MWSUG 2016 – Paper HW02

A Hands-on Introduction to SAS® DATA Step Hash Programming Techniques

Kirk Paul Lafler, Software Intelligence Corporation, Spring Valley, California

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.

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

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.

	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	Christmas Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	Ghost	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegge	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet

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.

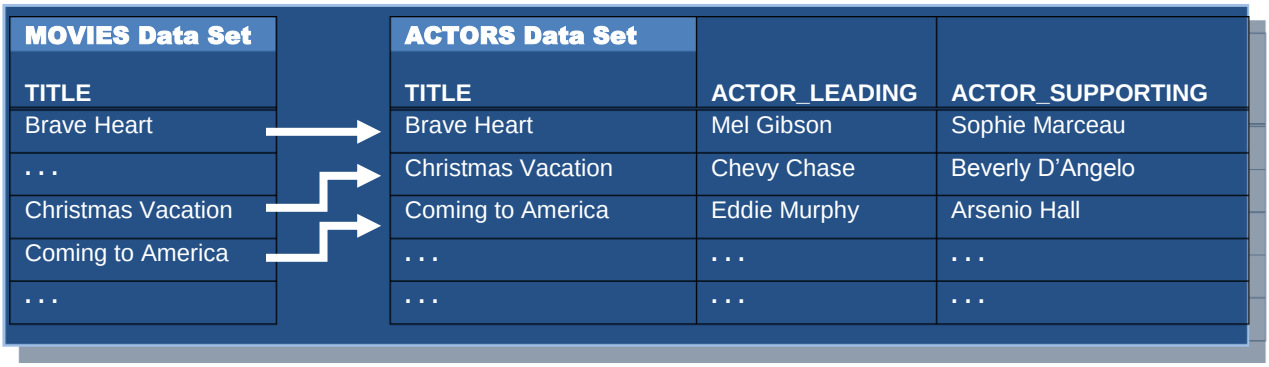


Figure 1. Table Lookup Operation with Simple Key

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 straight forward 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.

Method	Description
ADD	Adds data associated with key to hash object.
CHECK	Checks whether key is stored in hash object.
CLEAR	Removes all items from a hash object without deleting hash object.
DEFINEDATA	Defines data to be stored in hash object.
DEFINEDONE	Specifies that all key and data definitions are complete.
DEFINEKEY	Defines key variables to the hash object.
DELETE	Deletes the hash or hash iterator object.
EQUALS	Determines whether two hash objects are equal.
FIND	Determines whether the key is stored in the hash object.
FIND_NEXT	The current list item in the key's multiple item list is set to the next item.
FIND_PREV	The current list item in the key's multiple item list is set to the previous item.
FIRST	Returns the first value in the hash object.
HAS_NEXT	Determines whether another item is available in the current key's list.
HAS_PREV	Determines whether a previous item is available in the current key's list.
LAST	Returns the last value in the hash object.
NEXT	Returns the next value in the hash object.
OUTPUT	Creates one or more data sets containing the data in the hash object.
PREV	Returns the previous value in the hash object.
REF	Combines the FIND and ADD methods into a single method call.
REMOVE	Removes the data associated with a key from the hash object.
REMOVEDUP	Removes the data associated with a key's current data item from the hash object.
REPLACE	Replaces the data associated with a key with new data.
REPLACEDUP	Replaces data associated with a key's current data item with new data.
SETCUR	Specifies a starting key item for iteration.
SUM	Retrieves a summary value for a given key from the hash table and stores the value to a DATA step variable.
SUMDUP	Retrieves a summary value for the key's current data item and stores the value to a DATA step variable.

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

```
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 */
❶     HashSort.DefineKey ('Length');  /* identify variable to use as simple key */
❷     HashSort.DefineData ('Title',
                           'Length',
                           'Category',
                           'Rating');   /* identify columns of data */
      HashSort.DefineDone ();   /* complete hash table definition */
   end;
   set mydata.movies end=eof;
❸  HashSort.add ();   /* add data with key to hash object */
❹  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

```
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;
```

SAS Log Results (Continued)

```
    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;

ERROR: Duplicate key.
NOTE: The data set WORK.SORTED_MOVIES has 21 observations and 4 variables.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 22 observations read from the data set MYDATA.MOVIES.
```

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

```
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 */

❶   HashSort.DefineKey ('Length', 'Title'); /* identify variables to use as
                                               composite key */

❷   HashSort.DefineData ('Title',
                       'Length',
                       'Category',
                       'Rating');   /* identify columns of data */
    HashSort.DefineDone ();  /* complete HashSort table definition */
  end;
  set mydata.movies end=eof;

