ABSTRACT
The number 1 rule of camping is “leave no trace”. When you leave a campsite, there should be no evidence remaining that you were ever there. As the author of an application or utility, you need to be sensitive to the SAS® environment in which you are operating. Your application should leave nothing behind, other than expected results, that shows it was ever there.

Your application may need to change certain environmental characteristics, generate some datasets, perhaps some formats, probably some macro variables, and maybe open some libnames or filerefs.

Items such as these left behind by a program can adversely affect the SAS environment in which it was called. Things like datasets, macros, formats, and macro variables left behind can clutter and confuse the environment. Changed SAS options can cause significant problems to subsequently running steps.

This paper discusses a set of macros that are designed to “remember” the environment at the start of your application or utility and then restore that environment at the completion of the program. Simply put, these macros maintain a list of existing SAS datasets, open libnames, open filenames, options, macros, macro variables, formats, informats, titles and footnotes, and then restores the environment and deletes anything the application created during its execution, effectively leaving no trace.

INTRODUCTION
“There are 100 different ways to do everything in SAS.” This paper describes only one way, which by no means suggests that it is the only way. The reader should feel free to explore other techniques that SAS offers to do the same things shown in this paper.

The programs described in this paper exist and have been validated; however the code is proprietary and cannot be shared in its entirety. The techniques and code shown herein offer the reader a glimpse into the overall program and process. The reader is encouraged to use the information here to build their own version of the program.

The programs and techniques shown in this paper are relative to SAS version 9.1.3.

The term ‘restore’ that is used throughout this paper does not imply that datasets, macro variables, or other ‘remembered’ objects will be restored to their previous state, but rather objects added since then will be removed. Some objects like options, titles, and footnotes can truly be restored to their previous state.

The ‘saving’, ‘remembering’, ‘restoring’, ‘cleaning up’ in this paper refer specifically to the WORK directory of the SAS session. Any permanently stored objects are unaffected.

OVERVIEW
Object
Clean-up anything your program created and restore options, titles, and footnotes to their previous state.

Process
1) Record SAS environment as it exists before program execution.
2) After execution compare environment to before state and remove anything new and restore options, titles, and footnotes to their previous values.
Software
The process consists of two programs: START_UP & CLEAN_UP.

The restoration phase, CLEAN_UP, is more complex than START_UP, the recording phase.

Stored Data Naming Convention
All the macro variables and the single dataset name generated by this process are uniquely named using the current datetime, a fixed character string, and a description of the contents. The fixed character string is used to easily identify all the related macro variables and datasets at the time of clean-up and restoration of stored information. For example, the fixed character string might be '_user_'. Then at 9:15 am on Monday, May 24, 2010, the libnames might be stored in a macro variable named _1590311700_user_libnames, where 1590311700 is the current datetime. This paper will refer to these uniquely named macro variable and dataset names using a macro variable reference to represent the datetime such as &dt._user_libnames where &dt equals an underscore and the SAS datetime value, for example _1590311700.

This naming convention supports nesting of the START_UP and CLEAN_UP process because the CLEAN_UP step will look for the most recent (i.e. largest) datetime valued set of macro variables and datasets to restore the environment. This is based on the assumption that the CLEAN_UP step is restoring the most recently stored environment.

In the event the START_UP and CLEAN_UP get out of synchronization because unequal number of START_UP calls and CLEAN_UP calls have been issued or some step deleted the stored data, the CLEAN_UP step should issue a warning indicating that stored data could not be located and offer suggestions how that might have occurred.

Dictionary Tables
Using Proc SQL and the dictionary tables is far more efficient than using the data step and the sashelp views. Accessing tables is always more efficient than accessing views. Only Proc SQL can access the tables directly through the dictionary libname.

When you query a DICTIONARY table, SAS gathers information that is pertinent to that table. Depending on the DICTIONARY table that is being queried, this process can include searching libraries, opening tables, and executing SAS views. Unlike other SAS procedures and the DATA step, PROC SQL can improve this process by optimizing the query before the select process is launched. Therefore, although it is possible to access DICTIONARY table information with SAS procedures or the DATA step by using the SASHELP views, it is often more efficient to use PROC SQL instead. – SAS 9.1.3 Language Reference: Concepts - Dictionary Tables and Performance (Online Documentation)

Therefore example code in this paper will use Proc SQL and the DICTIONARY libname to access the dictionary tables.

