Paper 066-2010

Taming the Herd: Wrangling Unruly Text Files in SAS®

Taylor Young for the CARe Project, The Broad Institute, Cambridge, MA

ABSTRACT

It may come to pass that a SAS programmer is asked to make simple modifications to a set of plain text files thereby producing a new set of plain text files. At first, this may seem like an innocuous task, perhaps only requiring summarization of the data or swapping in a new set of IDs with a quick import and export cycle. Closer inspection of the files, however, may reveal unsightly hazards like a large number of files, long file names, long column headers, no column headers, or non-conforming missing values that make reproducing the structure of the original files difficult. This paper will guide you through several techniques to ensure that your files make it to the other side without losing one byte of information.

INTRODUCTION

Although the topics in this paper are discussed broadly, they originate from the NHLBI's CARe project which brings together genotype and phenotype data from 50,000 participants across 9 longitudinal, epidemiological studies and makes the data available to a community of several hundred investigators (Musunuru et al. 2010). One feature of CARe is a standardized analysis pipeline to which CARe investigators can submit their data for analysis. The pipeline requires data to be submitted in text-tab files, which for confidentiality concerns, are encrypted with IDs unique to each investigator. As a result, the author found himself importing and exporting unruly herds of text files time and time.

The herds were eventually tamed and this paper will demonstrate several techniques used for importing and exporting plain text files that don't necessarily follow established SAS conventions. Issues discussed include managing a large number of files, long file names, long column headers, no column headers, or non-conforming missing values. The solutions presented and discussed range from graceful UNIX and PERL command line arguments to heavy-handed, process intense file management.

ISN'T UNIX AN ANTIQUE?

If your data is already in SAS and you want to batch process a library of data sets, it's fairly common to use the CONTENTS procedure or the SQL procedure's dictionary tables to create a list of data sets, grab them as sequential macro variables, and then loop through each data set. If you are starting with plain text files, however, the options aren't so clear and however you manage to feed your file names into your DO loop, you are faced with manually curating a list of files. It might not be so bad for 5 to 10 files, but clearly this is a job for a computer, not a human. Just like a finely crafted antique, it is easy to appreciate the gracefulness of a single UNIX command line argument:

```
x ls -1 > FileNames.txt;
```

In the above UNIX argument, the 'ls' command lists the contents of the current directory, the '-1' option forces the output to one entry per line, and the '>' routes the output into a file in the current directory named FileNames.txt. Since the files you want to process may not be in the current directory, it's always good idea to include an absolute reference to the directory where the files exist and an absolute file path for the FileName.txt file you are creating:

```
x ls -1 '/directory/path' > '/file/path/FileName.txt';
```

Hint: The file does not need to be written to the same directory as the files you are listing.

Here is a quick macro using the 'ls' argument that imports the list of file names and prints them to the SAS log:

```
%macro FileNames;

x ls -1 '/directory/path' > '/file/path/FileName.txt';

data FileNames;
    infile '/file/path/FileName.txt' firstobs=1 ;
    informat FileName $256. ;
    format FileName $256. ;
    input FileName $;
```

```
    run;

proc sql noprint;
select FileName, count(FileName) into :File1 - :File999, :FileTot from FileNames;
    quit;

%do i=1 %to &FileTot;
%put &&File&i;
%end;

%mend FileNames;

%FileNames;
```

By now, you may have noticed that the 'ls' command returns more than filenames and that you don't have a tidy list of .txt files to work with. One option would be to add a WHERE= clause to the DATA step so that you are only importing file names that end in .txt:

```
data FileNames (where=(index(FileName, '.txt')>0));
    infile '/file/path/FileName.txt' firstobs=1 ;
    informat FileName $256. ;
    format FileName $256. ;
    input FileName $;
    run;
```

Another option would be to manually clean up the directory but since we're already trying to avoid calluses and dirt under our nails, this isn't the best solution. To be more selective when creating the list of files the UNIX 'find' command line argument is a great choice:

```
x find '/directory/path' –name '*.txt' > '/file/path/FileName.txt';
```

Before running off and adding this to your macro, beware that the 'find' command will dig through your directory structure interrogating every folder it comes across so we're going to have to tone it down a little. In order to keep the argument from recursively walking the directory structure we're going to add '-maxdepth 1' as a primary to the command line argument:

```
x find '/directory/path' –maxdepth 1 –name '*.txt' > '/file/path/FileName.txt';
```

