Paper CT01

Hash-ing It Out: Bringing SAS® to the Next Level

Dr. Lara Suckut, Chrestos GmbH, Essen, Germany
Veronika Schmidt, Chrestos GmbH, Essen, Germany

ABSTRACT

As data structures and utilization become larger and more complicated every day, efficient, time-saving solutions are needed to extract targeted information from multiple, possibly differently structured data sets.
SAS® hash tables provide quick and flexible ways to merge, sort, and look up data by taking information directly from the source instead of generating unnecessary data duplication that slows down processing and clutters up storage space. Additionally, using hash tables can be beneficial by saving time and reducing data steps during programming. We provide a thorough introduction to the basic definitions, concepts, and benefits of hash objects. By using examples that are practical and applicable to the work of a statistical programmer in the pharmaceutical industry, we aim to provide the audience with a firm grasp of the opportunities offered by this method and ideas on how to use hash tables to solve their current data challenges.

INTRODUCTION

Saving space and increasing the speed of analyses is a common goal of programmers for all kinds of applications. In the pharma world specifically, data sets get larger and analyses more complicated every day, e.g. when biomarkers are involved. Therefore, the need for efficient and time-saving solutions is a relevant topic, especially when working with large or complex data sets and differing data structures that need to be combined to gather all necessary information.
SAS offers hash tables as a possible solution to be able to quickly merge, sort, and look up data from other tables (SAS Institute Inc., 2023). Instead of using up disk space, hash objects provide a memory-based approach. While it is possible to use other methods like a data step merge, hash objects add the advantage of not having to sort data sets before merging while being run-time efficient. As demonstrated by Oreshko (2019), in a setting with 1,000,000 observations, using hash tables can be up to 30 times quicker than the standard method. Additionally, it can lead to simpler code that is shorter and easier to read when compared to other methods.
This paper gives an introduction to hash tables and how they can be applied in real-life scenarios. A basic understanding of its structure, its usage, and when and why they are helpful is provided.

WHAT IS A HASH OBJECT?

A Hash object is a data step construct that behaves similarly to a SAS array (Lafle, 2011). It connects multiple data sets using a key value. As a temporary object that is memory-based, dynamically sized, and deleted at the end of a data step, it can be used to quickly match and merge information from the respective data sets. Because they are only stored in memory, hash tables don't take up disk space, minimizing disk I/O.
As opposed to merging data in data steps, using hash tables does not require the data sets to be sorted. Hash objects use binary search trees to match the key variables, which is one reason why they work so fast.

The data set defined in the hash object is commonly referred to as the hash table. Table 1 is an example of what a hash table can look like. It consists of several key variables and data variables. Key variables are those variables that are used to match and identify the observations while the data variables contain the values that are of interest for the operation the hash table will be used for.

Keyvar1	Keyvar2	…	Datavar1	Datavar2	…
Key 1	Key A	…	Value 1	Value A	…
Key 1	Key B	…	Value 2	Value B	…
Key 1	Key C	…	Value 3	Value C	…
Key 2	Key B	…	Value 4	Value D	…
Key 2	Key D	…	Value 5	Value E	…
Key 3	Key A	…	Value 6	Value F	…
Key 3	Key B	…	Value 7	Value G	…
Key 4	Key A	…	Value 8	Value H	…
…	…	…	…	…	…

Table 1: Hash Table Example

HOW CAN HASH OBJECTS BE USED?

The following part of this paper describes how to create a hash object and how to use it in a SAS data step. Realistic examples based on real-life problems of statistical programmers show the advantages of using hash in everyday programming situations.

BASIC STRUCTURE

A hash object is set up within a data step and can only be used within said data step. Code 1 provides a basic example of how a hash object can be created and used.

```
1  data _NULL_;
2   if 0 then set hashdata;
3   if _N_ eq 1 then do;
4    declare hash h (dataset: 'hashdata (where = (datavar1 > 10))');
5    h.definekey('keyvar1','keyvar2');
6    h.definedata('datavar1','datavar2');
7    h.definedone();
8   end;
9  run;
```

Code 1: Basic structure

Note the data step construct can be saved, but does not need to be, e.g. in case the goal of the data step is to manipulate and output the hash table. As mentioned above, the hash table itself will not be saved, as it is only created to read data into memory, and the hash object as well as its contents disappear at the end of the data step (Lafle, 2011).

Before defining the hash table, it is important to recognize that all variables that will be used in the hash object need to be declared up front. This can be achieved by either using `length` or `attrib` statements followed by a `call missing` for key and data variables or by tricking SAS into reading the dataset without observations, as demonstrated in **line 2** of the provided code. Otherwise, SAS will give an error during the initialization of the hash object because the variable defined, e.g. as a key variable, is unknown to the program data vector.