There are a number of good papers on the SAS dictionary tables and several have been referenced in the RECOMMENDED READING section of this paper.

START_UP
The START_UP macro needs no parameters. START_UP uses as input the existing SAS environment. It collects a list of datasets, macros, macro variables, formats, informats, libnames, filenames, current option settings, titles, and footnotes and stores them. The outputs from START_UP include a number of uniquely named macro variables and one uniquely named work dataset.

Macro Variables
The dictionary table macros stores the current macro variables, user and system, local, global, and automatic. Selecting the macro variable name where the scope is not ‘AUTOMATIC’ will collect all the user macro variables. If a macro variable exists in both the local and global symbol table, the distinct function will select a single name effectively preserving both. If the macro variable exists in only one symbol table and the program adds a macro variable of the same name to the other symbol table, the additional macro variable will not be cleaned up. There are a few macro variables that should be excluded: SQLOBS, SQLOOPS, SQLXOBS, SQLRC, and SYSODSPATH.
These macro variables appear as a result of using Proc SQL.

```sql
proc sql;
    select distinct name into :&dt._user_macrovars
    from   dictionary.macros
    where  scope ^= 'AUTOMATIC'
    and    name not in('SQLOBS', 'SQLOOPS', 'SQLXOBS', 'SQLRC', 'SYSODSPATH');
```

Datasets
The dictionary table `tables` will provide the user datasets in the libname 'WORK'.

```sql
proc sql;
    select memname into :&dt._user_datasets
    from   dictionary.tables
    where  libname = 'WORK';
```

Titles and Footnotes
The dictionary table `titles` will provide the titles and footnotes. This data should be collected as a dataset instead of a macro variable because multiple components are required to identify titles and footnotes and their order. The variables in this dictionary table are type, number, and text. Certainly it is possible to store this data as a series of macro variables, but using a dataset is more concise.

```sql
proc sql;
    create table &dt._user_titles as
    select *
    from   dictionary.titles
    order by type, number;
```

The `order by` clause is used to ensure the dataset is in type and number order for ease of restoration by the CLEAN_UP macro.

Libnames
The dictionary table `libnames` will provide all the open libnames

```sql
proc sql;
    select libname into :&dt._user_libnames
    from   dictionary.libnames;
```

Filenames
The dictionary table `extfiles` provides the open filerefs. Care needs to be given to not collect the #LN files which are temporary files created by SAS and not user files. The code below and similar code in the CLEAN_UP phase avoids the #LN files. In both cases the Proc SQL wildcard % is used to represent 'any number of other characters'.

```sql
proc sql;
    select fileref into :&dt._user_filenames
    from   dictionary.extfiles
    where  fileref not like '#LN%';
```

Macros
The dictionary table `catalogs` provides user defined macros.

```sql
proc sql;
    select objname into :&dt._user_macros
    from   dictionary.catalogs
    where  libname = 'WORK'
    and    objtype = 'MACRO';
```
Formats and Informats
The dictionary table *formats* provides user formats. The format information is best extracted in four groups, one for each of the following: character formats, numeric formats, character informats, and numeric informats. This is done because during CLEAN_UP each of the format types must be deleted separately each using its unique entry type. This will be discussed further in the CLEAN_UP section.

```sql
proc sql;
    select fmtname into :&dt._user_formats /* numeric formats */
    from   dictionary.formats
    where  libname = 'WORK'
    and    substr(fmtname,1,1) ^= '$'
    and    fmttype = 'F';

    select fmtname into :&dt._user_formatcs /* character formats */
    from   dictionary.formats
    where  libname = 'WORK'
    and    substr(fmtname,1,1) = '$'
    and    fmttype = 'F';

    select fmtname into :&dt._user_infmts /* numeric informats */
    from   dictionary.formats
    where  libname = 'WORK'
    and    substr(fmtname,1,1) ^= '$'
    and    fmttype = 'I';

    select fmtname into :&dt._user_infmtcs /* numeric informats */
    from   dictionary.formats
    where  libname = 'WORK'
    and    substr(fmtname,1,1) = '$'
    and    fmttype = 'I';
```

SAS Options
Only choice options should be saved, those most likely to influence the program operation or output, for example CENTER, PAGESIZE, and ORIENTATION.

