SESUG Paper FCS025-2025

Spring Cleaning for CSVs

Brooke Ellen Delgoffe, M.S, Marshfield Clinic Health System

ABSTRACT

Not all data starts in SAS and a common place it goes when coming from tertiary systems is a Comma Delimited Value (CSV) file. While a CSV is a great interoperability tool, it also comes with its share of pitfalls. Free text with special characters, space padding, and HTML tags are just a few of the things "clogging up" an easy PROC IMPORT and often turning programmers to use the wonderful DATA step. Without the automatic metadata and structure capabilities, using the DATA step can be time consuming – even if wonderful.

In this paper I plan to teach readers about the INFILE, FILE statements and the use of BINARY read-in settings that allow them to input and output to a CSV – allowing either an "in place" fix or creation of a cleaned CSV. Use of parsing functions will help to identify pesky characters, name their replacements, or eliminate them completely.

A macro will be presented to demonstrate the concepts and provide an executable solution to this pesky CSV problem. An industry application will be provided to numerically show the benefits of implementing this tool in a real-world way.

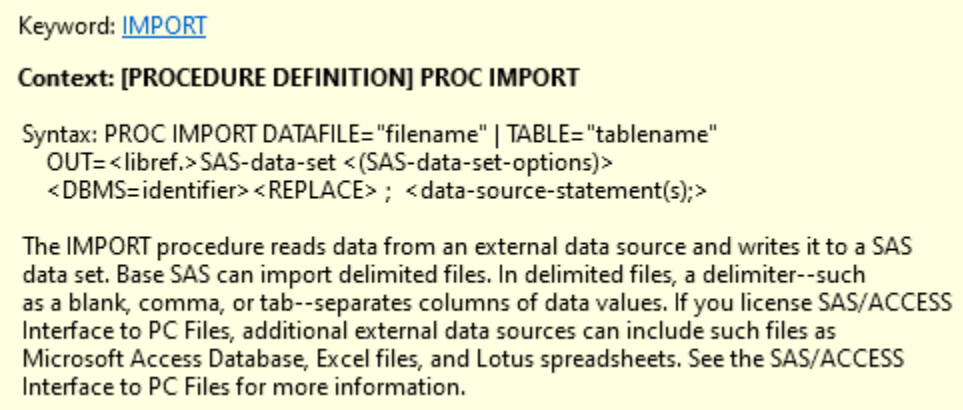
CONTENTS

INTRODUCTION

White space characters are those that create the appearance of horizontal or vertical space between characters amongst text and/or within text files (Mozilla Foundation, 2025). Examples include the space, tab, or line feed. Mozilla Inc. provides examples of how each of those characters are represented differently in different programming languages on their website (*See Recommended Reading: Mozilla Glossary – Whitespace)*. While we use white space characters in text editors like Microsoft Word © or even SAS Programs to provide organization and enhance readability in their native environment (primary use), they do not generally add those qualities once being used as data for analysis (secondary use). In fact, the very carriage returns used in this paper to separate the paragraphs, could cause failure to put its contents into a single field of a SAS dataset. The ability to analyze text data such as survey responses, medical notes, or addresses present the uncanny opportunity for input of these pesky characters and yet a quite common source of knowledge and data to analyze.

Just as common as the use of white space characters, is the use of delimited text files. As SAS explains, "A delimited text file is a file in which the individual data values contain embedded delimiters, such as quotation marks, commas, and tabs. A delimiter is a character that separates words or phrases in a text string that defines the beginning or end of a contiguous string of character data" (SAS Institute, 2016). This common source of data certainly has many solutions for use in SAS. The most skilled solution being the cornerstone DATA step, which is so beloved and powerful, yet programmatically much more typing and much more demanding of programming background. There is of course a SAS Enterprise Guide® (EG) Task called "Import Data" which will give users a point-and-click alternative, but this requires human clicking before it generates the code. Remaining is one of the most entry level, low effort, high reward solutions: The Import Procedure (PROC IMPORT).

As shown in Figure 1. EG Pop-up for PROC IMPORT, the syntax is short, the options plentiful, and the procedure powerful to detect attributes and quickly turn your external data into a SAS dataset. But just like any solution, it has strengths and weaknesses.


Keyword: IMPORT

Context: [PROCEDURE DEFINITION] PROC IMPORT

Syntax: PROC IMPORT DATAFILE="filename" | TABLE="tablename"
OUT=<libref.>SAS-data-set <(SAS-data-set-options)>
<DBMS=identifier><REPLACE> ; <data-source-statement(s);>

The IMPORT procedure reads data from an external data source and writes it to a SAS data set. Base SAS can import delimited files. In delimited files, a delimiter--such as a blank, comma, or tab--separates columns of data values. If you license SAS/ACCESS Interface to PC Files, additional external data sources can include such files as Microsoft Access Database, Excel files, and Lotus spreadsheets. See the SAS/ACCESS Interface to PC Files for more information.

