Using Hash Tables for Creating Electronic Codebooks
Raghav Adimulam, Westat

ABSTRACT
In projects that span multiple years, (e.g., longitudinal studies) there are usually thousands of new variables introduced at the end of every year or at the end of each phase of the project. These variables usually have the same stem or core as the previous year’s variables. However, they differ only in a digit or two that usually signifies the year number of the project. Therefore, every year, there is this extensive task of comparing thousands of new variables to older variables for the sake of carrying forward key database elements corresponding to the previously defined variables. These elements can include the length of the variable, data type, format, discrete or continuous flag, and so on.

In our SAS program, hash objects are efficiently used to cut down not only the time lapsed, but also the number of DATA and PROC steps used to accomplish the task. The resulting clean and lean code is much easier to understand. A macro is used to create the data set containing new and older variables. For a specific new variable, the FIND method in hash objects is used in a loop to find the match to the most recent older variable. Overall, what was originally taking about a dozen PROC SQL steps is now brought down to a single data step using the hash tables’ idea.

INTRODUCTION
For the past five years, the author has been involved in a multi-year longitudinal study that collects and publishes tens of thousands of variables yearly. The study uses Electronic Codebook software to allow users to identify and examine variables available in the published data file. Each of these variables has a fixed set of attributes assigned to it in the previous years of the study, like label, length, data type, formats, discrete/continuous flags, number of decimal places, description text, ID, etc. Tens of thousands of variables are collected each year over the duration of the study. In the present scenario, over a period of 6 years, the study has produced around 14000 variables. So, in the new year of study, i.e. in the seventh year, a new batch of around 2300 variables are being introduced.

Variables in the study are prefixed in the format Xn. The X is always a character and can be either a P, C, T, S or W. The n is always a number and can be one of 1, 2, 3, 4, 5 or 6 depending on the year of the study that the variable belongs to. The current batch of 2300 variables all have n equal to 7. The rest of the variable [substr(var,3)] after stripping the first two characters is herewith referred to as the core or stem of the variable. An example of the variables used in the study are shown in Figures 1 and 2.

PROBLEM STATEMENT
Our task is now to find as many possible matches of the 2300 new variables to that of the existing set of 14000 old variables. The new variables are present in an Excel file called NewVars.xlsx, whereas the existing 14000 variables are present in an Access database metadata, wherein they are stored along with the other attributes related to the variables. The desired matching for a new variable should be the older variable from the most recent year (as shown in Figure 1).

The traditional approach of matching involves using Proc SQL. The Proc SQL matching method takes at least 12 clauses for the whole process. First we extract all the X6 variables from the Meta Access database and match them with NewVars excel file to get matches and non-matches. Then we take the non-matches from the previous step and match them to the X5 variables. This process of taking the non-matches from the previous step and matching to the next round of variables will continue until all rounds are matched up, which is until the X1 variables.
Finally, the matches from all the steps above are accumulated to get the grand total of matches. It is expected that the matches found are close to 90%. The non-matches (~200) are the ones that are sent to the project team for a decision on the attributes assignment.

Figure 1. Desired mapping of the new variables to the older variables

THE HASH SOLUTION

The alternative to the Proc SQL solution is the Hash Object solution that accomplishes the task with just two Data steps – One with Hash Objects and another with a Macro. The workflow of the Hash program is given in Figure 2.
option mlogic mprint validvarname=upcase nofmterr compress=binary noxwait fullstimer ps=55 ls=120 nocenter;

libname newyr "H:\MY DOCUMENTS\SASGlobalForum\NewVars.xlsx" access=readonly;
libname oldvar "H:\MY DOCUMENTS\SASGlobalForum\ALLVAR.accdb" access=readonly;

data _null_; x=sleep(2); run;

/* Import ECB metadata from the Access table containing Older variable names*/
data Access;
  set oldvar.variables
    (keep=FieldName DescriptionText Length Type Decimal DiscreteFlag FormatName VarLabel);
run;

/* Import New variable names stored in an Excel file to SAS dataset NEWVAR */
%macro varn;
  data newvar ;
    length var_name $30 F1 - F6 $3 Core Y1 - Y6 $30;
    set newyr."Sheet1$"n ;
    if not missing(var_name) and substr(var_name,2,1)='7' ;
    core = substr(var_name,3);
    %do i=1 %to 6;
      F&i. = cats(substr(var_name,1,1),&i.) ;
      if not missing(F&i.) then Y&i. = cats(F&i.,core) ;
    %end;
  run ;
%mend;
%varn;

