

How to Unravel Hierarchical Data Using SAS®

Maxine Heinitz, U.S. Food & Drug Administration, Minneapolis, and

Sue Hakomaki, Computing Information Services, University of Minnesota

Abstract

Many companies are faced with converting data from specialized systems into a form compatible with modern, relational structures. Hierarchical data consists of a header record and detail records. This paper discusses the application of SAS® programming to unravel the hierarchy into a form which may be imported into relational database tables.

The trailing @ was used to evaluate whether the record type is header or detail. Specialized processing unique to FDA's data involved use of the RETAIN statement while reading in the header to convert the record identification number to one which related to 'sample' numbers used by Agency-wide systems. Procedures were applied to the detail records to write those details to the header. Tips for efficient processing and debugging will be discussed.

What is Hierarchical Data?

Many files, particularly files from legacy systems, are *hierarchical* in structure, consisting of

- ☐ a header record
- ☐ one or more related detail records.

FDA uses a specialized system to collect detailed information on illness-causing bacteria in foods. SAS was applied to process the data into a form which could be merged with records from data in a relational database structure. Typically, the files contained about 100,000 records per year over a period of twenty years. Efficient processing was essential since the specialized system continues to generate new data files which are accessed repeatedly. Other industries, such as insurance or pharmaceutical, may also be faced with processing hierarchical files.

This paper discusses one way to process a hierarchical file using SAS. Each record contained a field which could be used to identify whether it was a header record or detail record. A series of hierarchical ASCII files were read into SAS data sets by creating one observation for each header record, one observation for each detail record. The trailing @ in a DATA step was used to evaluate the record type - header or detail - and check for errors. Subsequent data steps read the detail records and created data sets. The data sets were merged based on the unique record number. The detail records were processed using Proc steps and the appropriate details were stored as part of that header observation. PUT statements were used to create ASCII files which could be used for importing into relational database tables.

Raw Data Example

In the example shown in Appendices 1 and 2, the file contained a record number unique for each record in columns 2-11 (variable called STRAIN). Descriptive data about the food sample such as product, geographic origin, date, and summary data on the type of laboratory analysis performed were reported in the header records. (Was the product positive for presence of *Salmonella*, *E. coli*, etc?) Detail records reported the detailed laboratory observations. For example, Was *Salmonella* isolated? If yes, what species? How many *E. coli* bacteria were found in this isolate?

Data Step 1

Here are the tasks SAS will perform in the first data step:

- ☐ Read a record, assign a value to a variable TYPE.
- ☐ Use the trailing @ to evaluate whether the record type is header or detail.
- ☐ Conditionally read data from the header records, check for errors, and perform other, specialized processing.
- ☐ Create a temporary SAS data set.

The code shown below is simplified to better illustrate to basic concepts of processing hierarchical data. In the original code the SAS System constructed other variables based upon the value of the variable IMPORT and other variables, which is beyond the scope of this paper.

```
libname mylib 'd:\data\saslib';

data one;
  attrib
    first      length=$1  label='1st digit of import number'
    sample     length=$10 label='FDA sample number'
    rec_id     length=$6  label='6 digit rec_id no.'
    isolate    length=$4  label='sub/comp no.'
    strain     length=$10 label='Micro-IS strain no.'
    import     length=$1  label='sample is an import'
    prod       length=$20 label='product description'
    geo        length=$20 label='country of origin or US ZIP';

  infile 'd:\data\mis94\path94a.txt' lrecl=50 pad missover;
  retain rec_id isolate import;
  input type $ 8-11 @;

  if type in ( ' ', 'A', 'C' ) then do;
    first = ' ';      rec_id = ' ';
    isolate = ' ';    import = ' ';

    input      first $ 1      rec_id $ 2-7      isolate $ 8-11
              strain $ 2-11  import $ 13      prod $ 17-43
              geo $ 45-69;

  end;

  if type ^ in ( ' ', 'A', 'C' ) then do;
    input      first $ 1      rec_id $ 2-7      isolate $ 8-11
              strain $ 2-11  import $ 13;

  end;

  .....more SAS statements.....

run;
```

tip 1

Use the ATTRIB statement to document your SAS code. Detailed variable description are helpful to someone who 'inherits' your code or when adding later enhancements.

The RETAIN statement causes a variable created in the DATA step to retain its value from the current observation into the next observation rather than being set to missing at the beginning of each iteration of the DATA step. It affects variables created in the current DATA step (for example variables created with an INPUT or assignment statement). For example, in this case we needed to read in certain header records in a particular way if the value for the variable TYPE was equal to ' ', A or C. If it wasn't, we needed to read in the record with the RETAINED values for variable REC_ID. SAS set up the PDV with REC_ID retained from

one observation to the next unless specifically told otherwise (as in the first scenario where TYPE was equal to ' ', A or C); in which case the INPUT statement was different.

tip 2 Use Proc CONTENTS and Proc PRINT for each data set created while building your code. They are useful tools for debugging and help verify SAS is processing the hierarchical data as you intended.

