td>
<td>115</td>
<td>Drama</td>
<td>1992</td>
<td>MGM / UA</td>
<td>PG</td>
</tr>
<tr>
<td>Rocky</td>
<td>120</td>
<td>Action/Adventure</td>
<td>1976</td>
<td>MGM / UA</td>
<td>PG</td>
</tr>
<tr>
<td>Starman</td>
<td>170</td>
<td>Action/Cop &amp; Robber</td>
<td>1983</td>
<td>Universal Studios</td>
<td>R</td>
</tr>
<tr>
<td>Starship Troopers</td>
<td>110</td>
<td>Drama</td>
<td>1999</td>
<td>Columbia TriStar</td>
<td>R</td>
</tr>
<tr>
<td>Star Trek</td>
<td>124</td>
<td>Action/SF</td>
<td>1979</td>
<td>Lucas Film Ltd</td>
<td>PG</td>
</tr>
<tr>
<td>The Hunt for Red October</td>
<td>135</td>
<td>Action/Adventure</td>
<td>1989</td>
<td>Paramount Pictures</td>
<td>PG-13</td>
</tr>
<tr>
<td>The Terminator</td>
<td>108</td>
<td>Action/SF</td>
<td>1984</td>
<td>Lion Entertainment</td>
<td>R</td>
</tr>
<tr>
<td>The Virtue of D2</td>
<td>101</td>
<td>Action/Adventure</td>
<td>1999</td>
<td>MGM / UA</td>
<td>PG</td>
</tr>
<tr>
<td>Titanic</td>
<td>191</td>
<td>Drama</td>
<td>1997</td>
<td>Paramount Pictures</td>
<td>PG-13</td>
</tr>
</tbody>
</table>

The second data set used in the examples is the ACTORS data set. It contains three columns: title, actor_leading, and actor_supporting, all of which are defined as character columns, and is illustrated below.

<table>
<thead>
<tr>
<th>Title</th>
<th>Actor_Leading</th>
<th>Actor_Supporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braveheart</td>
<td>Mel Gibson</td>
<td>Sophie Marceau</td>
</tr>
<tr>
<td>Christmas Vacation</td>
<td>Chevy Chase</td>
<td>Beverly D'Angelo</td>
</tr>
<tr>
<td>Coming to America</td>
<td>Eddie Murphy</td>
<td>Jason Mamoa</td>
</tr>
<tr>
<td>Dances</td>
<td>Tom Hanks</td>
<td>Sally Field</td>
</tr>
<tr>
<td>Directed to Kill</td>
<td>Patrick Swayze</td>
<td>David Morse</td>
</tr>
<tr>
<td>Forrest Gump</td>
<td>Mel Gibson</td>
<td>Danny Glover</td>
</tr>
<tr>
<td>Jumanji</td>
<td>John Travolta</td>
<td>Andy MacDowell</td>
</tr>
<tr>
<td>Lethal Weapon</td>
<td>Chevy Chase</td>
<td>Beverly D'Angelo</td>
</tr>
<tr>
<td>National Lampoon's Vacation</td>
<td>Chevy Chase</td>
<td>Beverly D'Angelo</td>
</tr>
<tr>
<td>Planes</td>
<td>Sylvester Stallone</td>
<td>Ed White</td>
</tr>
<tr>
<td>Schindler's List</td>
<td>Anthony Hopkins</td>
<td>Jude Law</td>
</tr>
</tbody>
</table>
WHAT IS A HASH OBJECT?
A hash object is a data structure that contains an array of items that are used to map identifying values, known as keys (e.g., employee IDs), to their associated values (e.g., employee names or employee addresses). As implemented, it is designed as a DATA step construct and is not available to any SAS PROCedures. The behavior of a hash object is similar to that of a SAS array in that the columns comprising it can be saved to a SAS table, but at the end of the DATA step the hash object and all its contents disappear.

