Paper 100-2008

How to Avoid Overwriting Work Data Sets and Work Formats - An Essential Macro Design Topic

Zizhong (James) Fan, MedImmune, Gaithersburg, MD

ABSTRACT

How to keep interim datasets or formats generated in macro programs from unexpectedly overwriting work data sets or work formats that are outside of the macro calls is a topic sometimes being overlooked. It can create confusions and sometimes lead to misleading results. This is an important macro design issue when developing public use or utility macro applications. This paper introduces an efficient technique that can be utilized to avoid conflicts between local macro data sets or formats and non-macro data sets or formats.

INTRODUCTION

When it comes to macro application development, one of the common practices is to set up separate macro symbol tables to avoid unexpected interactions between local macro variables across different macro calls in the same SAS® session as well as to avoid conflicts with global macro variables. This can be done by using %local statement. But how to keep temporary data sets or formats generated in macro programs from unexpectedly overwriting work data sets or work formats that are outside of the macro calls is a topic sometimes being overlooked. This is a potential problem that can create confusions and sometimes lead to misleading results. This is an important macro design issue when developing public use or utility macro applications that have potentially large number of users.

Here is one simple and hypothetical example. Let's say we have a table generator macro %AeTab for creating adverse event (AE) tables. After steps of data manipulation, we have an input data set called AeReport.

Hypothetical Example

```
data ….. run;
proc ….. run;
 …
data AeReport;
  …..
run;

%AeTab (indsn=AeReport, grpvar=trtgrp, subtitle=Related AE)
%AeTab (indsn=AeReport, grpvar=trtgrp, whr=%str(aetype=2), subtitle=Serious AE)
        …
%AeTab (indsn=AeReport, grpvar=trtgrp, whr=%str(relaten in (3 4 5), subtitle=Related AE)
```

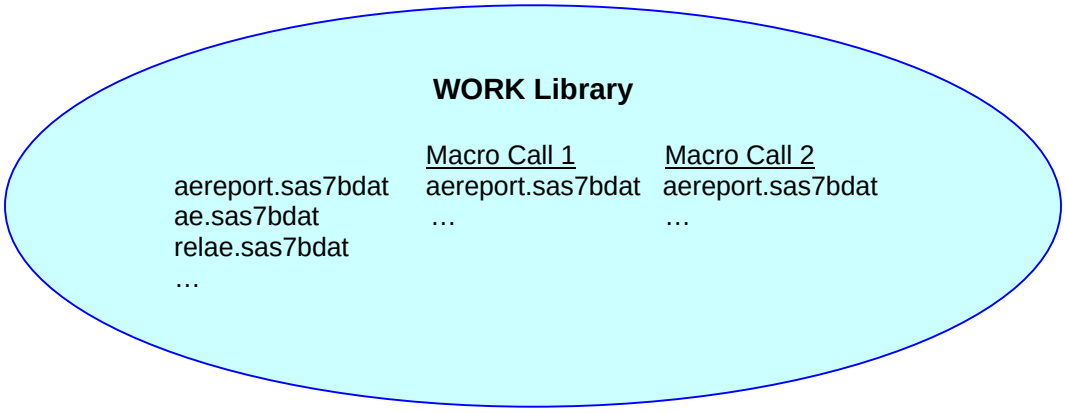
If we call the macro just one time in one SAS session to create one AE table, it works fine. But if we make multiple macro calls to create multiple tables in one SAS session, we could have problems. What if there is a data set called AeReport generated in the %AeTab? This input data set will be overwritten after the first macro call. The later macro calls will not work properly. This is a simple example of a project oriented macro for the purpose of illustrating the potential problem. And the real world situations could be much more trickier, especially when developing public use or utility macro programs. Plus, SAS already has restrictions on data set and format names. Users should not have another layer of restrictions in using data set or format names. From a developer's perspective, we should make our product robust and work-data-set-overwritten proof. We certainly do not want to put in a list of data set names or format names that should not be used in the user's menu.

