Re-programming a many-to-many merge with Hash Objects
CS05 PhUSE 2012 Budapest

D. J. Garbutt

17 October 2012
Why? 1

In this talk I will demonstrate how you can re-program a many-to-many merge using hash objects instead of the various other methods already known.

- to illustrate the advantages of hash object for typical SAS programming
What?

- Two worked examples in the paper and on Wiki page
  1. a many to many merge — CM back into AE
  2. a CDISC data validation — check vocabularies across domains and variables

  One data step with no macro variables at all.

- In this presentation I will look at just the second example which comes from a paper at PhUSE last year by Guido Wendland. Check it out.
How and When to use a hash object

**How?**
Tips and syntax details — See the paper. lots of further reading

**When?**
Hashes can be imported into any data step and can be used in many instances where macro variables would be normally be needed.
Hashes are *not* just for look up of large tables
Up Next

1. Introduction and History
2. Check a CDISC Domain for invalid values
   - The problem
   - The data
   - The programming
   - Adding other variables
   - Adding other datasets
3. SAS programming after the hash object
   - SAS Memory model
   - Hash Object myths
   - Issues with SAS program development
   - Solved!
   - Questions
Hash arrays and Content Addressable Arrays

1977

AWK and Snobol4 in 1977
Currently in KornShell 93, bash, Ruby, PowerShell, .NET, Java, R, etc.,
Dorfman showed hashing could be programmed with a data step in the 90’s but it is not simple and only worth the effort for very large datasets.
Now since 2008 (9.2) SAS has built-in support for the hash object.
AWK program 1

```
xargs file | awk '{
    $1=NULL;
    t[$0]++;
}
END {
    for (i in t) printf("%d\t%s\n", t[i], i) }
' | sort -nr
```
How does the hash array look?

```plaintext
name[‘Dave’] + 1;
index: [Dave][dave][Greg][Urs][n/a]
address: 2334 504 31 230 42
Count: 3 1 4 2 33
```

An Array: is a storage area and a rule:
Allocate area of contiguous memory then to access it calculate storage location from the index.
How does it work? 1

**normal array rule:** take the memory base address and add \( \text{index} \times (\text{bytes} - \text{in} - \text{integer}) \)

**hash array rule:** take the memory base address, now run \( \text{index} = \text{”character string“} \) through hash function to get an integer, add it to base address.
## How does it work? 2

A hash function is a bit like the MD-5 key or random number generator — it does bit operations on the characters, in this case to get an integer.
How does it work? 3

A hash array is not a list data structure.

It has no links between members and cannot grow indefinitely. And insertion / deletion operations take constant time $O(1)$. 
Check variables for correct values

- Some SDTM variables (data elements) have fixed content
  - so we would like to check and make sure they conform.

Ingredients

1. Data domain e.g. AE
2. list of variables to check with the name of their code-list
3. SDTM controlled data list
   http://evs.nci.nih.gov/ftp1/CDISC/SDTM/SDTM/Terminology.odm.xml
A solution was very clearly presented at last year’s PhUSE by Guido Wendland

The problem

Generate a data step fragment like

data violations (keep=variable error) ;
set ae ;
if aeacn not in ('DOSE INCREASED','DOSE NOT CHANGED','DOSE REDUCED',...)
then do;
  variable = 'AEACN' ;
  error = cats('value of AEACN',aeacn,' not found in dictionary');
output ;
end;
/* next test follows ...*/
The problem

To get this code

Transpose the code list
then substitute it into an in() test

Repeat for each variable.
Thousands of lines of code for SAS to interpret and execute
### VIEWTABLE: Work.Codelist

<table>
<thead>
<tr>
<th>Codelist Code</th>
<th>Codelist Name</th>
<th>CDISC Submission Value</th>
<th>CDISC Synonym(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DOSE INCREASED</td>
<td></td>
</tr>
<tr>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DOSE NOT CHANGED</td>
<td></td>
</tr>
</tbody>
</table>