Reading Detail Records

Here are the tasks SAS will perform in the next data steps:

- ☐ Read all records, treating them as detail records. Create a temporary data set.
- ☐ Combine data sets into a permanent data set, matched BY the unique value of STRAIN.
- ☐ Count the number of isolates having XSALM=+ by each record of REC_ID using Proc FREQ and create a new variable COUNT.
- ☐ Check for and eliminate duplicate records by values of REC_ID.
- ☐ Check for and output records having errors.

```
data two;

  attrib      strain      length=$10  label='Micro-IS strain number'
            xsalm      length=$1    label='Isolate is Salmonella positive'
            organism    length=$15   label='Genus & species'
            ecoli       length=$1    label='E. coli-3 tube positive'
            ec3_mpn     format=9.1   label='E.coli 3 tube MPN/g';

  infile 'd:\data\mis94\path94b.txt' lrecl=50 pad misover;
  input strain $ 2-11      xsalm $ 17  organism $ 16-30
        ecoli $ 33        @36 ec3_mpn 9.1  ...additional variables;

run;
```

tip 3 Use SAS CONNECT and TCP/IP connectivity to access the raw data on the host system and process using PC SAS. The infile statement would be specific for the host operating system.

```
proc sort data=two;
  by strain;
run;

data mylib.fy94path (label='1994 Salmonella');
  merge one two;
  by strain;
run;

*Count number of isolates having XSALM=+ and output dataset bb;
proc freq data=mylib.fy94path;
  where xsalm='+';
  tables xsalm / out=bb(drop=percent) noprint;
  by rec_id;
run;

data cc;
  merge mylib.fy94path bb;
  by rec_id;
run;
```

tip 4 To conserve computing resources, only use Proc SORT when necessary.

```

proc sort data=cc nodupkey;
  by rec_id;
run;

* Check for errors in Salmonella data;
data badata;
  set cc;
  if (xsalm = '+' and organism = ' ') then output badata;

  .....more SAS statements.....

```

In this example, SAS checked for errors in the raw data. Data set BADATA was printed and the reports were sent to the original laboratory for correction of the file. Techniques for reduction of other types of detail records and other types of hierarchical files will be discussed.

Write to an External File

Here are the tasks SAS will perform in the next step:

- ☐ Read data in a data step.
- ☐ Point to an external file to which the PUT statement will write.
- ☐ Indicate how to write the lines of data using column pointer controls.

Note in the example below a delimiter was used between the columns of data. The delimiter may not be necessary, depending upon the relational database software into which the data is imported.

```

filename xxx 'd:\data\output\salmpos.txt';

data _null_;
  set cc;
  file xxx;
  put @1 rec_id @8 ' , ' @10 count .....output other variables...;
run;

```

References

SAS Institute Inc.(1991), SAS® Language and Procedures: Usage 2, Version 6, *First Edition*, Cary NC: SAS Institute Inc.

SAS Institute Inc.(1992), SAS® Programming Course Notes, Cary NC: SAS Institute Inc.

SAS and SAS CONNECT are registered trademarks of SAS Institute, Inc., in the USA and other countries.

® indicates USA registration.

Contact Information:

Maxine L. Heinitz, Ph.D.
 U.S. Food & Drug Administration
 240 Hennepin Ave.
 Minneapolis MN 55401-1999

(612)334-4100 Ext 126
 BITNET: gvl@nihcu
 Internet: heinitzm@mwr65.mwr.ora.fda.gov

Sue Hakomaki
 Computing Information Services
 152 Shepherd Labs
 University of Minnesota
 Minneapolis MN 55455
 (612)626-8670
 hakomaki@uminn1
 hakomaki@vm1.spcs.umn.edu

Appendix 1: Header Data

```
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8-----+-----9-----+-----0-----
001000      - - BLUE CHEESE          53274      .....more variables.....
00100001
001000010
00100002
00100003
00100004
00100005
00100006
00100007
00100008
00100009
001010      - - VANILLA ICE CREAM    56159
0010101C15
0010102C15
001101      + - LOBSTER MEAT        CANADA
0011011C15
0011012C15
```

Appendix 2: Header & Detail Records

```
001000      + .....more variables.....
00100001      23.0
001000010     230.0
00100002      4.0
00100003     980.0
00100004     24000.0
00100005     9800.0
00100006     780.0
00100007     640.0
00100008     1200.0
00100009      47.0
001010      +
0010101C15
0010102C15
001101
0011011C15   + SLML NEWPORT
0011012C15   +
```