Variable Cross-Referencing Macros – Tools for When Base SAS Isn't Enough

Frank C. DiIorio
CodeCrafters, Inc. Chapel Hill NC

Sometimes, it's just hard to comprehend how robust the SAS tool set is – it seems like every time you have an esoteric need for data analysis or display, there's a function, procedure option, or some other nicety to make your working life productive and easy. All the marketing hype appears to be true.

But then there are the other times. You have a need for something which seems so common, so routine, that you spend an unreasonably long time going through the documentation because what you want just *has* to be there. You eventually discover that yes, indeed, there is no built-in piece of the SAS System that does what you need. Your obvious and routine task will have to be hand-crafted.

This article addresses one of the "other times." It briefly describes the functionality we needed, and then presents one of what are probably many solutions. Examples are also discussed.

The Problem

The impetus for writing the macros that we describe here was straightforward. The client's project required combining data from several studies into a single table. The data were stored in SAS data sets, but their contents varied, usually only slightly: not all variables were in all studies; a variable might have a different length or data type from one study to the next.

What we needed was a visual overview of the studies' contents. We wanted to display the variable lists for each study side-by-side, noting which studies had like-named variables. We also wanted to easily identify discrepancies in data type and/or length.

Let's look at an edited CONTENTS listing for a three tables, one per study, and each holding similar data:

study_one

#	Variable	Type	Len	Pos
1	clinic	Char	4	16
4	dob	Num	8	0
5	enroll	Num	8	8
2	patient	Char	5	20
3	race	Num	3	26
6	smoker	Num	3	29
7	treatment	Char	1	25

study_two

#	Variable	Type	Len	Pos
3	age	Num	8	0
9	age_at_onset	Num	8	16
1	clinic	Char	5	28
5	dob	Num	4	24
6	enroll	Num	8	8
2	patient	Char	5	33
4	race	Char	1	38
7	smoke	Num	3	40
8	tmt	Char	1	39

study_three

#	Variable	Type	Len	Pos
3	age	Num	3	24
1	clinic	Char	5	12
5	dob	Num	4	8
6	enroll	Num	8	0
2	patient	Char	5	17
4	race	Char	1	22
7	smoke	Num	3	27
8	tmt	Char	1	23

A visual inspection of these snippets reveals a few problems. CLINIC is character, length 4 in STUDY_ONE, but length 5 in the other tables. AGE_AT_ONSET is present only in STUDY_TWO. These and all others are not too hard to locate when there are few variables involved. Consider, however, the potential for tedium and error if many tables and/or variables have to be compared. Clearly, there is a need for automation.

The Approach

The initial design of the cross-reference utility was simple. Read the dictionary tables – SAS's meta data – and extract information about two or more data sets. Massage the data so that each variable in the "report" data set represents a table, and each observation represents a variable. If a particular study-variable combination exists in the meta data, the value in the report data set becomes "`[ok]`". If not, it is blank. So, if the studies meshed perfectly, the report data set would be a matrix filled with "`[ok]`".

Now we add a small wrinkle to the design – how do we identify type and length mismatches? Since this information is also in the meta data, extraction doesn't present a problem. Presentation is tricky, though. We want to report *mismatches*, but also don't want to overload the reader with cell after cell of data for *matches*. Essentially, we wanted to highlight the bad news and report the good news sparingly. The decision was to create variable TYPE_LEN, containing type and length. If the types and lengths for all the data sets matched, TYPE_LEN would have that information and the corresponding data set fields would be set to "`[ok]`". If we had a mismatch between two or more datasets, TYPE_LEN would be blank, and the data set fields' individual values would contain information about type and length.

Another twist that presented itself early on was what, exactly, to report from the report data set. In some situations, a listing of all variables might be helpful. In other cases, only mismatches are of interest. Other views of the data would be required as the project progressed. To provide this flexibility, a series of 0/1 indicator variables was attached to the data. These are used as filters for the REPORT procedure code that actually does the display.