### VIEWTABLE: Work.Codelist_md

<table>
<thead>
<tr>
<th>Codelist Code</th>
<th>Codelist Extensible (Yes/No)</th>
<th>Codelist Name</th>
<th>CDISC Submission Value</th>
<th>CDISC Synonym(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C66767</td>
<td>No</td>
<td>Action Taken with Study Treatment</td>
<td>ACN</td>
<td></td>
</tr>
<tr>
<td>C66768</td>
<td>No</td>
<td>Outcome of Event</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C66780</td>
<td>Yes</td>
<td>Age Span</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C66781</td>
<td>No</td>
<td>Age Unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C66797</td>
<td>No</td>
<td>Category for Inclusion/Exclusion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C66784</td>
<td>No</td>
<td>Common Terminology Criteria Events</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VIEWTABLE: Work.Meplist

<table>
<thead>
<tr>
<th>ruleID</th>
<th>variable</th>
<th>Codelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT0001</td>
<td>AEACN</td>
<td>ACN</td>
</tr>
<tr>
<td>CT0002</td>
<td>AESEV</td>
<td>AESEV</td>
</tr>
<tr>
<td>CT0003</td>
<td>DOMAIN</td>
<td>DOMAIN</td>
</tr>
<tr>
<td>CT0007</td>
<td>AEOUT</td>
<td>OUT</td>
</tr>
<tr>
<td>CT0037</td>
<td>AEBODSYS</td>
<td>SOC</td>
</tr>
<tr>
<td>CT0064</td>
<td>AESER</td>
<td>NY</td>
</tr>
</tbody>
</table>
The data

The goal is to check values 1

data violations (keep=variable error);
set ae;
The goal is to check values 2

```sas
if aeadn not in('DOSE INCREASED','DOSE NOT CHANGED','DOSE REDUCED',...)
then do;
    variable = 'AEACN';
    error = cats('value of AEACN',
                   ,aeadn
                   ,',' not found in dictionary');

    output ;
end;
```
The data

The goal is to check values 3

/* next test follows ...*/
run;
The goal is to check values 4

Could we do a fast lookup of the right code? We would put all the vocabulary in one hash object and use the check() method to test for presence.
Something like this: 1

```sas
notfound = codelst.check(
    key: CODELIST_CODE,
    Key: AEACN
);
```
Something like this: 2

```sas
if notfound then do;
    variable = 'AEACN';
    error = cats('Value of AEACN',aeacn,' not found in dictionary');
output ;
/* next variable... */
```
The data

Something like this: 3

Notes: no quotes round AECN
We need the code lists....
<table>
<thead>
<tr>
<th>Code</th>
<th>Code List Code</th>
<th>Codelist Name</th>
<th>CDISC Submission</th>
</tr>
</thead>
<tbody>
<tr>
<td>C66767</td>
<td></td>
<td>Action Taken with Study Treatment</td>
<td>ACN</td>
</tr>
<tr>
<td>C49503</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DOSE INCREASED</td>
</tr>
<tr>
<td>C49504</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DOSE NOT CHANGED</td>
</tr>
<tr>
<td>C49505</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DOSE REDUCED</td>
</tr>
<tr>
<td>C49501</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DRUG INTERRUPTED</td>
</tr>
<tr>
<td>C49502</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>DRUG WITHDRAWN</td>
</tr>
<tr>
<td>C48660</td>
<td>C66767</td>
<td>Action Taken with Study Treatment</td>
<td>NOT APPLICABLE</td>
</tr>
</tbody>
</table>
Split into two datasets
read the XML file

filename SDTMCT
  ’H:\garbutt\DATA\Projects\CS05-djg\SDTM_
filename SXLEMAP
  ’H:\garbutt\DATA\Projects\CS05-djg\CDISC_
libname SDTMCT xml
xmlmap=SXLEMAP
access=READONLY;
The data manipulate the datasets