HOW DOES A HASH OBJECT WORK?
A hash object permits table lookup operations to be performed considerably faster than other available methods found in the SAS system. Unlike a DATA step merge or PROC SQL join where the SAS system repeatedly accesses the contents of a table stored on disk to perform table lookup operations, a hash object reads the contents of a data set into memory once allowing the SAS system to repeatedly access it, as necessary. Since memory-based operations are typically faster than their disk-based counterparts, users generally experience faster and more efficient table lookup operations. The following diagram illustrates the process of performing a table lookup using the Movie Title (i.e., key) in the MOVIES data set matched against the Movie Title (i.e., key) in the ACTORS data set to return the ACTOR_LEADING and ACTOR_SUPPORTING information.

![Figure 1. Table Lookup Operation with Simple Key](image)

Although one or more hash tables may be constructed in a single DATA step that reads data into memory, users may experience insufficient memory conditions preventing larger tables from being successfully processed. To alleviate this kind of issue, users may want to load the smaller tables as hash tables and continue to sequentially process larger data sets containing lookup keys.

HASH OBJECT SYNTAX
Users with DATA step programming experience will find the hash object syntax relatively straightforward to learn and use. Available in all operating systems running SAS 9 or greater, the hash object is called using methods. The syntax for calling a method involves specifying the name of the user-assigned hash table, a dot (.), the desired method (e.g., operation) by name, and finally the specification for the method enclosed in parentheses. The following example illustrates the basic syntax for calling a method to define a key.

```
HashTitles. DefineKey ('Title');
```

where:

HashTitles is the name of the hash table, DefineKey is the name of the called method, and ‘Title’ is the specification being passed to the method.

HASH OBJECT METHODS
The author has identified twenty six (26) known methods which are alphabetically displayed, along with their description, in the following table.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Adds data associated with key to hash object.</td>
</tr>
<tr>
<td>CHECK</td>
<td>Checks whether key is stored in hash object.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Removes all items from a hash object without deleting hash object.</td>
</tr>
<tr>
<td>DEFINEDATA</td>
<td>Defines data to be stored in hash object.</td>
</tr>
<tr>
<td>DEFINEDONE</td>
<td>Specifies that all key and data definitions are complete.</td>
</tr>
<tr>
<td>DEFINEKEY</td>
<td>Defines key variables to the hash object.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes the hash or hash iterator object.</td>
</tr>
<tr>
<td>EQUALS</td>
<td>Determines whether two hash objects are equal.</td>
</tr>
<tr>
<td>FIND</td>
<td>Determines whether the key is stored in the hash object.</td>
</tr>
<tr>
<td>FIND_NEXT</td>
<td>The current list item in the key’s multiple item list is set to the next item.</td>
</tr>
<tr>
<td>FIND_PREV</td>
<td>The current list item in the key’s multiple item list is set to the previous item.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Returns the first value in the hash object.</td>
</tr>
<tr>
<td>HAS_NEXT</td>
<td>Determines whether another item is available in the current key’s list.</td>
</tr>
<tr>
<td>HAS_PREV</td>
<td>Determines whether a previous item is available in the current key’s list.</td>
</tr>
<tr>
<td>LAST</td>
<td>Returns the last value in the hash object.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Returns the next value in the hash object.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Creates one or more data sets containing the data in the hash object.</td>
</tr>
<tr>
<td>PREV</td>
<td>Returns the previous value in the hash object.</td>
</tr>
<tr>
<td>REF</td>
<td>Combines the FIND and ADD methods into a single method call.</td>
</tr>
<tr>
<td>REMOVE</td>
<td>Removes the data associated with a key from the hash object.</td>
</tr>
<tr>
<td>REMOVEDUP</td>
<td>Removes the data associated with a key’s current data item from the hash object.</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replaces the data associated with a key with new data.</td>
</tr>
<tr>
<td>REPLACEDUP</td>
<td>Replaces data associated with a key’s current data item with new data.</td>
</tr>
<tr>
<td>SETCUR</td>
<td>Specifies a starting key item for iteration.</td>
</tr>
<tr>
<td>SUM</td>
<td>Retrieves a summary value for a given key from the hash table and stores the value to a DATA step variable.</td>
</tr>
<tr>
<td>SUMDUP</td>
<td>Retrieves a summary value for the key’s current data item and stores the value to a DATA step variable.</td>
</tr>
</tbody>
</table>
SORT WITH A SIMPLE KEY