/* Use hash objects to find matches and nonmatches */
data match (keep=var_name FieldName DescriptionText Length Type Decimal DiscreteFlag FormatName VarLabel FlagMatch)
  nonmatch (keep=var_name);
  if 0 then set Access;
  if _n_=1 then do;
    declare hash h(DATASET:"Access");
    h.definekey('FieldName');
    h.defineData('FieldName','DescriptionText','Length','Type','Decimal','DiscreteFlag','FormatName','VarLabel');
    h.defineDone();
  end;
  set Newvar ;
  if h.find(key:Y6) = 0 then do; FlagMatch=6; output match; end;
  else if h.find(key:Y5) = 0 then do; FlagMatch=5; output match; end;
  else if h.find(key:Y4) = 0 then do; FlagMatch=4; output match; end;
  else if h.find(key:Y3) = 0 then do; FlagMatch=3; output match; end;
  else if h.find(key:Y2) = 0 then do; FlagMatch=2; output match; end;
  else if h.find(key:Y1) = 0 then do; FlagMatch=1; output match; end;
  else output nonmatch;
run;

libname _all_ clear;
The VARN Macro prepares the dataset that will be subsequently used by the Hash Objects Data Step. The macro contains a Data Step reading the Excel file NewVars using the Newyr Libname engine. Six new variables F1 – F6 are created containing the two-byte alpha-numeric prefix Xn, where n indicates the Year number as discussed previously. The core of the variables is extracted using the formula `core = substr(var_name, 3);`

Another set of six new variables are created, called Y1 – Y6, using the CATS function to concatenate the core with F1 – F6 variables respectively.

```sas
%do i=1 %to 6;
F&i. = cats(substr(var_name, 1, 1), &i.);
if not missing(F&i.) then Y&i. = cats(F&i., core);
%end;
```

The %DO - %END clause in the macro eliminates the redundancies in the code by creating the new F1 – F6 and Y1 – Y6 variables with minimal macro code. At the end of the execution of the VARN macro, we create a dataset called NEWVAR that is shown below in Figure 3.

<table>
<thead>
<tr>
<th>var_name</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>Y1</th>
<th>Y2</th>
<th>Y3</th>
<th>Y4</th>
<th>Y5</th>
<th>Y6</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREAD</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7READ</td>
</tr>
<tr>
<td>TWRITE</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7WRITE</td>
</tr>
<tr>
<td>TLANG</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7LANGUAGE</td>
</tr>
<tr>
<td>TIMEAS</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7TIMEAS</td>
</tr>
<tr>
<td>TSCIENCE</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7SCIENCE</td>
</tr>
<tr>
<td>TSSCHOOL</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
<td>T7</td>
<td>T8</td>
<td>T9</td>
<td>T10</td>
<td>T11</td>
<td>T12</td>
<td>T7SSCHOOL</td>
</tr>
<tr>
<td>S7PUBLIC</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7PUBLIC</td>
</tr>
<tr>
<td>S7MAGNET</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7MAGNET</td>
</tr>
<tr>
<td>S7Y1</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y1</td>
</tr>
<tr>
<td>S7Y2</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y2</td>
</tr>
<tr>
<td>S7Y3</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y3</td>
</tr>
<tr>
<td>S7Y4</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y4</td>
</tr>
<tr>
<td>S7Y5</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y5</td>
</tr>
<tr>
<td>S7Y6</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>S8</td>
<td>S9</td>
<td>S10</td>
<td>S11</td>
<td>S12</td>
<td>S7Y6</td>
</tr>
</tbody>
</table>

Figure 3. New variables (NEWVAR) table arranged for Hash processing

The last data step in the program uses the Hash Objects to create two datasets called MATCH and NONMATCH by keeping only the relevant fields in them.

In the data step, the first IF statement, which is highlighted below, is a non-executable statement because of the presence of 0 in the IF clause. However, during compile-time, the SAS compiler reads the Access table's metadata of the variables and adds it to the PDV, thereby avoiding having to use the LENGTH statement to define the variables’ metadata.

```sas
if 0 then set Access;
```

The `declare hash` statement creates the Hash object 'H' in memory and loads the hash object from the dataset ACCESS.

```sas
declare hash h(DATASET:"Access");
```
The `definekey` method defines a set of hash object keys, which is `FieldName` in this case.

The `defineData` method defines the data elements - `DescriptionText`, `Length`, `Type`, `Decimal`, `DiscreteFlag`, `FormatName` and `VarLabel` - to be stored in the hash object.