SOLUTION

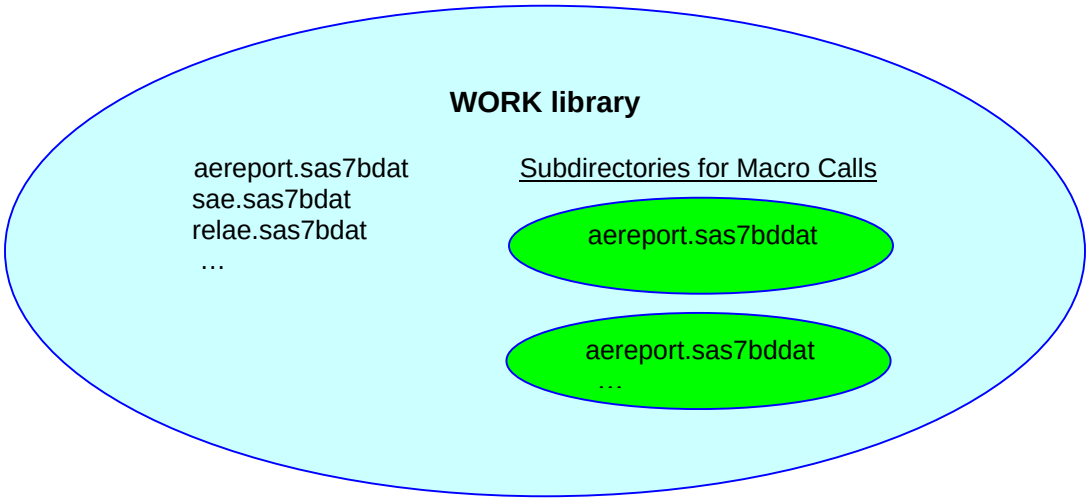
There are different ways to tackle this problem. Some people may use a lot of underscores and try to be as creative as possible to come up with unusual data set names in their macro programs to reduce the chance of conflicting. But there is always a chance that the data sets with underscores or strange names can find their 'name twins' in the WORK library in the 'free world'. Remember we human beings are equally creative. The names that you can think of, others can do the same. Some people may check existing WORK data set names or format names against data set and format names in the macro. This requires additional comprehensive programming planning, especially when the macro program is data driven and generates a huge number of interim data sets and formats. Another common practice is to delete all the temporary data sets created within the macro program at the end of the macro run. This even could delete work data sets with the same names as the local (macro) data sets.

The clean and hassle free solution this paper introduces is creating a subdirectory under the current SAS session WORK library directory and designate it as USER library to store all the interim data sets and the format catalog generated in macro program. The design is illustrated in the figures bellow.

No Subdirectory and Risk of Overwriting



Subdirectory and No Risk of Overwriting



This approach should be implemented at the very top of the macro program before creating any interim data sets or formats. And at the very end of the macro program, remove the subdirectory and reset the USER library to the original directory. The beauty of this approach is that there will never be a chance that the inner work world can collide with the outer work files.

Here's the sample code illustrating the above design in the MS Windows® environment.

Sample Code 1

```
/*============== Step 1==============*/
%local loopn subwork origxwt origuser origfmt;

/*============== Step 2==============*/
%let loopn = 0;
%do %while (%sysfunc(libref(subw&loopn)) = 0);
  %let loopn = %eval(&loopn + 1);
%end;
%let subwork = subw&loopn;

/*============== Step 3==============*/
%let loopn = 0;
%do %while
  (%sysfunc(fileexist(%sysfunc(pathname(work))\subw&loopn)) ne 0);
   %let loopn = %eval(&loopn + 1);
 %end;

/*============== Step 4==============*/
 %let origxwt = %sysfunc(getoption(xwait)) %sysfunc(getoption(xmin))
      %sysfunc(getoption(xsync));
 %let origfmt = %sysfunc(compress(%sysfunc(getoption(fmtsearch)), ')('));
 %let origuser = %sysfunc(getoption(user));
 options noxwait xmin xsync fmtsearch=(user &origfmt);

/*============== Step 5==============*/
 options noxwait xmin xsync;
 x "md ""%sysfunc(pathname(work))\subw&loopn""";
 libname &subwork "%sysfunc(pathname(work))\subw&loopn";
 options user=&subwork;

/*==============================*
 |       CORE MACRO CODE        |
 *==============================*/

/*============== Step 6==============*/
 libname &subwork clear;
 x "rd/s/q ""%sysfunc(pathname(&subwork))""";

/*============== Step 7==============*/
 %if %length(&origuser) ne 0 %then %do;
   options user=&origuser;
 %end;

 %if %length(&origuser)=0 %then %do;
   options user=work;
 %end;

 options &origxwt fmtsearch=(&origfmt);
```

Let's break it down to 7 steps and take a closer look.

- In step 1, define all the macro variables local in order to void interactions with the other macro variables outside of the current macro call.

- In step 2, check and create a new libname that has never been used in the current SAS session and put it into the macro variable &subwork.