The Code

The program was divided into two macros. The first, CROSS_REF_SETUP, parses the list of data sets to compare, makes sure they exist, then builds an analysis data set for each of them. The code is shown below (with numerous stylistic concessions due to space constraints):

```
%macro cross_ref_setup / parmbuff;
%global n_data;
%let line   = &syspbuff.;
%let length = %length(&line.);
%let line   = %qsubstr(&line, 2,
                       %eval(&length.-2));
%let n_data = 0;
%do num = 1 %to 99;
    %let piece = %scan(&line., &num., %str( ) );
    %if &piece. ^= %then %do;
        %if %index(&piece., .) = 0 %then
            %let piece = work.&piece.;
        %if %sysfunc(exist(&piece.)) %then %do;
            %let n_data = %eval(&n_data. + 1);
            %let libname&n_data. =
                 %upcase(%scan(&piece., 1, .)) ;
            %let memname&n_data. =
                 %upcase(%scan(&piece., 2, .)) ;
            %end;
        %end;
        %else %if &piece. = %then %let num = 100;
%end;
%if &n_data. = 0 %then %goto bottom;

proc sql;
     %do i = 1 %to &n_data.;
         create table _temp&i._ as
         select libname,
                memname,
                name,
                trim(type) || ' ' ||
```

```
               left(put(length, 5.)) as specs&i.
         from dictionary.columns
         where   libname = "&&libname&i."
               & memname = "&&memname&i."
         order by name;
     %end;
quit;
data report;
     merge %do i = 1 %to &n_data.;
               _temp&i._(in=_&i)
           %end;
           ;
     by name;
     length compare $10;
     array specs(&n_data.) $10;
     status       = 1;
     done_compare = 0;
     do i = 1 to dim(specs);
        if specs(i) ^= ' ' & ^done_compare
           then do;
           compare      = specs(i);
           done_compare = 1;
           end;
        if specs(i) ^= ' ' &
           specs(i) ^= compare then do;
           status = 0;
           leave;
           end;
     end;
     if status = 1 then do;
        do i = 1 to dim(specs);
           if specs(i) ^= ' ' then do;
              type_len = specs(i);
              specs(i) = '[ok]';
              end;
        end;
        end;
     if sum(of _1-_&n_data.) = &n_data. then do;
        inall    = 1;
        notinall = 0;
        end;
        else do;
             inall    = 0;
             notinall = 1;
             end;
     if type_len = ' ' then mismatch = 1;
        else                mismatch = 0;
     if ^mismatch & inall  then matchall = 1;
        else                    matchall = 0;
     if ^mismatch then matchfound = 1;
        else          matchfound = 0;
     label %do i = 1 %to &n_data.;
               specs&i.="&&libname&i &&memname&i"
           %end;
           type_len = "Specs"
           ;
run;
%bottom: ;
%mend;
```

The macro accepts blank-delimited data set names as its parameter. We use the EXIST function to test for the presence of the data set. If found, we increment N_DATA and create macro variables LIBNAMEn and MEMNAMEn. We are, in effect, building two arrays, each N_DATA elements in length.

The macro then constructs N_DATA data sets using CREATE TABLE statements in SQL. Data sets _TABLE1_ to _TABLE_n contain library, member, and variable names, and TYPE_LEN, which looks like "`char  40`", "`num  8`", and so on. The last piece of the process is the DATA step creating data set REPORT. Here we combine the tables and create the indicator variables used for filtering. REPORT can now be used as input to display, or other, procedures.

The second macro in the program, CROSS_REF_USE, accepts an optional filtering parameter and writes an HTML file containing a report.

```
%macro cross_ref_use(type=);
%let type = %upcase(&type.);
%if &type. = MATCHFOUND %then
    %let title2 = Matching Whenever Found;
    %else %if &type. = MATCHALL %then
          %let title2 = Matching in Every
                        Data Set;
    %else %if &type. = INALL  %then
          %let title2 = In Every Data Set;
    %else %if &type. = MISMATCH %then
          %let title2 = Mismatches Only;
    %else %if &type. = NOTINALL %then
          %let title2 = Missing From At
                      Least One Data Set;
    %else %let title2 = All Variables Were
                        Selected;
proc report headline nowindows
     data=report
          %if &type. ^= %then (where=(&type.=1)) ;
     split=' ';
columns name type_len
        %do i = 1 %to &n_data.;
           specs&i.
        %end;
        ;
%do i = 1 %to &n_data.;
   define specs&i. / display center ;
%end;
title "Variable Comparisons Taken at &systime. on
&sysday., &sysdate.";
%if &title2. ^= %then title2 "&title2." %str(;) ;
run;
%mend;
```

