RTSUG - March 23, 2010

SAS Programming Techniques for Decoding Variables on the Database Level

Chris Speck, PAREXEL International, Durham, NC

Abstract

Thinking about SAS data in terms of entire libraries or databases couldn't be less intuitive for beginning programmers. When learning SAS, the dataset is the obvious starting point, but it unfortunately is also the ending point for many since it requires a shift of understanding to realize how to program beyond it. This goes beyond mere macro programming and involves interacting with entire libraries of data as one would a dataset. One *could* approach making uniform changes to an entire library of data armed with nothing but a dataset mentality. But such a road would be fraught with errors and wasted motions as Program A uses one algorithm to deal with Dataset A while Program B uses a slightly different one to adjust Dataset B, and so on. Why not do it all in one program with less work and a more reliable result?

This paper will provide methods by which SAS programmers working in Base SAS can quickly "decode" variables within an entire library by making new variables equal to the formatted values of these variables. For instance, the numeric variable AESEV may have a value of 1 for "mild", 2 for "moderate", and 3 for "severe" with a format representing these character values. Decoding this variable means creating a character variable AESEVC equivalent to the formatted values of AESEV. The method described in this paper appears in a program called the MetaEngine, and will include heavy use of the SAS macro facility and SAS dictionary tables. For more information on the MetaEngine, see my previous paper SAS Programming Techniques for Manipulating Data on the Database Level.

Introduction

Decoding hundreds of variables in a library can be a daunting task. Does this require 1) determining which formatted variables need decoding, 2) investigating the types of these formats, 3) deriving the decode variables with put() statements in a data step, and 4) double-checking each put() statement to make sure the correct format got assigned? Does this also entail 5) manually assigning lengths of these decode variables based on the longest value in the format (so no decode values will get truncated)? If so, then a programmer should prepare for many tedious hours of work and rework.

There are, however, programming techniques that will allow quick decoding of entire libraries with reasonable setup time and upkeep efforts. Furthermore, the MetaEngine is portable, and once a programmer masters the concepts behind it, it can be used almost anywhere.

The Engine Skeleton

The MetaEngine uses a simple macro loop which iterates through a library dataset by dataset. The macro variable &THISDS refers to the dataset that the engine is dealing with at any given time.

```
*******************************************************;
** MetaEngine Macro
** LIB=Library in which study datasets exist
** OUTLIB=Library into which FDA-Ready datasets will be saved
*******************************************************;
%macro MetaEngine(lib=, outlib=);

  ** Gets list and number of datasets in library in macro variable DSNAMES **;
  proc sql noprint;
    select memname
    into :dsnames separated by '~'
    from dictionary.columns
    where libname="&LIB" and varnum=1;
  quit;

  ** Loops through library with scan macro function **;
  %let i = %eval(1);
  %do %while (%scan(&dsnames,&i,~) ne );
    %let thisds = %scan(&dsnames,%eval(&i),~);

    ** << Bulk of programming is performed >> ***;

    %let i = %eval(&i+1);
  %end;

%mend MetaEngine;

%MetaEngine(lib=MYLIB, outlib=NEWLIB);
```

For a quick breakdown, the PROC SQL block uses the SAS dictionary table COLUMNS to place a list of all datasets in a library into the macro variable &DSNAMES. This macro variable is used in the %DO loop to produce the name of each dataset, one by one, for processing. Datasets that do not require analysis can be omitted using a WHERE statement in the PROC SQL, as well as in an additional macro parameter. As the loop iterates, the macro variable &THISDS resolves to dataset names in the &DSNAMES list one by one. This is how each dataset gets processed in the MetaEngine. Keep in mind that the library entered as a parameter must be in all caps for it to be understood by PROC SQL.

Preserve Dataset Labels

Experienced SAS programmers know that the data step will not preserve dataset labels. Since the MetaEngine employs many data steps within its loop, by the time all variables in a dataset have their decodes the original dataset label will be gone. This is why we must first preserve these labels first in the “bulk of programming is performed” area of the MetaEngine.

```
** Reset dataset label to missing **;
%let dslabel=;

** Gets dataset label for each dataset **;
proc sql noprint;
  select memlabel
  into :dslabel
  from dictionary.tables
  where libname="&lib" and memname="&thisds";
quit;
%let dslabel=%trim(&dslabel); ** trims label **;
```

Later on, when constructing the final version of the dataset represented at any time by &THISDS, "&DSLABEL" should appear in the final data step's LABEL option. Notice that the macro variable used for this is reset to missing before any action takes place.

