Paper 215-27

Making Variables Less Variable: Standardizing SAS® Data Sets

William C. Murphy
Howard M. Proskin & Associates, Inc., Rochester, NY

ABSTRACT

The new data arrives and its similar to dozens of SAS data sets you processed in the past. So you can save time and effort by recycling old programs to do the analysis and generate reports. But the new data set has different variable names than those used in previous studies. In order to use the previously generated programs, you use PROC DATASETS or a DATA step to resolve the variable naming differences. But then another data set arrives for a similar study with yet another different set of variable names. Again you use PROC DATASETS or a DATA step to resolve the differences. Then another data set arrives and … (How many names can they think of for the same variable?!). Enough! A simple macro using a PROC FSEDIT window can be used to review each new data set and enforce a standard naming convention. Furthermore using a PROC FORMAT lookup table, the macro can make suggestions at what the variable names should be and what variables should or shouldn't be included in the data set.

INTRODUCTION

As a small statistical consulting firm in western New York, we have developed ongoing relationships with a number of clients. For these clients, we build and analyze databases from data supplied by laboratory or clinical studies of a variety of consumer and medical products. In several cases, the types of data supplied and the analysis requested are nearly identical to projects that we have worked on in the past. Therefore, it should be only a small effort to use old programs to conduct our work. However, even similar studies from the same client have different variable names. In fact, even in dozens of similar studies from the same client we have received data sets with different names for the same variables! To use our old programs, our first step must be to change these variables to standard names that we use in all studies.

OLD WAY

When we receive new data sets from a client, the first thing that we do is performing a series of PROC CONTENTS to determine the names of the variables in each of the data sets. We compare these names to the study protocol and our in-house naming conventions. Then we make a list of the variable names that disagree with the standard names that we use in our programs. We type these names into a RENAME statement that is contained in a PROC DATASETS or a DATA STEP in a SAS program. After running this program, we can then proceed with the project using our old, well-tested programs.

Even thought this method works just fine, it involves too many steps that can lead to errors in our database. Furthermore, although there are different variable names supplied each time, not all the names supplied are necessarily original. In fact, after several similar studies with the same client we have developed a look-up table, mapping a variety of their variable names to our preferred variable names. Our goal is to automate this process as much as possible: review the variable contents of a data set, rename variables with data from our lookup table, and rename variables that are not in the lookup table. To pursue this goal, and since macros allow for ease of automation, all code discussed below will be contained within a macro program called %rename.

LOOK-UP TABLE

To generate a look-up table that maps various permutations of variables names to our preferred names, we decided to use a PROC FORMAT statement:

```
proc format;
    value $MyName
        /*  Project  */
        'projnum'     = 'project'
        'study'       = 'project'
        'studyid'     = 'project'
        /*  Subject */
        'patnum'      = 'subject'
        'ptid'        = 'subject'
        'subjnum'     = 'subject'
        …;
```

This PROC FORMAT maps the names they supply in their data sets (left side) to our standard names of 'project' and 'subject'. The actual list we use in our studies is well over a hundred lines long. We delineate separate variable groups by comments with the preferred variable names, so that we may easily add new names for the same parameter without confusion or duplication.

In addition to these formats, we also create one informat:

```
    invalue $drop
        'y','Y'   =   'Y'
        other     =   ' ';
```

and a format for variables that we wish to ignore:

```
    value $MyDrop
        'entryid'='Y'
        'entrydate'='Y'
        other=' ';
```

These formats and informat will be use to drop variables that are not needed in our database. This is necessary because the source of the data sets typically put in some entry and audit control variables that only have meaning to them.

In actual production, this look-up table is created outside of the %rename macro. The CNTLOUT option is used with the PROC FORMAT to save the table as a SAS data set in our central library. To call up the table inside the macro program, we simply write

```
proc format cntlin=biblio.vartab;
    run;
```

This method allows us to compile our macro program and save it to our central library [Murphy, 1998]. Then, when we make a change to our format look-up table, we do not need to recompile and resave our macro program.

VARIABLE LIST

The next step in our process is to make a one-observation data set containing variable names and drop parameters:

```
data __1;
    id=open("&data");
    nvar=attrn(id,'NVARS');
    call symput('NVar',nvar);
    if nvar>&MaxVar then
        nvar=&MaxVar;
    array Old{&MaxVar} $12;
    array New{&MaxVar} $12;
    array Drop{&MaxVar} $1;
    do i=1 to nvar;
        Old[i]=varname(id,i);
        New[i]=put(Old[i],$MyName.);
        Drop[i]=put(Old[i],$MyDrop.);
        end;
    id=close(id);
    drop id i ;
    run;
```

where Old1, Old2, … are the variable names supplied in our client's data set; New1, New2, … are the standard variable names that we use in our data base; and Drop1, Drop2, … are parameters use to determine if we should keep a variable. We use the SCL command OPEN to choose the data set referred to by the macro &Data. The SCL function ATTRN is used to determine the number of variables in the data set and the function VARNAME is used to acquire the names of the variables. The PUT statement operating on the 'old' variable name is used to assign a 'new' name and a 'drop' parameter from our format look-up table. If the user desires variables with more than 12 characters in their name then the variable lengths in the arrays must be changed. The macro variable &NVar controls the number of rows used in our screen editing; it can not be larger than what the screen was designed for (&MaxVar).

