Paper AD15

Mapping Clinical Data to a Standard Structure: A Table Driven Approach

Nancy Brucken, i3 Statprobe, Ann Arbor, MI
Paul Slagle, i3 Statprobe, Ann Arbor, MI

ABSTRACT

Clinical Research Organizations (CROs) are often hired for projects involving mapping multiple disparate studies into a single submission database, or mapping legacy studies to Clinical Data Interchange Standards Consortium (CDISC) standards. Before programming begins, documentation containing details about the transformation is created. Then, programs are developed which incorporate the documents into logic, a process which takes considerable time and resources. Very often, such mapping systems have been developed as a series of custom macros or programs, which will work for mapping a single study into the target specifications, but break down when applied to multiple studies or projects. This paper describes a self-documenting table-driven mapping system developed by i3 Statprobe to be reusable across multiple studies and projects, which incorporates that effort into a single process.

BASIC STRUCTURE

Transformations required for mapping original fields to new fields were classified into 4 types:

- Global, or fields that need to be mapped or derived for every dataset in the study, such as a unique subject identifier.
- Direct, or fields that are merely renamed or re-formatted.
- Linear, or fields that can be derived from a single calculation in the right-hand side of an assignment statement in a DATA step.
- Complex, or fields that must be derived using multiple statements in a single DATA step, or multiple DATA or PROC steps.

Tables were set up for each type of transformation. These tables were then read at different points in the system, and variables in the original database processed according to the instructions embedded in the tables. The core of the system was a transpose performed on the original dataset, completely denormalizing it so it could be merged with the various transformation tables by variable name.

GLOBAL TRANSFORMATION TABLE

The global table consisted of the name of the new field; its attributes, such as type, length and label; and the formula for creating it. This table was rather small, so all information was entered manually. Sample rows from this table are shown in Table 1 below.

Table 1. Global transformation table

Study	Output Variable	Variable Type	Formula	Macro	Output Variable Label	Output Variable Format
1	USUBJID	CHAR	trim(left(put(input("&prot", 2.), z2.))) || '0' || trim(left(substr(site, 5, 2))) || trim(left(pid))		Unique Subject Identifier	$8
2	USUBJID	CHAR	trim(left(pid))		Unique Subject Identifier	$8

DIRECT TRANSFORMATION TABLE

The table of direct transformations was by far the largest, consisting of one record per dataset and variable in the input database. We created this table from the SAS® COLUMNS dictionary table; for each variable in the original database, it contained the type and associated format, along with the name of the remapped variable, the dataset it belonged to, and its type and associated format. The table also noted which variables from the input dataset were to be dropped from the remapped dataset.

This table served two purposes. The mapping system itself merged it with a completely normalized version of the original dataset, by variable name, to perform the direct transformations described in the table. Because of this merge, this table was restricted to one record per input dataset and variable. In addition, since it contained a record for every variable in the original database, it served as documentation of what happened to every variable when the database is re-mapped. Sample rows from this table are shown in Table 2.

Table 2. Direct transformation table

Study	Input Dataset	Input Var	Input Var Label	Input Var Type	Input Var Format	Output Dataset	Output Var	Output Var Label	Output Var Type	Output Var Format
1	DEMO	DOB	Date of birth	Num	Date9.	DM	BRTHDTC	Date of birth	Char	$8
1	DEMO	GENDER	Gender	Num	Gen.	DM	SEX	Sex	Num	Sex.
1	DEMO	RACE	Race	Num	Race.	DM	RACE	Ethnicity	Num	Race.
1	DEMO	CONSENT	Date of Inf. Consent	Num	Date9`					* DROP
1	DEMO	WT	Weight	Num	5.1	VS	WTKG	Weight (kg)	Num	5.1

Several things happened in this example. DOB (date of birth) was changed from a numeric to a character value of length 8. Values of GENDER were remapped from those specified according to the GEN. format to their equivalents using the SEX. format. RACE was brought over unchanged, except that the variable label became 'Ethnicity'. CONSENT was dropped from the output dataset; the presence of '* DROP' in the format field was adopted as a convention for indicating variables to be dropped, since it is not a valid format name. Finally, WT was moved from the DEMO (demographics) dataset to the VS (vital signs) dataset.