We also want to be sure that every file found is a file suitable for import to SAS so we'll add '-type f' as a primary to the command line argument:

```
x find '/directory/path' –maxdepth 1 –name '*.txt' –type f >
'/file/path/FileName.txt';
```

Finally, the 'find' argument prints the entire file path and while it can easily be parsed later in the macro, the formatted print primary '-printf' can accomplish this from within the command line argument as well:

```
x find '/directory/path' –maxdepth 1 –name '*.txt' –type f -printf "%f\n" >
'/file/path/FileName.txt';
```

When using '-printf' a print format must be specified and by default, it does not add a new line at the end of a string. The '%f' in the format indicates that we wish to print only the last element of the file name, not the entire file path. The '\n' adds a new line to the end of each file name so that only one file name is printed per line in our output file.

Here is the previous macro updated with the 'find' argument:

```
%macro FileNames;

x find '/directory/path' –maxdepth 1 –name '*.txt' –type f -printf "%f\n" >
'/file/path/FileName.txt';

data FileNames;
    infile '/file/path/FileName.txt' firstobs=1 ;
    informat FileName $256. ;
    format FileName $256. ;
    input FileName $;
```

```
    run;

proc sql noprint;
select FileName, count(FileName) into :File1 - :File999, :FileTot from FileNames;
    quit;

%do i=1 %to &FileTot;
%put &&File&i;
%end;

%mend FileNames;

%FileNames;
```

While the 'find' command line argument is more difficult to write than 'ls', it is an excellent tool for creating a list of files to be imported into SAS. The '-name' primary, for example can be modified to include any string that would identify a set of files for processing. Furthermore, a set of files can be identified by many other factors such as user, group, or modification date. See the UNIX man pages for a full reference.

Note: For windows users, a similar process is outlined using SAS external file function in paper 054-2007 by Linda Libeg.

SOMETIMES_IMPORTANT_DATA_IS_STORED_IN_THE_FILENAME.TXT

Occasionally, important information about the file is stored in the file name. Since there isn't a globally accepted file naming convention, there's also a good chance that the file names will be longer than the 32-character limit for SAS data set names or that they will contain spaces. If you're not storing a permanent copy in a SAS data set, long file names won't cause too much trouble. Once the file names are stored as macro variables (as is the case in the previous macro), any import and export statements can reference the file names stored in macro variables and the name given to the temporary data set being processed is irrelevant. If, however, a permanent copy will be stored, it's a good idea to create a link to a 'safe for SAS' file name and the actual file name before importing. Using the DATA step from the previous macro, this can be accomplished when importing the list of file names:

```
data FileNames;
    infile '/file/path/FileName.txt' firstobs=1 ;
    informat FileName $256. ;
    format FileName $256. ;
    input FileName $;
    SafeName='Safe_FileName_'||left(_n_);
    run;
```

By concatenating the "Safe_FileName_" prefix with the _N_ automatic variable, a sequential list of file names is created that can be referenced from within the existing DO loop:

```
%do i=1 %to &FileTot;
%put Safe_FileName_&i :: &&File&i;
%end;
```

NOT ALL COLUMN HEADERS MAKE GREAT VARIABLE NAMES

Similar to file names, important information about a variable may be stored in a column header that is too long to use as a SAS variable name once the file is imported. SAS, however, won't complain. It will just truncate the variable name to the first 32 characters. This makes it difficult to export a file with the same headers that existed when the file was imported. You also run the risk of creating duplicate variable names due to the truncation, in which case, SAS will use the automatic VAR*n* naming convention for the duplicate variables. To overcome this limitation and to use the SAS automatic variable names, the PROC IMPORT GETNAMES= option should be set to NO:

```
proc import datafile='FileName.txt' out=ImportFile replace;
    getnames=no;
    run;
```

While this will create a SAS data set with VAR1 - VAR*n* for variable names, we still need to obtain the column headers from the imported file so they can be written to the exported file. The best option is to import the file again in a separate DATA step where only the first observation containing the headers is imported. That data set can be

transposed to create a data set that links the automatic variable names with the actual header values, which can then be gathered into macro variables using PROC SQL.