❸ HashSort.add ();  /* add data with key to HashSort table */

❹ 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).

```
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.
```

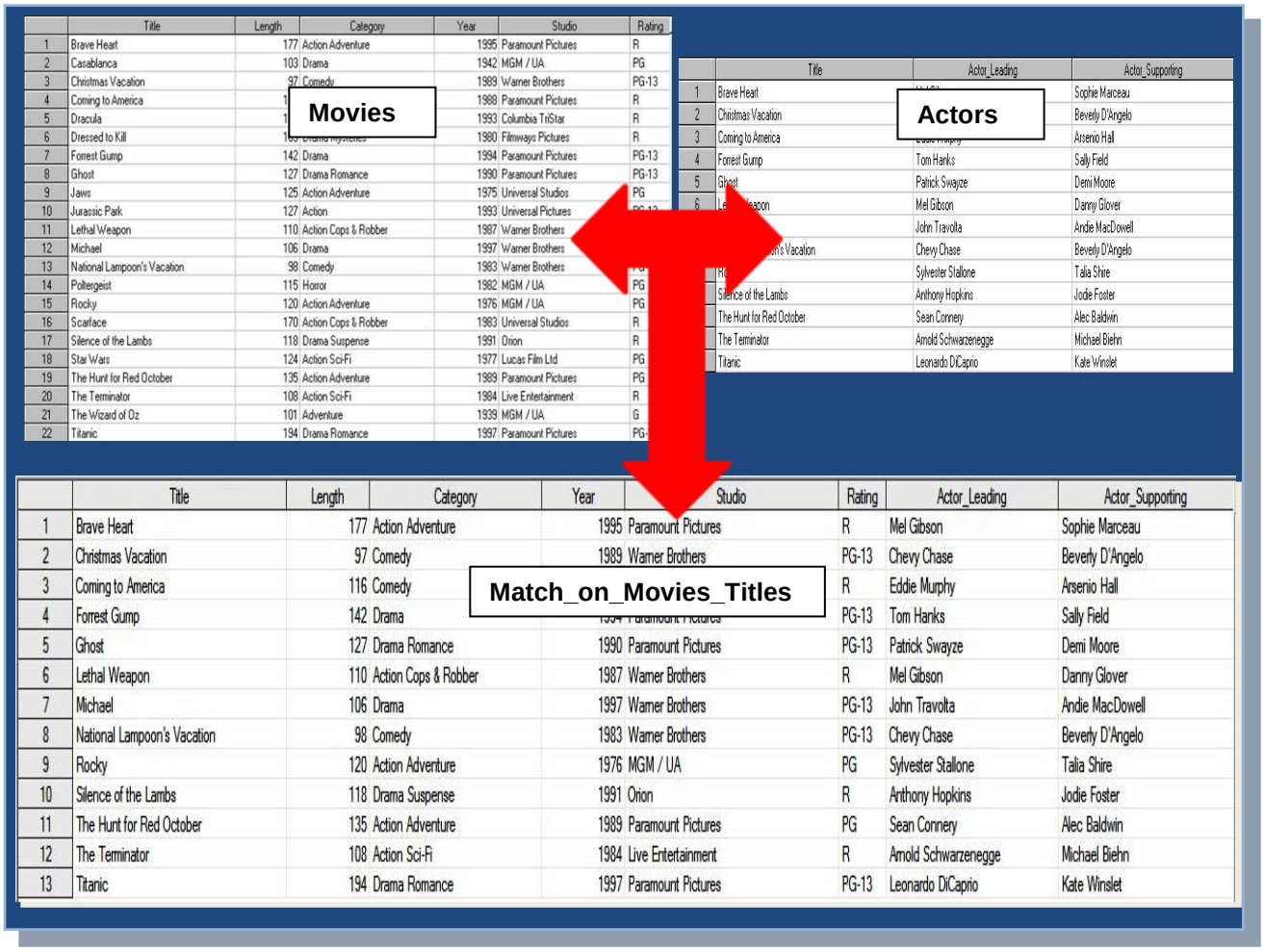
Search and Lookup with a Simple Key

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.

```
data match_on_movie_titles(drop=rc);

❶ if 0 then set mydata.movies
               mydata.actors;  /* load variable properties into hash tables */

  if _n_ = 1 then do;
❷   declare Hash MatchTitles (dataset:'mydata.actors'); /* declare the name
                                                          MatchTitles for hash */

❸   MatchTitles.DefineKey ('Title'); /* identify variable to use as key */
    MatchTitles.DefineData ('Actor_Leading',
                            'Actor_Supporting'); /* identify columns of data */
     MatchTitles.DefineDone (); /* complete hash table definition */
   end;

   set mydata.movies;

❹ if MatchTitles.find(key:title) = 0 then output;  /* lookup TITLE in MOVIES table
                                                      using MatchTitles hash */
run;
```

Results

The match-merge (or join) process is illustrated using the following diagram.