Figure 1. EG Pop-up for PROC IMPORT

A delimited text file or specifically a comma delimited values file is one of the most basic data interoperability tools between software systems. Not only does SAS have ways to use a CSV, but so does about every data related software system and every data centric programming language. Even if PROC IMPORT is not your next step, nor SAS your continued use case, the solutions covered can produce a CSV file with the same potential destinations as the original CSV and provoke the same advances. Martin Mincey from SAS Technical Support has a great article on SAS Blogs about how to use SASPy® and the SAS® kernel to interact with SAS® and SAS® Viya® via Python (SAS Institute & Mincey, 2022), making this an easy step in your Jupyter notebook or any other tool like it with the same capabilities.

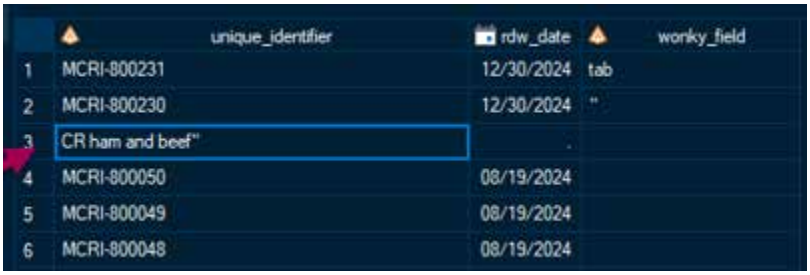
Before applying the subsequent solutions to your delimited data, you should be aware of when it will be helpful and have the knowledge to supply appropriate information to the macro. While the solutions covered can be applied to any type of delimited file, the focus here will be on the CSV. A CSV that has excessive whitespace characters will benefit, but missingness will not be addressed.

TO REMOVE WHITE SPACE OR NOT TO?

Typically, data cleaning is performed using code so that source data remains the ground truth, but there are times when it makes sense to modify the source data coming from CSV files in the CSV itself. The presence of white space characters is an opportunity. Of the reasons: 1) Issues importing data with PROC IMPORT, 2) Decreasing the Size of the File, and 3) Readability are the ones that stand out.

ISSUES IMPORTING DATA WITH PROC IMPORT

PROC IMPORT is the easiest connection between SAS and your CSV-based data. Its options and skill allow SAS to easily interpret the attributes of the data, make best guesses on structural metadata, and deposit the data into a SAS Dataset. However, there are a few times when PROC IMPORT cannot perform like the DATA step. One of those such issues is the use of white space characters, which may cause spillover issues like you see in Figure 1.



	unique_identifier	rdw_date	wonky_field
1	MCRI-800231	12/30/2024	tab
2	MCRI-800230	12/30/2024	"
3	CR ham and beef"		
4	MCRI-800050	08/19/2024	
5	MCRI-800049	08/19/2024	
6	MCRI-800048	08/19/2024	

Figure 2. Flow Over Issue Caused by Line Feed/Carriage Return

DECREASING THE FILE SIZE

Storage space is often a concern and storing data in its “thinnest” state can greatly improve the use of that data by more than just SAS. Management of whitespace characters can greatly reduce the size of the files without losing the meaning of the data inside. While they may have been needed when the data were in a different form, there is likely not a downside to removing them at the time of data analysis.

READABILITY

At first glance, data that are prone to white space padding may also appear completely missing or prove challenging to review analytically without first removing the whitespace. When speaking specifically about Excel, you may find that a field appears truncated or empty. When clicking on the cell the values do appear (Figure 2).

