**Paper SA-11-2014**

# A Hands On Tutorial For Automating a Tedious Process

Eric Barnitt, Minnesota Department of Health, St. Paul, MN

## ABSTRACT

Once, I inherited a program which required the programmer to manually update over 100 separate %LET statements with 87 different file names. Even worse, this program was required to be updated and run twice a year. Rather than suffer through the tedious and laborious process of manually updating these statements, I decided to write a SAS® program which does it automatically. This paper provides an example of how various SAS system commands and programming steps can be used together to fully automate the process of transforming multiple flat text files into one SAS data set. First, a description of the motivation and background for solving the problem is given. Next, I describe the mechanics of how SAS can read the names of files from a folder and translate that information into a SAS data set. I then show how to use the data set of file names in a macro to cycle through the list of files and read them into SAS. Finally, I provide the reader with a few lessons learned and words of caution in performing this type of process automatically. This paper presumes the reader is knowledgeable with basics of how to convert text files into data sets and with intermediate-level macro programming skills. Therefore, the ideal audience is a programmer at approximately an intermediate level of experience. Programmers who are closer to a beginner skill level can benefit from being exposed to an example of how SAS can be used to automate and update existing programs. All programming work is done on SAS run in a Windows environment using version 9.13.

## INTRODUCTION

In a different life, I once inherited a program which pulled together over 100 individual text files and read them all into one SAS data set. The process by which these files were converted in SAS was that an unlucky programmer had to manually update all of the individual file references in the program. This proved to be problematic in a number of ways. First, manually changing all the individual file references proved to be very time-intensive. Worse, the process of updating each file reference proved to be error-prone; even a careful programmer might mistakenly copy the same file name twice. The second problem with the program is that it needed to be updated and run twice a year, thus doubling the amount of time dedicated to a tedious and problematic task.

Through a process of trial and error, a process including a few simple steps was developed which alleviated these frustrations completely. These steps are:

1. Get a list of file names from a specified directory into a SAS data set.

2. Count the number of files in the directory so the process can be fully automated.

3. Establish a macro name for each file in the directory, all of which have sequential names (e.g. file1, file2, file3…)

Once these steps are complete, the user has the complete freedom to manipulate the files in the directory in any way they wish. This example takes text files and converts them to a SAS data set, but this process could easily be altered slightly to work on any sort of automated task or procedure a programmer wishes to perform on a number of files. Following the step-by-step explanation of how the steps listed above are accomplished, a few other use cases from my professional experience are discussed; these highlight the flexibility this approach has on a number of different applications.

One final note is that this paper is intended to be informative for beginning SAS users. As such, some of the deeper technical explanations for certain procedures are presented in a topical fashion. Many of these ideas have broader and deeper applications which can be explored by programmers seeking to learn more about them.

## STEP 1: CREATING A LIST OF FILE NAMES

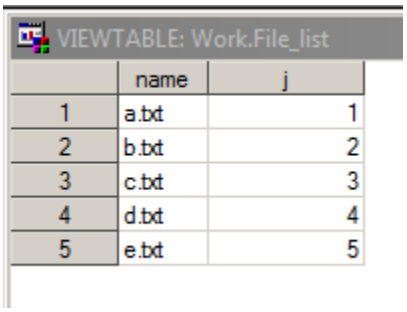
For purposes of illustration, I have created a dummy file directory with a few text files named A, B, C, D, and E which I am trying to dynamically read into SAS. The first step to for this to happen is to create a list of the files in the directory where they are stored. To accomplish this, I chose to use a SAS pipe within a filename step. A pipe is an option which allows users to go outside of a SAS session to access data, programs, or other files. In this case, use of a pipe is a bit more straightforward. There are a few text files stored in a folder called “sas_training” located in the specified directory. The following code goes to that directory, grabs the names of the files stored there, and creates a data set with one row per each file name.

```
filename indata pipe "dir f:\hep\HSR\sas_training /b"

data file_list;
    length name $9;
    infile indata;
    input name 9.;
    j = _n_;
run;
```

Note that in this DATA step, each row is being assigned an identifying variable referred to as "j". This is used in the next data step.

The result of the preceding data step is a data set which looks like this.