LINEAR AND COMPLEX TRANSFORMATION TABLE

Linear and complex transformations were stored together in a common table. That table consisted of one record per output dataset and variable name, with fields for specifying the attributes, associated format, either a formula or macro invocation for calculating the variable, whether it was to be derived before or after the dataset transpose, and a sequence number, allowing for control over the order in which the new variables were derived. Some variable values had to be calculated first, so they could be accessed by other derivations in the dataset, and the sequence number allowed the order to be specified. Table 3 shows sample rows from this table.

Table 3. Linear/complex transformation table

Study	Output Dataset	Output Var	Output Var Label	Type	Output Var Format	Macro	Formula	Timing	Input Dataset	Seq Number
1	DM	BMI	Body Mass Index	Ext	4.1	htwt (prot = 1, inds = DEMOG, outds = DEMOG)		AFT	DEMOG	1
1	DM	BMIC	Body Mass Index Category	Ext	BMIC.	* htwt (prot = 1, inds = DEMOG, outds = DEMOG)		AFT	DEMOG	1
1	DM	BRTHDT	Birth date (num)	Int	DATE9.		INPUT (brthdtc, date9.)	AFT	DEMOG	1
1	DM	AGE	Age	Ext	3.	Age (inds = DEMOG, birthdt = BRTHDT)		AFT	DEMOG	2

In this example, BMI was calculated by calling the macro %HTWT after the dataset is transposed, and this computation should take place during the processing of the DEMOG input dataset. BMIC was also calculated by the same macro; the '*' in front of the macro call indicated the macro did not need to be invoked twice, as the calculation was already performed, but the record describing BMIC had to be created so its creation was documented. These were complex transformations, because the variables required for computing BMI had to be pulled from other datasets, and their baseline values identified. They also had to be called outside of a DATA step.

On the other hand, BRTHDT was created using a linear transformation, and the formula for its derivation was specified in the table. Finally, age was calculated using the %AGE macro, which required as input the SAS date for birthdate, so it could not be called until after BRTHDT was created. This was also considered to be a complex transformation, requiring a macro that had to be executed outside of a DATA step.

PROCESS

Once the tables were set up, a generic macro was written and executed to automate the mapping process. In order to avoid having to hard-code information related to specific studies, all study-specific code was handled in the mapping tables. Key variables, uniquely determining a record in the output dataset, were specified in the macro call, along with the name of the output dataset`

The macro first built macro variables from the various transformation tables, such as lists of all of the input datasets containing variables required for the output dataset, all of the fields to be dropped from the output dataset, and character and numeric input variables.

PROC SQL was used for this task, since it is very easy to create space-delimited lists with SELECT/INTO. For example, the following code generated a list of all of the numeric variables in the original dataset that were to be included in the output dataset:

```
PROC SQL NOPRINT;
  SELECT name INTO :xvarc SEPARATED BY ' '
  FROM DICTIONARY.COLUMNS
  WHERE libname="&libref" AND memname="&inds" AND type='char'
    AND upcase(name) IN (%upcase(&outvlist));
QUIT;
```

Key variables were excluded from this list, since they were not transposed, but were used as BY-variables in the PROC TRANSPOSE.

The key to the system was that it transposed the input dataset itself into two temporary datasets, each consisting of one record per variable. All numeric variables were transposed into one dataset, and all character variables were transposed into the other. These normalized tables were concatenated, and then merged with the direct transformation table by input variable; the variable name was then changed to the name of the desired output variable, the type and length were adjusted, and the new label applied. The following code accomplished this step:

```
PROC TRANSPOSE DATA=&inds
               OUT=trans&varitype (RENAME=(_name_=invar));
  BY &keyvars;
  VAR &&xvar&vartype;
RUN;

DATA trans1&vartype (DROP=col1 RENAME=(col1c=col1));
  MERGE trans&vartype (IN=int) vars&prot (IN=inv);
  BY prot inds outds invar;
  IF int AND inv;

  LENGTH col1c $ 200;

  *** Format coded values, convert numeric values to character,;
  *** then recode to integrated dataset codes.;
  IF infmt ne ' ' THEN DO;
    col1c = TRIM(LEFT(PUT&vartype(col1, infmt)));
    IF outfmt NOT IN (' ', "&dateft", '* DROP') THEN DO;
      IF SUBSTR(outfmt, 1, 1)='$' THEN
         col1c = INPUTC(col1c, outfmt);
      ELSE col1c = INPUTN(col1c, outfmt);
    END;
  END;

%IF &vartype=n %THEN %DO;
   ELSE IF intype='num' THEN col1c = put(col1, best16.);
%END;
%ELSE %IF &vartype=c %THEN %DO;
   ELSE col1c = col1;
%END;
RUN;
```

Many times, custom formats differed between the input and remapped datasets. For example, sex might be coded as 1=Female and 2=Male in the input dataset, while the output dataset required 1=Male and 2=Female. All coded variables were decoded; the decoded values were then recoded to the new format.

