A Sample Delivery System for A Multi-Site Study Using SAS® On A UNIX Platform

Gabriel Cano, MCNC Environmental Modeling Center, Research Triangle Park, NC

ABSTRACT

Planning a monthly sample delivery system for a multi-site study is a non-trivial undertaking. Management can be cumbersome over time if basic issues are not addressed prior to sampling. Issues of concern at the outset of the system are the computing environment, sampling design, major processes, and data management. Once data have been received there may be very few chances to make adjustments to the system. It is for this reason that the most generalized approach is best so that needed changes can be accommodated throughout the life of the system. Monthly data receipt and sample delivery makes system development challenging. This presentation will focus on methods for generalizing SAS® code, large-scale data management issues, reacting to unforeseen problems in a study, and taking advantage of the UNIX Operating System.

KEYWORDS

sampling, data management, UNIX

INTRODUCTION

This presentation formulates a generalized approach taken to manage a sampling system with monthly delivery. The following items will be outlined for the discussion: background and sampling rules, a generalized approach, conceptualizing the system, defining mechanisms, useful functions and macros, peripherals and additions to the system, and future considerations.

BACKGROUND AND SAMPLING RULES

Motivation for this discussion is provided by recent work on the National Study of Child and Adolescent Well-Being (NSCAW). When this study began it was well known that there was going to be a high volume of data to be processed. One hundred PSUs contained in approximately 50 files would be received a month for about one year. Each file would contain records of information regarding children and families receiving services from the agencies. One month's data would be received within the first two weeks of the following month. A sample would then be drawn from a PSU using a Stratified Probability Proportional to Size sample selection scheme. Sample would then be delivered to the Case Management System (CMS) for entry and tracking for interviews. In summary, this system would achieve the second stage of sampling for the NSCAW.

The Sampling rules of this study governed system development. They were as follows: (1) PSUs were broken up into 9 strata; (2) eight analysis domains were constructed for sampling (by age and services); (3) a child will have one chance of selection throughout the life of the system; (4) at most one child per household was selected throughout the life of the system; (5) selection should give priority to children by domain; (6) 60 interviews should take place per each of the 100 PSUs for the entire study. There was also an issue of timing that existed. Data were received with a time stamp from the site agency. Once data were sampled it should be released to the tracking system within a month of that time stamp regardless of the date it was actually received.

Sampling was originally expected to take place for 1 year but took place for 15 months ending in Q2 of 2001.

A GENERALIZED APPROACH

In a truly generalized approach the line of logic is more complex than expected. Over time so many things can go wrong. It is better to build redundancy checks into the system than to react hastily. With this in mind the success of identifying possible problem areas depends on how well the team explores (or questions) the system without getting bogged down in the details of it.

Things to keep in mind for conceptualizing a generalized system are:

- What information can change?
- What information should never change?
- What are the major processes?
- What are the inputs and outputs for those processes?

The issues associated with these questions are frequently intertwined and complex. These aspects can be discussed for long periods of time without reaching any certain conclusions. These questions should motivate concepts such as global information, standardized mechanisms (macros), and any other hidden functions that developers deem useful.

CONCEPTUALIZING THE SYSTEM

Now begins the iterative process of conceptualizing the system. Pressing out the most general aspects of system logistics into pseudo code was quite challenging. Generally speaking, too much discussion can lead to fatigue and too little discussion may lead to oversight of devastating situations.

With large brush strokes this was the line of logic:

- Get monthly data file.
- Select a sample from the file.
- Deliver sample to the tracking system.

Taking a step into this logic the following points formed these system processes:

- Apply global information to each program
- Get data frame and compare to frame history
- Get sample sizes and adjust them by the available data
- Select the sample and compare to sample history
- Release to tracking and update historical information

It can be seen in the 2nd and 4th bullet above that a recurring task is to compare the frame to the frame history and the sample to the sample history, respectively. Recognized early was the situation that some sites would be late entries into the system. This coupled with a varying number of files to be delivered each

month would cause data management problems over time.