	Title	Length	Category	Year	Studio	Rating	Actor_Leading	Actor_Supporting
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R	Mel Gibson	Sophie Marceau
2	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
3	Coming to America	116	Comedy			R	Eddie Murphy	Arsenio Hall
4	Forrest Gump	142	Drama			PG-13	Tom Hanks	Sally Field
5	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13	Patrick Swayze	Demi Moore
6	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R	Mel Gibson	Danny Glover
7	Michael	106	Drama	1997	Warner Brothers	PG-13	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
9	Rocky	120	Action Adventure	1976	MGM / UA	PG	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	118	Drama Suspense	1991	Orion	R	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	Sean Connery	Alec Baldwin
12	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R	Arnold Schwarzenegge	Michael Biehn
13	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13	Leonardo DiCaprio	Kate Winslet

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:

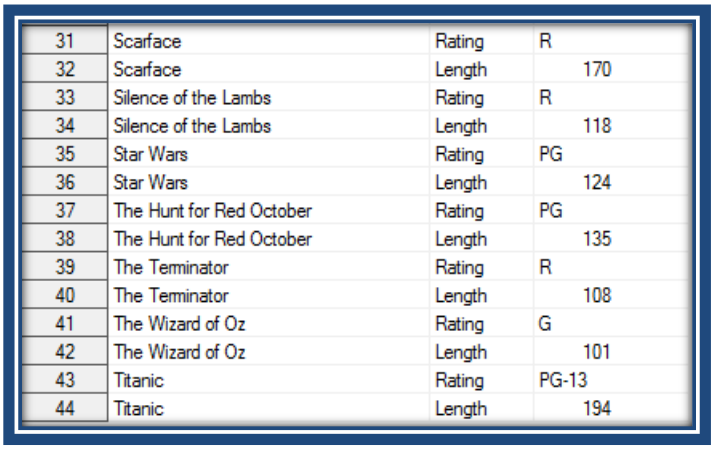
```
libname mydata "e:\workshops\workshop data" ;

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.

	Title	_NAME_	COL1
1	Brave Heart	Rating	R
2	Brave Heart	Length	177
3	Casablanca	Rating	PG
4	Casablanca	Length	103
5	Christmas Vacation	Rating	PG-13
6	Christmas Vacation	Length	97
7	Coming to America	Rating	R
8	Coming to America	Length	116
9	Dracula	Rating	R
10	Dracula	Length	130
11	Dressed to Kill	Rating	R
12	Dressed to Kill	Length	105
13	Forrest Gump	Rating	PG-13
14	Forrest Gump	Length	142
15	Ghost	Rating	PG-13
16	Ghost	Length	127
17	Jaws	Rating	PG
18	Jaws	Length	125
19	Jurassic Park	Rating	PG-13
20	Jurassic Park	Length	127
21	Lethal Weapon	Rating	R
22	Lethal Weapon	Length	110
23	Michael	Rating	PG-13
24	Michael	Length	106
25	National Lampoon's Vacation	Rating	PG-13
26	National Lampoon's Vacation	Length	98
27	Poltergeist	Rating	PG
28	Poltergeist	Length	115
29	Rocky	Rating	PG
30	Rocky	Length	120

Transposed_Movies Data Set created with PROC TRANSPOSE

Transposed_Movies Data Set (continued)



31	Scarface	Rating	R
32	Scarface	Length	170
33	Silence of the Lambs	Rating	R
34	Silence of the Lambs	Length	118
35	Star Wars	Rating	PG
36	Star Wars	Length	124
37	The Hunt for Red October	Rating	PG
38	The Hunt for Red October	Length	135
39	The Terminator	Rating	R
40	The Terminator	Length	108
41	The Wizard of Oz	Rating	G
42	The Wizard of Oz	Length	101
43	Titanic	Rating	PG-13
44	Titanic	Length	194

Transposed_Movies Data Set created with PROC TRANSPOSE (continued)

Transposing with the DATA Step Hash Method

My objective for using Hash methods in creating a restructured transposed data set is to emulate was 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. ❶ The next statement, "IF 0 THEN SET MYDATA.MOVIES" tells SAS t[illegible]ad variable properties into the hash object located in real me[illegible]ry. ❷ The DECLARE HASH statement provide 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. ❸ The "DECLARE HITER" statement defines and initializes the hash object for traversing the object in memory. ❹ The DEFINEKEY method identifies the variable (or variables) to use as the key. ❺ 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). ❻ The DEFINEDONE method completes the hash table definition. ❼ The FIRST() method tells SAS to return the first value stored in the defined hash object. ❽ The DO WHILE loop iterates repeatedly as long as there is data stored in the hash object. ❾ The LINK OUTLONG statement tells SAS to execute the OUTLONG subroutine. ❿ The NEXT() method tells SAS to return the next value from the defined hash object. ⓫ The STOP statement tells SAS to terminate the DATA step. ⓬

