Partially Transforming Hierarchical Data Sets for Sequential Processing Using Arrays
Richard L. Downs, Jr. and Pura A. Peréz
U.S. Census Bureau, Washington, D.C.

ABSTRACT
This paper explains a simple framework for applying traditional sequential processes (reformat, edits, imputation, etc.) to a hierarchical group of SAS® data sets. Using a series of macros, you define the data set hierarchy or a subset of that hierarchy. The macros transform all data sets under the top level of the hierarchy, creating top-level views of these data sets. Inside the process data step(s) another series of macros build the hierarchy-related portions of the data statement, all array statements referencing variables from transformed data sets, and the merge and output statements that reference the hierarchy data sets. After the processing data step(s) a final macro reverts the transformed data to its original format and updates the original data set(s). This framework isolates the complexity of relating the data sets from the complexity of the actual process, it maintains the data set hierarchy, eliminates the need for any post-process processing (all output data sets are complete at the end of the processing framework), and allows access to data at peer levels of the hierarchy.

INTRODUCTION
Most U.S. Census Bureau demographic personal or telephone interview surveys have switched from paper questionnaires to electronic instruments run using either the Computer-Assisted Survey Execution System (CASES) or Blaise. We extract data from CASES or Blaise and, for most surveys, convert the output into one or more SAS data sets for processing. Processing varies from survey to survey, but usually it involves taking the output data through a series of sequential steps: reformat, edits, imputation, weighting, recodes, table production, and internal/external user file production. In the past, we processed on a mainframe environment running 3GL (FORTRAN) programs against one or more flat or hierarchical files. Redesigned processing is on Solaris/Linux workstations running SAS programs against a hierarchical series of data sets.

In this paper we explain and demonstrate a simple framework for applying traditional sequential processing steps to hierarchical data sets while largely isolating the complexity of the data set relationships from the complexity of the actual processes.

SURVEY OUTPUT DATA
Conceptually, survey output data is organized at the case level and in rosters. Case level information is usually information about the household. A roster is a repeating group of data items. Each roster is a “child” of either the case level or another roster; instruments have a case level and may have multiple roster levels. For example, if we have a survey that collects information on jobs, each interview has a case level with information about the entire household. Each household can have several persons, so we have information about each person stored in a roster at a second level. Each person could have several jobs, so we have information about each job stored in another roster at a third level, etc.

When we translate survey output to SAS data sets each repeating group becomes a separate data set. Depending on the complexity and size of a survey, the results can be upwards of 100 data sets in a hierarchy several levels deep. An example data set hierarchy taken from the Medical Expenditure Panel Survey (MEPS) is shown in figure 1.
The data sets are related and uniquely identified by common, key variables. We can see this by looking at a subset of the MEPS data sets from figure 1, shown in figure 2.

- The **household** data set contains a variable `ctrlnum` that uniquely identifies each household.
- **Persons** has a variable `ctrlnum` that links the person to the household; it also has a variable `persons` that uniquely identifies that person within that household.
- **Events** (in this example an event is a health care event, such as a doctor visit) has variables `ctrlnum` and `person` that link that event to a person; it also has a variable `events` that uniquely identifies the event for that person.

This framework assumes that, for data sets under the top-level of the hierarchy, the key variables will be numeric and valued 1, 2, 3, etc. on successive observations for every unique parent key value. For example, the first person in a household would have `persons=1`, the second person in the household would have `persons=2`, etc.

Relationships between the data sets are either one-to-one or one-to-many.

**REQUIREMENTS**

Our issue is how do we implement the traditional sequential survey processes using SAS given the data hierarchy? Our solution should allow us to maintain the hierarchy, largely isolate the complexity of relating the data sets from the complexity of the survey processing, eliminate the need for any post-processing at the completion of the processing framework, and allow access to data at peer levels of the hierarchy.

By maintaining the hierarchy, we require that you do not collapse the hierarchy into “one big file” by either amalgamating the data or creating temporary data sets. Amalgamating the data (creating a physical top-level representation) is wasteful because of the number of blank values created in each observation. Creating a temporary data set based on the lowest level of the hierarchy, in our MEPS example `events`, is also wasteful because of the number of values that repeat over multiple observations. Also, for larger surveys, transforming or creating a temporary data set to represent the input relationships may be impossible due to resource constraints.