Sorting is a common task performed by SAS users everywhere. The SORT procedure is frequently used to rearrange the order of data set observations by the value(s) of one or more character or numeric variables. A feature that PROC SORT is able to do is replace the original data set or create a new ordered data set with the results of the sort. Using hash programming techniques, SAS users have an alternative to using the SORT procedure. In the following example, a user-written hash routine is constructed in the DATA step to perform a simple ascending data set sort. As illustrated, the metadata from the MOVIES data set is loaded into the hash table, a DefineKey method specifies an ascending sort using the variable LENGTH as the primary (simple) key, a DefineData method to select the desired variables, an Add method to add data to the hash object, and an Output method to define the data set to output the results of the sort to.

Hash Code with Simple Key

```sas
Libname mydata 'e:\workshops\workshop data' ;
data _null_;  
  if 0 then set mydata.movies; /* load variable properties into hash tables */
  if _n_ = 1 then do;
    declare Hash HashSort (ordered:'a'); /* declare the sort order for hash */
    HashSort.DefineKey ('Length'); /* identify variable to use as simple key */
    HashSort.DefineData ('Title',
                       'Length',
                       'Category',
                       'Rating'); /* identify columns of data */
    HashSort.DefineDone (); /* complete hash table definition */
  end;
  set mydata.movies end=eof;
  HashSort.add (); /* add data with key to hash object */
  if eof then HashSort.output(dataset:sorted_movies); /* write data using hash HashSort */
run;
```

As illustrated in the following SAS Log results, SAS processing stopped with a data-related error due to one or more duplicate key values. As a result, the output data set contained fewer results (observations) than expected.

SAS Log Results

```sas
Libname mydata 'e:\workshops\workshop data' ;
data _null_;  
  if 0 then set mydata.movies; /* load variable properties into hash tables */
  if _n_ = 1 then do;
    declare Hash HashSort (ordered:'a'); /* declare the sort order for hash */
    HashSort.DefineKey ('Length'); /* identify variable to use as simple key */
    HashSort.DefineData ('Title',
                         'Length',
                         'Category',
                         'Rating'); /* identify columns of data */
    HashSort.DefineDone (); /* complete hash table definition */
  end;
```
SORT WITH A COMPOSITE KEY

To resolve the error presented in the previous example, an improved and more uniquely defined key is specified. The simplest way to prevent a conflict consisting of duplicate is to add a secondary variable to the key creating a composite key. The following code illustrates constructing a composite key with a primary variable (LENGTH) and a secondary variable (TITLE) to reduce the prospect of producing a duplicate key value from occurring (collision).

Hash Code with Composite Key

```sas
data _null_; if 0 then set mydata.movies; /* load variable properties into hash tables */ if _n_ = 1 then do; declare Hash HashSort (ordered:'a'); /* declare the sort order for HashSort */ 1 HashSort.DefineKey ('Length', 'Title'); /* identify variables to use as composite key */ 2 HashSort.DefineData ('Title', 'Length', 'Category', 'Rating'); /* identify columns of data */ HashSort.DefineDone (); /* complete HashSort table definition */ end; set mydata.movies end=eof; 3 HashSort.add (); /* add data with key to HashSort table */ 4 if eof then HashSort.output(dataset:sorted_movies); /* write data using hash HashSort */ run;
```

SAS Log Results

As shown on the SAS Log results, the creation of the composite key of LENGTH and TITLE is sufficient enough to form a unique key enabling the sort process to complete successfully with 22 observations read from the MOVIES data set, 22 observations written to the SORTED_MOVIES data set, and zero conflicts (or collisions).
SEARCH AND LOOKUP WITH A SIMPLE KEY