For more information on how to adjust dataset labels using the MetaEngine, see my previous paper SAS Programming Techniques for Manipulating Data on the Database Level.

Calling the %Decode Macro

The next step is to call the macro that will decode the variable. The code should look like this:

```
** STEP 1: Creating base dataset in WORK **;
data ds0;
  set &lib..&thisds;
run;

** STEP 2: Building list of decoded macros **;
%let fv=; ** FV stands for FORMAT VARIABLES **;

** STEP 3: Calling the %DECODE macro once for every variable needing decoding **;
```

```
%if &thisds=DATA1 %then %do;
  %decode(fmtlib=&lib,ds=ds0,   newds=ds0_00,var=D1VAR1,newvar=D1VAR1C); %let fv=&fv D1VAR1;
  %decode(fmtlib=&lib,ds=ds0_00,newds=ds0_01,var=D1VAR2,newvar=D1VAR2C); %let fv=&fv D1VAR2;
  %decode(fmtlib=&lib,ds=ds0_01,newds=ds0_02,var=D1VAR3,newvar=D1VAR3C); %let fv=&fv D1VAR3;
  %decode(fmtlib=&lib,ds=ds0_02,newds=ds1   ,var=D1VAR4,newvar=D1VAR4C); %let fv=&fv D1VAR4;
%end;
%if &thisds=DATA2 %then %do;
  %decode(fmtlib=&lib,ds=ds0,   newds=ds0_00,var=D2VAR1,newvar=D2VAR1C); %let fv=&fv D2VAR1;
  %decode(fmtlib=&lib,ds=ds0_00,newds=ds0_01,var=D2VAR2,newvar=D2VAR2C); %let fv=&fv D2VAR2;
  %decode(fmtlib=&lib,ds=ds0_01,newds=ds0_02,var=D2VAR3,newvar=D2VAR3C); %let fv=&fv D2VAR3;
  %decode(fmtlib=&lib,ds=ds0_02,newds=ds0_03,var=D2VAR4,newvar=D2VAR4C); %let fv=&fv D2VAR4;
  %decode(fmtlib=&lib,ds=ds0_03,newds=ds0_04,var=D2VAR5,newvar=D2VAR5C); %let fv=&fv D2VAR5;
  %decode(fmtlib=&lib,ds=ds0_04,newds=ds0_05,var=D2VAR6,newvar=D2VAR6C); %let fv=&fv D2VAR6;
  %decode(fmtlib=&lib,ds=ds0_05,newds=ds0_06,var=D2VAR7,newvar=D2VAR7C); %let fv=&fv D2VAR7;
  %decode(fmtlib=&lib,ds=ds0_06,newds=ds0_07,var=D2VAR8,newvar=D2VAR8C); %let fv=&fv D2VAR8;
  %decode(fmtlib=&lib,ds=ds0_07,newds=ds1   ,var=D2VAR9,newvar=D2VAR9C); %let fv=&fv D2VAR9;
%end;
%** STEP 4: In case a dataset has no variable needing decoding **;
%else %do;
  data ds1;
    set ds0;
  run;
%end;
```

STEP 1: Sets up the base dataset DS0 which is identical to whichever dataset is iterating through the macro loop at any given time. This is the dataset before all its variables are decoded.

STEP 2: Creates the FV macro variable which ultimately contains a list of all the variables in a dataset which need decoding. It will be used later on to strip formats off the original variables. It could also be used to establish uniform attributes for the variables or to order them a certain way in the dataset. For every new &THISDS dataset, FV is first set to missing.

STEP 3: Calls the %DECODE macro a certain number of times depending on which &THISDS dataset is iterating through the library loop. Foreknowledge of what variables need decoding becomes useful here. For instance, a programmer must know that the dataset DATA1 has a variable D1VAR1 which requires a decode. If there are 30 datasets in a library needing decodes then there will be 30 %IF-%THEN-%ELSE statements (unless two or more datasets have the same variables needing decodes).