Reading data from an external file from within a data step requires that the variable names be specified with an input statement. Since the file was first imported with GETNAMES=NO, we can now reference the variables using a variable list**.** The problem is that we don't yet have a value for the last variable (VAR*n*) but referencing the DICTIONARY.COLUMNS automatic table easily solves that:

```
proc sql noprint;
    select count(name) into :NumVar from dictionary.columns
           where libname='WORK' and memname='IMPORTFILE';
    quit;
```

Then all we need to do is import the first observation with a data step and transpose it:

```
%let NV=&NumVar;  *Trims the leading blanks from the numeric value;

data Vars;
    infile 'FileName.txt' lrecl=32767 firstobs=1 obs=1;
    format VAR1 - VAR&NV $256.;
    informat VAR1 - VAR&NV $256.;
    input VAR1 - VAR&NV $;
    run;

proc transpose data=Vars out=Vars;
    var _all_;
    run;
```

Another option that may seem attractive is to set the PROC IMPORT DATAROW option to 1 so that SAS will import the headers as the first observation in the data set. Then PROC TRANSPOSE can be used on the first observation only to create a data set that contains the variable name and the header from the import file:

```
proc import datafile='FileName.txt ' out=ImportFile replace;
    datarow=1;
    getnames=no;
    run;

proc transpose data=ImportFile (obs=1) out=Vars;
    var _all_;
    run;
```

The downside to this method is that it will most likely import every variable as character, which is not the preferred variable format for numeric data. It could also inflate the size of your dataset substantially or you may lose some information if the variable length is too short based on the number of guessing rows.

Now that we have a tidy list of variable names in a data set, we'll need to establish a procedure for writing the stored macro values to the new external file. One option is to add the headers as variable labels and use the new PROC EXPORT option to export variable labels rather than variable names. If your goal is to reproduce the original file, however, this option produces unsatisfactory results since variable labels are double quoted when exported. To avoid the double quotes, the best option is to export the file with the automatic variable names and then execute a quick PERL one liner from the command line to search for the automatic variable names and replace them with the original headers. This is not for the faint of heart:

```
proc export data=ImportFile outfile='ExportedFile.txt'
    dbms=tab replace;
    run;

proc sql noprint;
    select _NAME_, COL1 into :OldVars separated by '09'x, :NewVars separated by
'09'x from Vars;
    quit;

x perl -pi -w -e "s/&OldVars/&NewVars/g" ExportedFile.txt;
```

This PERL one liner will execute the search on OldVars and replace with NewVars in the file specified, editing it in place. We've selected both sets of variable names into macro variables separated by a tab to create a unique string of text that will almost certainly prevent us from mistakenly replacing a data value with a variable name. We could have, for example, executed the PERL 'search and replace' from within a DO loop one variable at a time but that increases our risk of errant replacing since we would be searching for a much smaller string of text (i.e. VAR1, VAR2, or VAR4). For the sake of discussion, that code would look something like this:

```
proc sql noprint;
    select _NAME_, COL1 into :OldVar1 - :OldVar999 separated by '09'x,
    :NewVar1 - :NewVar999 separated by '09'x from Vars;
    quit;

%do i=1 %to 999;
x perl -pi -w -e "s/&&OldVar&i/&&NewVar&i/g" ExportedFile.txt;
%end;
```