Options like CENTER have an on or off value of CENTER or NOCENTER. They may be saved as is because no other information is needed. Options like PAGESIZE expect a value such as PAGESIZE=63. These options should be saved with the option name and value so that they can be restored without further manipulation. ORIENTATION also requires a value, but on some machines restoring the ORIENTATION value noticeably slows down the process. Therefore ORIENTATION should be stored in a separate location so it may be conditionally restored which will be discussed further in the CLEAN_UP section.

SAS options are stored in the dictionary table *options*.

```sas
data _null_;
    length _options_ $500;
    retain _options_;
    set sashelp.voption end=last;
    where optname in ('CENTER'    'DATE'  'FONT'   'LINESIZE'    'MLOGIC'
                    'MPRINT' 'NOTES' 'NUMBER' 'ORIENTATION' 'PAGESIZE'
                    'SYMBOLGEN' 'XSYNC' 'XWAIT');
    if optname in ('PAGESIZE' 'LINESIZE' 'FONT') and setting ^= ''
        then _options_ = trim(_options_) || ' ' || trim(optname) ||
            '=' || trim(setting);
    else if optname = 'ORIENTATION'
        then call symput("&dt._user_orientation",trim(setting));
    else _options_ = trim(_options_) || ' ' || trim(setting);
    if last;
    call symput("&dt._user_options",_options_);
```
The code above follows the logic described above and produces two macro variables, one with ORIENTATION only, containing only the setting, and looks like

```
landscape
```

and one with all the other options and their appropriate reset settings that looks like

```
nocenter date linesize=145 pagesize=63 number
```

SAS offers a procedure for recording and restoring the options, specifically Proc Optsave and Proc Optload. The differences between these procedures and the approach described in this paper are:

1) The procedures save and restore all options which takes more time than just a few choice options, and
2) The procedures will save and unconditionally restore the ORIENTATION option, and noticeably slow the system every time.

**CLEAN_UP**

The CLEAN_UP macro needs no parameters. CLEAN_UP uses as input the stored information recorded by the START_UP macro, specifically a number of macro variables and one work dataset. The output from CLEAN_UP is a restored SAS environment.

CLEAN_UP first must locate the data saved by the START_UP macro. If it cannot find the saved data, then a warning should be printed to the log to indicate that CLEAN_UP could not function and suggest some reasons why, specifically:

1) That there may have been more calls to CLEAN_UP than there have been to START_UP, or
2) Another process has deleted the information stored by START_UP.

Macro variables are cleaned up last because most of the saved information are stored as macro variables.

**Titles and Footnotes**

Titles and footnotes must be restored before the datasets are deleted because the titles and footnotes have been recorded as a dataset. Once the titles and footnotes dataset has been processed, it can be deleted by a subsequent step. With other restoration steps, the macro must first see what the current environment contains and then remove the new objects. In restoring the titles and footnotes there is no concern what the current values are since they are simply being reset to the previous state.

Because of how the title and footnote data are stored, a code fragment needs to be generated and run to actually restore the titles and footnotes. Each observation in the dataset include a type, 'T' for title and 'F' for footnote, a number corresponding to the title or footnote number, and the actual title or footnote text. The dataset was preserved in type and number order, so all the titles are together and in numeric order, as are the footnotes. A data _null_ can be used to create the code fragment containing the title and footnote statements. This is a straightforward and common technique that is detailed in the paper “Programming Squared” referenced in the RECOMMENDED READING section of this paper.

```
file restore temp;

data _null_;
set &dt._user_titles;
file restore;
if type = 'T'
  then rec = 'Title' || compress(put(number,2.)) || ' "' || trim(text) || '";';
else rec = 'Footnote' || compress(put(number,2.)) || ' "' || trim(text) || '";';
put rec;
run;

run;
```

```
title;
```
The null TITLE and FOOTNOTE statement remove any pre-existing values. This is necessary in the event that no titles or footnotes were stored or that the stored titles or footnotes begin at a level higher than what exists. For instance, if the current titles are

```r
  title1 'Report 7.1';
  title2 'Adverse Event Summary';
  title3 'By Event Severity';
  title4 'Open Label Phase';
```

If the stored titles were null, then no title statement would be created by the code above and the existing titles would remain. If the stored titles started at title3, then the pre-existing title1 and title2 would remain.