FULL SCREEN EDITING

Once we have this list, we must review the list of variables, the suggested new names, and the suggested drops. To do this we use a screen from PROC FSEDIT:

```
proc fsedit data=__1
        screen=biblio.vistas.rename nr=36;
    informat Drop1-Drop30 $drop. ;
    run;
```

The user screen is a simple three columns screen. The first column has the variables names from the data sets as we have received them (the 'old' names). The second column initially is a list of the suggested names (the 'new' names). The third column initially is a list which flags the suggested parameters we do not need. The informat limits the value of the drop parameters to 'Y' or missing. The actual screen (biblio.vistas.rename) that we created contains an attached program to add color and protection to various screen elements (see Appendix A), but can be readily customized to the user's desires. We can now use this screen to make the changes that our suggestions do not cover and overrule the suggestions that are inappropriate.

REVISED VARIABLE LIST

We now use our data set from our PROC FSEDIT session to create a list of the old names (OldList), new names (NewList), and a the names of variables to be dropped (DropList):

```
data _null_;
    set __1;
    length oldlist newlist droplist $300;
    array new{&NVar};
    array old{&NVar};
    array Drop{&NVar};
    DropList='';
    NewList='';
    OldList='';
    NRename=0;
    do i=1 to dim(new);
        if Drop[i]='Y' or Drop[i]='y' then
            droplist=trim(droplist)||' '||left(old[i]);
            else if new[i] ne '' and new[i] ne old[i]
                then do;
                    NRename+1;
                    OldList=
                        trim(OldList)||' '||left(Old[i]);
                    NewList=
                        trim(NewList)||' '||left(New[i]);
                end;
            end;
    call symput('DropList',DropList);
    call symput('NewList',NewList);
    call symput('OldList',OldList);
    call symput('NRename',NRename);
    run;
```

where we have used the concatenation operator to form our lists. Then using CALL SYMPUT the lists are stored in macro variables. We have generated macro variable containing a list of variables to be dropped (&DropList); a list of old variable names (&OldList) to be renamed; a list of the new variable names to be used (&NewList); and a count of the number of variables to be renamed (&NRename).

NEW DATA SET

Finally, we apply our generated macro variables to the supplied data set and save the result under the same name in a new library:

```
Data new.&data;
    set &data;
    %if &DropList ne %then %str(drop &DropList;);
    %do j=1 %to &NRename;
        %let alpha=%scan(&OldList,&j,' ');
        %let omega=%scan(&NewList,&j,' ');
        %str(rename &alpha=&omega;);
        %end;
    run;
```

where we have generated a series of drop and rename statements base on the values contained in our macro variables. The %IF statement inserts a DROP statement into the data step if a drop list exists. The %DO loop goes through the list of old and new variable names using the %SCAN function and inserts the appropriate RENAME statements.

MACRO DETAILS

The above procedures and data steps describe the heart of our rename macro. However, some additional points should be made. Good form dictates that the macro program should be self-contained. Therefore, we explicitly declare the macro variables created in this program as local;

```
%local maxvar DropList OldList NewList
       NRename i alpha omega NVar;
```

We also define the maximum number of variables (i.e rows) that our screen can hold:

```
%let maxvar=30;
```

This value can readily be varied to fit your own need, but any change would necessitate a change to the screen.

Finally, we should clean up the data sets and formats used in the macro when it is finished:

```
proc datasets nolist library=work;
    delete __1;
    run;
    quit;

proc catalog catalog=formats;
    delete drop.infmtc MyDrop.fmtc
                       MyName.fmtc;
    run;
    quit;
```

CONCLUSIONS

The macro %rename has streamlined the creation of our database from the data sets supplied by our clients. It allows easy review of the variable in the supplied data sets; a direct method for standardizing the variable names; and a procedure for disposing of unneeded variables.

