A Central Data Storage and Reporting System for Statewide Local Health Dept Visits in SAS

Authors: Bruce Nawrocki, Kathy Dail and Julie A. Walker

Background

The North Carolina (NC) Department of Public Health was searching for an alternative to an expensive third-party database system that could import data and produce reports on Local Health Dept patients, visits and services provided. Our home-grown system would have to import column-specific TXT files from disparate Electronic Health Record (EHR) systems, provide feedback on data errors, and create reports on the data collected.

Implementation

We chose to use our existing SAS software running on Windows. First, we set up an object-oriented dataset design. Then we created TXT import file formats and got buy-in from the EHR vendors to produce these files for us. Next step was to write SAS code to import these TXT files, check for errors and produce feedback, all while keeping data privacy issues in mind. We gathered all the SAS programs (import/error-checking/feedback) into one SAS Enterprise Guide (EG) process flow.

Results

After creating the new system in SAS, we now have more accurate data and fewer data errors, since we catch errors before they are imported to our datasets. We can also provide improved error feedback and error descriptions to the counties via email, which encourages them to send their data to us more accurately. State-level program managers can report on fresher data.

Choice of Data Processing and Storage Software

We didn't have any licensed database software, nor did we have IT support (nor a Database Administrator) to set up a database. We also didn't have the budget for additional software.
So, we went with the software we already licensed and knew – SAS. For our purposes, SAS datasets resemble databases in many ways. They both store metadata along with the data – that is, field names and other attributes. They both support searching for records, updating existing records, and adding new records. They both support the concepts of primary keys and foreign keys.

Inputs

NC's public health departments and the electronic health records of their patients are decentralized. By law, the state cannot require the Local Health Departments (LHDs, of which there are nearly 100 of them) to use specific Electronic Health Records (EHR) software vendors. Thus, each LHD chooses their own EHR software vendor and implementation partner, who spends time to install and customize, among other things, the data-entry forms into which they collect patient and patient-visit information.

The North Carolina Department of Health and Human Services (NC DHHS) has certain Federal reporting requirements for the LHD data. And so, we can require LHDs to send us monthly updates, showing details of patient services they have provided, and can determine the file format of these updates. NC DHHS staff held multiple meetings with the LHD vendors, describing the file format requirements.

Our SAS Datasets

We decided to use an "object-oriented" approach to our SAS datasets. There are nine objects we track. But to simplify things for this paper, let's focus on these three objects, each saved into its own SAS dataset:

Details about each LHD patient are stored in the **Patient Master Record** (PM).

Details about each patient's LHD visit are stored in the **Service Record** (SR). Each patient may have 1 or more service records for a given service visit date.

If a patient received family planning services, there would also be a **Family Planning Record** (P1), in addition to the service record. We added this dataset because there are a lot of additional Family Planning Service-specific fields we capture for Family Planning visits – too many to add to the Service Record itself.

Input Files

The files we receive from each LHD vendor are in the form of column-specific text (.TXT) files, where field values are within a range of columns. For example, Patient Last Name must reside within columns 10-50 of a "Patient Master" record.

We requested that each LHD vendor send us all these records' information within a single TXT file each month. So, Patient Master (PM) data would reside in the same file as their Service (SR) records, etc. The first two characters of each row determine the record type.

For example, an incoming TXT file might look like this:

```
PM  123456789 JONES    VANESSA  . . .
SR  123456789 12-11-2022 service-code . . .
SR  123456789 12-15-2022 service-code . . .
P1  123456789 12-15-2022 family-planning info . . .
```

Secure FTP Server

Each LHD (or their EHR vendor) uploads one or more of these TXT files regularly (once or twice a month) to our secure FTP server.

