SAS® Hash Object Programming *Made Easy*

Michele M. Burlew

SAS® Press

Contents

About This Book vii

About The Author xi

Acknowledgments xiii

Preface xv

Chapter 1: An Overview of Hash Objects 1

What Are Hash Objects? 2

Introducing a Simple Hash Object Application 2

Chapter 2: Hash Object Terminology and Concepts 7

What Is a SAS Hash Object? 7

Defining Terms Associated with Hash Objects 8

Writing Code That Works with Hash Objects 9

Understanding How SAS Stores Hash Objects in Memory 10

Understanding How Long Hash Objects Persist 11

Specifying the Contents of Hash Objects 11

Initializing Variables in a DATA Step That Contains a Hash Object 14

Illustrating How the Program Data Vector Connects DATA Step Variables and Hash Object Items 15

Chapter 3: Basic Hash Object Applications 25

Using a Hash Object As a Lookup Table 26

Defining a Hash Object 26

Finding Key Values in a Hash Object 28

Defining the Key Structure in a Hash Object 30

Understanding How the FIND and CHECK Methods Alter the Values of DATA Step Variables and Hash Object Data Items 30

Application Example: Verifying Presence of Key Values 33

Application Example: Returning Data Items from a Hash Object 36
Application Example: Defining the Key Value During Processing of the DATA Step 39
Application Example: Searching for a Key Value in Multiple Hash Objects 41
Application Example: Combining Data from Multiple Sources 44
Using Multiple Key Items to Look Up Data 49
Traversing Hash Objects 53
Specifying a Hash Iterator Object 53
Understanding the Methods That Control Traversal of a Hash Object 54
Illustrating How the Hash Iterator Object Traverses a Hash Object 55
Application Example: Traversing a Hash Object from Beginning to End 59
Application Example: Linking Hierarchically Related Data Sets 62
Application Example: Performing a Fuzzy Merge Using a Hash Iterator Object 66

Chapter 4: Creating Data Sets from Hash Objects and Updating Contents of Hash Objects 71

Creating a Data Set from a Hash Object 72
Adding, Modifying, and Removing Data from a Hash Object 73
Defining the Key Structure in a Hash Object 74
Understanding How to Specify the KEY and DATA Argument Tags 75
Identifying the Variables That the OUTPUT Method Writes to a Data Set 77
Understanding the Interaction between DATA Step Variables and Hash Object Data Items When Replacing Data in a Hash Object 78
Replacing Key Item Values When a Key Item Is Also a Data Item 80
Comparing the DATA Statement and the OUTPUT Method 83
Application Example: Finding the Unique Values of a Variable in a Data Set 87
Application Example: Ordering Observations by Variables Not Saved to a Data Set Created from a Hash Object 90
Application Example: Using Hash Objects to Apply Transactions to Master Data Sets—Part 1 92
Application Example: Using Hash Objects to Apply Transactions to Master Data Sets—Part 2 97
Application Example: Summarizing Data with the Hash Iterator Object 102
Application Example: Summarizing Hierarchically Related Data 106

Chapter 5: Hash Objects with Multiple Sets of Data Items per Key Value 113

Understanding the Concepts of Duplicate Key Values and Multiple Sets of Data Items per Key Value in a Hash Object 114

Defining Hash Objects That Process Multiple Sets of Data Items per Key Value 115

Illustrating How the MULTIDATA and DUPLICATE Argument Tags Affect Hash Object Processing 117

Understanding the Methods That Look for Multiple Sets of Data Items per Key 121

Understanding How SAS Stores Multiple Sets of Data Items per Key 123

Comparing Retrieval of Data from a Hash Object That Allows Multiple Sets of Data Items per Key Value to a Hash Iterator Object 123

Modifying Data in a Hash Object That Allows Multiple Sets of Data Items per Key Value 129

Summarizing Data in Hash Objects That Allow Multiple Sets of Data Items per Key Value 143

Application Example: Summarizing and Sorting a Data Set 144

Application Example: Creating Data Sets Based on a Series of Observations 147

Application Example: Creating a Data Set That Contains All Combinations of Specific Variables When the Number of Combinations Is Large 151

Application Example: Linking Hierarchically Related Data Using a Hash Object That Allows Multiple Sets of Data Items per Key Value 154