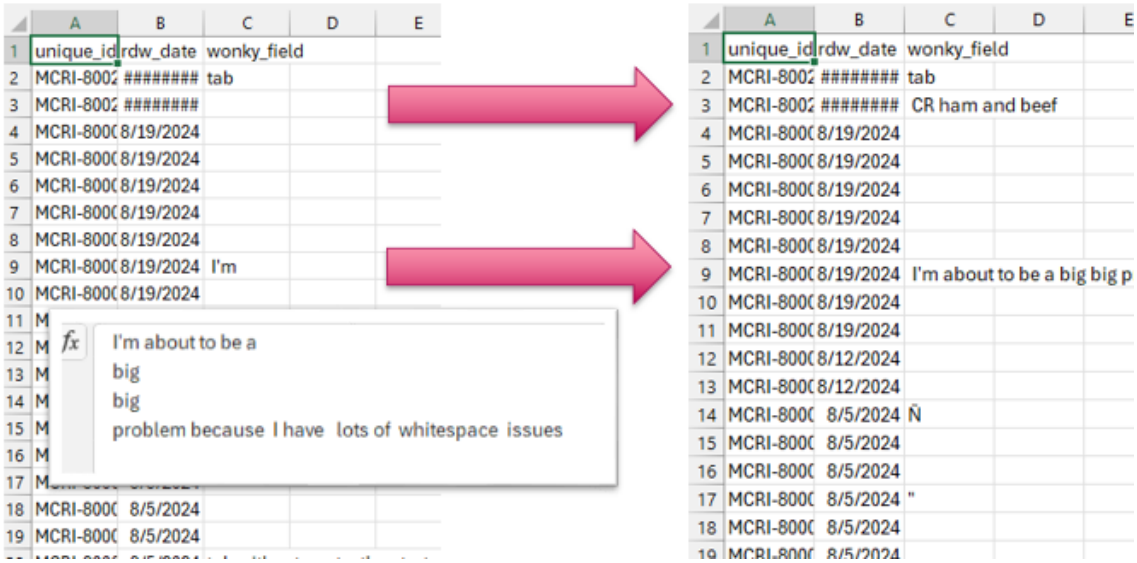


Figure 3. White space padded data "hide" in Excel.

MISSINGNESS VS. WHITE SPACE

MISSING DATA CODES

Missing data is demonstrated in CSVs by having nothing between column/record separators or empty quotes/cells. By definition, the comma is the column separator/delimiter, and the carriage return the row separator. In Figure 4, notice that rows end in a comma, character, or quote. Lack of characters after the comma and before the carriage return (next row) indicate missing data for the third column. In all other scenarios, a value exists for column three.

In rows 2-3 of Figure 4 (Notepad ++ side), we see a carriage return in a quoted value that creates a new row. Different applications will handle that scenario differently. In Excel, the quoted carriage return is ignored/included to a single cell because it is within double quotes. SAS is more literal, and sees the carriage return as a row separator – causing the import issue seen in Figure 1.

In SAS code these are empty quotes (character field) or a period (numeric field), that show as empty cells or periods in SAS datasets (as demonstrated in Figure 1). To check if a field is missing, you can use the ISNULL() function (See *Recommended Reading: IS NULL Function*).

WHITE SPACE CHARACTERS

In SAS, white space characters can be referred to by their HEX Code (See Figure 1 and *Recommended Reading: ASCII Whitespace Character Codes)* using the syntax:

```
'0D'x
```

Where the x designates the value inside the quotes as a hex code. In this case, 0D is a Carriage Return (created by the “Enter” key on a QWERTY keyboard).

Code	HEX	Symbol	HTML Number	HTML Name	HTML Name2	Name
9	09					Horizontal Tab
10	0A	
	Line Feed
11	0B		&#11;			Vertical Tabulation
12	0C					Form Feed
13	0D	
		Carriage Return
32	20					Space

Figure 4. Screen capture of HEX Codes from ascii-code.com

These are the codes that are expected to be found and replaced or deleted. If your data does not contain those characters, then it will be unaffected when providing them as inputs.

One of the quickest ways to check for whitespace character issues is to open the data in a viewer. As seen in Figure 2, when opening a CSV and expecting to find a new line for each new observation you would see carriage returns have ruined that expectation. Be careful when opening that data in Excel, as Excel may hide the data at first glance, but provide it in the equation bar when clicking into the cell.