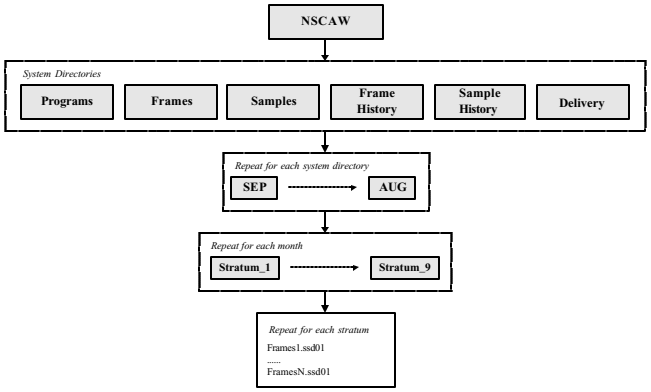
After an exhaustive discussion of needs, desires and requirements it was decided that the following points were most important to a successful system:

- Minimize the amount of code.
- Identify and automate key mechanisms for public use.
- Use UNIX scripts to generate, file and run programs.

How was the implementation of this system, with the concepts and requirements outlined so far, going to be carried out? It was decided that SAS® and the UNIX operating system were the choices for our computing environment. SAS® was chosen because no other application combines the flexibility of data management and statistical analysis better. UNIX was the operating system of choice for its richness.

DEFINING MECHANISMS

The file system is shown here to motivate our method:



The graphic above shows how files existed in the Frames directory. This structure scheme was duplicated for all system directories. All programs and data were filed by month and stratum ID. This organized the system into a somewhat uniform fashion allowing the ability to create mechanisms for file management.

Clearly as the time in the study proceeded from month to month the number of files to be examined in the system also increased. This increase imposed a degree of system complexity that had to be addressed and controlled. There was a welling concern of walking a tightrope to manage this system once data collection began. A generalized and automated method for comparing a monthly file to its history was needed. (A macro!) This was needed for the data frame to satisfy the sampling rule of not duplicating selection. This was also needed for the sample to satisfy the rule of not selecting more that one child per household. Observe for a moment the file system above. This macro was driven by a stratum ID and a month and was to read in an unknown number of files. The number of files in each directory varied for different reasons. PSUs were released to the tracking system upon request. PSU holds were at times requested for a number of reasons such as field managers or staff needing to catch up on interviews, files needing to be retransmitted for various data problems, etc. The 'good' files in the system were expected to propagate to the CMS as normal.

Sampling was carried out by PSU within a stratum. This system was set up so that all PSUs in a file could be sampled by their availability. This implied that one can sample even if there was a failure to acquire all files (namely the PSUs) comprising a particular stratum. For example look at the following table:

Table 1

Files and PSU counts by Strata

Counts

Strata	Files	PSUs
Stratum 1	1	5
Stratum 2	1	5
Stratum 3	1	7
Stratum 4	1	7
Stratum 5	1	5
Stratum 6	1	8
Stratum 7	5	5
Stratum 8	1	12
Stratum 9	38	51
	50	100

Frame & Sample	Frame & Sample histories

Index Maximums

The counts shown here are maximum numbers that the user supplied to the run scripts. These scripts received the proper global information as arguments the user at program runtime. The Files Counts column indicates the maximum number of frames (or samples) to expect while the PSUs Count column indicates the number of history files that can be generated. Luckily, the counts never went much over 10 for either the frame or the history files. For most of the strata, except stratum 9, the files were invariably present for processing and entered together as one frame. PSU exit, on the other hand, was a completely different matter. Having the capability to release sample upon request made processing so much easier for the entire system.

Side note 1: There is a distinction between the frame (or sample) and the history files. The monthly frame was a hot register of data to be sampled and delivered. The history files were the same files as the frame (or sample) but from all months prior to the current month being sampled. Because the frame (or sample) contained all information (much of it redundant) about the current data it was trimmed down significantly for historical archive.