/* the map defines two tables */
data codeList_M;
  SET SDTMCT.CodeList_M;
run;

data codelist;
  set sdtemct.codelist;
  codelist_name= scan(oid,3,’.’);
run;
Read the control dataset
maplist data as used by Wendland

data cs05l.maplist;
  length ruleID $6 variable $8
     Codelist $411;
input ruleid @10 variable @20 codelist $10.;
datalines;
CT0001    AEACN    ACN
CT0002    AESEV    AESEV
CT0009    DOMAIN   DOMAIN
CT0027    AEOUT    OUT
CT0037    AEBODSYS SOC
CT0064    AESER    NY
;
The programming

Setup the hash objects; Lookup the data

```sas
varleft = vars_to_check.first();

rc = codelist_md.find();

notfound = codelst.check(
    key: CODELIST_CODE,
    Key: AEACN);
```
## Program flow and PDV contents

<table>
<thead>
<tr>
<th>Source</th>
<th>Dataset :AE</th>
<th>Maplist</th>
<th>Maplist</th>
<th>Maplist</th>
<th>Codelist</th>
<th>Codelist</th>
<th>PDV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Codelist_md</td>
<td>Codelist_md</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>AEACN</td>
<td>RuleID</td>
<td>Variable</td>
<td>Codelist</td>
<td>Codelist_code</td>
<td>codelist_value</td>
<td>RC</td>
</tr>
<tr>
<td>Sample Value</td>
<td>Treatment</td>
<td>CT00001</td>
<td>AEACN</td>
<td>ACN</td>
<td>C66797</td>
<td>INCREASED</td>
<td>0</td>
</tr>
</tbody>
</table>

```r
set ae;
variable = Treatment
'AEACN';
rc = maplist.find();
variable = Treatment
rc = CT00001
rc = AEACN
rc = ACN
codelist_md.find0;
codelist_value= vvaluex(variable);
rc = CT00001
rc = AEACN
rc = ACN
c66797
rc = Treatment
rc = 2365
codelist.check();
```
Iterating through the datasets

add hash iterator to the maplist hash object

/* Maplist matches variable and codelist (num id) */
declare hash maplist(dataset: "cs05l.maplist"

    , hashexp:4
    , ordered: ’yes’);
rc + maplist.DEFINEKEY(’VARIABLE’) ;
rc + maplist.DEFINEDATA(’VARIABLE’,’RULEID’,’CODELIST’);
rc + maplist.DEFINEDONE();
call missing(VARIABLE, ruleid, codelist) ;
    if rc then
        putlog ’ERRO’
            ’R: problem creating hash obj maplist’ ;

/* define iterator for maplist to enable looping */
declare hiter vars_to_check(’maplist’) ;
Iterating with `first()` and `next()`

```sas
varleft = vars_to_check.first();
do while (varleft = 0 ) ;
RC = 0;
notfound = 1 ;
call missing(ruleid,
    ,codelist
    ,codelist_value
    ,codelist_code
  ) ;
varleft = vars_to_check.next() ;
end;
```
Look up the variable value to check

valtotest = vvaluex(variable);  

notfound = codelst.check(  
    key: CODELIST_CODE,  
    Key: valtotest );
Loopying across datasets 1

Compile the data step and generate with call execute (1 of 4 ways)

```sas
*--- compile program ---- ;
data violations /PGM=work.violations ;
run;
*--- and generate list of members in vmembers from maplist--;
```
Adding other datasets

Looping across datasets 2

Compile the data step and generate with call execute (1 of 4 ways)

data _null_;
set vmembers;
call execute(
    'data PGM=work.violations ;'
);
call execute('execute ;');
Looping across datasets 3
Compile the data step and generate with call execute (1 of 4 ways)

```sas
call execute(
catt('redirect INPUT AE=
    , memname
    ,');'
  ) ;
```
Looping across datasets 4