Notice that the programmer decides what to name the decode variables. This may become necessary when a variable name is already 8 characters long and simply adding a “C” at the end of it for the decode variable may not be feasible.

As for the macro itself, a dataset (DS) goes into the macro without a decode variable and a different dataset (NEWDS) comes out with one. NEWDS is identical to DS except for the presence of the new decode variable. VAR determines which variable gets decoded, and NEWVAR determines the name of the decode variable. To the far right of the line, we increment the list of decoded variables in the FV macro variable.

FMTLIB specifies the library in which the variable’s format catalog resides (which a programmer must also know beforehand). If the format catalog is named anything other than “formats” then the programmer should enter the library name, followed by a period, followed by the catalog name (e.g., &MYLIB..FMTS). Note that &MYLIB is only appropriate if the format exists in that library. If it exists in a different library, then that library name can be entered. A programmer does not have to use &MYLIB for this purpose.

STEP 4: Regardless of whether a dataset has variables requiring decoding, DS1 gets created from DS0.

The %DECODE Macro

The %DECODE macro is only effective if it can be quicker and more reliable that simply using PUT() statements in data steps. It accomplishes this by creating a low research bar for the programmer. Decoding a library’s worth of data with nothing but a dataset mindset would require the programmer to follow the 5 steps described in the introduction for each variable. The MetaEngine only requires the programmer perform step 1 (with possibly the additional half-step of knowing where the appropriate format catalog is). Once the variable in need of a decode is named, the % DECODE macro completes the programmer’s research and then acts accordingly.

```
********************************************************;
** Decode Macro
** FMTLIB=Library holding the variable format catalog
** DS    =Dataset going into the macro
** NEWDS =Dataset produced by the macro
** VAR   =Formatted variable requiring a decode
** NEWVAR=Name of new decode variable
********************************************************;
%macro Decode(fmtlib=, ds=, newds=, var=, newvar=);

  /* STEP 1 */
  proc sql noprint;
    select format into :fmt from dictionary.columns
    where libname="WORK" and memname=%upcase("&ds") and name="&var";
  quit;
  /* STEP 2 */
  proc format noprint cntlout=fmt (keep=length) library=%upcase(&fmtlib) fmtlib;
    select %substr(&fmt,1,%length(&fmt)-1); * removes period at end of format;
  run;
  /* STEP 3 */
  data _null_;
    set fmt;
    if _n_=1 then call symput('len',cats(put(length,best.)));
  run;
  /* STEP 4 */
  data _null_;
    set &ds;
    if _n_=1 then do;
      if length(vlabel(&var))<=38 then
        call symput('newlabel',cats(vlabel(&var)) || "-C");
      else if length(vlabel(&var))=39 then
        call symput('newlabel',cats(vlabel(&var)) || "C");
      else if length(vlabel(&var))>=40 then
        call symput('newlabel',substr(cats(vlabel(&var)),1,39)||"C");
    end;
  run;
  /* STEP 5 */
  data &newds;
    length &newvar $&len;
    set &ds;
    &newvar=put(&var,&fmt);
    label &newvar="&newlabel";
  run;
  /* STEP 6 */
  proc datasets nolist lib=work memtype=data;
    delete fmt &ds;
  run; quit;

%mend Decode;
```

STEP 1: Finds variable's format using the SAS Dictionary table COLUMNS and assigns it to the FMT macro variable.

STEP 2: Gets the maximum length of the format values (which is contained in the LENGTH variable produced by the FMTLIB option) and stores it in the control-out dataset FMT.

STEP 3: Assigns maximum length value to macro variable LEN. This will be used later in a length statement to prevent a formatted value from being truncated.

STEP 4: Uses the VLABEL() function to retrieve the variable's label and then adjusts it in a uniform way (in this case, by adding "-C" to the end) to indicate that it is a decode variable. Note that in order to keep all labels within 40 characters, the algorithm adds only "C" to labels that are 39 characters long, and replaces the 40th character with a "C" and truncates the remainder of the label in case the label is 40 characters or longer. This convention may vary depending on the needs of the sponsor.

STEP 5: Creates new dataset (&NEWDS), derives the new decode variable with the variable's format (&FMT) in a PUT() statement, and assigns it a length (&LEN) and a label (&NEWLABEL).