The process is straightforward. Assign TITLE2 on the basis of TYPE, then filter REPORT, created by macro CROSS_REF_SETUP, and create the report. The general-purpose nature of REPORT allows multiple calls to CROSS_REF_USE with only one call to CROSS_REF_SETUP. Valid report types (parameter TYPE values) are:

- MATCHFOUND Restrict the listing to variables with matching characteristics in at least one of the data sets. Contrast with MATCHALL, below.
- MATCHALL Restrict the listing to variables with matching characteristics in each of the data sets. Contrast with MATCHFOUND, above.
- INALL Restrict the listing to variables found in every data set, regardless of characteristic matching status.
- MISMATCH Restrict the listing to variables with type and/or length mismatches in two or more data sets.
- NOTINALL Restrict the listing to variables that are not found in every data set, regardless of characteristic matching status.

If TYPE is not assigned, it defaults to a null value. This creates a listing of all variables in the data sets.

Examples

Suppose we want to list all variables

```
ods html body= "C:\papers\var_xref\var_xref.htm";
%cross_ref_use
ods html close;
```

Output, using our sample data, is found in Figure 1. Variables are displayed in alphabetical order, one variable per row. A blank SPECS value signals type/length conflicts between in two or more data sets (AGE, CLINIC, DOB, and RACE have this problem). The presence of a matching variable in a data set is indicated by "`[ok]`", while present, but conflicting variables, have the type and length for that variable in the data sets.

Figure 1: Using Default Value of TYPE

Variable Comparisons Taken at 09:49 on Thursday, 07FEB02
All Variables Were Selected

Column Name	Specs	WORK STUDY_ONE	WORK STUDY_TWO	WORK STUDY_THREE
age			num 3	num 8
age_at_onset	num 8			[ok]
clinic		char 4	char 5	char 5
dob		num 8	num 4	num 4
enroll	num 8	[ok]	[ok]	[ok]
patient	char 5	[ok]	[ok]	[ok]
race		num 3	char 1	char 1
smoke	num 3		[ok]	[ok]
smoker	num 3	[ok]		
tmt	char 1		[ok]	[ok]
treatment	char 1	[ok]		

To create a report containing variables that are found in every data set, we enter the following code. Output is in Figure 2.

```
ods html body= "C:\papers\var_xref\var_xref.htm";
%cross_ref_use(type=matchfound)
ods html close;
```

Figure 2 Variables Found in Every Data Set

Variable Comparisons Taken at 09:49 on Thursday, 07FEB02
In Every Data Set

Column Name	Specs	WORK STUDY_ONE	WORK STUDY_TWO	WORK STUDY_THREE
clinic		char 4	char 5	char 5
dob		num 8	num 4	num 4
enroll	num 8	[ok]	[ok]	[ok]
patient	char 5	[ok]	[ok]	[ok]
race		num 3	char 1	char 1

Finally, to list nothing but mismatches data types and lengths, we enter the following. Output is found in Figure 3.

```
ods html body= "C:\papers\var_xref\var_xref.htm";
%cross_ref_use(type=mismatch)
ods html close;
```

Figure 3 Display Only Mismatches

Variable Comparisons Taken at 09:49 on Thursday, 07FEB02
Mismatches Only

Column Name	Specs	WORK STUDY_ONE	WORK STUDY_TWO	WORK STUDY_THREE
age			num 3	num 8
clinic		char 4	char 5	char 5
dob		num 8	num 4	num 4
race		num 3	char 1	char 1

Contact

If you have comments, contact me at FCD1@MINDSPRING.COM.
You're welcome to use the macros, so long as attribution is made.