Compile the data step and generate with call execute (1 of 4 ways)

call execute(  
catt('redirect output violations =viol_  
    , memname  
    ,''; run;'  
  )  
);  
run;
Introduction and History

Check a CDISC Domain for invalid values

SAS programming after the hash object

Adding other datasets

**Full code is**

- about 120 lines
  - 50 lines of hash object declaration
- not a *single* macro variable
- headed for the wiki...
SAS memory model — Till now
SAS Memory Models with hash objects

During Data Step only

Define hash object

hash_obj = K1 DATA

Program Input Stack

program.sas

define hash str()

str:find()

output()

check()

has_next()

find_next()
8 Hash Object myths dispelled

1. They are just for lookup with millions of rows — I don’t need that syntax mess!

2. Useless — you can’t have duplicate keys!

   - multidata:’yes’

3. If keys are duplicated you get no warning!

   - duplicate:’error’
8 Hash Object myths dispelled 2

4. You can’t sort them!
   ordered: ‘yes’

5. You’ll run out of memory!
   4 GB is 5.3 million 800 byte records

6. You have to build the hash with add(), delete() etc. Tedious!
   dataset:’ work.dave (where myths > 9)’
8 Hash Object myths dispelled 3

7. Can’t be used outside a data step! → output() method will write to dataset

8. Merge, Proc SQL, set key=, and formats work fine for merging → hash is faster and advantage increases with size of data
8 Hash Object myths dispelled 4

9

Syntax is verbose and opaque

10

Can’t be used with procs, macros, the DDV or proc IML
### Issues with SAS program development

#### Longstanding issues since Version 6 and earlier

Why is developing large scale systems so hard with SAS? My list of reasons is:

1. separation of procs and data steps from each other by the disk transfer wall
Fundamental Issues with SAS program development

Longstanding issues since Version 6 and earlier

2. macro is purely a text generator (cf Korn Shell, Perl, Ruby)

3. macro variables are simple, individual structures. Other structures have to be emulated by naming conventions and collision avoidance. Or not.
Issues with SAS program development

Fundamental Issues with SAS program development 3
Longstanding issues since Version 6 and earlier

4. in data steps only scalars are allowed (ie single named elements of PDV). Arrays are just list of variable names, not data structures!

5. lack of truly encapsulated functions or subroutines (goto label: – are you kidding me?).
Fundamental Issues with SAS program development

Longstanding issues since Version 6 and earlier

6 lack of an exec() function that can execute code in place and return the result
Solved!

Issues with SAS program development 1 — solved!

In Version 9.2 SAS programming has jumped into the present and the previous issues addressed:

Hash Object adds matrix objects to data step, with fast search, size independent performance and flexible data types
Issues with SAS program development 2 — solved!

Proc FCMP allows functions to encapsulate user code and for it to be used as normal SAS function. It supports arrays and call by reference.

run_macro, dosub & dosubl allow data steps and procs to be called from a data step and pass results back via macro variables.
Issues with SAS program development 3 — solved!

J Secosky, Executing a PROC from a DATA step, SAS GF 2012,
The hope for meta data

Meta data programming and meta data systems are the next wave of change
The hash object opens new doors

- shorter code
- easy to debug (can use data step debugger)
- easier to maintain

For designing and building systems, especially based on metadata, these three new features will make an enormous difference to SAS programming and how it is done.

It is the 21st century, SAS programming is fun again. Datasets and data steps can drive processing without macro.
And the next doors:

please SAS:

1. Allow hash objects to be passed to proc FCMP functions
2. Allow hash objects in open code and to persist outside data steps
3. Allow procs to be passed hash objects as if datasets
4. Allow procs to write hash objects as output
Questions

? 

david@garbuttconsult.ch
References, further reading

PhUSE Wiki
Where you can also add code, examples, and discussions.

NOTE:
Watch out for papers written before V9.2
— They may now be misleading
— duplicates are allowed,
— they may also have workarounds that are no longer needed.