- In step3, find out the directory for the WORK library and create a folder name that has not been created, and put it into the macro variable subw&loopn.

- In step 4, capture and store the current system options xwait, xmin, xsync, user and fmtsearch. These options will be reset in step 7.

- In step 5, set system options as noxwait xmin xsync in order to run operating system commands to create the subdirectory for storing the data sets and formats that will be created in the current macro call. In Appendix, figure 1 shows the subdirectory is library SUBW0 with a path of C:\SASWork\SasTempFiles_TD3372\subw0. The actual WORK library for this session is one level up - C:\SASWork\SasTempFiles_TD3372. And the USER library reflects the same path as SUBW0. Now the macro has its own independent and isolated space where we can store all the temporary data sets and formats generated by the macro program. We can use any data set names or format names we want without worrying about overwriting existing data sets or formats. And after the core macro code between step 5 and step 6, we can move on to the bottom of the macro program to remove the temporary subdirectory and all its contents.

- In step 6, clear the libname and delete the subdirectory.

- In step 7, reset all the system options including USER library and fmtsearch to their original values. Figure 2 shows that library SUBW0 is cleared and the USER is the same as WORK.

This sample code can be modified to fit personal preferences. For example, if you are not familiar or comfortable with using the USER library, you can ignore the part of assigning USER to the subdirectory. Instead, you need to use &subwork. as the libref in front of all the temporary data sets in the macro program to alert yourself that you are reading and writing in a different WORK library. More information about USER library can be found at
http://support.sas.com/onlinedoc/913/docMainpage.jsp

Another possible alternation is modulating step 1 to step 5 into one utility macro program, for example called %entry, and step 6 and 7 into another, maybe called %exit. Then you can just call this pair of utilities at the top and bottom respectively when developing any macro applications.

```
%macro AeTab;
    %entry
     …MyOwnMacCode…
    %exit
%mend AeTab;
```

One special attention should be given when creating temporary formats. In order to store formats into the user library, LIBRARY=USER has to be used in the macro code. The sample code below modifies format labels by adding group population numbers. It is a data driven process, which means any variable with format that plugs in can be modified to show the original formatted value with the added population numbers.

Sample Code 2

```
proc sql noprint;
 select format into :v_fmtname
     from dictionary.columns
      where libname="USER" and memname="%upcase(&inputdsn)" and
       name="%upcase(%scan(&by_var, -1))";
 quit;

 proc sql noprint;
  create table totn as
     select distinct %scan(&by_var, -1) as start, "&v_fmtname" as fmtname,
         trim(put(%scan(&by_var, -1), &v_fmtname. -l))||
          " (N="||trim(put(totn, best. -l))||")" as label
      from report;
quit;

proc format library=user cntlin=totn;
run;
```

Let's say the last &by_var variable is treatment group. It's a numeric variable originally formatted by WORK format TRTGRP as 0=' PLACEBO', 1= 'DOSE 1 ', and 2='DOSE 2'. What the sample code 2 does is: First, it retrieves the format name from the dictionary table COLUMNS. Then it prepares a data set to be used to replace the existing format for the target variable, in this case, the last variable in the &by_var list - TRTGRP. Then it uses PROC FORMAT CNTLIN to create the modified format into the USER library without touching the original (work or permanent) format. After the current macro call, the USER library, which is in the temporary subdirectory under WORK, will be deleted so that the original format labels can be modified again from the source format, not the modified (USER) one. The result table will have formatted values as:

First macro call for ITT Population

PLACEBO (N=318)
DOSE 1 (N=320)
DOSE 2 (N=325)

Second macro call for Safety Population

PLACEBO (N=310)
DOSE 1 (N=317)
DOSE 2 (N=324)

Fail to store the modified format in the temporary subdirectory could have consequences of changing the outside original format as well as ending up with result as following:

First macro call for ITT Population

PLACEBO (N=318)
DOSE 1 (N=320)
DOSE 2 (N=325)

Second macro call for Safety Population

PLACEBO (N=318) (N=310)
DOSE 1 (N=320) (N=317)
DOSE 2 (N=325) (N=324)

Third macro call for PK Evaluable Population

…

CONCLUSION

Avoiding overwriting existing WORK data sets or format catalog is an important macro design issue. Creating and designating a subdirectory under WORK library is a simple and efficient solution to avoid overwriting existing data sets or formats in the same SAS session. This technique should be recommended and introduced to all macro application developments. And the benefits include:

1. No possibility of overwriting any existing data sets or formats.
2. No extra storing space is needed because this approach is creating a subdirectory under the WORK library umbrella. The WORK library will be used anyway. It just separates the macro working space away from the rest of the WORK library.
3. No extra typing is needed by designating USER library to the subdirectory. Otherwise, all the interim data sets created in macros requires a libref in the front.
4. No extra thinking about creating creative and dummy data set or format names is needed. We can use any meaningful data set names within the macro without worrying about overwriting existing data sets or formats outside of macros.
5. No interferences with current session option settings. The original system options are captured beforehand and restored afterwards.