Once the `&dt._user_titles` dataset has been processed it can be deleted along with other datasets.

Datasets
Proc Datasets is used to delete all the datasets created since START_UP. Once again, the dictionary table `tables` is accessed, this time to see what datasets have been created since START_UP. A macro variable named `deletes` will be created that contains only the new dataset names, each separated by a single space.

```r
proc sql;
  select distinct memname into :deletes separated by " "
  from   dictionary.tables
  where  libname = 'WORK'
  and    memname not in (&dt._user_tables);
```

Then Proc Datasets can use the macro variable `deletes` to delete the datasets created since START_UP

```r
proc datasets library = work nolist;
  delete &deletes;
  quit;

%symdel deletes / nowarn;
```

The `symdel` macro function will delete the macro variable `deletes`. Because the same macro variable is used to identify all the objects to be deleted, it should be created and deleted as necessary with each step.

Libnames
Closing libnames opened since START_UP is done in a fashion similar to the titles and footnotes. A code fragment needs to be generated to close each libname, but unlike the processing of the titles and footnotes, it is not necessary to create an external file to accomplish this because the libname statements can go directly into the job stream. Title and footnote statements may also go directly into the job stream, but creating those statements is a bit more complex, thus the use of an external file to help clarify and simplify the process. For libnames the program must identify the libnames that have been opened since START_UP that still need to be closed by accessing the dictionary table `libnames` using code similar to that used for the work datasets above. The list of libnames to be deleted can be used to create the necessary statements like this:

```r
%let cnt = 1;
%do %while (%scan(&deletes,&cnt) ne  );
  libname %scan(&deletes,&cnt) clear;
%let cnt = %eval(&cnt + 1);
%end;
```

This will produce libname statements that look like this:

```r
  libname alpha clear;
  libname beta clear;
```
Filerefs
When determining what new filerefs are present since START_UP, the program needs to be certain not to get SASAUTOS or any #LN files which are SAS temporary files and not user files.

```
proc sql;
    select distinct fileref into :deletes separated by " "
    from dictionary.extfiles
    where fileref not in (&dt._user_filerefs)
    and fileref ^= 'SASAUTOS'
    and fileref not like '#LN%';
```

The list of filerefs can then run through code similar to that for the libnames producing statements like this:

```
filename alpha clear;
filename beta clear;
```

Macros
When finding the new macros created since START_UP, the program should look in the dictionary table catalogs where libname = 'WORK' and objtype = 'MACRO' and objname is not in the saved list of user macros. That list can then be used by Proc Catalog to delete the macros from the work macro library, specifically work.sasmacr.

```
proc catalog catalog=work.sasmacr;
   %let cnt = 1;
   %do %while (%scan(&deletes,&cnt) ne );
       delete %scan(&deletes,&cnt) / et=macro;
   %let cnt = %eval(&cnt + 1);
   %end;
quit;
```

This macro loop will produce statements like:

```
delete alpha / et=macro;
delete beta / et=macro;
```

Formats
Since formats were collected in four groups, they must be processed in four groups. The technique is the same for all four groups with changes in format type, format name, and entry type. For character formats

```
... where libname = 'WORK'
and substr(fmtname,1,1) = '$'
and fmttype = 'F'
and fmtname not in (&dt._user_formatcs);
```

For character informats

```
... where libname = 'WORK'
and substr(fmtname,1,1) = '$'
and fmttype = 'I'
and fmtname not in (&dt._user_infmtcs);
```

For numeric formats

```
... where libname = 'WORK'
and substr(fmtname,1,1) ^= '$'
and fmttype = 'F'
and fmtname not in (&dt._user_formats);
```
For numeric informats

... where libname = 'WORK'
and substr(fmtname,1,1) ^= 'S'
and fmttype = 'I'
and fmtname not in (&dt._user_infmts);

Then Proc Catalog can do the deletes much the same way as was done for macros except the entry type is set to
't format' for numeric formats, 'infmt' for numeric informats, 'formatc' for character formats, and 'infmtc' for character
informats.

delete %scan(&deletes,&cnt) / et=format;
delete %scan(&deletes,&cnt) / et=infmt;
delete %scan(&deletes,&cnt) / et=formatc;
delete %scan(&deletes,&cnt) / et=infmtc;

SAS Options
With other restoration steps, the macro must first see what the current environment contains and then remove the
new objects. In restoring the SAS options there is no concern what the current values are since they are simply
being reset to the previous state.

