

SAS Programming Techniques for Manipulating Metadata on the Database Level

Chris Speck, PAREXEL International, Durham, NC

ABSTRACT

One great leap that beginning and intermediate SAS programmers must take in order to achieve a more expert level of programming is to think in terms of libraries, or entire databases, rather than only datasets. This has recently become much more crucial now that programmers are under greater pressure to apply regulatory standards to clinical research data. In the United States, the FDA more and more requires that data meet the strict standards set by the Clinical Data Interchange Standards Consortium (CDISC). This has caused many SAS programmers to rethink their purposes and practices while preparing study data for FDA submission. It forces them to deal with metadata, not just on the dataset level, but on the level of entire libraries as well.

This paper will provide methods by which SAS programmers working in Base SAS can quickly manipulate the data and metadata of entire libraries. These methods, compiled in a program called the "Meta-Engine", will include heavy use of the SAS macro facility, SAS dictionary tables, and SAS quoting techniques.

INTRODUCTION

Converting a database of study data to CDISC standards can be a big job -- especially when dealing with legacy data. What's a programmer to do when faced with a library of 55 large datasets, almost all of which contains missing or overly long labels, non-native SAS formats, character variables extending over 200 characters, special characters in dataset names, and other seemingly innocuous transgressions that ten years ago would not have stopped a study from being submitted? Of course, "know thy data" as an axiom applies here as much as ever. As a result, many programmers will find methods to the madness. For example, a certain problematic variable reappears across multiple datasets, or perhaps a certain problematic dataset reappears across multiple libraries. Or not, as in the worst cases. Either way, however, there are programming techniques that will allow a quick transformation of entire libraries into standards compliant data with reasonable setup time and upkeep efforts. In other words, the Meta-Engine is portable, and once a programmer masters the concepts behind it, it can be used almost anywhere.

THE ENGINE SKELETON

The Meta-Engine 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.

```
*****;
** Meta-Engine Macro
** LIB=Library in which study datasets exist
** OUTLIB=Library into which FDA-Ready datasets will be saved
*****;
%macro MetaEngine(lib=, outlib=);
** Puts list & number of datasets in macro variables DSNames and DSNUM **;
proc sql noprint;
  select memname into :dsnames separated by ' '
    from dictionary.columns
    where libname="&lib" and varnum=1
  ;
  select count(memname) into :dsnum
    from dictionary.columns
    where libname="&lib" and varnum=1
  ;
quit;

%put &dsnames;
%put &dsnum;

** Loops through library **;
%let i = %eval(1);
%do %while (%eval(&i) <= %eval(&dsnum));
  %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 uses the SAS dictionary table COLUMNS to place a list of all datasets in a library into the macro variable &DSNAMES as well as the number of elements in this list into the &DSNUM macro variable. These two macro variables are used in the %DO loop to produce the name of each dataset, one by one, for processing. If there are some datasets that do not require analysis they can be omitted using a WHERE statement in the PROC SQL step, as well as with an additional macro parameter. As the loop iterates, the macro variable &THISDS equals the dataset names in the &DSNAMES list one by one. This is how each dataset gets processed in the Meta-Engine. Keep in mind that the library entered as a parameter must be in all caps for it to be understood by the PROC SQL block.

ADJUSTING DATASET LABELS

The first thing one sees when scrutinizing a study's metadata as it is presented in an organized format, such as a Case Report Tabulation (CRT), is the presence or absence of dataset labels. The following code assigns corrected dataset labels to a macro variable (%DSL2), which will be employed later in the program. It should be applied in the "bulk of programming" section above.

```

** Gets dataset label for each dataset **;
proc sql noprint;
    select memlabel
    into :dsl
    from dictionary.tables
    where libname="&lib" and memname="&thisds"
;
quit;

** Trims label in macro variable **;
data _null_;
    length temp $200;
    temp="&dsl";
    call symput('dsl2',cats(temp));
run;

** Adjusting DS labels **;
    %if &thisds=DATA1 %then %let dslabel=This is the label for DATA1;
%else %if &thisds=DATA2 %then %let dslabel=This is the label for DATA2;
%else %let dslabel=&dsl2;
;

```

In the %IF %THEN block above, DATA1 and DATA2 refer to dataset names. Later on, when constructing the final version of the dataset represented at any time by &THISDS, "&DSLABEL" should appear in the data step's LABEL option. Notice that this approach of reassigning dataset labels every time the program runs avoids the error of appending the corrected label to the end of the original one.

ADJUSTING DATA AND METADATA

The next step is to adjust the actual data and metadata within each dataset. This is performed within a single dataset DS0 which will, of course, represent a completely different dataset with every iteration through the macro loop. Macro logic within the data steps will execute whatever changes that need to be made. Note that we don't yet apply the &DSLABEL macro variable in a LABEL= option since we have not yet reached our final data step. Note also that the code below also uses the hypothetical datasets DATA1 and DATA2.

