SAS1573-2015

Don't Be a Litterbug: Best Practices for Using Temporary Files in SAS®

Richard D. Langston, SAS Institute Inc., Cary, NC

ABSTRACT

This paper explains best practices for using temporary files in SAS® programs. These practices include using the TEMP access method, writing to the WORK directory, and ensuring that you leave no litter files behind. An additional special temporary file technique is described for mainframe users.

INTRODUCTION

Many SAS programs use temporary files. These files do not pertain to WORK data sets but to files written using the FILE statement in the DATA step and referenced by the FILENAME statement. If you use filenames that refer to permanent files, but the files are used only temporarily within the program, then you must remember to delete these files within the SAS program or they will remain after the program completes. Such files can be referred to as "litter". In this paper, we explain how to minimize your file litter and be a good SAS programming citizen.

USING THE TEMP DEVICE NAME IN THE FILENAME STATEMENT

Consider the following SAS code that creates a temporary file of test data:

```
filename myfile 'c:\temp\tempfile.txt';
data _null_;
  file myfile;
    do i=1 to 5; put i; end;
run;
```

There are at least three flaws with this program:

1) It assumes that you are running on Windows because the file systems on UNIX and z/OS would not allow filenames like c:\temp\tempfile.txt.
2) It assumes that you have a directory called c:\temp, which might not be present on another user's system.
3) You have to add code later on in the program to delete the file (otherwise, you leave litter).

Consider instead this FILENAME statement:

```
filename myfile temp;
```

The DATA step that follows is exactly the same. But now this SAS code is portable and will work equally well on UNIX and z/OS. There is no assumption about file location because the SAS host code will determine the proper location of the file for you. And the SAS host code will delete the file for you when the SAS program terminates or when the MYFILE fileref is reused or cleared.

If you need to know the actual pathname associated with the TEMP file, you can use the PATHNAME function with the fileref in order to obtain that pathname. For example, suppose you needed to call an external program called mypgm with the filename, as in this code:

```
filename myfile temp;
data _null_; x=1; y=2; file myfile; put x y; run;
x " mypgm ""%sysfunc(pathname(myfile))"" ";
```

Note the use of the double quotes. The entire string for the X command is in double quotes, and we want to quote our pathname for the mypgm invocation using double quotes as well, so we pair the double quote with another double quote. Note also that this quoting is necessary because the pathname might have embedded blanks, and without the quotes the command line might be misinterpreted. (This is especially true on Windows).

HOW TO DELETE FILES

If you need to create a file that is explicitly named, but you want to delete the file by the end of the SAS program, you can do so portably with the FDELETE function. Use this instead of the X command. For example, suppose you have created myfile.txt and want to delete it. Do not do this:

```
x 'del myfile.txt'; /* on Windows */
x 'rm myfile.txt'; /* on UNIX */
x 'del myfile.txt'; /* on TSO on z/OS - cannot do this in batch! */
```

Instead do this, which is the same regardless of operating system:

```
filename xyz 'myfile.txt';
data _null_; rc = fdelete('xyz'); run; /* do this */
%let x = %sysfunc(fdelete(xyz)); /* or this */
filename xyz clear;
```

Note that if the file doesn't exist, you won't get any warning or error message. That is useful in case you want to add this code to the top of your program to ensure that the file is deleted before you start, regardless of whether it exists. Note also that XYZ is an arbitrarily chosen fileref. You can use either a DATA step or a %SYSFUNC invocation. (Note that no quotes are used with the fileref when using %SYSFUNC). Clearing the fileref is not completely necessary but makes for cleaner code.

Note also that you should minimize your use of the X command. Many applications, especially those on workspace servers, might disallow the use of the X command using NOXCMD. The FDELETE function can still be used when NOXCMD is in effect. Of course, you will need access permission to delete files.

MAINFRAME CONSIDERATIONS

If you are running on z/OS, keep in mind the FILESYSTEM option. That value can be MVS or HFS. If you use MVS, the TEMP device refers to OS data sets. If you use HFS, the TEMP device refers to HFS files. You can change this option back and forth easily. For example:

```
options filesystem=hfs;
filename xyz temp;
options filesystem=mvs;
```

Even though you have changed the FILESYSTEM option back to MVS, the XYZ fileref continues to refer to a temporary file in HFS space.

CREATING DIRECTORIES

The TEMP device is meant for creating an individual file. But you might want to create a directory and write multiple files to it and then delete the directory by the end of the SAS program. You cannot use TEMP for this, but you can create a directory under the WORK directory using the DCREATE function and the GETOPTION function. Here is an example of doing this:

```
data _null_;
  mydir = dcreate('newdir',getoption('work'));
  call symputx('mydir',mydir);
  run;
%put &mydir;
```

```
data _null_; file "&mydir/abc.txt"; put 'x'; run;
```

The %PUT statement produced the following when running a test:

```
C:\Users\Rick\AppData\Local\Temp\SAS Temporary Files\_TD5696_RICK-PC_\newdir
```

It created a macro variable called MYDIR with this value. And then a file called abc.txt was written into that directory.