Besides sorting, another essential action frequently performed by users is the process of table lookup or search. The hash object as implemented in the DATA step provides users with the necessary tools to conduct match-merges (or joins) of two or more data sets. Data does not have to be sorted or be in a designated sort order before use as it does with the DATA step merge process. The following code illustrates a hash object with a simple key (TITLE) to merge (or join) the MOVIES and ACTORS data sets to create a new dataset (MATCH_ON_MOVIE_TITLES) with matched observations.

```sas
data _null_;  
if 0 then set mydata.movies;  /* load variable properties into hash tables */
if _n_ = 1 then do;
   declare Hash HashSort (ordered:'a'); /* declare the sort order for HashSort */
   HashSort.DefineKey ('Length', 'Title'); /* identify variable to use as composite key */
   HashSort.DefineData ('Title',
      'Length',
      'Category',
      'Rating'); /* identify columns of data */
   HashSort.DefineDone (); /* complete HashSort table definition */
end;
set mydata.movies end=eof;
HashSort.add (); /* add data using key to HashSort table */
if eof then HashSort.output(dataset:'sorted_movies'); /* write data using HashSort */
run;

NOTE: The data set WORK.SORTED_MOVIES has 22 observations and 4 variables.
NOTE: There were 22 observations read from the data set MYDATA MOVIES.
```

```sas
data match_on_movie_titles(drop=rc);

1 if 0 then set mydata.movies
   mydata.actors; /* load variable properties into hash tables */
if _n_ = 1 then do;
2 declare Hash HashActors (dataset:'mydata.actors'); /* declare the name HashActors for hash */
   HashActors.DefineKey ('Title'); /* identify variable to use as key */
   HashActors.DefineData ('Actor_Leading',
      'Actor_Supporting'); /* identify columns of data */
   HashActors.DefineDone (); /* complete hash table definition */
end;
set mydata.movies;
3 if HashActors.find(key:title) = 0 then output; /* lookup TITLE in MOVIES table using HashActors object */
run;
```
Results
The match-merge (or join) process is illustrated using the following diagram.

TRANSPOSING WITH THE TRANSPOSE PROCEDURE
In the paper; SAS on a Shingle, Flippin with Hash (2012); Miller and Lafler illustrate two key points: 1) how PROC TRANSPOSE is used for converting SAS data set structures and 2) how hash programming techniques are used to emulate the PROC TRANSPOSE process. The objective was to demonstrate the programming techniques and select hash methods that were used to successfully create a transposed data set. For those unfamiliar or with limited experience using PROC TRANSPOSE, the SAS Base procedure gives SAS users a convenient way to transpose (or restructure) any SAS data set structure. Popular uses for PROC TRANSPOSE include:

✓ Converting the observations of a data set structure to variables, sometimes referred to as changing a vertical (long or thin) data structure to a horizontal (wide or fat) data structure;
✓ Converting the variables of a data set structure to observations, sometimes referred to as changing a horizontal (wide or fat) data structure to a vertical (long or thin) data structure.

Although experienced SAS users may use any number of approaches in lieu of the TRANSPOSE procedure to restructure a data set, these alternate techniques can require more time for programming, testing and debugging. The PROC TRANSPOSE syntax to restructure (or transpose) selected variables into observations is shown, below. After sorting the MOVIES data set in ascending order by TITLE, PROC TRANSPOSE then accesses the sorted MOVIES data set. The BY statement tells PROC TRANSPOSE to create BY-groups for the variable TITLE. The VAR statement specifies the variables, RATING and LENGTH, to transpose into observations. The result of the transpose process is then written to a data set called, Transposed_Movies.
PROC TRANSPOSE Code:

libname mydata "e:\workshops\workshop data" ;
proc sort data = mydata.movies
  out = sorted_movies ;
  by title ;
run ;
proc transpose data = sorted_movies
  out = transposed_movies ;
  by title ;
  var rating length ;
run;

The resulting Transposed_Movies data set from running the TRANSPOSE procedure, below, contains three variables: TITLE, _NAME_ and _COL1. With closer inspection, the data set contains duplicate TITLE values (observations), a distinct _NAME_ value for “Rating” in the first observation of COL1 and a distinct _NAME_ value for “Length” in the second observation of COL1 for each BY-group.