The hash object is initialized in **line 4 to 7** of the code. In **line 4**, the hash object is named and the data set loaded. It is possible to use data set options. Alternatively, the code can be written as "`dcl hash h (…);`".

The keys for the hash object are defined using the name of the hash object and adding the method "`.definekey(…)`" (see **line 5**). The key values can be a single variable or a combination of variables (composite key). For example, in SDTM creation, where sets of key variables are very clearly defined, hash tables are easily applicable.

Line 6 is optional: In case no associated data needs to be fetched, it can be left out. If data needs to be fetched, the associated values can be defined using the name of the hash object and adding the method "`.definedata(…)`".

The method "`.definedone()`" added to the hash object name ends the initialization of the hash object (**line 7**).

While constructing the hash object, it is important to make sure the hash object is created only once and not for every iteration of the data step. For example, the code can be put within an if-clause that will trigger only once (see **lines 3 and 8** of the provided code) or a stop statement can be used.

Any manipulation of the hash object as well as any fetching and checking of data is done after the initialization of the hash object. In this case, any additional lines of code can be added between **line 8 and 9** of the provided code. Examples of how to manipulate hash objects and what code to use to do so can be found in the following sections.

EXAMPLE 1: CHECK VALUES

The following example shows how to use a hash object to check whether data from one data set can be found in another data set.

As depicted in Figure 1, the set of one or more key variables of the original data set are matched to the respective variables in the data set that was loaded as the hash table. The key variables in the original data set uniquely identify one single line in the hash table. For each row in the original data set, a request to match the key variables is sent. Information on whether there was a successful match is returned and, in this example, saved in a variable in the original data set. A value of 0 represents a successful match and any other number an unsuccessful match. The data variables do not need to be included in the hash table definition. Internally, SAS uses the key variable as a data variable to keep the required structure (Dorfman and Henderson, 2019).

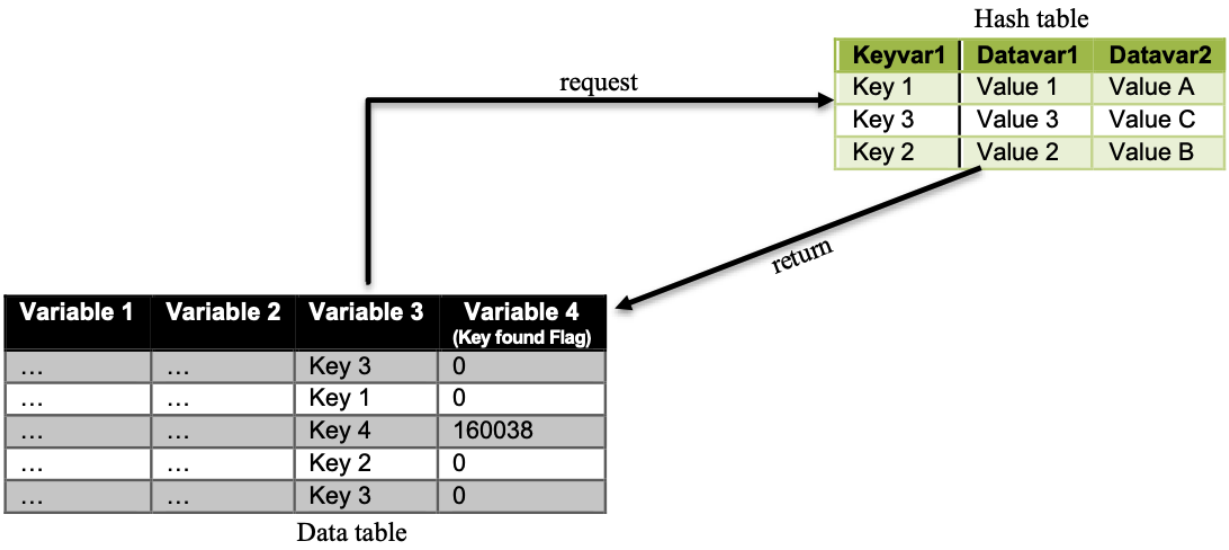

Figure 1: Check Values

For example, imagine a situation in SDTM creation where the names of laboratory tests need to be checked against CDISC SDTM controlled terminology. In this case, a data set containing the valid CDISC submission values (`sdtm_ct`) is provided.
One way to deal with this problem would be to sort both the SDTM table and the terminology table and then merge them together to determine whether the laboratory test name is in the provided file. Depending on the size of the table and the size of the provided file, this could take some time. In this example, the provided file is instead declared as a hash table.

```
1     data checked issues;
2           set tocheck;
3           if 0 then set sdtm_ct(keep=cdisc_submission_value);
4           if _N_ eq 1 then do;
5                 declare hash h(dataset:'sdtm_ct');
6                 h.definekey('cdisc_submission_value');
7                 h.definedone();
8           end;
9           rc = h.check(key:lbtest);
10          row_num = _N_;
11          if rc ne 0 then output issues;
12          output checked;
13    run;
```