```
libname mydata 'e:\workshops\workshop data' ;
❶ data hash_long_movies (drop=rc Rating Length) ;
❷  if 0 then set mydata.movies(keep=Title Rating Length) ;
   if _n_ = 1 then do ;
❸     declare Hash Hash_movies(dataset:'mydata.movies',
                               ordered:'ascending') ;
❹     declare Hiter Hi_movies ('Hash_movies') ;
❺     Hash_movies.DefineKey ('Title') ;
❻     Hash_movies.DefineData ('Title', 'Rating', 'Length') ;
❼     Hash_movies.DefineDone () ;
   end ;
❽  rc = Hi_movies.first() ;
❾  do while (rc = 0) ;
❿     link outlong ;
⓫      rc = Hi_movies.next() ;
   end;
⓬  stop ;
return ;
```

```
⑩outlong: ;
     Title ;
     Label = 'Rating' ;
     Value = Rating ;
     output hash_long_movies ;

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

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 LABLE value for "Rating" in the first observation of VALUE and for "Length" in the second observation of VALUE for each BY-group.

	Title	Label	Value
1	Brave Heart	Rating	R
2	Brave Heart	Length	177
3	Casablanca	Rating	PG
4	Casablanca	Length	103
5	Christmas Vacation	Rating	PG-13
6	Christmas Vacation	Length	97
7	Coming to America	Rating	R
8	Coming to America	Length	116
9	Dracula	Rating	R
10	Dracula	Length	130
11	Dressed to Kill	Rating	R
12	Dressed to Kill	Length	105
13	Forrest Gump	Rating	PG-13
14	Forrest Gump	Length	142
15	Ghost	Rating	PG-13
16	Ghost	Length	127
17	Jaws	Rating	PG
18	Jaws	Length	125
19	Jurassic Park	Rating	PG-13
20	Jurassic Park	Length	127
21	Lethal Weapon	Rating	R
22	Lethal Weapon	Length	110
23	Michael	Rating	PG-13
24	Michael	Length	106
25	National Lampoon's Vacation	Rating	PG-13
26	National Lampoon's Vacation	Length	98
27	Poltergeist	Rating	PG
28	Poltergeist	Length	115
29	Rocky	Rating	PG
30	Rocky	Length	120

Hash_Long_Movies Data Set created with Hash Methods

Hash_Long_Movies Data Set (continued)

31	Scarface	Rating	R
32	Scarface	Length	170
33	Silence of the Lambs	Rating	R
34	Silence of the Lambs	Length	118
35	Star Wars	Rating	PG
36	Star Wars	Length	124
37	The Hunt for Red October	Rating	PG
38	The Hunt for Red October	Length	135
39	The Terminator	Rating	R
40	The Terminator	Length	108
41	The Wizard of Oz	Rating	G
42	The Wizard of Oz	Length	101
43	Titanic	Rating	PG-13
44	Titanic	Length	194

Hash_Long_Movies Data Set created with Hash Methods (continued)

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, and transposes.

References

Dorfman, Paul, and Marina Fridman (2010). "Black Belt Hashigana," Proceedings of the 2010 North East SAS Users Group (SESUG) Conference.

Dorfman, Paul and Peter Eberhardt (2010). "*Two Guys on Hash*," Proceedings of the 2010 South East SAS Users Group (SESUG) Conference.

Dorfman, Paul (2009). "*The SAS® Hash Object in Action*," Proceedings of the 2009 South East SAS Users Group (SESUG) Conference.

Dorfman, Paul, Lessia S. Shajenko and Koen Vyverman (2008). "*Hash Crash and Beyond*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Dorfman, Paul, and Koen Vyverman (2006). "*DATA Step Hash Objects as Programming Tools*," Proceedings of the Thirty-First SAS Users Group International Conference.

