Paper 051-2012

Generation Why: How Generation Data Sets Can Help

Lisa Eckler, Lisa Eckler Consulting Inc., Toronto, ON

ABSTRACT

Generation data sets are powerful tools available in SAS® to manage versions of data. This paper will address what generation data sets are, how they are defined and referred to, and why you should use them to improve programming productivity and achieve simple and elegant management of historical versions of data in SAS.

INTRODUCTION

A generation data set is one in a related a series of SAS data sets in a single library, sharing a single name data set name, distinguished by a version number. Generation Data Sets (also described as Generation Groups[1]) provide a way to attach a single name to a series of data sets representing historical versions of the data. These data sets may be accessed using either an absolute reference (to a specific generation of data) or a relative reference (to the most recent generation or an offset from the most recent). Each time a new generation is created, the relative generations are aged. Once the pre-defined maximum number of generations has been reached, the oldest (greatest absolute number[2]) will automatically be dropped as the newest one is created. The Generation Data Set concept is not new to SAS and has been discussed previously in the user community (see Lafler, 2006) but doesn't seem to be widely used or appreciated. This means that we still encounter situations where the programmer has written many programming steps, copying data sets to different data set names or libraries to accomplish precisely what would occur naturally with a generation data set, just by using a single specification of the number of generations to maintain. This specification comes from the presence of GENMAX, a very helpful data set option which can be used when a data set is created, through a DATA step or procedure output. GENNUM, described below, is the corresponding option to refer to a particular generation of a data set.

The chart on the following page shows what happens each time the data set name EXAMPLE is used as output from a SAS PROC or DATA step, where EXAMPLE has been defined with `GENMAX = 3`.

[1] SAS Generation Data Sets behave much like Generation Data Groups (GDGs) in the mainframe MVS operating environment. The SAS implementation of this functionality offers more flexibility as to when the group is defined, but otherwise may seem quite familiar to those who have worked in MVS.

[2] Since version numbers are only 3 digits, the absolute version number is represented modulus 1000. See Figure 1 on the following page for an illustration.

Here's an illustration of how data gets aged through a series, where `GENMAX = 3`.

Program action	Name of newest data set	Relative generation(-1)	Relative generation (-2)	What happens
Create first generation	EXAMPLE			New EXAMPLE is the only data set in the series.
Create second generation	EXAMPLE	EXAMPLE#001		EXAMPLE is aged to relative generation -1, also accessible as absolute generation 1. New EXAMPLE is created.
Create third generation	EXAMPLE	EXAMPLE#002	EXAMPLE#001	EXAMPLE#001 is aged to relative generation -2, also accessible as absolute generation 1. EXAMPLE is aged to relative generation -1, also accessible as absolute generation 2. New EXAMPLE is created.
Create fourth generation	EXAMPLE	EXAMPLE#003	EXAMPLE#002	EXAMPLE#001 is dropped. EXAMPLE#002 is aged to relative generation -2, also accessible as absolute generation 2. EXAMPLE is aged to relative generation -1, also accessible as absolute generation 3 New EXAMPLE is created.
…	…	…	…	…
Create 1000th generation	EXAMPLE	EXAMPLE#999	EXAMPLE#998	…
Create 1001st generation	EXAMPLE	EXAMPLE#000	EXAMPLE#999	… Note that the maximum generation number allowed is 999, so the absolute reference number rolls over to 000. New EXAMPLE is created.

HOW TO DEFINE A GENERATION GROUP

Specify GENMAX = <n>[3] when creating a SAS data set, to define it as a generation data set and establish the maximum number of generations to keep. That single keyword is all that's required to define a SAS data set as being part of a generation group.

For example, to define a generation group with twenty-four generations and populate the initial generation:

```
data RESULTS_MONTHLY (genmax = 24);
        set work.MY_SUMMARIZED_DATA;
run;
```

Once the first couple of generations in a generation group exist, they can be accessed programmatically by referring to the relative generation number[4] using the GENNUM option, for example:

```
proc sql;
        select *
        from RESULTS_MONTHLY(gennum=1);
quit;
```

or by referring to the absolute generation number[5]:

```
proc sql;
        select *
        from RESULTS_MONTHLY(gennum=-1);
quit;
```

WHY USE GENERATION GROUPS?

Generation data sets are simple to set up and powerful and elegant to use in programs, but tend to be under-utilized. Why should we use them?

- Whether in a formal development environment, in production, or for prototyping or ad-hoc analysis, maintaining automatic back-ups of data provides for easy data recovery if a step doesn't turn out as required.
- Always having a prior version of data easily accessible makes it easy to validate new data by running PROC COMPARE, comparing the new data set with the previous version.
- Keeping the same format of data for previous time periods in data sets with the same name makes it easy to summarize, analyze or report on time series data. This is a particularly rare, yet highly effective use of generation groups, as we will see in the example below.

[3] <n> can be any positive integer from 1 to 1000.

[4] When only the first generation exists, it can be accessed as either GENNUM = 1 or GENNUM = 0.

[5] Normally, a positive number is an absolute reference and a negative number is a relative reference, but 0 is a relative reference to the latest (or current) created version.

Any or all of the above are easily accomplished with minimal coding and no need to manually delete old data, by taking advantage of generation data sets. Keeping back-ups via generation groups is more efficient than creating your own named copies of data, since the data doesn't need to be re-processed to create an archive.