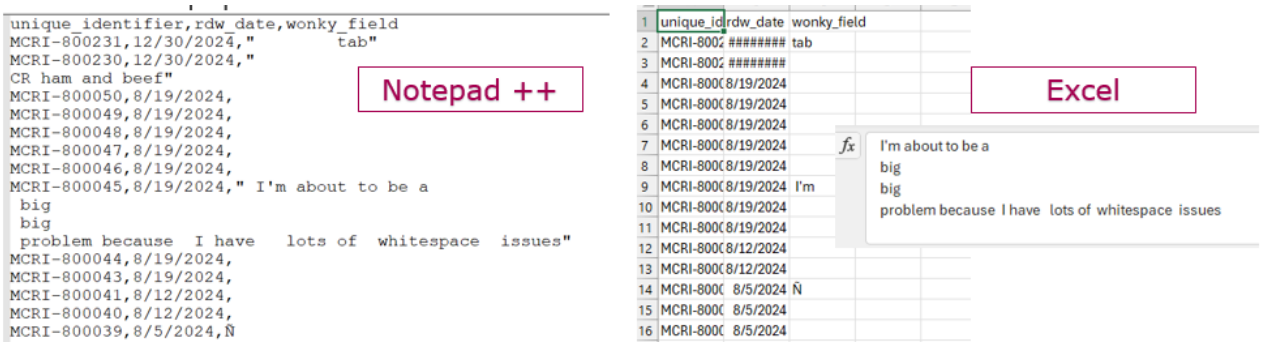


Figure 5. Sample Input Data Demonstrating Whitespace Character Issues

FIND (), PRXCHANGE (), TRANWRD (), AND TRANSLATE () FOR FINDING WHITE SPACE CHARACTERS

While not necessarily the fix provided in this paper, there are a variety of methods for finding and fixing white space characters once inside SAS. Try these out if you cannot figure out which “pesky characters” you are attempting to find and adjust. These are also good techniques if you prefer to fix the data once inside SAS.

FIND PESKY CHARACTERS

If you can get your data into SAS, there are several helpful functions to help you find and oust the culprits. The FIND function allows you to search a string for another string, including one referenced as a hex. For example,

```
DATA WONKY_FINDER;
set SESUG25_IMPORT1;
if FIND (wonky_field,'09'x) gt 1 then HAS_WONKY=1;
RUN;
```

OUST THE PESKY CHARACTERS

Once you have found the characters, you can also use SAS functions to change them. Fun functions like PRXCHANGE () will let you use a pattern to oust these characters. The typical TRANSLATE () will turn one character into another and TRANWRD () will change a series of characters to another.

```
DATA NO_WONKY_PLEASE;
set WONKY_FINDER;
where HAS_WONKY=1;

*find and replace the tab with a space;
LESS_WONKY = translate (wonky_field,' ','09'x);

*remove any html tags;
retain rx1;
if _n_=1 then rx1=prxparse("s/<.*?>//");
call prxchange(rx1,-1,wonky_field);
run;
```

MACRO: %CSVPREP_PROCIMPORT STEP-BY-STEP

The macro in APPENDIX 1 defines a solution that will replace or remove whitespace characters from a CSV. Optionally it will create a new CSV, either as requested or if it must. A new CSV must be created if deletions are occurring.

This solution started with the code provided by Nat Wooding as part of Southeast SAS Users Group Conference 2023 (Wooding, 2023) The base of this code is present in several other posts in Stack Overflow, SAS Communities, and previous SAS papers (*See* Recommended Reading *LexJansen.com).*

THE MACRO CALL

To run this code, you will first need to run the definition code either using an %INCLUDE statement (as below) or by copying the code into your program. Once you do so, you will be able to run this code using the following call and parameters. These are the “instructions” you are giving to the macro to help guide its actions as it decides and then removes or replaces pesky characters.

```
*point to location of definition code;
%let filepath = [Location of the Code];
%INCLUDE "&filepath.\CSVprep_PROCIMPORT.sas";


*Run Code;
%CSVprep_PROCIMPORT (IN_CSV=%bquote ()
      , OUT_CSV=IN_CSV
      , rep_with=' '
      , rep_chars = %str('0D'x,'0A'x)
      , delete_multi_chars = %str('20'x,'09'x)
      );
```

Parameter (*= Required)	Description & Guidance	Default
***IN_CSV**	File path to original CSV file. Use %bquote () if path contains spaces or problematic characters. Example: C:\Users\tester\Desktop\CSVprep_PROCIMPORT.sas	None
OUT_CSV	File path to new/output file OR cue to use the value of IN_CSV. Provide a file path in the same syntax as IN_CSV or the value IN_CSV to point back to the input file.	IN_CSV
REP_WITH	The single character you would like to replace the characters provided in REP_CHARS. This can be any quoted character or a hex character with syntax '[hex code]'x. List only one character.	Space
REP_CHARS	A comma-delimited list of characters (no limit) to be replaced with the character from REP_WITH. This can be any quoted character or a hex character with syntax '[hex code]'x resolving to a length of 1 (single character). Use the %STR () function as a wrapper to avoid issues with commas in the value. Example: '$','09'x Not Allowed: '_!' (length 2+), 8 (unquoted values)	Carriage Return and Line Feed
DELETE_MULTI_CHARS	A comma-delimited list of characters (no limit) to be removed when the character before it is also in the list. This can be any quoted character or a hex character with syntax '[hex code]'x resolving to a length of 1 (single character). Use the %STR () function as a wrapper to avoid issues with commas in the value. **NOTE:** This executes after the replacement. Sequence does not have to be of the same value (Example – under defaults - space tab carriage tab space à 1) Replace to: space tab space tab space à 2) Remove to: Space - since subsequent space/tabs are in the delete multi list).	Space and Tab