The `defineDone` method indicates that the key and data definitions are complete.

```plaintext
h.definekey('FieldName');
h.defineData('FieldName','DescriptionText','Length','Type', 'Decimal','DiscreteFlag','FormatName','VarLabel');
h.defineDone();
```

All the above methods are executed only once at `_n_`=1 which is the first iteration of the data step.

The `find()` method determines whether the given key has been stored in the hash object ‘H’. It will not need any argument tags if it is using the current value of the `FieldName` column, since we defined the hash object with `FieldName` column as the key. However, in our case, the name of the variable to lookup (`Y1 – Y6 variables`) are not the same as the name of the key item in the hash. Therefore, we use the `key:` argument tag followed by one of the variable names.

In the Hash IF..THEN..ELSE clause, we first try to find matches of the New variables to any variables from the immediately preceding year, which is year 6. Then from there onwards we use the ELSE condition to go in descending order of the year number as in 5, then 4, then 3, then 2 and finally year 1.

```plaintext
if h.find(key:Y6) = 0 then do; FlagMatch=6; output match; end;
else if h.find(key:Y5) = 0 then do; FlagMatch=5; output match; end;
else if h.find(key:Y4) = 0 then do; FlagMatch=4; output match; end;
else if h.find(key:Y3) = 0 then do; FlagMatch=3; output match; end;
else if h.find(key:Y2) = 0 then do; FlagMatch=2; output match; end;
else if h.find(key:Y1) = 0 then do; FlagMatch=1; output match; end;
else output nonmatch;
```

We start the search for the older variables by first IF statement using the `key:Y6`. If the key `Y6` is found in the hash object, the data variables are updated and the return code is set to zero. In addition, the `FlagMatch` variable is set to the value 6 and the variable values are output to the dataset MATCH. If the key is not found, the return code is non-zero and the ELSE statement looks for the next key match in Y5 and so on in the order (`Y4 Y3 Y2 Y1`). If none of the Yn keys are found, the remaining new variable names are sent to the dataset NONMATCH, which was found to have 219 variable names out of a total 2300 new variable names that needed matching.

The purpose of the `FlagMatch` variable is to be an indicator of the year the new variable found the match in. The number of matches from each year can be seen in Table 1.

```plaintext
proc freq data=match; tables FlagMatch; run;
```

<table>
<thead>
<tr>
<th>FLAGMATCH</th>
<th>Frequency</th>
<th>Cumulative Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>142</td>
<td>211</td>
</tr>
<tr>
<td>6</td>
<td>1870</td>
<td>2081</td>
</tr>
</tbody>
</table>

Non-matches = 219

Table 1: Frequency of the FlagMatch variable.
CONCLUSION

Hash tables are powerful tools when performing a lot of match-merging between the SAS® data sets. In the paper, both the HASH solution and the traditional Proc SQL solution are given so that the user may get the comparative sense of how both these solutions look side-by-side in comparison and how many lines of code can be saved by adopting the Hash object techniques.

Too often, the problem with programmers embracing Hash solutions is the absence of this stark comparison of listing the HASH code route alongside the traditional alternative of DATA/PROC steps. The table below illustrates this point. The first column is the last Data step in the Hash solution that replaced all of the code in second column – all of the twelve Proc SQL clauses and one data step.