STEP 6: Performs garbage collection since this macro will most likely be called often. Purging this library of the no-longer-needed datasets that stretch the gulf between DS0 and DS1 would be a good idea, especially if you're dealing with large datasets to begin with. Imagine a 500MB dataset with 30 variables to decode. Without deleting each dataset after it gets processed by the %DECODE macro, the MetaEngine would create 15GB of datasets. Such spacing hogging could possibly tax whatever shared server your SAS system operates on.

Making DS2 and Completing Work

After constructing DS1, some space should be dedicated to making specific changes to datasets that cannot easily be automated.

```
** Perform any other changes that cannot be automated **;
data ds2;
  set ds1;
  %if &thisds=DATA3 %then %do;
                 /*1234567890123456789012345678901234567890*/
    label D3VAR1C="Trunc. decode label which was too long-C";
  %end;
run;
```

Any further processing can continue after this point (using DS3, DS4, etc.), just as long as in the final data step uses the &DSLABEL macro variable in a LABEL statement to (re)assign the dataset's original label. This is also where &FV can be used in case you need to remove or strip off formats from all the variables that were decoded.

```
** Copy dataset into the new folder with appropriate dataset label **;
data &outlib..&thisds %if %length(&dslabel)>0 %then (label="&dslabel");;
  set ds2;
  format &fv;
run;
```

The MetaEngine in its entirety will be presented at the end of this paper.

Possibilities and Challenges

It is possible to automate the MetaEngine further. A PROC SQL could easily produce a list of all variables in a dataset with non-native SAS formats. This list could then inform an inner loop which calls %DECODE() once for each variable and uses the naming conventions described above. What one gains in automation, however, one loses in adaptability. Not all formats may exist in the same catalog. Not all variable labels may be less than or equal to 39 characters. Not all variable names may be 8 characters long. And not all formatted variables may require decodes. The MetaEngine as shown above foregoes ultimate push-button automation to allow the programmer to handle the sticky situations on a case by case basis.

Conclusion

There are other aspects of data manipulation at the library level not discussed here, namely the adjusting of variable data and metadata as well as the ordering of variables in a dataset. These processes were discussed in my previous paper SAS Programming Techniques for Manipulating Data on the Database Level and can be integrated in the program rather quickly. What the MetaEngine offers is a quick and streamlined approach for a programmer to begin thinking about metadata on the library level and to begin manipulating it intuitively it as if it were data in a dataset.