By isolating the complexity of relating the data sets, we want you to have to build minimal relationship logic into their process step(s) code. Ideally you can reference the related data sets as if they are “one big file.” Also, you should have to build minimal output control logic into their process step code to create updated versions of the appropriate input data sets.

We want to eliminate the need for any post-processing. This means that all data sets produced by the processing data step(s) must be complete at the end of the framework.

Finally, we want to allow access to data at peer levels of the hierarchy. This means that we can not restrict our processing to one simple branch of the data set hierarchy. Rather, we require the ability to process a subset of the hierarchy or potentially the entire hierarchy at once.

**SOLUTION CONCEPTS**

A solution that meets all of our requirements is constructing top-level views of the data sets under the top-most data set in the hierarchy or a subset of the hierarchy, processing these views, and translating from any resulting transformed data set(s) back to the original data set hierarchy. By top-level view we mean a representation with one observation for each instance of the top-most data set in the hierarchy or subset of the hierarchy. Variables in the data sets under this top-most data set become arrays or multi-dimensional arrays. For example, if we look at the data set hierarchy shown in figure 2, a top-level view of the `events` data set has one observation per household instead of one observation per event. Variables in the `events` data set become multi-dimensional arrays in the transformed data set. We explain the process and details of creating a top-level view in the preprocessing section below. A top-level view solution is necessary for the following reasons:

- Given the nature of the data, a low-level view (based on the lowest data set in the hierarchy or subset of the hierarchy) does not allow for access to data at peer levels of the hierarchy.
- Only a top-level view guarantees simultaneous access to data at all levels of the hierarchy; using low-level views we have one data set observation from each data set associated with a top-level data set observation in the PDV at any one time.
Conceptually, our solution is a framework covering three general areas: preprocessing, processing, and postprocessing.

**PREPROCESSING**

Preprocessing consists of selecting a top-level data set in the hierarchy as our processing base, then selecting various data sets under the top-level data set in the hierarchy and creating top-level views of those data sets.

Figure 3 shows a subset of the data sets from the MEPS hierarchy shown in figure 1. The *household*, *persons*, and *events* data sets were described earlier. The *presmeds* data set contains information about a person’s prescription medications. The *pregna01* data set contains a person’s pregnancy information. In this example, the *household* data set is our top-level. Our framework then needs to create household representations of the *persons*, *presmeds*, *pregna01*, and *events* data sets.

We create these representations by transforming each data set under our top-level data set. Cody (1999) and Virgile (1999) provide good fundamental descriptions and examples of simple transformation using PROC TRANSPOSE and in data steps using arrays. These transformations take data sets with a single key variable and one or more non-key variables where there are one or more instances of a key value and output a data set with one observation per unique key value where each non-key variable is transformed into an array.

In our framework we have multiple data sets in a hierarchy. We do not transform the top-level data set (in figure 3 *household*) since it is already a top-level representation (hence we say the data is only partially transformed); this data set may have multiple key variables. We transform each data set under the top-level data set similarly to Cody and Virgile’s simple examples. We combine possible multiple observations with identical top-level key variable values into a single observation, transforming each variable on the data set into an array. For each data set more than one level below the top-level data set in the hierarchy we transform each variable into a multidimensional array.

To illustrate this concept further, let us look at how this transformation applies to the *persons* data set shown in figure 3. A subset of the variables from the *persons* data set is shown in table 1.

<table>
<thead>
<tr>
<th>obs.</th>
<th>ctrlnum</th>
<th>persons</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Rick Downs</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Pura Peréz</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>John Smith</td>
</tr>
</tbody>
</table>

Table 1

For this simple example, we will assume that there are no more than two persons per household. The transformed *persons* data set is shown in table 2.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Rick Downs</td>
<td>Pura Peréz</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td>John Smith</td>
<td></td>
</tr>
</tbody>
</table>

Table 2

Note from examining tables 1 and 2 that the persons and name variables each became arrays dimensioned 1 to the maximum number of persons per household in the resulting data. Also note that there are missing values in the second persons information in table 2 since there was only one person in the second household. The data shown in table 2 is a top-level (household) representation of the persons data set.

Let us look at another example, this time with multi-dimension arrays, of how transformation applies to the *events* data set in figure 3. Just the key variables from the *events* data set are shown in table 3.
Table 3

<table>
<thead>
<tr>
<th>obs</th>
<th>ctrlnum</th>
<th>persons</th>
<th>events</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For this simple example we will assume that there are no more than two persons per household and no more than two events per person.