Each LHD is given a three-digit number, starting with 001. We chose a three-digit number (stored in SAS as a character field) because there might potentially be more than 100 (but less than 1000) LHDs. . Within each LHD's folder there are 4 sub-folders. The folder structure of our FTP server looks like this:

LHD Folders	Sub-Folders
001	> Upload to State > In_process > Processed > Download from State
002	> Upload to State > In_process > Processed > Download from State

LHD personnel can only their view their own LHD-numbered folder and its sub-folders. They cannot see other LHD-numbered folders. The LHDs upload their TXT files to their own "Upload to State" folder, and later retrieve any Excel error files from the "Download from State" folder. "In_process" and "Processed" folders are for internal use.

Processing Steps

We process the TXT files we receive via a SAS Enterprise Guide project flow named **ProcessRecords.egp**, which looks like this:

0. Setup

This step sets up some global macro variables – such as our FTP server's address and user account, folders where we will copy any .TXT files downloaded from the FTP site, and where we store "in-process" .TXT files.

We also define our SAS work library location there, so when we build temporary files containing lists Microsoft Windows commands, SAS can find those in our SAS WORK library: `%let wpath=%sysfunc(pathname(work));`

This step displays a parameter screen, so we can enter the LHD number whose files we plan to process (which is then saved to &LHD macro variable) and point to the location of our LHD .TXT and output files, such as P:\LHD_data.

1. Move files on FTP

This step consists of SAS code sending SSH commands to our FTP server. These commands just move one LHD's incoming .TXT files to our from the "Update to State" folder to the "in_process" folder on the FTP server, so we don't accidentally re-process them later.

```
 * Go to remote folder cmd;
put "cd ""/&LHD/Upload to State""";
* Move all files to in_process folder;
put "mv *.* ""/&LHD/in_process""";
```

These SSH commands are sent to our FTP server for execution via Pageant.exe, an SSH command processor that's part of the free PuTTy software suite. SAS talks to Pageant via a private key file loaded into the computer's memory before the SAS Enterprise Guide process starts. This avoids having to enter the secure FTP server password in either the SAS code, or at runtime.

2. Download Files

More commands are generated and sent to our FTP server for processing. This time, on our local Windows computer, we change to our "infileDir", then copy the "in_process" .TXT files from FTP server to our local Windows server.

```
put "lcd ""&infileDir""";
* Go to remote folder cmd;
put "cd ""/&LHD/in_process""";
* Move all files to in_process folder;
put "mget *.TXT";
```

3. ProcessVendorRecords.sas

This step is where most of the SAS data processing takes place. This calls an external SAS code file (designated by curly arrow in left-bottom corner of its step icon above) which processes records (rows) of the TXT file in this sequence:

- Patient Master Records
- Service Records
- Family Planning Records

This step also generates these items, providing feedback for the LHD personnel:

1. Excel files containing detailed error reports. These files are uploaded to our secure FTP server, where authorized LHD personnel can download and view the data. We send these files via FTP because they contain personal data, so we cannot send it via email.
2. An email showing a summary of processing results for their TXT file, which looks like this:

District/County: 043 - HARNETT

File: TEST043.txt

Date Processed: 09/13/2023 4:03 PM

Record Type	Records Entered	Records Accepted	Errors Found	Records Rejected	Percent Rejected
P1 (FP)	10	1	9	9	90.00%
Patient Master	10	10	0	0	0.00%
Service Record	10	10	0	0	0.00%

These are the results from the file you uploaded.

If there were Records Rejected, error reports were sent to the FTP server (in 'Download from State' folder) - one error file for each Record Type.

It is your responsibility to fix these errors, and then upload the fixed records to us.

You can read more about this SAS code in the **ProcessVendorRecords.sas Details** section below.

