Creating Data-Driven Data Set Names in a Single Pass Using Hash Objects

Jack Hamilton
Kaiser Foundation Health Plan
Oakland, California

The Topic Today

The topic today is:

Creating multiple output data sets

In a single pass through an input data set

Where the output data set names are based on values in the input data set and are not known in advance

Why Would You Want To Do This?

If you're working strictly in SAS, you probably wouldn't need to – BY groups, CLASSes, and some reporting procedures can handle data in groups.

But you might have external requirements that you can't change:

- Separate files for external programs
- Separate files for external reporting
- Pomity-haired bosses

And Another Reason

This topic gave me a handle to talk about two things I'm interested in: DOW loops and Hash Objects.

And I'm not making this up: I started looking at this after someone asked a question on SAS-L.

Some Sample Data

```
data have;
   do state = 'KY', 'CA', 'MS';
      do b = 1 to 3;
         do c = 200, 100;
            output;
         end;
      end;
   end;
   stop;
run;
```

Some Of The Data

Data Set HAVE

Obs	state	b	c
1	KY	1	200
2	KY	1	100
3	KY	2	200
4	KY	2	100
5	KY	3	200
6	KY	3	100
7	CA	1	200
8	CA	1	100
9	CA	2	200
10	CA	2	100
11	CA	3	200

Out_KY

Out_CA

The Old Way To Do It

```
data ex2_KY ex2_CA ex2_MS;
    set have;
    select (state);
        when ('KY') output ex2_KY;
        when ('CA') output ex2_CA;
        when ('MS') output ex2_MS;
        otherwise error 'Unknown
  State ' state=;
    end;
run;
```

But…

But this requires that we know the state codes in advance, and we've already stipulated that we don't.

So we have to generate code dynamically.

Why Is Dynamic Code Bad?

It's not an unmitigated evil, of course, and sometimes dynamic code generation is by far the best solution to a problem, but…

- You can't see the code that's going to execute, and
- In this case, it requires an extra read (or more) through the data.

I'll show you how to do it anyway…

Here's One Way To Do It

Let's look back at the base code:

```
data ex2_KY ex2_CA ex2_MS;
    set have;
    select (state);
        when ('KY') output ex2_KY;
        when ('CA') output ex2_CA;
        when ('MS') output ex2_MS;
        otherwise error 'Unknown State '
  state=;
    end;
run;
```

One Way To Do It

The changeable part of the code is localized to two places. All we need to do it to generate those pieces of code and plug them in.

Three approaches (at least):

- Use PROC SQL to generate macro variables
- Use a data step to write code that you %INCLUDE
- Use a data step to CALL EXECUTE code.

PROC SQL Approach

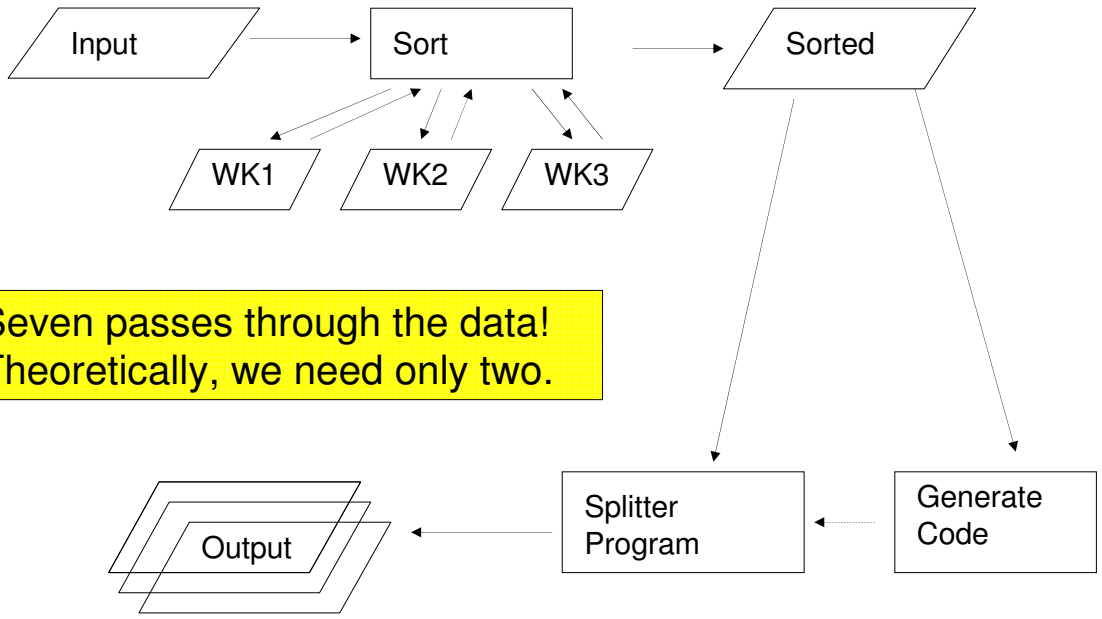
```
proc sql noprint;
    select distinct
              'ex3_' || state,
              cats('when (',
                   quote(state),
                   ') output ex3_',
                   state, ';')
    into
              :DATASETS separated by ' ',
              :WHENS    separated by ' '
    from      have;
quit;
```