Side note 2: The Files Counts column represents an indexing integer that would be supplied by the user at run time and appended to the frame and sample file names upon entry to the system. Once the sample had been selected it was ready for release. A PSU Counts integer was also be provided by the user for historical archiving and delivery.

With that said the more detailed discussion regarding global information, scripts, and collecting the history files can now take place.

GLOBAL INFORMATION

Applying the global information to a program may seem like a trivial task. But, in creating an automated system this became an area where encapsulation of this task removed many programming headaches. So many errors and, even more, the devastating loss of system configuration could occur here even if one is careful. Global information is defined as the attributes that uniquely identify an instance of sampling. The global information, which described an instance for sampling looked like this template:

```
/* global template information */
%let strata        = _strata_;
%let month         = _month_;
%let frame_index   = _frameindex_;
%let release_index = _releaseindex_;
%let sample_type   = _sampletype_;
```

This global information preceded all programs as a heading to be used for sampling a PSU. A complete program run would look like this with the respective arguments:

```
> do_frame   REGION_1 SEP 1 1 SAMPLE_A
> do_sample  REGION_1 SEP 1 1 SAMPLE_A
> do_release REGION_1 SEP 1 1 SAMPLE_A
```

Each stratum represented a region in the U.S. transmitting data for the study. The month argument was the month that the file was time stamped for. Indexes frame_index & release_index defined how a frame entered and exited the system, respectively. The indexes became somewhat cumbersome to manage over time. This was due to the user having to manually probe the system to determine the correct indexes for the next round of sampling. Nevertheless, this was the method we chose and for the most part it was successful. SAMPLE_A, and later SAMPLE_B, were file attribute labels appended to the files. These samples are described in more detail in the PERIPHERALS AND ADDITIONS TO THE SYSTEM section under the OVERLAYING ADDITIONAL SAMPLE subsection.

SCRIPT 1: DO_RELEASE

UNIX scripts were utilized to generate SAS® programs and fill them with the user supplied global information. In regards to minimizing the code only one set of program code was maintained or touched at any time. A program would be filled with the global information, copied to the program's directory, and then run from that directory. The following script was just one in a series of run scripts used to run the sampling system from start to finish. Here is the do_release script:

```
#! /bin/csh

# name:    do_release
# purpose: to run the final step of
sampling.
#          create history files
#          and delivery files.

# error: check for correct number of args
if ( $#argv < 4 ) then

     echo "error[$0]: incorrect number of
args"
```

```
# run final step of sampling
else
     # set up output directory path name
     set outdir=$2/$1

     # set up path/program name
     set program=$outdir/release$3

     # set up sed substitution string
     set cmd = ('s/_strata_/'$1'/g;
                 s/_month_/'$2'/g;
                 s/_frmidx_/'$3'/g;
                 s/_histidx_/'$4'/g')

     # insert global inputs,
     # create program and file it
     sed $cmd templates/release.tmp >!
         ../$program.sas

     # run script arguments, output
     # output directory and program name
     runprog ../$outdir release$3$4

     # script2 args: stratum id, month
     countfiles $1 $2

     # copy result and
     # save it in the work directory
     cp ../$program.sas work/finalqc.sas
     cp ../$program.log work/finalqc.log
     cp ../$program.lst work/finalqc.lst

endif
```

The other scripts do_frame and do_sample follow similarly in their processing logic. The following is an example of how the global template was generated by the scripts into program code headings:

```
/* global information */
%let strata        = REGION_1;
%let month         = SEP;
%let frame_index   = 1;
%let release_index = 1;
%let sample_type   = SAMPLE_A;

/*  other initializations computed */
/* from the globals information    */

/*  include libraries, etc.  */

/*  code here  */
```

These run scripts removed many burdens from system management:

- One set of code was maintained minimizing handling.
- Programs were automatically filed by the scripts.
- Fewer debugging layers.
- Handled multiple users running several different sampling tasks each.