4. Upload Error Files

This step uploads the error Excel (.xlsx) files, if needed, to our FTP server.

```
put "lcd ""&errfileDir""";
* Go to remote folder cmd;
put "cd ""/&LHD/Download from State""";
* Move all files to Processing folder;
put "mput *.xlsx";
```

5. Cleanup FTP Files

This step moves the .TXT files from "in_process" to "processed" folder on FTP server.

```
put "cd ""/&LHD/in_process""";
* Move all files to in_process folder;
put "mv *.* ""/&LHD/processed""";
```

6. Cleanup Local Files

This step calls a Windows command to move any error files to an archive folder for backup:

```
x "move ""&errfileDir\*.*"" ""&errfileDir\archive""";
```

ProcessVendorRecords.sas Details

The first part of this SAS code sets up the LIBNAMES and various macro values used, such as primary keys for each SAS dataset. Each dataset is kept sorted by these fields:

```
** Primary Keys - sort vars;
%let primKeyPM = county local_patient_id;
%let primKeySR = county local_patient_id service_date service_code;
%let primKeyFP = county local_patient_id service_date;
```

The next section of code counts how many TXT files an LHD might have sent to us (sitting in our &infileDir), and assigns the filenames found to macro values &read1, &read2, etc.

```
%let command =%unquote(%str(%')dir "&infileDir.\*.txt" /b%str(%')) ;
filename DIRLIST pipe &command;

* Create a SAS dataset with filenames found;
data dirlist ;
   infile dirlist lrecl=200 truncover;
   input file_name $100.;
run;

* Create macros named read1,read2,etc. (one for each file found) and numFiles
(contains number of files found);
*  read1 is fully qualified file name (incl. dir) of input file from vendor;
data _null_;
   if nobs=0 then do;  * Special case where dirlist is empty;
      call symputx('numFiles',0); * Set to 0 readin macro loop will not run;
      stop;
   end;
   set dirlist nobs=nobs end=eof;
   count+1;
   call symputx('read'||put(count,4.-L),cats("&infileDir.\",file_name));
   if eof then call symputx('numFiles',count);
run;
```

We then call a %readin macro, which sets up a big loop for each TXT file, and processes each record type in sequence, then saves TXT files to backup folder. (Note: some of the original code has been skipped below, as signified by . . .)

```
%macro readin;
   * The DO loop is run for each file, and processes the record types in each file in
sequence;
   %local i; * Define i as a local variable, so it does not overwrite any i value in
any calling code;

   %do i=1 %to &numFiles;

      %let timestamp = %unquote(%sysfunc(datetime(),mydtfmt.));

      ** Set up macros, such as FileReportedOn, which is the filename portion of the
      full directory path in DirFileName;
      . . .
      ** Before processing the file, delete any existing loadstat records from today,
      to avoid duplicate records;
        . . .

      %processPatientMasterRecs(DirFileName=&&read&i, primKey=&primKeyPM);
```

```
      %processServiceRecs(DirFileName=&&read&i, primKey=&primKeySR);

      %processFamilyPlanningRecs(DirFileName=&&read&i, primKey=&primKeyFP);
      . . .

      %sendEmailConfirmation(DirFileName=&&read&i);

      * Copy the input file to a backup (processed) folder ----;
      options noxwait; * Automatically return to the SAS code after running an MS-DOS
      command. No need to type EXIT in MS-DOS window;
      data _null_;
      outFile = tranwrd(%unquote(%str(%')&&read&i%str(%')), "\in_process\",
   "\&processedSubDir\");
      call symput('outFile', trim(outFile)); * Convert data step variable to macro
   variable;
      run;
      ** We need another data step because macros defined in the data step above are
   NOT available until data step ends;
      data _null_;
        %let command =%unquote(%str(%')move "&&read&i" "&outFile" %str(%')) ;
        rc = system(&command);
        put rc=;
      run;
   %end;
%mend readin;
```

Each of the "%processXxxRecs" macros (referenced above, but not shown) follows the same general structure. For example, %processPatientMasterRecs performs these tasks on the Patient Master (PM) records, where &LHD is the LHD number, such as 043:

1. Create two empty SAS datasets – validPM_&LHD and errorPM_&LHD
2. Call a validate macro, such as `%ValidatePatientMasterRecs`, that runs a series of validation checks against that record's data.
3. If a record contains any errors, then these errors (including error-type and error-message) are written to SAS dataset errorPM_&LHD. The final version of this dataset is saved to an Excel file. These records are NOT loaded to our master file.
4. If a record contains no errors, the record is written to validPM_&LHD
5. After looping through all PM records, if validPM&LHD contains any records, we sort it by primary key (in this case &primKeyPM), then update our "master" production PM SAS dataset with these new records:

```
data prod.PM_Master;

   update prod.PM_Master validPM_&LHD  updatemode=nomissingcheck;

   * By default, missing values in a transaction record do NOT replace existing
values in the master data set. Nomissingcheck indicates we DO want missing
values in transaction file to replace existing values in master;

   by &primKey;

run;
```

This code does several things for us – for each new PM record in validPM_&LHD, it compares its values of the primary key fields to those of the existing records in prod.PM_Master. If the primary key is not found in the production file, the new record is added to the production file. If the primary key already exists in production, it updates that production record's fields with the field-values from the new PM record.

For us, this is a necessary check, because sometimes patient records are evolving. An LHD may send us a partially filled-out patient record, and then later may send us an update on that same patient record, with more optional-entry fields filled in.

Implementing Foreign Keys

Our sequence of processing record-types is important because we must ensure that:

- Patient Master (PM) records are processed and imported to production first. Without a PM record in our production file, we cannot load any other record types for that patient.
- Service (SR) records are processed next. We cannot accept or load an SR record before we have a corresponding production PM record for that patient. If that happens, we flag the SR record as an error.

We ensure that we don't load a Service record without an existing Patient Master record via the use of foreign key lookup. That is, before we load the Service record, we look at its values of the Patient Master's primary key fields (County and Local Patient ID), and check that a Patient Master record already exists with those key fields.

So, while the %processServiceRecs macro generally follows the 4 steps outlined in %processPatientMasterRecs above, it must also add this additional check between steps 2 and 3:

```
if recCount = 1 then do; ** Hash only needs to be defined once;
  declare hash lkup(dataset: 'prod.PM_Master');
  lkup.DefineKey('County','Local_Patient_ID');
  lkup.DefineDone();
end;

if lkup.find() ne 0 then do;  ** Record was NOT found in hash (PM_Master) table;
  Error_Data = catx('/', County, Local_Patient_ID);
  Error_Number = 'SR000';
  Error_Message = 'Service Record requires existing Patient Master record:
County/Local_Patient_ID';
  Record_Has_Errors='Y';
  output errorSR_&LHD;
end;
```

The first time through this DATA step, the code above declares a hash called LKUP, referencing the PM Master SAS dataset. It also defines the (primary) key for that dataset. We are not returning any data from that dataset (we're only using the hash to tell us if a primary key is found), so we don't need to add a lkup.defineData() statement.

Each cycle through this DATA step, we call lkup.find(). If the return code from find() is not 0 (zero), we flag an error, meaning the currently in-process Service Record's values for County and Local_Patient_ID are NOT FOUND in the PM Master SAS dataset.

We run a similar hash lookup for most of our other record-type processing, such as in %processFamilyPlanningRecs.

Outputs

The SAS Enterprise Guide process creates these outputs:

- Email to LHD personnel, showing processing results summary
- Error report in Excel format, showing primary key fields of any records containing errors, along with an error code and error description, which should help LHD personnel to find and fix these issues in their own EHR system, before attempting to resend these records back to us.

Since the production data is stored in SAS datasets, we can use our in-house SAS expertise to analyze it, writing annual (or quarterly or monthly) reports showing a statewide number of LHD patients and services provided. These reports are helpful for both internal use and Federal reporting requirements.

CONTACT INFORMATION

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

Bruce Nawrocki
North Carolina Division of Public Health, Injury and Violence Prevention Branch
5505 Six Forks Rd, Raleigh NC 27613
E-mail: bruce.nawrocki@dhhs.nc.gov