Restoring the SAS options is a two step process since the options were stored in two places. The first step is
unconditional and simple, just issue an options statement with the appropriate macro variable

options &dt._user_options;

If the macro variable was created correctly the statement above should generate code that looks something like this:

options nocenter date linesize=145 pagesize=63 number;

The second step deals with the ORIENTATION option. On some systems setting the ORIENTATION option
noticably slows the process, even if the setting is not changing. Therefore the CLEAN_UP macro should make the
resetting of ORIENTATION conditional, then the slow down is only encountered when actually necessary. Using the
SAS GETOPTION function the reset can be made conditional

%if %sysfunc(getoption(orientation)) ^= &dt._user_orientation 
  %then %do;
    options orientation = &dt._user_orientation;
  %end;

ORIENTATION was originally saved by START_UP as only the value so that it could be used in the condition above
and in the options statement, though the options statement then required the ORIENTATION= phrase be included.

Macro Variables
Because so much of the stored data are kept as macro variables, the macro variables have to be processed last. Get
the list of macro variables to be deleted by first looking in the dictionary table macros and comparing against the
saved list

proc sql;
  select name into :deletes separated by " "
  from (select name
        from dictionary.macros
        where scope ^= 'AUTOMATIC'
        and name not in ("&dt._user_macrovars")
  ...
Macro variables would be accounted for except the one actually containing the macro variable list. Therefore, the code below, in one form or another will always be necessary.

```sql
... UNION
  select name
  from dictionary.macros
  where name like "&_dt_.~_USER~_%" escape '~'
);
```

To search for actual percent or underscore characters in your text using the LIKE operator, you must use an ESCAPE clause. The ESCAPE clause in the LIKE condition enables you to designate a single character string literal, known as an escape character, to indicate how Proc SQL should interpret the LIKE wildcards (%) and _ when SAS is searching within a character string. – SAS SQL 1: Essentials Course Notes, 2009, section 2.3, slide #94.

Excellent examples using the ESCAPE clause are available in the SAS Online Documentation under SQL Procedure, LIKE Condition, Escape Clause.

LIMITATIONS AND RESTRICTIONS

This system does not control changes to pre-existing items. Therefore, if START_UP records that a dataset named DEMOG exists prior to running a program and that program alters the contents of the dataset DEMOG, there is no way to identify this, prevent it from happening, or of restoring the original data.

The START_UP and CLEAN_UP macros have been designed for use with reporting programs, permanent dataset creation programs, or other larger applications, not day-to-day utilities that produce fragments of code or SAS environment modifications such as options or titles. Objects created in the WORK directory by the calling program will be deleted by the CLEAN_UP macro. This includes datasets, macros, macro variables, titles, and other objects. Permanently stored objects are not affected by the CLEAN_UP macro.

If a macro variable exists in both the local and global symbol table, the distinct function will select a single name effectively preserving both. If the macro variable exists in only one symbol table and the program adds a macro variable of the same name to the other symbol table, the additional macro variable will not be deleted during CLEAN_UP.

CONCLUSION

It is vitally important to "clean up after yourself" when you write programs that are called by other users and programs. Leaving trash behind may adversely affect any process running after your program. Running as invisibly as possible and cleaning up will be appreciated by all using your program.

REFERENCES

- SAS Online Documentation 9.1.3.

RECOMMENDED READING

ABOUT THE AUTHOR

Jim Johnson has been programming with SAS in the Pharmaceutical Industry since 1986. He has presented at many local, regional, and national conferences and has been teaching in the SAS Certificate Program at Philadelphia University since its inception in 1997. Jim has a reputation as a “problem solver” and efficiency enthusiast. His recent work includes large SAS systems, writing programs that write programs, an SDTM compliance verification system, infrastructure programming, and advanced validation and documentation skills.

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

Jim Johnson
jimmy2960@verizon.net

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.