Table 4

<table>
<thead>
<tr>
<th>obs</th>
<th>ctrlnum</th>
<th>events{1,1}</th>
<th>events{1,2}</th>
<th>events{2,1}</th>
<th>events{2,2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Note from examining tables 3 and 4 that the events variable becomes a two-dimensional array dimensioned from 1 to the maximum number of persons per household and from 1 to the maximum number of events per person. Also note the missing data values in table 4. The missing values are a result of the first person in household 1 having only one event, the first person in household 2 having only one event, and there not being a second person in household 2. The data shown in table 4 is top-level (household) representation of the events data set.

For the data set hierarchy shown in figure 3 the processing framework needs to do the following transformations:

1. **persons**: one observation per household with each variable becoming an array in the form variable\{person number\}.

2. **events**: one observation per household with each variable becoming a two dimensional array in the form variable\{person number, events number\}.

3. and 4. We process presmeds and pregna01 similarly to events.

**PROCESSING**

Processing consists of a data step or data steps that merge the top-level data set with the top-level views of the other selected data sets. The data step(s) output the appropriate hierarchical data set(s) and assign the correct variables to each data set.

In the data step(s), we reference variables from data sets under the top-level data set as arrays. The array names have the same names as the original data set variables. The arrays are indexed by the appropriate key variables for the data set, the data set's parent, and the parent data set's parent. For example, looking at the data sets shown in figure 3, a variable coming from events translates into an array variable\{persons number, events number\}. Events number is the key variable for the events data set and persons number is the key variable for events' parent data set.

**POSTPROCESSING**

If the processing data step updates any of the transformed data sets then the processing framework must revert the resulting transformed data to its original format. To illustrate this, again let us look at tables 1 and 2. Reverting the data back to its original format in the hierarchy is a reversal of the transformation described under preprocessing. In our example, the transformed data shown in table 2 becomes the original data set shown in table 1. Note from the tables that there is no observation in table 1 for the second person in household 2, since that persons number is missing (indicating no person).

For the data set hierarchy shown in figure 3 the processing framework would need to do the following reversions:

1. **persons**: one observation per person where the original person number is not missing with each variable set equal to the transformed variable\{person number\}.
② **events**: one observation per event where the original events number is not missing with each variable set equal to the transformed variable (person number, events number).

③ and ④ We process *presmeds* and *pregna01* similarly to *events*.

At first look our solution seems to violate our requirement against amalgamating the data. However, the way we implement the solution meets the requirement by:

- allowing you to specify a subset of data set variables for processing.
- creating the transformed data as SAS data views.
- optimally determining the maximum occurrences of each roster, and, correspondingly, each transformed data set array dimension.
- allowing you to specify data sets in the hierarchy as read only.

**SOLUTION IMPLEMENTATION**

We implement the processing framework using ten SAS macros. We show the macro calls in the following sections. Again, these macros cover three general areas: preprocessing, processing, and postprocessing. We list the complete source code for these macros along with several example programs showing how to use the various macros on the Internet at the following URL: [http://ptrans.dusia.com/](http://ptrans.dusia.com/).

**PREPROCESSING**

① `%PTRANS(data=, dim=, goodlist=, in=, number=, out=, parent=, postfix=, prefix=, readonly=, labels=, keyvar=)`

%PTRANS registers a data set with the processing framework.

**DATA=** *data set*  
Specify the data set name, complete with libname reference.

**DIM=#**  
Set the dimension for this level of the hierarchy to #; if # is blank then %PTRANS will determine the optimal # value. For example, if we know there is a maximum of 16 persons per household, then we would set DIM=16.

**GOODLIST=file**  
Specify an ASCII file with a subset of the variable names that will be processed, one variable name per line.

**IN=** *variable*  
Specify a variable that indicates whether the data set contributed data to the current observation.

**NUMBER=#**  
Specify the unique number that will be used to identify this data set in the hierarchy.

**OUT=** *data set*  
Specify the data set output name (if different from the input name), complete with libname reference.

**PARENT=#**  
Specify the data set number (see above) of the data set’s parent data set. For example, looking at the data set shown in figure 3, if we define persons as data set number 2, then the parent for *presmeds*, *pregna01*, and *events* is 2.