VIEWTABLE: Work.File_list

| | name | j |
|---|---|---|
| 1 | a.txt | 1 |
| 2 | b.txt | 2 |
| 3 | c.txt | 3 |
| 4 | d.txt | 4 |
| 5 | e.txt | 5 |

**Figure 1: The list of files in a DATA set**

## STEP 2: OUTPUT COUNT OF FILES AS A MACRO VARIABLE

For this process to be fully automated, SAS needs to know how many files are in a particular folder. It is true that a programmer might know before they are working on a project how many files they need to manipulate in a process, but adding this step allows for this process to be fully dynamic. No matter how many files are in the folder, or if the programmer doesn't know, this step alleviates the burden of establishing a hard-coded cutoff.

```
data file_list2;
    set file_list;
    call symputx("filecount",j);
run;

%put &filecount;
```

Because of the way SAS works behind the scenes, the macro variable filecount is established with a value of 1 when the first line of the input file file_list is read in. Then, when each subsequent line of the data set is read in, the value of filecount is overwritten with the new value of j. When the final row is read in from the input data set, the variable filecount has the maximum value of j; in this case, this means that the filecount variable is the number of records in the specified folder.

## STEP 3: CREATE A MACRO VARIABLE FOR EACH FILE NAME

At this point, we have a SAS data set which contains the names of all the files we wish to work with. We know how many files there are in the folder, and we have stored that number as a macro variable in SAS. This allows us to fully automate our manipulations of the files in this folder without having to manually specify how many files to work with. The next step in the process is to create a macro variable for each file in the folder. Ultimately, since we wish to cycle through the files to perform data manipulations, it makes sense to set up these names in a way which allows for iteration. The easiest way to do this is to give each file a macro variable name with a sequential numbering pattern (e.g. file1, file2, file3…)

This is accomplished using the following short macro.

```
%macro setupvar;
%do i=1 %to &filecount;
data _null_;
    set file_list2;
    if &i. = j then do;
           call symput("file&i.",name);
    end;
run;

%end;
%mend;

%setupvar;

/* test */

%put &file1;
%put &file5;
%put &&file&filecount;
```

```
Log - (Untitled)
242
243  %put &file1;
a.txt
244  %put &file5;
e.txt
245  %put &&file&filecount;
e.txt
```

**Figure 2: Testing that macro variables are being assigned correctly**

It is good practice to test that the file names are being assigned correctly. In this case, we happen to know that there are five files in the folder which are going to be converted into a SAS data set. We can use this information to strategically choose which file names we wish to check. In this example, we are checking that file1 and file5 are correctly identified as A and E respectively. We also want to check that using the macro token filecount in this setting produces the same result as file5.

### STEP 4: PUTTING IT ALL TOGETHER

At this point, each file name has been assigned to a sequential macro variable, thus making iterative looping a viable way to work through each data set. The goal of this exercise is to concatenate these five files into one data set. One short and final macro will do the job. There are multiple ways to do this, but my personal preference is to create a null data set and append each file onto a growing data set.

```
data base1;
    set _null_;
    var1 $8 var2 var3;
run;

%macro createdata;

%do i=1 %to &filecount;

filename b "F:\HEP\sas_training.&&file&i..txt";

data temp;
```

```
    infile b;
input @1 var1 $8. var2 var3;
run;

proc append base=base1 data=temp; run;

%end;
%mend;
```

## CONCLUSION

While this example is simplistic, there are lessons here for a beginning SAS programmer. The most significant one I hope to impart on such a reader is that there are relatively simple tricks and logic which can save users a lot of time and effort. The process of manually updating all the file references was time-intensive and fraught with possible errors. With this automated process complete, now every file is read in correctly every time the program is run.

The second lesson in this paper is that sometimes ideas used for one purpose can be useful in other settings. This process began because I wanted to figure out a way to avoid copying and pasting dozens of file names. What I ended up with is a powerful tool I use frequently in my daily work. One of the most powerful aspects of SAS, in my opinion, is that it is flexible enough to work with various data and file types within the same program. This same process could be adapted slightly to work with any number of files which SAS can work with.

In closing, as I went through the process of writing this paper, I also came across a few limitations which are important to share. This process, while flexible, depends on an underlying similarity in data. In my case, I assumed that my data all had the same column structure. This allowed me to fully automate the process of appending the files together. In reality, a more nuanced coding approach to handle slight differences in file structure might be necessary. If a programmer were to attempt to use this process to automate a process to check the distribution of a data element in every file in a directory, for example, they might wish to adapt this logic to handle cases where a file does not contain that data element. As this paper is intended for beginning SAS programmers and not those at a more advanced level, this more nuanced approach is slightly beyond the scope of this paper.

## CONTACT INFORMATION

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


Name: Eric Barnitt
Enterprise: Minnesota Department of Health
Address: 85 East 7th Place
City, State ZIP: St. Paul, MN 55101
Work Phone: 651 201-3553
E-mail: eric.barnitt@state.mn.us