APPENDIX A

The following is a listing of the program attached to out PROC FSEDIT screen that colors the old variable names (green), the new variable names (blue), and the drop list (magenta) and also protects certain areas of the screen.

```
fseinit:
  call wname('Rename or Drop Variables');
  protect old1-old30;
  unprotect new1-new30 drop1-drop30;
  return;


init:
  array  xo{30} $5 _temporary_
   ('old1'  'old2'  'old3'  'old4'  'old5'
    'old6'  'old7'  'old8'  'old9'  'old10'
    'old11' 'old12' 'old13' 'old14' 'old15'
    'old16' 'old17' 'old18' 'old19' 'old20'
    'old21' 'old22' 'old23' 'old24' 'old25'
    'old26' 'old27' 'old28' 'old29' 'old30');
  array  xn{30} $5 _temporary_
   ('new1'  'new2'  'new3'  'new4'  'new5'
    'new6'  'new7'  'new8'  'new9'  'new10'
    'new11' 'new12' 'new13' 'new14' 'new15'
    'new16' 'new17' 'new18' 'new19' 'new20'
    'new21' 'new22' 'new23' 'new24' 'new25'
    'new26' 'new27' 'new28' 'new29' 'new30');
  array  xd{30} $6 _temporary_
   ('drop1'  'drop2'  'drop3'  'drop4'  'drop5'
    'drop6'  'drop7'  'drop8'  'drop9'  'drop10'
    'drop11' 'drop12' 'drop13' 'drop14' 'drop15'
    'drop16' 'drop17' 'drop18' 'drop19' 'drop20'
    'drop21' 'drop22' 'drop23' 'drop24' 'drop25'
    'drop26' 'drop27' 'drop28' 'drop29' 'drop30');
  do i=1 to nvar;
     rc=fldcolor(xo[i],'green',' ',1,12);
     rc=fldcolor(xn[i],'blue',' ',1,12);
     rc=fldcolor(xd[i],'magenta',' ',1,1);
     end;
  do i=nvar+1 to dim(xo);
     rc=fldcolor(xo[i],'white',' ',1,12);
     rc=field('PROTECT',xn[i]);
     rc=fldcolor(xn[i],'white',' ',1,12);
     rc=field('PROTECT',xd[i]);
     rc=fldcolor(xd[i],'white',' ',1,1);
     end;
  return;

main:
  return;

term:
  return;
```

APPENDIX B

The following is a complete listing of the %rename macro. To use it you must have created a format control data set (biblio.vartba) that contains the needed formats; a screen for changing variable names (biblio.vistas.rename); and a LIBNAME new to hold the output data set.

```
%macro Rename(data);

    %local maxvar DropList OldList NewList
                  NRename i alpha omega NVar;

    %let maxvar=30;

    proc format cntlin=biblio.vartab;
        run;

    %*** Create Data Set with List of Number of Variables ***;
    %*** Old Names, New Names , and Drop flags  ***;
    data __1;
        id=open("&data");
        nvar=attrn(id,'NVARS');
        call symput('NVar',nvar);
        if nvar>&MaxVar then
            nvar=&MaxVar;
        array Old{&MaxVar} $12;
        array New{&MaxVar} $12;
        array Drop{&MaxVar} $1;
        do i=1 to nvar;
            Old[i]=varname(id,i);
            New[i]=put(Old[i],$MyName.);
            Drop[i]=put(Old[i],$MyDrop.);
            end;
        id=close(id);
        drop id i ;
        run;

    %*** Get Preferences from User in Full Screen ***;
```

```
proc fsedit data=__1
                screen=biblio.vistas.rename nr=36;
        informat Drop1-Drop30 $drop. ;
        run;

%*** Create List of Drop Variables, Old Names, ***;
%*** New Names, and Number of Renames ***;
data _null_;
    set __1;
    length oldlist newlist droplist $300;
    array new{&NVar};
    array old{&NVar};
    array Drop{&NVar};
    DropList='';
    NewList='';
    OldList='';
    NRename=0;
    do i=1 to dim(new);
        if Drop[i]='Y' or Drop[i]='y' then
            droplist=trim(droplist)||' '||left(old[i]);
            else if new[i] ne '' and new[i] ne old[i]
                then do;
                    NRename+1;
                    OldList=
                        trim(OldList)||' '||left(Old[i]);
                    NewList=
                        trim(NewList)||' '||left(New[i]);
                end;
            end;
    %*** Convert Lists to Macros ***;
    call symput('DropList',DropList);
    call symput('NewList',NewList);
    call symput('OldList',OldList);
    call symput('NRename',NRename);
    run;

%*** Copy Data Set to New Library and ***;
%*** Apply Drops and Renames ***;
data new.&data;
    set &data;
    %if &DropList ne %then %str(drop &DropList;);
    %do j=1 %to &NRename;
        %let alpha=%scan(&OldList,&j,' ');
        %let omega=%scan(&NewList,&j,' ');
        %str(rename &alpha=&omega;);
        %end;
    run;

%*** Clean Up ***;
proc datasets nolist library=work;
    delete __1;
    run;
    quit;

proc catalog catalog=formats;
    delete drop.infmtc MyDrop.fmtc
                        MyName.fmtc;
    run;
    quit;

%mend;
```

REFERENCES

Murphy, W. C. (1998), "Creating and Maintaining a Central SAS® Library for Health Care Management", *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*, Cary, 23, 1128-1130.

CONTACT

William C. Murphy
Howard M. Proskin & Associates, Inc.
2468 E. Henrietta Rd.
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com