Now that we've established a method for importing a file without getting the variable names, but rather by pulling the headers into macro variables and writing the stored headers to an external file, we can update the previous macro integrating the IMPORT and EXPORT procedures:

```
/* Open the macro */
%macro FileNames;

/* Get a list of files to import */
x find '/directory/path' –maxdepth 1 –name '*.txt' –type f  -printf "%f\n"
    > '/file/path/FileName.txt';

/* Import the list of files */
data FileNames;
    infile 'FileNames.txt' firstobs=1 ;
    informat FileName $256. ;
    format FileName $256. ;
    input FileName $;
    SafeName='Safe_FileName_'||left(_n_);
    run;

/* Get file names as macro variables */
proc sql noprint;
    select FileName, count(FileName) into :File1 - :File999, :FileTot from
FileNames;
    quit;

/* Do loop for all files */
%do i=1 %to &FileTot;
    %put Processing Safe_FileName_&i :: &&File&i;

/* Import the file */
proc import datafile="&&File&i" out=Safe_FileName_&i;
    getnames=no;
    run;

/* Count the number of variables */
proc sql noprint;
    select count(name) into :NumVar from dictionary.columns
           where libname='libref' and memname="Safe_FileName_&i";
    quit;

%let NV=&NumVar;  *Trims the leading blanks from the numeric value;

/* Import the first observation only (headers) */
data Vars;
    infile ="&&File&i"  lrecl=32767 firstobs=1 obs=1;
```

```
    format VAR1 - VAR&NV $256.;
    informat VAR1 - VAR&NV $256.;
    input VAR1 - VAR&NV $;
    run;

/* Transpose the variable names and headers */
proc transpose data=Vars out=Vars;
    var _all_;
    run;

/* Export the file */
proc export data=Safe_FileName_&i outfile="&&File&i"
    dbms = tab replace;
    run;

/* Get unique strings of variable names and headers */
proc sql noprint;
    select _NAME_, COL1 into :OldVars separated by '09'x, :NewVars separated by
'09'x from Vars;
    quit;

/* Replace SAS automatic variable names with proper headers */
x perl -pi -w -e "s/&OldVars/&NewVars/g" "&&File&i";

%end;  *Close the do loop;

/* Close the macro */
%mend FileNames;

/* Call the macro */
%FileNames;
```

WAIT, NOW WITHOUT COLUMN HEADERS?

If you've made it this far, then importing and exporting files without headers probably won't seem as daunting as when we started. The import is pretty straightforward, and building on the previous import discussion, we can use PROC IMPORT with DATAROW=1 and GETNAMES=NO to import the file:

```
proc import datafile='FileName.txt' out=ImportFile replace;
    datarow=1;
    getnames=no;
    run;
```

Next we'll just need to export a file without writing the SAS variable names or labels to the first row in the external file. This can't be done with PROC EXPORT so we'll need to write the data to an external file from within a DATA step. We can use PROC SQL and the VARS data set from the previous example to create a list of variables to write to the exported file:

```
proc sql noprint;
    select _NAME_ into :VarList separated by ' ' from Vars;
    quit;

data _null_;
    file 'ExportedFile.txt' delimiter='09'x;
    put &VarList;
    run;
```

There, that wasn't so bad.

NON-CONFORMING MISSING VALUES

In SAS, missing values are clearly designated as a **space** for character variables and as a **period** for numeric variables. But that doesn't mean there isn't a file specification out there that calls for all missing values to be coded as "Missing" or as some other convention not easily supported by SAS. Using a text string to identify missing values

won't bother SAS if it's already importing a variable as character. But, when SAS is importing a numeric variable and comes across a text string, it will throw an error into the log and assign the observation its rightful missing value, a period. It's also common for missing values to be assigned a faux numeric code such as .A, -9, -999, or -999.999 and while SAS won't have trouble importing any of these, you might not know that they are present. This will cause headaches, particularly if types of missing values have been specified such as .A, .B, .C, and .D. While SAS will read the values without errors, when the data set is exported, those values lose their period and become A, B, C, and D respectively which would be a considerable change from the source file. Since one of our main goals is to export the files from SAS in the same format as the imported file, the best option for accommodating nonconforming missing values is to force SAS to import all of the data into character variables.

The pitfalls of importing numeric data as character data are twofold. First, storing data as character instead of numeric can increase the file size tremendously. In general, importing all of the variables as character with a length of 8 will result in a file the same size as if all of the variables were as numeric with a length of 8. Numeric variables, however, are stored as floating point numbers and a value like 123456789.101112 can be stored with a length of 8 whereas a character value would require a length of 16 roughly doubling the amount of space required to store the variable. On the other hand, if the numeric data is limited to one or two characters, storing it in a character variable with a length of 2 will use less space in comparison to the default length of 8 or the minimum length of 3 for numeric variables.

There's no easy way to get around the risk of truncating numeric values when importing them into character variables. Data truncation even remains a risk when they're imported to numeric variables. The PROC IMPORT option GUESSINGROWS is one of our most helpful tools but it also has its limitations. 32,767 rows to be precise, which isn't too comforting if you're routinely dealing with files that have 32,768 or more rows.