SCRIPT 2: COUNTFILES

How many files are in a history from month to month? Suppose there is one file in the Frames directory and one file in the Frames History directory. The following UNIX script is run at the release stage of the system:

```
#! /bin/csh

# name:    countfiles
# purpose: count the number of files in
the
#          month and stratum directory

# error: check that 2 fields are provided
if ( !( $#argv == 2 ) ) then

    echo "usage[$0]: incorrect number of
args

# create files counts
else
    # set up directory variables
    programs = programs/$2/$1
    frames   = frames/$2/$1
    fhist    = frameshist/$2/$1

    # frame/sample file count for month
    echo "%let frame"$2"="`ls
                ../$frames/frame*.ssd01
|
                awk '{print NR}' |
                tail -1`";" >!
                ../$programs/fbatch.sas

    # history file count for month
    echo "%let fhist"$2"="`ls
            ../$fhist/$2/$1/fhist*.ssd01
|
            awk '{print NR}' | tail -
1`";" >>
            ../$programs/fbatch.sas
endif
```

This script insured that the counts drawn for a particular directory were accounted for and up-to-date. The following 2 lines of code would be generated by this script into a file. That file would then be included during the gathering of a history file:

```
%let framesep = 1;
%let fhistsep = 1;
```

COLLECTING THE HISTORY FILES

Once all count variables were defined gathering a history was the next challenge. A valid list of all possible sampling months was defined in a single list, ALLMONTHS. The following code created a substring of all months to be gathered for a particular history file:

```
%let ALLMONTHS = SEP … DEC JA2 … FE2;
```

```
%let idx
=%eval(%index(&ALLMONTHS,&month)-2);

%let monstr = %substr( &ALLMONTHS, 1,
&idx );
```

Now that the list of months was defined the process of including the files created by the script COUNTFILES could begin. The collecting of the file count macro variables began as shown in the following loop below:

```
/* include all file count macro variables
*/
%let nummon = %numtok( &monstr, ` ` );
%do k = 1 %to &nummon;

  %let mon = %scan( &monstr, &k );
  %include
"&path/&month/&strata/fbatch.sas";

%end;
```

The file count macro variables were now included. Setting all the files together by looping through each month and each of the files in the directory took place as follows:

```
/* history = framehist or samplehist */
data &history;
    set
    %do i = 1 %to &nummon;
       %let mon = %scan(&monstr,&i);
       %let num = %getivar(&&fhist&mon);

       %if &num > 0 %then
       %do j = 1 %to &num;

           &fhlib&mon..fhist&j

       %end; /* j loop for files */

    %end; /* i loop for months */

/* site specific information here */

run;
```

Once the system was in place a macro call to read the history was invoked as follows:

```
/* gather all frame history files for all
*/
/* months prior to the current month
*/
%gethist(framehist)

/* gather all sample history files for
all */
/* months prior to the current month   */
%gethist(samplehist)
```

USEFUL FUNCTIONS AND MACROS

MACRO: GETIVAR

This was quite useful when a macro variable had been initialized to a blank. This occurred often if no files existed in a directory.

```
/* adds 0 to the macro variable     */
/* and returns the number           */
%macro getivar(invar);

%do; %eval(&invar + 0) %end;

%mend getivar;
```

MACRO: NUMTOK

```
/* returns the number of delimited */
/* tokens from a given string      */
%macro numtok(instr,xdelim=%unquote(`
`));

%let num = 0;
%do %while( %scan(&instr,&num,&delim) >
"" );
        %let num = %eval(&num + 1);
%end;

&num

%mend numtok;
```

MACRO: XMOD & X1MOD

```
/* returns the number of      */
/* times denom divides numer  */
/* returns values: 0..denom-1 */
%macro xmod(numer, denom);

%do;

    %eval( &numer - %eval( %sysevalf(
           &numer/&denom, floor) *
       &numer) )

%end;

%mend xmod;

/* returns the number of     */
/* times denom divides numer */
/* returns values: 1..denom  */
%macro x1mod( numer, denom );

%let modval = %xmod( &numer, &denom );
%if &modval = 0
%then %do; %eval( &denom ) %end;
%else %do; %eval( &modval ) %end;

%mend x1mod;
```