Eberhardt, Peter (2011). *"The SAS® Hash Object: It's Time to .find() Your Way Around,"* Proceedings of the 2011 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2016). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2016 South East SAS Users Group (SESUG) Conference.

Lafler, Kirk Paul (2016). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2016 Iowa SAS Users Group (IASUG) Conference.

Lafler, Kirk Paul (2015). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2015 South Central SAS Users Group (SCSUG) Conference.

Lafler, Kirk Paul (2011). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2011 South East SAS Users Group (SESUG) Conference.

Lafler, Kirk Paul (2011). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2011 PharmaSUG Conference.

Lafler, Kirk Paul (2011). "An Introduction *to SAS® Hash Programming Techniques*," San Diego SAS Users Group (SANDS) Meeting, February 16th, 2011.

Lafler, Kirk Paul (2010). "An Introduction *to SAS® Hash Programming Techniques*," Bay Area SAS (BASAS) Users Group Meeting, December 7th, 2010.

Lafler, Kirk Paul (2010). "An Introduction *to SAS® Hash Programming Techniques*," Proceedings of the 2010 South Central SAS Users Group (SCSUG) Conference.

Lafler, Kirk Paul (2010). "An Introduction *to SAS® Hash Programming Techniques*," Awarded "Best" Contributed Paper, Proceedings of the 2010 Western Users of SAS Software (WUSS) Conference.

Lafler, Kirk Paul (2010). *"DATA Step and PROC SQL Programming Techniques,"* Ohio SAS Users Group (OSUG) One-Day Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2010). *"Exploring Powerful Features in PROC SQL,"* SAS Global Forum (SGF) Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009). *"DATA Step and PROC SQL Programming Techniques,"* South Central SAS Users Group (SCSUG) 2009 Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Lafler, Kirk Paul (2009). *"DATA Step versus PROC SQL Programming Techniques,"* Sacramento Valley SAS Users Group 2009 Meeting, Software Intelligence Corporation, Spring Valley, CA, USA.

Loren, Judy and Richard A. DeVenezia (2011). "*Building Provider Panels: An Application for the Hash of Hashes,*" Proceedings of the 2011 SAS Global Forum (SGF) Conference.

Loren, Judy (2006). "How *Do I Love Hash Tables? Let Me Count The Ways!,*" Proceedings of the Nineteenth Northeast SAS Users Group Conference.

Miller, Ethan and Kirk Paul Lafler (2012), *"SAS® on a Shingle, Flippin with Hash,"* Proceedings of the 2012 Western Users of SAS Software (WUSS) Conference Proceedings, SRI International, Menlo Park, CA, and Software Intelligence Corporation, Spring Valley, CA, USA.

Muriel, Elena (2007). *"Hashing Performance Time with Hash Tables,"* Proceedings of the 2007 SAS Global Forum (SGF) Conference.

Parman, Bill (2006). "*How to Implement the SAS® DATA Step Hash Object*," Proceedings of the 2006 Southeast SAS Users Group Conference.

Ray, Robert and Jason Secosky (2008). "*Better Hashing in SAS® 9.2*," Proceedings of the Second Annual SAS Global Forum (SGF) Conference, SAS Institute Inc., Cary, NC, USA.

Secosky, Jason (2007). "*Getting Started with the DATA Step Hash Object*," Proceedings of the 2007 SAS Global Forum (SGF) Conference, SAS Institute Inc., Cary, NC, USA.

Acknowledgments

The author thanks Dave Foster and Chuck Kincaid, Hands-On Workshops Section Chairs, for accepting my abstract and paper; Richann Watson, MWSUG 2016 Academic Chair; Adrian Katschke, MWSUG 2016 Operations Chair; the MidWest SAS Users Group (MWSUG) Executive Board; and SAS Institute for organizing and supporting a great conference!

Trademark Citations

About The Author

Kirk Paul Lafler is an entrepreneur, consultant and founder of Software Intelligence Corporation, and has been using SAS since 1979. Kirk is a SAS Certified Professional, provider of IT consulting services, advisor and professor at UC San Diego Extension and educator to SAS users around the world, mentor, and emeritus sasCommunity.org Advisory Board member. As the author of six books including Google® Search Complete (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written hundreds of papers and articles; been an Invited speaker and trainer at hundreds of SAS International, regional, special-interest, local, and in-house user group conferences and meetings; and is the recipient of 23 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

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