Code 2: Check Values

The data set `tocheck`, containing the variable `lbtest`, is set and the relevant variables from the hash table are initialized in **lines 2 and 3** of Code 2. Since the only variable used from `sdtm_ct` is `cdisc_submission_value`, only this specific variable needs to be initialized.
In **lines 4 to 8**, the hash object `h` is declared, as previously described in Code 1. It includes all observations from `sdtm_ct` and `cdisc_submission_value` is defined as the key variable. No data variables are defined, as only the key value is needed to determine whether the `lbtest` values occur in the terminology table.
The hash table is then used to check whether the value of the key variable `lbtest` from `tocheck` can be found among the values of the key variable from the hash table `h` (**line 9**). The newly created variable `rc` indicates whether or not the key value could be found. A value of 0 indicates a successful find; any other values indicate no find.
Lines 10 and 11 are not strictly necessary, but helpful for the identification of issues later. A variable `row_num` is created containing the row number of the original data set. If the returned value of the check method is not 0, i.e. the observation seems to not have a matching key value in the hash table, the observation is written to an extra data set issue that can be looked at for an overview of all misspelled laboratory tests.
The complete original data set with the two additional variables `rc` and `row_num` is written into the `checked` data set (**line 12**).

The example shows how, using a hash table, a variable can be easily created that identifies observations with key values that are not present in the comparison data set. On top of that, an extra data set only containing those observations without a matching key is created with the original line number included for easier identification.

Note: The comparisons with the check methods are case sensitive. When only the letters and not their cases are relevant, the key variables from both data sets should be set to lower or upper case.

EXAMPLE 2: FETCH VALUES

The following example shows how data can be fetched from a hash table and be added to a data table.
As depicted in Figure 2, the set of one or more key variables of the original data set is matched to the respective key variables in the data set that was loaded as the hash table. The key variables in the original data set uniquely identify one single line in the hash table. For each row of the original data set, a request to match the key variables is sent and the respective values of the data variables from the hash table are selected. Information on whether there was a successful match can be saved as a variable in the data set with 0 representing a successful match and any other number an unsuccessful match. The data variable values are returned from the hash table and added in the original data set.