MACRO: SCAN (AVAILABLE IN SAS/MACROS®)

```
/* SAS®built-in function. Returns the */
/* Ith token from a string arg    */
%scan[1]( &ALLMONTHS, I <,delimiter >)
```

PERIPHERALS AND ADDITIONS TO THE SYSTEM

The sampling system described throughout this paper was but one part of a larger effort to collect data from state agencies participating in the study and to deliver a sample to the CMS. Except for the CLEANING THE DATA section each of the following components was highly based on the ability to gather a history file. Here is a brief description of some of those other components:

CLEANING THE DATA

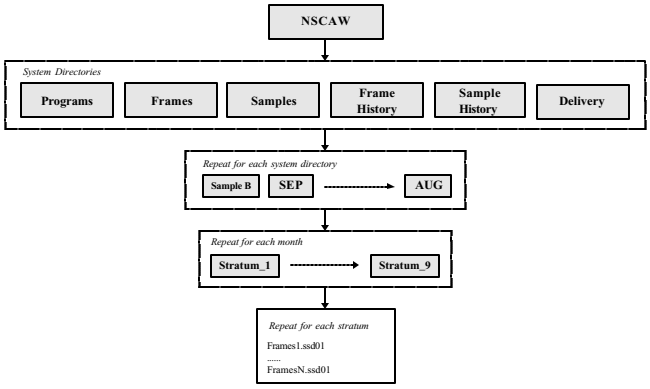
A frame preprocessor existed in the form of another application that statistical programmers used to edit data for frame entry. This was a mostly automated, and in its own right, a sophisticated system. Itincluded decrypting the data received from the agency sites, formatting it into a SAS dataset, QC checks to insure data integrity, adding our sampling system time stamp information from the agency's time stamp, and transferring it to the sampling system.

SUBSAMPLING A PSU

This item came in the form as a need after the system was in place. Once the study started some field staff became inundated with interviews for various reasons. The solution was to preserve the sample and system design by cutting back on sample delivered to the field but not sample drawn. Basically the sample process was repeated with a few modifications: a smaller set of sample sizes was used and a sample file was not generated since one already exists. This component was automated within our system and only generated a list of IDs that were delivered to the CMS. In essence the system remained unchanged by this addition.

OVERLAYING ADDITIONAL SAMPLE

SAMPLE_A can be described as a report of children (and their families) receiving a set of services from a site agency. SAMPLE_B can be described as a report of children (and their families) receiving a particular services from a site agency. SAMPLE_B was actually an *a priori* need. Luckily, the additional sample was needed only once in the lifetime of the system. But, how it was achieved also brought a few challenges. Because each agency had different capabilities and availability of data it was impossible to depend on SAMPLE_B being delivered at the same time for every PSU. Basically SAMPLE_B was drawn when data became available for whatever month that might be in. A level of complexity was now introduced and there was no turning back. Once SAMPLE_B was to be drawn for a particular month there would not be a SAMPLE_A drawn for that month. Clearly we are now dealing with a missing data issue as far as programming goes. This was anticipated and accounted for in the file count variables being generated by the scripts. However, we did encounter some problems as far as bookkeeping goes. The system had to be modified to differentiate between the new sample tag, SAMPLE_B, and the actual month that sampling was occurring. At various times in the system the name of the sample had to be used as a month for output. On the other hand the actual name of the month was needed for history retrieval and file time stamps. You can see how these dual meanings can cause inconsistencies and confusion in maintenance. Despite any problems encountered in adding this additional sample SAMPLE_A was to continue to be drawn monthly for sites that were not providing SAMPLE_B data. This was definitely one of the most challenging aspect to the sampling system. The following graphic show how the system changed due to SAMPLE_B.



MONITORING UTILITIES FOR SPECIAL REPORTS

This item came in the form as a desired quality once the system was in place. Over time tracking the data became unwieldy and the need for reports was a must. These reports were provided mostly for field staff to have an idea of what was to be anticipated as far as workload was concerned. Reports were added to flag when PSUs were not generating a sufficient amount of sample. From month to month sample sizes became a concern in the design affect.