<table>
<thead>
<tr>
<th>Equivalent HASH object solution</th>
<th>Equivalent Proc SQL code solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data match nonmatch;</td>
<td></td>
</tr>
<tr>
<td>if 0 then set Access;</td>
<td></td>
</tr>
<tr>
<td>if <em>n</em>=1 then do;</td>
<td></td>
</tr>
<tr>
<td>declare hash h(DATASET:&quot;Access&quot;);</td>
<td></td>
</tr>
<tr>
<td>h.definekey('FieldName');</td>
<td></td>
</tr>
<tr>
<td>h.defineData('Text');</td>
<td></td>
</tr>
<tr>
<td>h.defineDone();</td>
<td></td>
</tr>
<tr>
<td>end;</td>
<td></td>
</tr>
<tr>
<td>set Newvar;</td>
<td></td>
</tr>
<tr>
<td>** Match Access DB with newVar and get match 6 and nomatch 6 */</td>
<td></td>
</tr>
<tr>
<td>proc sql noprint;</td>
<td></td>
</tr>
<tr>
<td>create table match 6 as</td>
<td></td>
</tr>
<tr>
<td>select distinct a.<em>,b.</em></td>
<td></td>
</tr>
<tr>
<td>from Access a, newVar b</td>
<td></td>
</tr>
<tr>
<td>where a.FieldName = b.Y6;</td>
<td></td>
</tr>
<tr>
<td>create table nonmatch 6 as</td>
<td></td>
</tr>
<tr>
<td>select * from newVar</td>
<td></td>
</tr>
<tr>
<td>where var_name not in</td>
<td></td>
</tr>
<tr>
<td>(select var_name from match 6) ;</td>
<td></td>
</tr>
<tr>
<td>if h.find(key:Y6) = 0 then do;</td>
<td></td>
</tr>
<tr>
<td>FlagMatch=6; output match; end;</td>
<td></td>
</tr>
<tr>
<td>else if h.find(key:Y5) = 0 then do;</td>
<td></td>
</tr>
<tr>
<td>FlagMatch=5; output match; end;</td>
<td></td>
</tr>
<tr>
<td>create table match 5 as</td>
<td></td>
</tr>
<tr>
<td>select distinct a.<em>,b.</em></td>
<td></td>
</tr>
<tr>
<td>from Access a, nonmatch 6 b</td>
<td></td>
</tr>
<tr>
<td>where a.FieldName = b.Y5;</td>
<td></td>
</tr>
<tr>
<td>create table nonmatch 5 as</td>
<td></td>
</tr>
<tr>
<td>select * from nonmatch 6</td>
<td></td>
</tr>
<tr>
<td>where var_name not in</td>
<td></td>
</tr>
<tr>
<td>(select var_name from match 5) ;</td>
<td></td>
</tr>
<tr>
<td>else if h.find(key:Y4) = 0 then do;</td>
<td></td>
</tr>
<tr>
<td>FlagMatch=4; output match; end;</td>
<td></td>
</tr>
<tr>
<td>create table match 4 as</td>
<td></td>
</tr>
<tr>
<td>select distinct a.<em>,b.</em></td>
<td></td>
</tr>
<tr>
<td>from Access a, nonmatch 5 b</td>
<td></td>
</tr>
<tr>
<td>where a.FieldName = b.Y4;</td>
<td></td>
</tr>
<tr>
<td>create table nonmatch 4 as</td>
<td></td>
</tr>
<tr>
<td>select * from nonmatch 5</td>
<td></td>
</tr>
<tr>
<td>where var_name not in</td>
<td></td>
</tr>
<tr>
<td>(select var_name from match 4) ;</td>
<td></td>
</tr>
</tbody>
</table>
else if h.find(key: \texttt{Y3}) = 0 then do;  
FlagMatch=3; output match; end;  
/** Match Access DB with nonmatch\_4 and get match\_3 and nonmatch\_3 */  
create table match\_3 as  
select distinct a.*,b.*  
from Access a, nonmatch\_4 b  
where a.FieldName = b.Y3;  
create table nonmatch\_3 as  
select * from nonmatch\_4  
where var\_name not in  
(select var\_name from match\_3);  
else if h.find(key: \texttt{Y2}) = 0 then do;  
FlagMatch=2; output match; end;  
/** Match Access DB with nonmatch\_3 and get match\_2 and nonmatch\_2 */  
create table match\_2 as  
select distinct a.*,b.*  
from Access a, nonmatch\_3 b  
where a.FieldName = b.Y2;  
create table nonmatch\_2 as  
select * from nonmatch\_3  
where var\_name not in  
(select var\_name from match\_2);  
else if h.find(key: \texttt{Y1}) = 0 then do;  
FlagMatch=1; output match; end;  
else output \texttt{nonmatch};  
run;  
/** Match Access DB with nonmatch\_2 and get match\_1 and nonmatch\_1 */  
create table match\_1 as  
select distinct a.*,b.*  
from Access a, nonmatch\_2 b  
where a.FieldName = b.Y1;  
create table nonmatch as  
select * from nonmatch\_2  
where var\_name not in  
(select var\_name from match\_1);  
quit;  
/** Merge all the match datasets to get overall matches */  
data MATCH;  
set match\_6 match\_5 match\_4 match\_3  
match\_2 match\_1;  
run;

<table>
<thead>
<tr>
<th>Table 2: HASH approach vs PROC SQL approach</th>
</tr>
</thead>
</table>

**REFERENCES**

SAS® 9.4 Component Objects: Reference, Third Edition; Dictionary of Hash and Hash Iterator Object Language Elements  


ACKNOWLEDGMENTS
The author would like to thank Michael Raithel for the suggestions and critiques which were greatly appreciated.

RECOMMENDED READING

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

Raghav Adimulam
Westat
1600 Research Blvd
Rockville, MD 20850
adimulam@gmail.com

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.