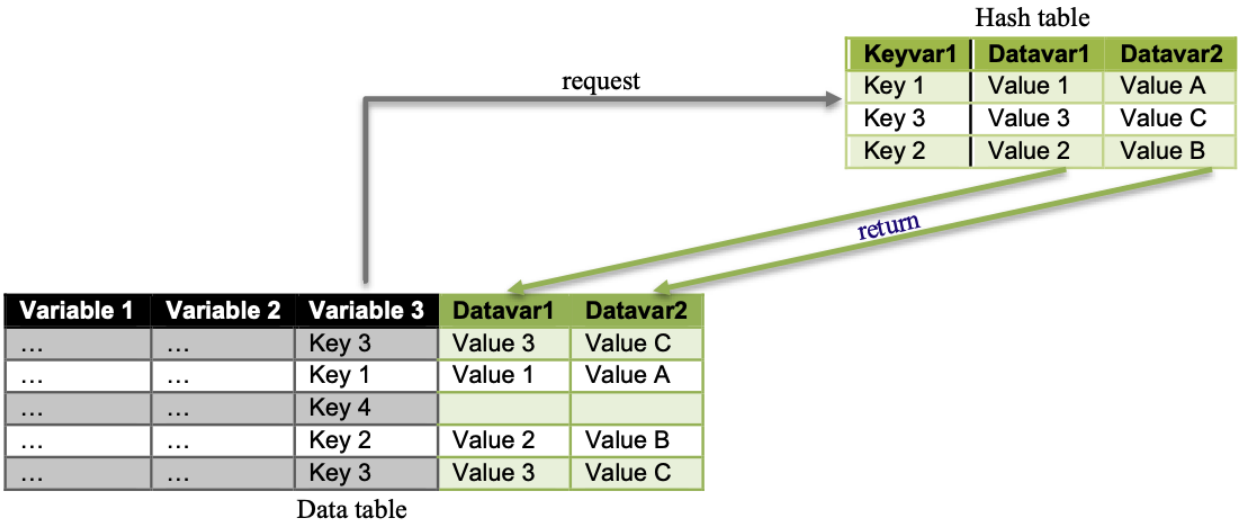

Figure 2: Fetch Values

As an example from statistical programming, imagine that an ADaM data set for concomitant medications (CM) is to be generated. An on-treatment flag needs to be created. On top of the SDTM domain CM (`indata.cm`) data set, some additional variables are needed from the subject-level data set ADSL (`indata.adsl`) to be able to identify when a subject was on treatment. The needed dates could, of course, be added by merging the two data sets together, but this would potentially require previous sorting and could take some time, depending on the size of the tables. Instead, a hash table can be used as a lookup table to fetch this information, which can then be used to identify concomitant medication taken during treatment.

```
1     data adcm;
2           set indata.cm(where=(length(cmstdtc) eq 10 and length(cmendtc) eq 10));
3           if 0 then set indata.adsl(keep=usubjid trtsdt trtedt);
4           if _N_ eq 1 then do;
5                 declare hash h(dataset:'indata.adsl(keep=usubjid trtsdt trtedt)');
6                 h.definekey('usubjid');
7                 h.definedata('trtsdt','trtedt');
8                 h.definedone();
9           end;
10          call missing(trtsdt,trtedt);
11          rc = h.find();
12          cmstd = input(cmstdtc,YYMMDD10.);
13          cmend = input(cmendtc,YYMMDD10.);
14          ontrtfn = ifn((cmstd ge trtsdt and cmstd le (trtedt + 14)) or
                          (cmend ge trtsdt and cmend le (trtedt + 14)), 1, 0);
15    run;
```

Code 3: Fetch Values

In **lines 2 and 3** of Code 3, the CM dataset `indata.cm` is set, and the data set `indata.adsl`, which will be the hash table, is initialized. Note that the data set is reduced to only the necessary variables to save memory.
In **lines 4 to 9**, the hash object `h` is declared, as previously described in Code 1. It includes all observations from `indata.adsl`, keeping only the `usubjid`, `trtsdt` and `trtedt variables`. The key is defined as `usubjid` in **line 6**. `trtsdt` and `trtedt` are defined as the data variables in **line 7**.
Line 10 is important to note, as it relates to a common mistake that is easily overlooked. In this line, the data variables are set to missing for each new row in the data set done by SAS. If this is not set to missing and a key is not found in the hash table, the variable used will not be overwritten and instead be carried over from the previous line. Therefore, leaving out this step might lead to wrong values used for the new data set.
After setting the data variables to missing for each observation, the find method is called to fetch the data variables from the hash table using the provided hash key and the data variables defined in the method call. While it is not strictly necessary to define the result as a variable, it is best practice, because it can help with debugging or to distinguish values where the key was not found and those where a key was found, but the data variable from the hash table has a missing value.
Once the `cmstd` and `cmend` variables are in the right format (**lines 12 and 13**), the fetched date variables are used to define an on-treatment flag for the concomitant medications (**line 14**).

This example demonstrates how easily hash tables can be used to fetch variables and use them in calculations. The data used in this example can be found in the PHUSE scripts repository (https://github.com/phuse-org/phuse-scripts/tree/master/data/sdtm/updated_cdiscpilot).

Note: The comparisons with the find methods are case sensitive, just as for the check method from Example 1. When only the letters and not their cases are relevant, the key variables from both data sets should be set to lower or upper case.

WHAT SHOULD BE KEPT IN MIND WHEN USING HASH TABLES?

This section is a short summary of things to keep in mind that were either already mentioned in previous sections or that may also be helpful.