Transposed_Movies Data Set created with PROC TRANSPOSE
TRANSPOSING WITH THE DATA STEP HASH METHOD

My objective for using Hash methods in creating a restructured transposed data set is to emulate what is created with the TRANSPOSE procedure. We'll begin with the statement, “DATA Hash_Long_Movies”, because the application of Hash methods is currently only available in a DATA step. The next statement, “IF 0 THEN SET MYDATA.MOVIES” tells SAS to load variable properties into the hash object located in real memory. The DECLARE HASH statement provides a name to the hash object being created in memory as ‘Hash_movies’, the name of the input data set, and how the data is ordered. The “DECLARE HITER” statement defines and initializes the hash object for traversing the object in memory. The DEFINEKEY method identifies the variable (or variables) to use as the key. The DEFINEDATA method informs SAS what variables to read into the hash object in memory (in our case all variables not removed with the DROP= or KEEP= data set option). The DEFINEDONE method completes the hash table definition. The FIRST() method tells SAS to return the first value stored in the defined hash object. The DO WHILE loop iterates repeatedly as long as there is data stored in the hash object. The LINK OUTLONG statement tells SAS to execute the OUTLONG subroutine. The NEXT() method tells SAS to return the next value from the defined hash object. The STOP statement tells SAS to terminate the DATA step.

```sas
libname mydata 'e:\workshops\workshop data';
data hash_long_movies (drop=rc Rating Length);
if 0 then set mydata.movies(keep=Title Rating Length);
if _n_ = 1 then do;
   declare Hash Hash_movies(dataset:'mydata.movies', ordered:'ascending');
   declare Hiter Hi_movies ('Hash_movies');
   Hash_movies.DefineKey ('Title');
   Hash_movies.DefineData ('Title', 'Rating', 'Length');
   Hash_movies.DefineDone ();
end;
rc = Hi_movies.first();
do while (rc = 0);
   link outlong;
   rc = Hi_movies.next();
end;
stop;
return;
```
The resulting Hash_Long_Movies data set created with the Hash methods, below, contains three variables: TITLE, LABEL and VALUE. As with the transposed data set created earlier, this data set contains duplicate TITLEs, a distinct LABEL value for “Rating” in the first observation of VALUE and for “Length” in the second observation of VALUE for each BY-group.

```sas
%outlong: ;
   Title ;
   Label = 'Rating' ;
   Value = Rating ;
   output hash_long_movies ;

   Title ;
   Label = 'Length' ;
   Value = Length ;
   output hash_long_movies ;
   return ;
run ;
```

---

Hash_Long_Movies Data Set created with Hash Methods
CONCLUSION

Users have a powerful hash DATA-step construct to sort data, search data sets, perform table lookup operations, and transpose data sets in the SAS system. This paper introduced the basics of what a hash table is, how it works, the basic syntax, and its practical applications so SAS users everywhere can begin to take advantage of this powerful memory-based programming technique to improve the performance of sorts, searches, table lookup operations, merge (or join) and transposes.

REFERENCES


ACKNOWLEDGMENTS
The author thanks the Western Users of SAS Software (WUSS) 2019 Hands-On Workshop Section Chair for accepting my abstract and paper; Hunter Glanz, WUSS 2019 Academic Program Chair; Cynthia Alvarez, WUSS 2019 Operations Chair; the WUSS Conference Committee; and SAS Institute for organizing and supporting a great conference!

TRADEMARK CITATIONS
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.
CONTACT INFORMATION
Kirk Paul Lafler is an entrepreneur, consultant, educator and author, and has used SAS software since 1979. Currently, Kirk is a lecturer and adjunct professor at San Diego State University; an advisor and adjunct professor at University of California San Diego Extension; and teaches SAS courses, seminars, workshops and webinars to users around the world. As the author of PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019), Google® Search Complete (Odyssey Press. 2014) and hundreds of SAS papers and articles; Kirk has served as an Invited speaker, educator, keynote and section leader at SAS user group conferences and meetings worldwide; and is the recipient of 25 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/
LinkedIn: https://www.linkedin.com/in/Order-of-Magnitude-Analytics/
Twitter: @sasNerd