The name of the WORK directory will differ each time. The GETOPTION function, if given the argument of WORK, will return the pathname for the current WORK directory. The DCREATE function is given the name of the subdirectory to be created as its first argument and the name of the directory under which the subdirectory is to be created as its second argument. The return value is the fully qualified directory name. Be aware that this name might have embedded blanks. If the return value is blank, the directory was not successfully created. Note also that if SAS has been invoked with the NOWORKINIT or NOWORKTERM options, this directory will be permanently created, which you likely will not want.

Note that in the preceding example, a forward slash was used (&mydir/abc.txt). This will work on Windows as well because the SAS host code manages the filenames properly. This is not true if you are generating code for the X command. You will have to use a forward slash (UNIX and z/OS HFS) or a backslash (Windows) as appropriate.

Note also that this mechanism of creating a directory cannot be used on z/OS unless FILESYSTEM is set to HFS.

Examples of using a temporary directory would be in a zip command command ,or the ZIP access method available in SAS@9.4, or when using FTP to upload or download all files in a directory.

And this temporary directory will be deleted as part of the WORK directory deletion when the SAS program terminates. (An exception is when the NOWORKTERM option has been given.)

DELETING SAS DATA SETS

It is always a good practice to delete your SAS data sets when you no longer need them. This is especially true with temporary SAS data sets that you might create within a macro. It doesn't matter if the data sets are small: it is always good to clean up as needed. But if they are large data sets, you absolutely need to delete what is no longer needed, even for WORK data sets, because you might run out of disk space unnecessarily.

I have always liked to use PROC DELETE, even though PROC DATASETS is promoted as the way to delete data sets. Perhaps this is because I was using SAS long before PROC DATASETS was created!

Just get in the habit of deleting what is no longer needed:

```
data temp; x=1; run;
data temp2; set temp; y=x; run;
proc delete data=temp; run; /* or */
proc datasets lib=work; delete temp; quit;
```

AVOIDING THE USE OF THE CURRENT DIRECTORY

It can be tempting to write files directly into the current directory. For example:

```
data _null_; file 'myfile.txt'; put 'x'; run;
```

This writes myfile.txt into your current directory. This is fine for small SAS programs that you need to run just for yourself, but for SAS code that will be run by others, you should try to avoid this practice. The current directory might not be writable when the SAS program is run. In addition, the user might not realize the files were written to the directory, which results in litter left behind.

USING DUMMY FILES

Dummy files can be preferable to TEMP if you will not need the files at all or if you want to dispose of the data entirely. The DUMMY device type can be given in SAS code for any operating system. It corresponds to /dev/null in UNIX and FILESYSTEM=HFS on z/OS, NUL: in Windows, and NULLFILE on z/OS.

For example, consider a macro that writes two files unconditionally. You only need one of the files, and you will completely ignore the other:

```
filename myfile temp;
filename bitbuckt dummy;
%getdata(file1=myfile,file2=bitbuckt);
```

(Note that I like the fileref of BITBUCKT because it refers to the "bit bucket", an old term indicating that you want unwanted bits to disappear.)

Another good example of dummy files is when you want to suppress your log output. Consider that you have SAS code you want to run that will display log messages that you don't want to be visible in the log (such as passwords). Here is an example of using dummy files to avoid displaying the log:

```
filename bitbuckt dummy;
proc printto log=bitbuckt new; run;
%include mycode; run;
proc printto; run;
filename bitbuckt clear;
```

Everything between the PROC PRINTTO invocations will be written to the dummy file and will not be seen at all.

DELETING MACRO VARIABLES AND MACRO DEFINITIONS

Just like with data sets that are no longer needed, macro variables and macro definitions can and should also be deleted. Macro variables take up memory (and possibly disk space), and macro definitions will take up disk space.

Note that setting a macro variable to an empty value does not delete the macro variable, as in this example:

```
%let abc=xyz;
%put _user_;
%let abc=;
%put _user_;
```

You will still see that the GLOBAL macro variable ABC is displayed with the second %PUT statement.

To truly delete a macro variable, use the %SYMDEL statement, as in this example:

```
%let abc=xyz;
%put _user_;
%symdel abc;
%put _user_;
```

For macro definitions, you can use the %SYSMACDELETE statement, as in this example:

```
%macro abc; %put this is abc; %mend;
%abc;
%sysmacdelete abc;
/* it is now an error to reference %ABC */
```

Note that when %SYSMACDELETE is invoked, the macro member is deleted from work.sasmacr. You cannot use %SYSMACDELETE on compiled macros in SASMSTORE or in SASHEP.

CONCLUSION

Always keep in mind that you might leave your files (your litter!) behind after your SAS program has completed. Unless there is a need for those files afterward, be vigilant in removing them. Use TEMP, DUMMY, temporary directories in WORK, and FDELETE to get the job done and don't be a litterbug!

CONTACT INFORMATION

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

Rick Langston
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Rick.Langston@sas.com
http://www.sas.com