Chapter 6: Managing Hash Objects 159

Creating, Deleting, and Clearing Hash Objects During Execution of a DATA Step 159

Determining the Number of Items in a Hash Object 164

Application Example: Creating a Data Set for Each BY Group 166

Comparing Two Hash Objects 170

Specifying Memory Structure Usage of a Hash Object 174

Determining the Size of an Entry in a Hash Object 174

Index 181

Chapter 1: An Overview of Hash Objects

What Are Hash Objects?.....2
Introducing a Simple Hash Object Application 2

Programming hash objects in SAS sounds like a geeky process. A quick review of the statements associated with hash objects shows that the statements look geeky too. Although the items that are associated with hash object programming seem to resemble statements or functions, they are not called statements or functions. Instead they are called *methods* and they're written in something called *dot notation*!

If you are reading this book, you probably already know how to write a DATA step. And you may have picked up this book to figure out if it's worth your time to learn a new way of processing data in the DATA step. You understand how SAS compiles and executes a DATA step. You know how to apply procedures like SORT, SQL, and FORMAT to structure your data the way you need to and to look up information efficiently.

If you are already a skilled SAS programmer and you know how to use SAS procedures effectively; why should you learn how to use hash objects? The answer is that they are an amazingly efficient tool for looking up information and joining information from data sets and tables.

Because you already know how SAS processes data, you can quickly learn how to write code that includes hash objects. The methods that define and work with hash objects are used in the DATA step in ways that are familiar to you.

This book shows you how to make hash objects fit into what you already know, and it often compares the hash solution to a solution that uses SAS language and procedures, one that you're likely already familiar with applying. This book introduces you to hash object programming and connects it to concepts you already understand.

If you write lots of SAS programs and use SAS procedures to structure and look up data, then it is worth it to take the time to learn how to use hash objects. Once you become familiar with the syntax, you will find many places where you can use a hash object. You are likely to save programming time and computer processing time when you incorporate hash object programming

in your SAS programming toolkit. If you've already used hash objects some, then this book offers you some examples for techniques that you may not have tried or understood how to use.

The examples in this book vary in complexity and are generally organized in order of increasing complexity. By studying how an example uses a hash object, you will find ways to adapt hash objects programming techniques to your own applications.

What Are Hash Objects?

Technically, hash objects, interchangeably called hash tables in this book, are data structures that provide a way to efficiently search data. Hash object programming is a feature in many programming languages. A hash object is a type of array that a program accesses using keys. A hash object consists of key items and data items. The programming language applies a hash function that maps the keys to positions in the array.

When you define an array in SAS, SAS allocates a fixed amount of memory based on the number of elements you specify and the attributes of the variables that make up the array. A hash object is a more dynamic structure. It grows and expands as you add and remove items from the table following the instructions in your programs.

A SAS hash object exists only within the DATA step in which it creates the hash object. When the DATA step ends, SAS deletes the hash object.

Introducing a Simple Hash Object Application

In general, SAS language in a SAS DATA step follows a linear, logical process. A typical DATA step processes a SAS data set one observation at a time or it reads a raw data file one record at a time. SAS executes the statements one after the other. While your statements can direct processing to different locations within a DATA step, a DATA step typically starts at the top and proceeds sequentially statement-by-statement to the bottom of the DATA step.

A common SAS programming task is to look up information based on data in the observation currently being processed. Many methods exist in SAS to look up information. Some of these include:

- IF-THEN and SELECT statements
- referencing elements of arrays
- applying formats
- merging data sets
- joining tables

Some of these methods require more than just a single DATA step or PROC step to achieve the lookup.

For example, as shown in the first table in Output 1.1, say you have data set EMPHOURS with hours worked for a group of employees where each employee is identified in each observation solely by employee ID. To complete the employee report, you need the employee's full name. This data is stored in another data set, EMPLOYEES, as shown in the second table in Output 1.1 that contains both the employee ID and demographic information about the employee.