REFERENCES:

SAS® online document V9.
http://support.sas.com/91doc/docMainpage.jsp

SAS® Guide to Macro Processing, Version 6, Second Edition; SAS Institute, Cary, NC, USA

CONTACT INFORMATION

Zizhong (James) Fan
Clinical Data Analyst III
Phone: (301) 398-5264
Fax: (301) 398-8264
Email: fanz@medimmune.com

MedImmune, Inc.
One MedImmune Way
Gaithersburg, MD 20878

DISCLAIMER

Appendix

Figure 1: Subdirectory created under WORK library directory

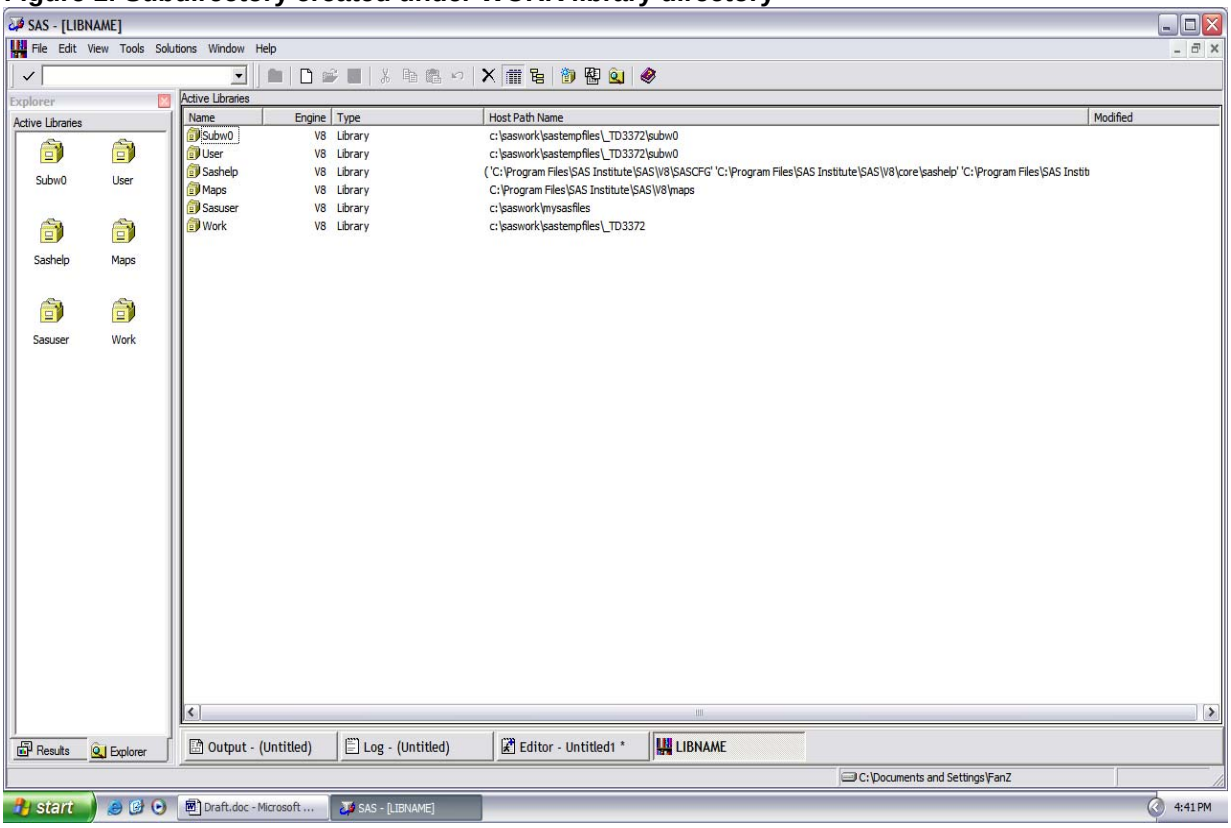


Figure 2: Subdirectory removed and USER library reset to WORK library