SAMPLE SIZE OPTIMIZATION

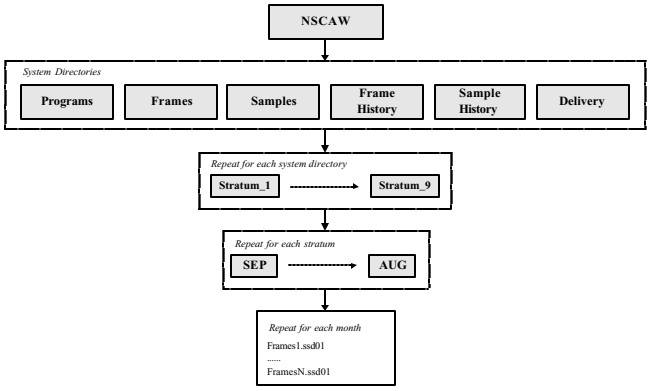
A certain amount of sample was needed per month. What was actually received and drawn was a different story. Optimizing this was somewhat of a dual effort. The biggest effort to controlling this was done externally as an independent system. Known proportions of data were used to gauge what was to be expected in the form of the design effect; required sample sizes were then produced by PSU and domain. In a lesser effort the actual sample sizes were then adjusted by data availability.

MAINTAINING IDS

From month to month each record released from the sampling system was given an ID for entry into the CMS. These sampling IDs were maintained by storing the last index used to generate an ID. That last PSU ID was then incremented and made the starting point for the next round of sampling. The last index used during sampling was then stored back upon release. This was, at one point, the source of many problems in the system because of the dual meaning of the sample type with the month. A problem created by the introduction of SAMPLE_B.

FUTURE CONSIDERATIONS

1) Change the directory structure by swapping the level of the stratum ID with the month. This will force only the month variable to be functional in history retrievals reducing a level of complexity in system management and thus somewhat more manageable. The new directory structure is illustrated as follows:



2) Create a way to generate macro variables to represent the list of files that exist in a directory instead of the counts. This would save macro variable names from being exhausted over time and reduce the loop by an order of magnitude. The exercise would then be to append the lists of file names. The problem case of no data would go away after the PSU was sampled once.

```
/* this convention would change */
%let framesep = 2;
%let fhistsep = 2;

/* this would then be generated */
%let framesep = frame1.ssd01
frame2.ssd01;
%let fhistsep = fhist1.ssd01
fhist2.ssd01;
```

3) Automate the frame and release indexes entered by the user. The problems continually faced that prevented anything of this nature were events such as bad data, no data, and bugs early in the system. Quite simply this was seen as a “like to have” feature rather than a need. A proposed solution however would be to create a mechanism to probe the data for file counts and maintain the indexes similar to how the sampling IDs were maintained.

REFERENCE

1: SAS® Macro Language: Reference, First Edition, 1997.

TRADEMARK INFORMATION

ACKNOWLEDGMENTS

The National Study of Child and Adolescent Well-Being was sponsored by the U.S. Department of Health and Human Services, Administration for Children, Youth and Families (June 2001). National Survey of Child and Adolescent Well-Being: State Child Welfare Agency Survey: Report. Washington, D.C.

Graphics design: Amanda Lewis-Evans, RTI

Team members that made this system a success:
Margie Byron, RTI
Larry Campbell, RTI
Sutapa Das, RTI
Ruby Johnson, RTI
Jun Liu, RTI
Suzanne MacDonald, formerly of RTI
Lisa McBroom, RTI
Amanda Lewis-Evans, RTI
Brian Sutton, formerly of RTI

CONTACT INFORMATION

Gabriel Cano
MCNC Environmental Modeling Center
P.O. Box 12889
3021 Cornwallis Road
Research Triangle Park, NC, 27709-28889
Work Phone: (919) 248-4110
Fax: (919) 248-9245
Email: gcano@mcnc.org
Web: www.emc.mcnc.org