Output 1.1 PROC PRINT of EMPHOURS (all 6 observations) and EMPLOYEES (first 20 observations)

EMPHOURS		
Obs	**empid**	**emphours**
1	6XBIFI	38.5
2	WA4D7N	22.0
3	VPA9EF	43.0
4	TZ6OUB	11.5
5	L6KKHS	29.0
6	8TN7WL	38.0

EMPLOYEES							
Obs	**empid**	**empln**	**empfn**	**empmi**	**gender**	**startdate**	**emppaylevel**
1	6XBIFI	Ramirez	Danielle	N	F	04/21/1989	AIb
2	AWIUME	Thompson	Catherine	D	F	06/18/1986	PIIIa
3	06KH8Q	Chang	William	T	M	07/23/2002	PIIa
4	WA4D7N	Garcia	Breanna	X	F	08/20/1982	AIb
5	OOQT3Z	Jones	Brooke	E	F	08/28/1994	MIIa
6	1JU28B	Smith	Matthew	I	M	08/22/1982	TIIIb
7	V8OARE	Hall	Samuel	B	M	05/25/2010	PIb
8	1GTXQ2	Parker	Nathaniel	S	M	08/12/1996	PIc
9	VPA9EF	Baker	Cheyenne	C	F	02/24/1990	AIIa
10	0IP7L6	Hughes	Alexander	N	M	08/08/1991	TIIb
11	Q1A4SU	Sanchez	Nathaniel	W	M	08/13/1998	TIId
12	ANWFGX	Green	Tyler	I	M	12/04/1991	TIc
13	L1I8Y7	Edwards	Angelica	O	F	11/18/1991	MIIIa
14	TZ6OUB	White	Heather	T	F	01/27/1999	AIIIa
15	235TWE	King	Briana	M	F	10/08/1992	TIIc
16	XYOJC7	Scott	Mark	T	M	01/15/2002	TIIa
17	8TN7WL	Miller	Tyler	J	M	08/31/1998	AIIIc
18	US3DZP	Brown	Sarah	U	F	12/11/2000	TIb

EMPLOYEES							
Obs	**empid**	**empln**	**empfn**	**empmi**	**gender**	**startdate**	**emppaylevel**
19	ODBAIZ	Jones	Rachel	T	F	10/11/1999	PIIb
20	A4GJG4	Johnson	Angelica	Z	F	01/01/1994	AId

Typing a series of IF-THEN or SELECT statements to find an employee's name seems prohibitive, especially if your demographic lookup data set is large. Probably a merge in the DATA step or a PROC SQL join would work to obtain the names of the employees based on their employee IDs. If you do the merge, both your hours-worked data set and your demographic data set must be sorted or indexed by the employee ID prior to the DATA step that does the merge.

While PROC SQL does not require that you sort your tables before you have PROC SQL join them, the step processes more efficiently if your tables are sorted or indexed by the columns that do the join. However, when you use PROC SQL, if you need to do other processing of the table, like calculations, it can be more difficult to code those statements in SQL than if you wrote SAS language statements in a DATA step. If you add a second SELECT statement to your PROC SQL step or if you follow the PROC SQL step with a DATA step, you've now processed your data set twice.

Example 1.1 presents sample code that shows a DATA step merge solution.

Example 1.1 DATA Step Merge

```
proc sort data=emphours;
  by empid;
run;
proc sort data=mylib.employees;
  by empid;
run;
data empnames;
  merge emphours(in=inhours)
        mylib.employees(keep=empid empln empfn empmi in=inall);
    by empid;
  length empname $ 60;
  if inhours;
  if inall then empname=catx(' ',empfn,empmi,empln);
  else empname='** Not Found';
run;
```

With hash object programming, you can achieve both the lookup of employee demographic data and additional SAS language processing in one DATA step. Your DATA step starts by loading the demographic data set into a hash object.

To further streamline the lookup process, you can tell SAS to load only certain variables and observations into the hash object. For this simple example, you load the names of the employees in the hash object, and you tell SAS that the key to find data in the hash object is the employee ID.

Perhaps you know that your employee data set has observations for employees that have a specific job classification. You could tell SAS to load the hash object with the names of employees only from that subset of the employee population.

Since hash objects reside in memory, it's highly likely that the DATA step with the hash object runs more quickly than any of the solutions in the list above, and this is especially true if your data sets are large.

Example 1.2 presents a DATA step that uses a hash object to find employee names for administrative employees. The values of EMPPAYLEVEL for administrative employees start with "A". While the language in this step may look strange to you, it is really easy to understand and code once you have a little knowledge.