```

** Adjusting data and metadata **;
data ds0;
    %if &thisds=DATA1 %then length LONGVAR $200;;
    set &lib..&thisds;
    label VAR1="The appropriate label"; *VAR1 label changed in all datasets;
    %if &thisds=DATA1 %then %do;
        format VAR2 myfmt.;
        label DATA1VAR="Corrected label"; *DATA1VAR label changed only in DATA1;
    %end;
    %if &thisds=DATA2 %then %do;
        drop VAR3 VAR4;
    %end;
** << continue for other datasets in database **;

```

```
run;
```

This approach works well if individual datasets require unique changes. But suppose there are many datasets that require the same change. Suppose many but not all datasets need to have the variables LIST1VAR and LIST2VAR dropped with only partial overlap? Suppose LIST2VAR should not be dropped from a particular dataset (for example TERM)? One could extend the %IF %THEN %DO block above into something unwieldy:

```
%if &thisds=DATA2 or &thisds=AE or &thisds=CONMED or &thisds=LAB1 or &thisds=LAB2 or
&thisds=LAB3 or &thisds=LAB4 or &thisds=ECG or &thisds=COMMENT1 or &thisds=COMMENT2
or &thisds=VITALS or &thisds=DEMOG or &thisds=KEYVARS or &thisds=MEDHX or
&thisds=DEATHS or &thisds=DISP or &thisds=TERM %then %do;
  drop LIST1VAR;
%end;

%if &thisds=DATA3 or &thisds=AE or &thisds=CONMED or &thisds=LAB1 or &thisds=LAB2 or
&thisds=LAB3 or &thisds=LAB5 or &thisds=COMMENT1 or &thisds=COMMENT3 or
&thisds=SUMMARY or &thisds=VITALS or &thisds=DEMOG or &thisds=KEYVARS or
&thisds=MEDHX or &thisds=DEATHS or &thisds=DISP %then %do;
  drop LIST2VAR;
%end;
```

One could continue with this approach for all changes required in the data. This would make for some pretty ugly code, not to mention difficult to read code since all the differences between these two %IF %THEN blocks are not obvious at first glance. Through the process of quoting, SAS allows a programmer to streamline this approach.

SAS QUOTING

By using the following macro, a programmer can automate the production of these complicated %IF %THEN statements. They will appear in macro variables which will be resolved in the data step for the DS0 dataset. A programmer should place a variant of the following code at the top of the Meta Engine macro:

```
*****;
** GetList Macro
** VAR =Variable SAS searches for to make list of datasets with that variable.
** WHERE =Condition limiting search if VAR stays the same in a certain dataset.
** Must be an entire WHERE statement in PROC SQL and begin with "and".
*****;
%global LIST1VAR LIST2VAR; ** list of macro variables you need to create **;
%macro GetList (var=, where=);

  ** Macro quoting function nrstr() keeps from resolving &thisds (masks the &) **;
  proc sql noprint;
    select '%nrstr(&thisds)= ' || cats(memname) || ' or '
    into :&var separated by ' '
    from dictionary.columns
    where libname="&lib" and name="&var" &where;
  ;
  quit;

  ** && resolves VAR from &LIST1VAR to '&thisds=XXX or' **;
  %put &&&var;

  ** Macro quoting function %qsubstr() masks the & when removing final OR;
  %let &var = %qsubstr(&&&var,1,%length(&&&var)-3);
  %put &&&&var;
%mend GetList;

%GetList(var=LIST1VAR, where=);
%GetList(var=LIST2VAR, where= and memname ne "TERM");
```

Instead of having to search an entire database to construct lengthy if-then lists, the programmer only has to enter the name of the variable being searched for, and then let SAS do all the work. In the convention used in the code above, the parameter VAR is the same as the actual variable SAS is searching the library for. So if the variable LIST1VAR appears in three datasets (e.g., AE, CONMED, and DEMOG) in the library, then the macro variable &LIST1VAR at first will resolve to:

```
&thisds=AE or &thisds=CONMED or &thisds=DEMOG or
```

The ultimate goal, of course, is to drop the final "or" and resolve &LISTVAR in the DS0 data step itself, thus constructing the long and unwieldy %if %then statements accurately and effortlessly at run time.

The confusing aspect of this occurs when resolving the ampersands. SAS has to be told when to resolve the ampersands and when not to. This is where quoting comes in. Without quoting, as in the NRSTR() function above, SAS would not enter the actual text of &THISDS into the &VAR macro but its resolved value (i.e., whatever dataset on the &DSNAMES list that is currently iterating through the loop at that particular time). But in this case we want the literal text "&THISDS" to go into &VAR since we are constructing code not resolving variables.

Aside from removing the trailing "or", The QSUBSTR() function does essentially the same thing as the NRSTR() function. It allows SAS to act on the actual text &THISDS rather than its resolved value. In other words, it "masks" the ampersand and thus treats it like any other character. Also note that because VAR is a parameter of the %GetList() macro, one ampersand will resolve it to the text value entered in the macro call (in this case, either LIST1VAR or LIST2VAR). This is clearly not what we want entered into the QSUBSTR() function. Three ampersands however allow an additional level of resolution, so it is not a substring of string literal "LIST1VAR" or "LIST2VAR" being created but a substring of

```
&thisds=AE or &thisds=CONMED or &thisds=DEMOG or
```

And once the final "or" is removed, the macro variable is finally ready to be applied in the DS0 data step.

Inside the DS0 data step, however, there is one final step. If one were to simply enter the following code:

```
%if &LISTVAR %then %do;
  drop LIST1VAR;
%end;
```

SAS would throw the following error:

```
ERROR: A character operand was found in the %EVAL function or %IF condition where a
numeric operand is required.
```

This error occurs because special characters like the ampersand held within the macro variable are still masked. SAS needs them unmasked in order to resolve them. The %UNQUOTE function does just this. It unmask a special character in a macro variable so that it is interpreted as a macro language element rather than as text. Typically, %UNQUOTE restores normal text values that had been altered by previous macro quoting functions. %UNQUOTE executes only during macro execution.

So in the DS0 data step, a programmer need only enter:

```
%if %unquote(&LIST1VAR) %then %do;
  drop LIST1VAR;
%end;
%if %unquote(&LIST2VAR) %then %do;
  drop LIST2VAR;
%end;
```

to have the same effect as:

```
%if &thisds=DATA2 or &thisds=AE or &thisds=CONMED or &thisds=LAB1 or &thisds=LAB2 or
&thisds=LAB3 or &thisds=LAB4 or &thisds=ECG or &thisds=COMMENT1 or &thisds=COMMENT2
or &thisds=VITALS or &thisds=DEMOG or &thisds=KEYVARS or &thisds=MEDHX or
&thisds=DEATHS or &thisds=DISP or &thisds=TERM %then %do;
  drop LIST1VAR;
%end;

%if &thisds=DATA3 or &thisds=AE or &thisds=CONMED or &thisds=LAB1 or &thisds=LAB2 or
&thisds=LAB3 or &thisds=LAB5 or &thisds=COMMENT1 or &thisds=COMMENT3 or
&thisds=SUMMARY or &thisds=VITALS or &thisds=DEMOG or &thisds=KEYVARS or
&thisds=MEDHX or &thisds=DEATHS or &thisds=DISP %then %do;
  drop LIST2VAR;
%end;
```

But with only a fraction of the time and work.

ALPHABETIZING VARIABLES

When presenting metadata to the FDA or clients, often in Case Report Tabulation (CRT) format, readability and searchability become paramount. Reviewers looking for a particular variable want to find it and find it quickly. When variables lack a certain order, this becomes a very tedious exercise, especially when variables lack intuitive names. Thus, the Meta-Engine calls an OrderVars macro to order variables alphabetically.

```
*****;  
** OrderVars Macro  
** DS      =Dataset being passed into macro  
** OUTDS   =Ordered dataset produced by macro  
*****;  
%macro OrderVars(ds=, outds=);  
  proc contents data=&ds out=alpha(keep=name) noprint; run;  
  proc sql noprint; select name into :avars separated by ' ' from alpha; quit;  
  data &outds;  
    retain &avars;  
    set &ds;  
  run;  
%mend;
```

Like %GetList, %OrderVars should appear at the top of the %MetaEngine macro itself. To break it down, a PROC CONTENTS produces a dataset ALPHA from the input dataset (which in this case would be DS0). The KEEP= statement ensures that ALPHA contain only the variable names. The PROC SQL puts these names in a macro variable list called &AVARS which is then fed into the RETAIN statement in the following data step. The RETAIN statement sets the alphabetical order of the variables as would a LENGTH statement, but without the extra effort of including lengths and dollar signs.

So within the programming section of the Meta-Engine, the code would look something like this:

```
%OrderVars(ds=DS0, outds=DS1);
```

Any further processing can continue after this point (using DS2, DS3, etc.), just as long as in the final data step uses the &DSLABEL macro variable in a LABEL= statement to assign the dataset's proper label.

The Meta-Engine in its entirety will be presented at the end of this paper.

CONCLUSION

There are other aspects of metadata manipulation at the library level not discussed here, namely the decoding of formatted variables and library testing for CDISC noncompliance. Further, the examples above do not include steps for deleting temporary variables and datasets. However these can be integrated in the program rather quickly. What the Meta-Engine 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 as if it were data in a dataset.

CONTACT INFORMATION

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

Name: Chris Speck
Enterprise: PAREXEL International
Address: 2520 Meridian Parkway, Suite 200
City, State ZIP: Durham, NC 27713
Work Phone: 919 294 5018
Fax: 919 544 3698
E-mail: chris.speck@parexel.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or 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.

META ENGINE (FULL CODE WITH SAMPLE DATA)

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

*****;
** GetList Macro
** VAR =Variable SAS searches for to make list of datasets with that variable.
** WHERE =Condition limiting search if VAR stays the same in a certain dataset.
**      Must be an entire WHERE statement in PROC SQL and begin with "and".
*****;
%global LIST1VAR LIST2VAR; ** list of macro variables you need to create **;
%macro GetList (var=, where=);

    ** Macro quoting function nrstr() keeps from resolving &thisds (masks the &) **;
    proc sql noprint;
        select '%nrstr(&thisds)= ' || cats(memname) || ' or '
            into :&var separated by ' '
            from dictionary.columns
            where libname="&lib" and name="&var" &where;
    ;
quit;

    ** &&& resolves VAR from &LIST1VAR to '&thisds=XXX or' **;
    %put &&&var;

    ** Macro quoting function %qsubstr() masks the & when removing final OR;
    %let &var = %qsubstr(&&&var,1,%length(&&&var)-3);
    %put &&&var;
%mend GetList;

%GetList(var=LIST1VAR, where=);
%GetList(var=LIST2VAR, where= and memname ne "TERM");

*****;
** OrderVars Macro
** DS =Dataset being passed into macro
** OUTDS =Ordered dataset produced by macro
*****;
%macro OrderVars(ds=, outds=);
    proc contents data=&ds out=alpha(keep=name) noprint; run;
    proc sql noprint; select name into :avars separated by ' ' from alpha; quit;
    data &outds;
        retain &avars;
        set &ds;
    run;
%mend;

** Puts list & number of datasets in macro variables DS NAMES and DS NUM **;
proc sql noprint;
    select memname into :dsnames separated by ' '
        from dictionary.columns
        where libname="&lib" and varnum=1
    ;
    select count(memname) into :dsnum
        from dictionary.columns
        where libname="&lib" and varnum=1
    ;
quit;

%put &dsnames;
%put &dsnum;
```

```

** Loops through library **;
%let i = %eval(1);
%do %while (%eval(&i) <= %eval(&dsnum));
  %let thisds = %scan(&dsnames,%eval(&i));

  ** Gets dataset label for each dataset **;
  proc sql noprint;
    select memlabel
      into :dsl
      from dictionary.tables
      where libname="&lib" and memname="&thisds"
  ;
  quit;

  ** Trims label in macro variable **;
  data _null_;
    length temp $200;
    temp="&dsl";
    call symput('dsl2',cats(temp));
  run;

  ** Adjusting DS labels **;
  %if &thisds=DATA1 %then %let dslabel=This is the label for DATA1;
%else %if &thisds=DATA2 %then %let dslabel=This is the label for DATA2;
%else %let dslabel=&dsl2;
;

** Adjusting data and metadata **;
data ds0;
  %if &thisds=DATA1 %then length LONGVAR $200;;
  set &lib..&thisds;
  label VAR1="The appropriate label"; *VAR1 label changed in all datasets;
  %if &thisds=DATA1 %then %do;
    format VAR2 myfmt.;
    label DATA1VAR="Corrected label"; *DATA1VAR label changed only in DATA1;
  %end;
  %if &thisds=DATA2 %then %do;
    drop VAR3 VAR4;
  %end;

  %if %unquote(&LIST1VAR) %then %do;
    drop LIST1VAR;
  %end;
  %if %unquote(&LIST2VAR) %then %do;
    drop LIST2VAR;
  %end;
run;

%OrderVars(ds=DS0, outds=DS1);

data &outlib..&thisds (label="&dslabel");
  set DS1;
run;

%let i = %eval(&i+1);
%end;
%mend MetaEngine;
%MetaEngine(lib=MYLIB, outlib=NEWLIB);

```