Even with these pitfalls taken into consideration, importing as character can still be an option to address the problem of nonconforming missing values. As previously noted we can use PROC IMPORT with GETNAMES=NO and DATAROW=1 options to nudge SAS into importing all data as character variables since we can expect the column headers are character. We'll also add GUESSINGROWS=10000 for good measure:

```
proc import datafile='FileName.txt ' out=ImportFile replace;
    datarow=1;
    getnames=no;
    guessingrows=10000;
    run;
```

There is a chance, however, that the column headers are themselves numeric and if the missing value codes are faux numeric (i.e. .A, .B) you're going to lose the period upon export. After the file has been imported once, we can query the DICTIONARY.TABLES table for the number of variables and re-import the file from within a DATA step assigning each variable a length that makes us feel a little more comfortable, say 32:

```
proc sql noprint;
    select numvar into :NumVar from dictionary.tables
    where libname='WORK' and memname='IMPORTFILE';
    quit;

%let NV=&NumVar;  *Trims the leading blanks from the numeric value;

data ImportFile;
    infile 'FileName.txt' lrecl=32767 firstobs=2;
    format VAR1 - VAR&NV $32.;
    informat VAR1 - VAR&NV $32.;
    input VAR1 - VAR&NV $;
    run;
```

It's very possible that 32 characters far more than required for the data we've imported and if we're storing a permanent copy of this data set we should at least attempt to reduce the file size. Using PROC SQL to create a table that links the variable name to the maximum length of each variable and importing the data with the variable length equal to the maximum length can do this:

```
proc sql;
    create table Vars as
    Name (char),
    Length (char);
```

```
    %do i=1 %to &NV;
    insert into Vars (Name) Values ("VAR&i");
    update Vars set Length=
    (select max(length(VAR&i)) from ImportFile) where Name="VAR&i";
    %end;

    select Length into :Length1 - Length999 from Vars;
    quit;

data ImportFile;
    infile 'FileName.txt' lrecl=32767 firstobs=2;
    %do i=1 %to &NV;
    format VAR&i $&&Length&i...;
    %end;
    %do i=1 %to &NV;
    informat VAR&i $&&Length&i...;
    %end;
    input VAR1 - VAR&NV $;
    run;;
```

It only took 3 imports, but we've managed to import all of our data as character values and optimize the variable length hopefully reducing the permanent file size.

CONCLUSION

This paper demonstrates that while SAS is a flexible and widely used tool for data management, occasionally, the SAS programmer has to implement some unconventional techniques such as UNIX and PERL command line arguments and multiple imports per file in order to achieve the desired goal. In the end, the flexibility of SAS ensures that no herd will go un-wrangled.

REFERENCES

Musunuru K, Lettre G, Young T, Farlow DN, Pirruccello JP, Ejebe KG, Keating BJ, Yang Q, Chen MH, Lapchyk N, Crenshaw A, Ziaugra L, Rachupka A, Benjamin EJ, Cupples LA, Fornage M, Fox ER, Heckbert SR, Hirschhorn JN, Newton-Cheh CH, Nizzari MM, Paltoo DN, Papanicolaou GJ, Patel SR, Psaty BM, Rader DJ, Redline S, Rich SS, Rotter JI, Taylor HA Jr, Tracy RP, Vasan RS, Wilson JG, Kathiresan S, Fabsitz RR, Boerwinkle E, Gabriel SB. Candidiate Gene Association Resource (CARe): design, methods, and proof of concept. Circ Cardiovasc Genet, in the press.

BSD General Commands Manual:

http://www.unix.com/man-page/Linux/1/man/

Linda Libeg. "054-2007: Functionsize: Process your External Files" The Proceeding of the SAS Global Forum 2007.

www2.sas.com/proceedings/forum2007/054-2007.pdf

Online Perl Documentation:

http://www.perl.org/docs.html

RECOMMENDED READING

SAS Institute Inc. 2010. **SAS OnlineDoc® 9.2.** Cary, NC: SAS Institute Inc.

Taylor Young. "200-2010: Facilitating Genetic Analysis: SAS®, the NHLBI and CARe" The proceeding of the SAS Global Forum 2010.

CONTACT INFORMATION

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

Taylor Young
The Broad Institute
7 Cambridge Center

Cambridge, MA 02142
tyoung@broadinstitute.org
www.broadinstitute.org