Example 1.2 DATA Step That Uses a Hash Object

```
data empnames;
  length empid $ 6 empln $ 30 empfn $ 25 empmi $ 1 empname $ 60;
  if _n_=1 then do;
   declare hash e(dataset: 'mylib.employees(where=(emppaylevel=:"A")');❶ ❷
   e.definekey('empid'); ❸
   e.definedata('empln','empfn','empmi'); ❹
   e.definedone();
   call missing(empln,empfn,empmi); ❺
  end;
  set emphours; ❻
  drop rc;
  rc=e.find();❼
  if rc=0 then empname=catx(' ',empfn,empmi,empln); ❽
  else empname='** Not Found'; ❾
run;
```

The step starts with an IF-THEN block that executes only on the first iteration. This code defines hash object E and loads it with demographic information from data set EMPLOYEES. Having been defined on the first iteration of the DATA step, hash object E remains in memory throughout execution of the DATA step so that you can quickly retrieve from it the demographic data you need for each observation in the EMPHOURS data set.

The statements that follow the IF-THEN block execute once for every observation in EMPHOURS. The statement with the FIND method retrieves information from hash object E based on the values of EMPID. The IF-THEN statement that follows it executes when SAS finds a match in hash object E. The ELSE statement executes when SAS does not find a match in hash object E.

Specifically, the statements in the IF _N_=1 block that executes on the first iteration of the DATA step do the following:

❶ Create hash object called E.
❷ Load into hash object E the observations in MYLIB.EMPLOYEES for administrative employees (EMPPAYLEVEL=:"A").
❸ Identify variable EMPID as the key to find data in hash object E.
❹ Load only variables EMPLN, EMPFN, and EMPMI from MYLIB.EMPLOYEES into hash object E as data.
❺ Initialize to missing the DATA step variables with the same names as the items that SAS loads into hash object E.

The statements outside of the IF _N_=1 block that execute on each iteration of the DATA step do the following:

❻ Read each observation from data set EMPHOURS.
❼ Look for a match in hash object E for the value of EMPID in the EMPHOUR observation currently being processed.
❽ When SAS finds a match in hash object E, concatenate the information retrieved from hash object E into variable EMPNAME.
❾ When SAS does not find a match in hash object E, assign informative text to the variable EMPNAME.

The LENGTH statement at the beginning of the DATA step adds to the Program Data Vector (PDV) the three variables whose values are retrieved from hash object E. SAS outputs these three variables, the employee ID (EMPID), and the new variable EMPNAME to data set EMPNAMES.

Example 1.2, which is the DATA step with the hash object, executes the same way as a DATA step without a hash object. You could think of the block of code that defined hash object E as similar to how you might define an array that provides lookup values. Yet, hash object programming provides much more functionality than arrays. In this simple DATA step in Example 1.2, you can see that in one statement the DATA step looks for a match in hash object E and it retrieves the values of three data items when SAS finds a match. If you used arrays, your code would likely require multiple arrays, or possibly multi-dimensional arrays.

About The Author

Michele M. Burlew, president of Episystems, Inc., designs and programs SAS applications for data management, data analysis, report writing, and graphics for academic and corporate clients. A SAS user since 1980, she has expertise in many SAS products and operating systems. Burlew is the author of seven SAS Press books: *SAS Hash Object Programming Made Easy; Combining and Modifying SAS Data Sets: Examples, Second Edition; Output Delivery System: The Basics and Beyond (coauthor); SAS Guide to Report Writing: Examples, Second Edition, SAS Macro Programming Made Easy, Second Edition; Debugging SAS Programs: A Handbook of Tools and Techniques;* and *Reading External Data Files Using SAS: Examples Handbook*.

Learn more about the author by visiting her author page at support.sas.com/burlew. There you can download free chapters, access example code and data, read the latest reviews, get updates, and more.

MICHELE BURLEW MAKES SAS® PROGRAMMING EASY

Order your copy of SAS® Hash Object Programming Made Easy today!

Learn more about Michele Burlew, read free chapters from her books, and access example code and data on her author page.

Browse our full catalog to find additional books that are just right for you.

Subscribe to our monthly e-newsletter to get the latest on new books, documentation, and tips—delivered to you.

Browse and search free SAS documentation sorted by release and by product.

Email us: sasbook@sas.com
Call: 800-727-3228

sas®
THE POWER TO KNOW®