**POSTFIX=**  
Specify a postfix to be applied to the data set's variables.

**PREFIX=**  
Specify a prefix to be applied to the data set's variables.

**READONLY=y or n**  
Y to specify the data set as read only; N or blank to specify the data set is output.

**LABELS=y or n**  
Specify to create labels for the transformed views; a label is the equivalent array cell created by the %ARRAY macro. Specify Y to create labels. Specify N or blank to not create labels.

**KEYVAR=** *variable [variable...]*  
Specify the variable(s) that uniquely identify each data set observation within the universe of its parent data set;
Looking back to the data sets shown in figure 3, the following %PTRANS reference defines the events data set within the hierarchy:

```
%PTRANS (NUMBER=3,
  DATA=DEFAULT.EVENTS,
  PARENT=2,
  KEYVAR=EVENTS,
  GOODLIST=EVENTS.GL);
```

From this statement we can see the following:

- **Events** is registered as data set number 3.
- **Events** parent data set (persons) is registered as data set number 2.
- **Events** is uniquely identified within its parent’s universe by the variable events.
- A subset of the variables in events will be processed; those variables names are in the ASCII file events.gl.
- %PTRANS is to determine the optimal dimension for events (dim= not specified).
- The processing framework will output default.events (readonly= not specified).
- %PTRANS will not create labels for the transformed variables (labels= not specified).

%PTRANS generates the following PROC SQL code to determine the optimal dimension for events:

```
PROC SQL;
RESET NOPRINT;
SELECT MAX(COUNT) INTO :DIM3 FROM
(SELECT PERSONS,CTRLNUM,COUNT(*) AS COUNT FROM DEFAULT.EVENTS GROUP BY
PERSONS,CTRLNUM);
```

Finally, %PTRANS generates the following data step code that creates a top-level (household) view of the events data set. Note %PTRANS transforms the variables in events in to 2 dimensional arrays (by person number and event number). Also note that %PTRANS transformed the original event number variable.

```
DATA VIEW3(KEEP=CTRLNUM __19 - __30) / VIEW=VIEW3;
ARRAY EDATA{3,2} $ 2 __19 - __24;
ARRAY EVENTS{3,2} 8 __25 - __30;
DO UNTIL(LAST.CTRLNUM);
  SET DEFAULT.EVENTS
    (KEEP=CTRLNUM PERSONS EDATA EVENTS
     RENAME=(EDATA=COL1 EVENTS=COL2 ));
  BY CTRLNUM;
  EDATA{PERSONS,COL2}=COL1;
  EVENTS{PERSONS,COL2}=COL2;
END;
```

%NEWVAR (NAME=, TYPE=, LENGTH=)

%NEWVAR declares a new variable that the framework outputs with the data set previously defined with the %PTRANS macro.

- **LENGTH**=# Length of the new variable.
- **NAME**=variable name Name of the new variable.
- **TYPE**=c or n Type of the new variable; C for character or N for numeric.

For example, keeping with our example of the events data set, the following %NEWVAR reference defines a new character variable, length 13, named edata2 on the events data set.

```
%NEWVAR (NAME=EDATA2,
  TYPE=C,
  LENGTH=13);
```

Please note that %NEWVAR must be called immediately after %PTRANS for a particular data set.
%OVERLAY (NAME=, SRC=)

%OVERLAY defines an array that overlays several variables on the previously defined data set.

NAME= array name Name of the new array that overlays other input variables/arrays.

SRC= variable [variable ...

The variables that will be overlayed. If these variables are in a "stored array", you may optionally specify them using array syntax. For example, var1 var2 var3 may be expressed as var{3}.

PROCESSING

%DATA (ROOT=)

%DATA generates the appropriate hierarchical data set references in the processing data step's data statement. This includes the complete keep clause. You reference %DATA in the actual data statement:

DATA %DATA;

We show an excerpt of the resulting SAS code from our events example below:

... EVENTS(KEEP=CTRLNUM __19 - __36)
...

ROOT=# Specify the number of the root data set for this macro. If you specify a root #, then the macro will process only that root data set and its children; if you do not specify a root #, the macro will process all root data sets and their children.