The MetaEngine (Full Code with Sample Data)

```
********************************************************;
** MetaEngine Macro
** LIB=Library in which study datasets exist
** OUTLIB=Library into which FDA-Ready datasets will be saved
********************************************************;
%macro MetaEngine(lib=, outlib=);

********************************************************;
** Decode Macro
** FMTLIB=Library holding the variable format catalog
** DS    =Dataset going into the macro
** NEWDS =Dataset produced by the macro
** VAR   =Formatted variable requiring a decode
** NEWVAR=Name of new decode variable
********************************************************;
%macro Decode(fmtlib=, ds=, newds=, var=, newvar=);

  /* STEP 1 */
  proc sql noprint;
    select format into :fmt from dictionary.columns
    where libname="WORK" and memname=%upcase("&ds") and name="&var";
  quit;
  /* STEP 2 */
  proc format noprint cntlout=fmt (keep=length) library=%upcase(&fmtlib) fmtlib;
    select %substr(&fmt,1,%length(&fmt)-1); * removes period at end of format;
  run;
  data _null_;
    set fmt;
    if _n_=1 then call symput('len',cats(put(length,best.)));
  run;
  /* STEP 3 */
  data _null_;
    set &ds;
    if _n_=1 then do;
      if length(vlabel(&var))<=38 then
        call symput('newlabel',cats(vlabel(&var)) || "-C");
      else if length(vlabel(&var))=39 then
        call symput('newlabel',cats(vlabel(&var)) || "C");
      else if length(vlabel(&var))>=40 then
        call symput('newlabel',substr(cats(vlabel(&var)),1,39)||"C");
    end;
  run;
  /* STEP 4 */
  data &newds;
    length &newvar $&len;
    set &ds;
    &newvar=put(&var,&fmt);
    label &newvar="&newlabel";
  run;
  /* STEP 5 */
  proc datasets nolist lib=work memtype=data;
    delete fmt &ds;
  run; quit;

%mend Decode;

  ** Gets list and number of datasets in library in macro variable DSNAMES **;
  proc sql noprint;
    select memname
    into :dsnames separated by '~'
    from dictionary.columns
    where libname="&LIB" and varnum=1;
  quit;

  ** Loops through library with scan macro function **;
  %let i = %eval(1);
  %do %while (%scan(&dsnames,&i,~) ne );
```

```
    %let thisds = %scan(&dsnames,%eval(&i),~);

    ** Reset dataset label to missing **;
    %let dslabel=;

    ** Gets dataset label for each dataset **;
    proc sql noprint;
      select memlabel
      into :dslabel
      from dictionary.tables
      where libname="&lib" and memname="&thisds";
    quit;
    %let dslabel=%trim(&dslabel); ** trims label **;

    ** STEP 1: Creating base dataset **;
    data ds0;
      set &lib..&thisds;
    run;

    ** STEP 2: Adding decodes using the %DECODE macro **;
    %let fv=; ** FV stands for FORMAT VARIABLES **;

    ** STEP 3: Calling the Decode macro once for every variable needing decoding **;
    %if &thisds=DATA1 %then %do;
      %decode(fmtlib=&lib,ds=ds0,   newds=ds0_00,var=D1VAR1,newvar=D1VAR1C); %let fv=&fv D1VAR1;
      %decode(fmtlib=&lib,ds=ds0_00,newds=ds0_01,var=D1VAR2,newvar=D1VAR2C); %let fv=&fv D1VAR2;
      %decode(fmtlib=&lib,ds=ds0_01,newds=ds0_02,var=D1VAR3,newvar=D1VAR3C); %let fv=&fv D1VAR3;
      %decode(fmtlib=&lib,ds=ds0_02,newds=ds1   ,var=D1VAR4,newvar=D1VAR4C); %let fv=&fv D1VAR4;
    %end;
    %if &thisds=DATA2 %then %do;
      %decode(fmtlib=&lib,ds=ds0,   newds=ds0_00,var=D2VAR1,newvar=D2VAR1C); %let fv=&fv D2VAR1;
      %decode(fmtlib=&lib,ds=ds0_00,newds=ds0_01,var=D2VAR2,newvar=D2VAR2C); %let fv=&fv D2VAR2;
      %decode(fmtlib=&lib,ds=ds0_01,newds=ds0_02,var=D2VAR3,newvar=D2VAR3C); %let fv=&fv D2VAR3;
      %decode(fmtlib=&lib,ds=ds0_02,newds=ds0_03,var=D2VAR4,newvar=D2VAR4C); %let fv=&fv D2VAR4;
      %decode(fmtlib=&lib,ds=ds0_03,newds=ds0_04,var=D2VAR5,newvar=D2VAR5C); %let fv=&fv D2VAR5;
      %decode(fmtlib=&lib,ds=ds0_04,newds=ds0_05,var=D2VAR6,newvar=D2VAR6C); %let fv=&fv D2VAR6;
      %decode(fmtlib=&lib,ds=ds0_05,newds=ds0_06,var=D2VAR7,newvar=D2VAR7C); %let fv=&fv D2VAR7;
      %decode(fmtlib=&lib,ds=ds0_06,newds=ds0_07,var=D2VAR8,newvar=D2VAR8C); %let fv=&fv D2VAR8;
      %decode(fmtlib=&lib,ds=ds0_07,newds=ds1   ,var=D2VAR9,newvar=D2VAR9C); %let fv=&fv D2VAR9;
    %end;
    %** STEP 4: In case a dataset has no variable needing decoding **;
    %else %do;
      data ds1;
        set ds0;
      run;
    %end;

    ** Perform any other changes that cannot be automated **;
    data ds2;
      set ds1;
    run;

    ** Copy dataset into the new folder with appropriate dataset label **;
    data &outlib..&thisds %if %length(&dslabel)>0 %then (label="&dslabel");;
      set ds2;
      format &fv;
    run;

    %let i = %eval(&i+1);
  %end;

%mend MetaEngine;

%MetaEngine(lib=MYLIB, outlib=MYNEWLIB);
```