EXAMPLE: ALTERNATIVE SOLUTIONS FOR A PROGRAMMING TASK

Let's explore a situation which may be common in ad-hoc data analysis or reporting to see how generation data sets might be especially helpful. There are several ways to accomplish the same result, but one of those ways will involve far less complexity and less code and be more flexible.

In this situation, we have a series of monthly data sets, with a new data set being created for each month, beginning with January 2010.

The data set name identifies what month of data is included, so we have

RESULTS _JAN2010
RESULTS _FEB2010
RESULTS _MAR2010
…
RESULTS _AUG2011

What if you received an ad-hoc request to measure changes (growth) by comparing activity in the past three months compared to the three months prior to that and also compared to the corresponding three months in the previous year?

Method 1: The quick "ad-hoc" approach

We can combine three months into one data set for each of the consolidations required.

```
data LATEST3MO;
       set    RESULTS_JUN2011
              RESULTS_JUL2011
              RESULTS_AUG2011
              ;
run;

data PREVIOUS3MO;
       set    RESULTS_MAR2011
              RESULTS_APR2011
              RESULTS_MAY2011
              ;
run;

data LASTYR3MO;
       set    RESULTS_JUN2010
              RESULTS_JUL2010
              RESULTS_AUG2010
              ;
run;
```

After consolidating results for each of LATEST3MO, PREVIOUS3MO and LASTYEAR3MO, we can combine the summarized data for reporting.

This code is simple enough and of course it will work fine for an ad-hoc requirement. It's also somewhat self-documented. But if this turned into a request to provide a new compilation of similar data every month, for rolling three-month windows – or worse, longer windows -- it would become bothersome. The code, although simple, would need to be altered every month.

Method 2: Using macros, parameters and functions to make the code more flexible

To add enough flexibility to accommodate rolling windows and make the program reusable, we could write code to

- determine what the current month is (either based on the current system date or a parameter)
- use the current month to generate a list of the data set names for the LATEST three months in a macro variable
- generate a list of the data set names for the PREVIOUS three months in a macro variable
- generate a list of the data set names for the relevant months from LAST YEAR in a macro variable.

The consolidation of data and summarizing and combining for reporting could be done, similar to Method 1 but with macro variables for lists of names instead of hard-coded data set names.

This style of coding is often viewed as more advanced than what was done in Method 1 and is a common solution to turning what was an ad-hoc program into a reusable one. It is certainly more complex, taking advantage of SAS date functions and macros, and although highly flexible and not needing to be altered each month, it would require a lot of coding – and testing – to begin with.

This approach also assumes that a new data set is added to the collection every month. At some point, you will need to manage the archiving or deletion of old monthly results data sets.

Method 3: Using Generation Data Sets

Generation data sets provide a much more eloquent way of accomplishing the same task. We would have data sets with sequential names RESULTS_, RESULTS_#001, RESULTS_#002 through RESULTS_#nnn, where RESULTS_ contains the data from the most recent month. RESULTS_nnn could be referred to as RESULTS_(gennum=0), the previous month would be RESULTS_(gennum=-1), etc. The earliest month of interest for our example would be RESULTS_(gennum=-14) for the beginning of the same three-month period in the previous year.

```
data LATEST3MO;
       set    RESULTS_(gennum=-2)
             RESULTS_(gennum=-1)
            RESULTS_(gennum=0)
              ;
run;

data PREVIOUS3MO;
       set    RESULTS_(gennum=-5)
              RESULTS_(gennum=-4)
              RESULTS_(gennum=-3)
              ;
run;

data LASTYR3MO;
       set    RESULTS_(gennum=-14)
              RESULTS_(gennum=-13)
              RESULTS_(gennum=-12)
              ;
run;
```

This solution will work for rolling three month windows without requiring any code changes, requires no macro variables, and can very easily be adapted to rolling reporting windows other than three months. Because the data sets are created as Generation Data Sets, once the maximum number of generations has been reached, the oldest version will automatically drop off whenever a new one is created, so it isn't necessary to periodically delete old data.

USAGE NOTES FOR GENERATION DATA SETS

- All of the examples in this paper are using work (temporary) data sets for simple illustration. The generation data set construct is, of course, only truly meaningful when used with permanent SAS libraries.
- A new generation is created any time the data set referred to would (had it not been a generation data set) have been overwritten, as output from a DATA step or a procedure.

- If a generation group is defined and then referred to later with a GENMAX specification smaller than what was in effect, the earlier generations will be dropped and the only the recent ones up to the new maximum will be maintained. If a larger GENMAX is specified, additional generations will be added, up to the new maximum before any more are dropped.
- The number of generations of data to be maintained can also be modified using PROC DATASETS. Refer to Raithel (2010) for information.

CONCLUSION

The use of generation data sets can save development time, reduce and simplify code (and therefore code maintenance), reduce run time and maintain automatic data back-ups for comparison or recovery. All of these are consistent with safe and efficient programming practices. They should be used whenever it is appropriate to maintain multiple versions of data, either as backups or for future analysis.

REFERENCES AND RESOURCES

Lafler, Kirk Paul.SAS Institute Inc. 2006.SUGI 31 Proceedings. Cary, NC: SAS Institute Inc.
http://www2.sas.com/proceedings/sugi31/253-31.pdf

Raithel, Michael, SAS Institute Inc. 2010. Proceedings of the SAS® Global Forum 2010 Conference. Cary, NC: SAS Institute Inc. http://support.sas.com/resources/papers/proceedings10/138-2010.pdf

CONTACT INFORMATION

Your comments are welcome.

Lisa Eckler
lisa.eckler@sympatico.ca