- For de-bugging or even for further calculations, it's helpful to define the result of the check or find method as a variable. This variable contains a numerical value that is 0 if there is a match, and a positive integer if no match is found.
- The check and find methods are case sensitive and this needs to be accounted for: for example, by setting all key variables to upper or lower case in both the hash table and the data set before using the methods.
- Only the data variables from the hash object are written to the data set, not the key variables. If the key variables are to be included in the data set, they need to be specified as key and data variables.
- The key variables should lead to a unique match. If duplicate keys are given, it is advisable to consciously set options for how to handle them. As a default, SAS uses the first match and discards the rest.
- If no matching key is found the data variable is not automatically set to missing and instead takes over the value from the previous observation. To avoid using wrong values, the variables should be manually set to missing, e.g. by using call missing or by checking the result of the find or check method in an if-clause before calculating.
- All data variables used from the hash table (and all variables from the data set, if one is loaded) need to have distinct names from the data set. If there is an overlap, the variables need to be renamed either in a previous data step or when defining the hash object.
- It is possible to adapt the hash search based on data set size, number of searches as well as time or memory constraints. For example, setting an appropriate `hashexp` value leads to a better performance. The higher the value, the more trees are used and the faster the search, but this also requires more memory. It might be necessary to try a few different values to find the sweet spot of not using too much memory but still have a good performance.
- To save memory, it is advisable to use the smaller data set as a hash table when merging.
- While defining the hash object, it's possible to use data set options like restricting the hash table to only key and data variables or subsetting it. The latter can be specifically useful for sub-analyses.
- While using hash tables can lead to a significant reduction of run-time in certain cases (see Introduction), Oreshko (2019) also provides an example for a comparison to using SQL joins. The speed increase of up to 30 times when using hash tables instead of a data step merge with 1,000,000 observations is much higher than the provided SQL example, where is is "only" 17 times faster. It is important to keep in mind that the

benefit of using hash tables largely depends on the situation and the alternative method it is compared to. Before re-structuring a well-running system, think about how much benefit there will be and whether it is worth the effort.

WHERE CAN YOU FIND MORE INFORMATION?

There are multiple great resources for further information on hash objects and more examples that can help in understanding how to use them. To get more input on hash tables and their use, refer to the following papers:

- Lafler (2011) provides information on how to use hash tables for sorting data sets.
- For hints on n-way summarization and how to use hash tables as a dynamic dictionary, refer to Dorfman and Vyverman (2005).
- Dorfman and Shajenko (2011) gives a good guide on how to store file pointers in hash tables.
- Accessing external data, e.g. from an ORACLE database, is discussed by Blackwell (2010).
- Dorfman (2016) shows how to deal with duplicate keys.
- The possibility to search for keys other than those originally defined is explored in Schacherer (2015).
- Dorfman and Vyverman (2005) as well as Dorfman and Henderson (2019) give information on how to save hash tables after manipulating them.
- Lindenbaum (2024) provides a good description of the syntax used for hash tables.
- Oreshko (2019) gives examples on defining baseline, detecting repeated adverse events, flagging records from an interval and fuzzy merging.
- Multiple hash objects can be used in one data step, but it is advised to keep memory limitations in mind and choose smaller tables as hash tables in that case (Lafler, 2011).

CONCLUSION

There are many applications in SAS programming where the hash tables can be of use. They provide an easy way to quickly and efficiently merge and lookup data. Their efficiency is mostly due to being memory-based, which additionally saves disk space. While there are some pitfalls to keep in mind to avoid incorrectly matched values, the method is overall a relatively simple way of working with data from multiple sources.

REFERENCES

BLACKWELL, John. Find() the power of Hash - How, Why and When to use the SAS® Hash Object. In: *Proceedings of the 2010 North East SAS Users Group (NESUG) Conference*. 2010.

DORFMAN, Paul M. Using the SAS® Hash Object with Duplicate Key Entries. In: *Proceedings of the 2016 SAS Global Forum (SGF) Conference*. 2016.

DORFMAN, Paul M.; HENDERSON, Don. The SAS® Hash Object: Well Beyond Table Lookup. In: *Proceedings of the 2019 SAS Global Forum (SGF) Conference*. 2019.

DORFMAN, Paul M.; SHAJENKO, Lessia S. Hash+ Point= Key. In: *Proceedings of the 2011 North East SAS Users Group (NESUG) Conference*. 2011.

DORFMAN, Paul M.; VYVERMAN, Koen. Data Step Hash Objects as Programming Tools. In: *SUGI*. 2005. S. 236-30.

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

LINDENBAUM, Daniel. Blink and You Will Miss It!. *Proceedings of PhUSE 2024*, 2024.

ORESHKO, Valeriia. Let's See Further than One Observation: Applying Hash Objects for Typical Clinical Programming Tasks. *Proceedings of PhUSE 2019*, 2019.

SAS® Institute Inc. 2023. SAS® Help Center. 2023. [Cited: 04.08.2025.] https://documentation.sas.com/doc/en/lrcon/9.4/n1b4cbtmb049xtn1vh9x4waiioz4.htm.

SCHACHERER, Chris. Introduction to SAS Hash Objects. *SAS GF*. 2015. S. 3024-2015.

CONTACT INFORMATION

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

Author Name: Dr. Lara Suckut, Veronika Schmidt
Company: Chrestos GmbH
Address: Girardetstr. 1-5, Essen, Germany, 45131
Email: lara.suckut@chrestos.de, veronika.schmidt@chrestos.de
Website: www.chrestos.de