%ARRAY (ROOT=)

%ARRAY generates the appropriate array statements to easily reference variables from the transformed data sets; the array names are the same as the variable names from the original data set(s). We show an excerpt of the array statements from our events example below; note the inclusion of the new edata2 variable defined by the earlier call to %NEWVAR:

... ARRAY EDATA{3,2} $ 2 __19 - __24;
ARRAY EVENTS{3,2} 8 __25 - __30;
ARRAY EDATA2{3,2} $ 13 __31 - __36;
...

%MERGE (REWIND=, ROOT=)

%MERGE generates the merge statement that merges the top-level data set with the transformed data views by the appropriate key variable(s). The merge statement contains an END= argument that will set the variable __done. We list the resulting SAS code from our example with the data sets in figure 4 below:

MERGE DEFAULT.HOUSEHLD VIEW2 VIEW3 VIEW4 VIEW5 END=__DONE;
BY CTRLNUM;

REWIND=# Specify the maximum number of times your processing data step will "rewind" the merged data set/views; see the %REWIND macro below. If you specify the rewind option the %MERGE macro build # copies of the merge statement within a select statement.

%REWIND (ROOT=)

%REWIND, working in conjunction with the REWIND= option of the %MERGE macro, allows you to "rewind" the merged top-level data set and transformed views back to the first observation. %REWIND increments a counter (__rewind) that causes the execution of a different merge statement in the select statement created by %MERGE.
%OUTPUT (ROOT=)

%OUTPUT outputs the top-level data set to its original or specified OUT= name/ location. Transformed data sets are output to the work library; the %UPDATE macro (see below) changes the transformed data sets back to their original format and updates the original data sets/output data sets. We list the resulting SAS code from our example with the data sets in figure 4 below:

OUTPUT DEFAULT.HOUSEHLD PERSONS PREMEDS
   PREGNA01 EVENTS;

POSTPROCESSING

%UPDATE (ROOT=)

%UPDATE reverses the amalgamation done by the %PTRANS macro and either updates the original data sets or creates specified output data sets. Coming back to our events example, %UPDATE generates the following data step code that changes the events data set from a top-level (household) view back to its original format. Note that %UPDATE will not create observations for missing events.

DATA EVENTS
   (KEEP=CTRLNUM PERSONS EDATA EVENTS EDATA2);
LENGTH EDATA $ 2;
ARRAY COL1{3,2} $ 2 __19 - __24;
ARRAY COL2{3,2} $ 8 __25 - __30;
LENGTH EDATA2 $ 13;
ARRAY COL3{3,2} $ 13 __31 - __36;
SET EVENTS;
DO PERSONS=1 TO 3;
   DO EVENTS=1 TO 2
      WHILE (COL2(PERSONS,EVENTS) NE .);
      EDATA=COL1{PERSONS,EVENTS};
      EDATA2=COL3{PERSONS,EVENTS};
      OUTPUT;
   END;
END;

Finally, %UPDATE updates the original events data set with modified values from the un-partially transformed events data set.

DATA DEFAULT.EVENTS;
UPDATE DEFAULT.EVENTS EVENTS;
BY CTRLNUM PERSONS EVENTS;

%CLEANUP

%CLEANUP clears all global macro variables created by the other macros.

CONCLUSION

We can apply this framework in situations where we match-merge input data sets that meet the criteria listed below and output one or more of those data sets:

1. The data sets must be a hierarchy of one-to-one or one-to-many relationships.
2. Each data set must contain key variables to uniquely identify each observation and associate it with its parent, grandparent, or great-grandparent observation as appropriate.
3. Each data set must be sorted by its appropriate key variables.

The macros described here fulfill our processing requirements. Specifically, we preserved the hierarchy by not physically combining data sets during processing. We removed most of the complexity of handling input and output data sets from processing. We ensured the output is complete at the end of the processing step(s). Finally, we allowed access to data at peer levels of the hierarchy.
DISCLAIMER

This paper reports the results of research and analysis undertaken by Census Bureau staff. It has undergone a more limited review than official Census Bureau publications. This report is released to inform interested parties of research and encourage discussion.

BIBLIOGRAPHY


CONTACT INFORMATION

Richard’s e-mail address is: Richard.Lee.Downs.Jr@census.gov.

Pura’s e-mail address is: Pura.A.Perez@census.gov.

The complete source code for the macros presented in this paper and several example programs are available on the internet at the following URL: http://ptrans.dusia.com/.

SAS and all other SAS Institute Inc. product or service names are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.