The system then merged the denormalized dataset with the table of linear and complex transformations. Linear transformations were executed directly, via an assignment statement built from the transformation table. Macro invocations for complex transformations that could be run inside of a DATA step were executed directly; those that had to be run outside of a DATA step were stacked up via the use of CALL EXECUTE, in the order specified by the sequence number in the transformation dataset.

Once all of the transformations were applied, the dataset was transposed to one record per unique combination of key variables as specified for the output dataset. Any linear or complex transformations to be run after the transpose were executed at this time, global transformations were applied, and temporary variables were dropped. This process was then repeated for each output dataset required for the remapped database. The following code was used to transform the dataset, and apply the complex transformations:

```
*** Transpose back to original structure;
PROC TRANSPOSE LABEL=_label_ DATA=transall (DROP=inds)
               OUT=&outds (DROP=_name_);
  BY prot &keyvars;
  VAR col1;
  ID outvar;
RUN;

*** Apply any dataset-specific derivations to be done after
*** transpose;
DATA _NULL_;
  SET MAPPING.derive (WHERE=(prot="&prot" AND outds="&outds" AND
                             timing="AFT"));
  BY prot outds DESCENDING type;

  *** If internal to DATA step, then generate DATA and
  *** statements;
  %IF &internal > 0 %THEN %DO;
    IF _N_=1 THEN DO;
      CALL EXECUTE("DATA &outds;");
      CALL EXECUTE("SET &outds;");
    END;
  %END;

  *** Generate either assignment statement or macro call, regardless
  *** of type;
  *** DERIVE has internal derivations before external;
  *** A * as the first character of MACRO indicates a placeholder
  *** record;
  IF formula NE ' ' THEN
    CALL EXECUTE(outvar || '=' || formula || ';');
  ELSE IF macro NE ' ' AND SUBSTR(macro, 1, 1) NE '*' THEN
    CALL EXECUTE('%' || macro);

  *** If internal to DATA step, then generate RUN statement;
  %IF &internal > 0 %THEN %DO;
    IF type='INT' THEN DO;
      IF outfmt NE ' ' THEN
        CALL EXECUTE('format ' || outvar || ' ' || COMPRESS(outfmt)
                     || '.;');
    END;
    IF LAST.type THEN CALL EXECUTE("RUN;");
  %END;
RUN;
```

CONSIDERATIONS

Our initial version of the mapping system stored the transformation tables as SAS datasets. The drawback to this process is that SAS datasets are more difficult to edit than Excel files, which was the other file format we had considered. However, we had multiple programmers working on the mapping project for which this was developed, and SAS/SHARE allowed us to have multiple people editing the transformation tables and running programs against them simultaneously.

We made sure that every variable in the input database was documented in the direct transformation table, including variables that were dropped from the remapped database, and all variables that were created were documented in either the global or linear/complex transformation tables. This allowed us to generate documentation showing exactly what happened to each input variable, and the source of each output variable, directly from the transformation tables themselves.

SUMMARY

Table-driven mapping is an extremely powerful technique for generating mapping applications that can be repeatedly reused across datasets and projects, without having to modify the underlying code. Instead, all of the custom code is stored in a series of tables, which can then be edited to adjust for differences between datasets and studies. Extensive use should also made of the SAS dictionary tables in the mapping system, again avoiding the need for placing study-specific code inside of the mapping programs.

TRADEMARK CITATION

CONTACT INFORMATION

Nancy Brucken
i3 Statprobe
Nancy.Brucken@i3statprobe.com
(734) 769-5000 x1231

Paul Slagle
i3 Statprobe
Paul.Slagle@i3statprobe.com
(734) 769-5000 x1164