PROC SQL Approach

```
data &DATASETS;
    set have;
    select(state);
        &WHENS
        otherwise;
    end;
run;
```

Details are left as an exercise for the reader, but Ron Fehd is doing a workshop tomorrow on SELECT … INTO.

Many, Many Reads of the Data



Seven passes through the data!
Theoretically, we need only two.

But There's Another Way

How can we achieve the same result with only two passes through the data?

- With Hash Objects
- And as a bonus, a DOW Loop

The DOW Loop

The DOW Loop was discovered by and named after our section co-chair, Ian Whitlock.

It stands for DO loop of Whitlock.

He didn't name it for himself. As far as I can tell, it was named by Paul Dorfman, who has also promoted it.

The DOW Loop

It's faster than the standard one-iteration-per-observation method of processing BY-groups in a data step.

It simplifies code which is executed at the beginning and end of BY-groups.

After you understand it, it's also simpler to use.

The DOW Loop

```
data _null_;
   /* Code to run before BY-group */
   putlog 'INFO: Starting BY-group';
   do until (last.state);
      set have;
      by state notsorted;
      /* Code to run for each observation */
      putlog '    ' b= c=;
   end;
   /* Code to run after BY-group */
   putlog 'INFO: Ending BY-group' state=;
run;
```

Traditonal Approach

```
data _null_;
   set have;
   by state notsorted;
   /* Code to run before BY-group */
   if first.state then
      putlog 'INFO: Starting BY-group';
   /* Code to run for each observation */
   putlog '    ' b= c=;
   /* Code to run after BY-group */
   if last.state then
      putlog 'INFO: Ending BY-group' state=;
run;
```

Advantages of New Way

- Fewer lines of code
- Faster
  - Data step iterated few times
  - No IF statements that will usually be False
- Easier to read (after you understand it) because you don't have to think about when IFs are executed (code can be read sequentially)
- Also, there are some really neat things you can do with double-DOW loops, but this is a 20 minute paper.

The Hash Object

- Also called an Associative Array, because it's somewhat similar to associative arrays in other languages such as REXX.
- Like an array, but with indexed keys instead of numbered subscripts.
- Unfortunately, saddled with a verbose syntax. It's also an unfamiliar syntax, but you'll get over that.
- Paul Dorfman is doing a paper on this in this room later this morning.

Using a Hash Object

- Declare it
- Initialize it
- Define the keys
- Define the data
- Add records
- Write the records back out

Three Key Differences

Three key differences between HOs and Arrays:

- Hash Objects use keys, not indices.
- Hash Objects are implemented as object-oriented functions, not as base language syntax.
- Most Hash Object functions accept parameters for data set names and variable names rather than data sets and variables

A Somewhat Obscure Point

That last point is somewhat obscure, I'll admit.

To state it differently:

Hash Objects let you decide on data set names and variable names at run time rather than compile time.

That's what makes this paper possible.

Let's Look at the Code

```
data _null_;
    declare hash group;
    group = _new_ hash(ordered: 'ascending');
    group.definekey('_unique_Key');
    group.definedata('State', 'b', 'c');
    group.definedone();
    do until (last.state);
       set have;
       by state notsorted;
       _unique_key + 1;
       group.add();
    end;
    group.output(dataset: 'ex5_' || state);
    group.delete();
run;
```

Some Restrictions

- Data must be in BY-group order in this example (but the groups themselves don't have to be in sort order).

  The example code contains an example of using two Hash Objects to get around this requirement.

- All the data for a group must fit into memory (old fashioned, maybe they'll fix it someday).

Acknowledgments

- Ian Whitlock, for discovering this technique
- Paul Dorfman for publicizing it
- Ron Fehd for for further publicizing it
- Mike Rhoads for setting a good example
- SAS-L for bringing up interesting questions

Contact Information

Web site:

www.excursive.com/sas/

Email:

jfh@alumni.stanford.org

Revised paper and code will be on web site in a week.