DECIDING AN OUTPUT: FILENAME STATEMENTS

FILENAME statements are used to provide paths to the CSV file(s). We are naming the first one “csvfile” and using the value of the macro variable “IN_CSV” as its path. This identifies the file at the path providing in the macro call as the first file. Afterwards, the code will reference that file by its name (csvfile) instead of with the path to the CSV.

```
filename csvfile "&IN_CSV.";
```

Next, we will use macro logic to determine if a second file needs a name. If using the value IN_CSV, then it names another file. A single file/path can be given multiple “filerefs”, so if a separate path is not provided, we give the path of the IN_CSV. In this case “csvfile” and “csvfile2” reference the same CSV.

```
%if &OUT_CSV. = IN_CSV %then %do;
      filename csvfile2 "&IN_CSV.";
      %end;
%else %do;
      filename csvfile2 "&OUT_CSV.";
      %end;
```

A macro variable named “REPLACE_IN_FILE” is also defined and given a value of Y or N to use in later macro logic.

```
      %let REPLACE_IN_FILE = Y;
```

Finally, a message writes to the log file with a %PUT statement to document the logical outcome and intended steps. By using the prefix of “NOTE:” or “WARNING:” I am also leveraging the power of the log to color code them (Figure 5) and utilize them just as a system generated note or warning.

```
%put NOTE: A new csv will be created with the requested changes.;
```

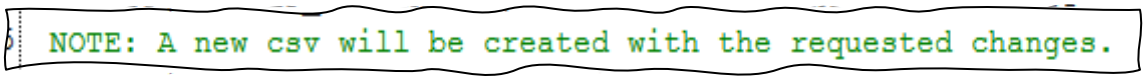


Figure 6 Preview of Log File with NOTE

Finally, the FILENAME statement option CLEAR, is used to get rid of the file reference so that there are no conflicts in subsequent runs. This disassociates the names csvfile and csvfile2.

READING IN THE FILE: BINARY FORMAT

A DATA step is used to do the processing, but instead of creating a SAS dataset we are going to use the CSV file(s). To do this we will use _NULL_ to stop the creation of the SAS dataset, a INFILE statement to identify the input, and a FILE statement to identify the output. The RECFM option will be set to N to request that the file be read in as a single stream with no record boundaries. Each byte will be considered one at a time. The SHAREBUFFERS option will conditionally be used on the INFILE statement when the input and output files are the same AND no deletions are occurring so that they share the same buffer. This cannot be done if they are different files or when deletions are occurring.

```
      data _null_;
        infile csvfile recfm=n
            %if &REPLACE_IN_FILE. = Y %then %do; SHAREBUFFERS %end;
            ;
      file csvfile2 recfm=n;
      ...
```

Next, we will set the input statement to treat each byte as a character value with a length of one and name the variable “a”.

COUNT VARIABLES & QUOTING

Since each “record” is only one character, we will use a RETAIN statement to retain information about previous actions and findings**.**

```
retain open 0 streak 0 deletes 0 changes 0;
```

As different actions are taken, we are using these variables to track them and updating macro variables with the same names as we go along. A macro variable is required to allow for communication outside the data step, so those are updated as we go along.

- **CHANGES** **:** Counts each time a character is changed to a different character.
- **STREAK** **:** Characters in a row belonging to the list we provided. Can be different characters but must be in a row.
- **DELETES** **:** Counts each time a character is eliminated from the stream.

The ^() syntax works only with binary indicators and changes the value from 0 to 1 OR 1 to 0. We are using the following to track whether the current byte is occurring within quotes.

```
if a = '"' then open = ^(open);
```

ACTIONS: REPLACE OR DELETE

Since we are considering the file one byte at a time, we do not need to perform any string functions for finding the pesky characters. The action of replacing or deleting the byte will occur using simple if/then logic. In the case of replacement, the value of the macro variable will resolve, and a simple variable definition statement will be used to overwrite the previous value. The system options MPRINT and SYMBOLGEN will cause these actions to be viewable in the log (Figure 6).

```
a = &rep_with.;
```

```
SYMBOLGEN:  Macro variable REP_CHARS resolves to '0D'x,'0A'x
MPRINT(CSVPREP_PROCIMPORT):   if a in('0D'x,'0A'x) then do;
MPRINT(CSVPREP_PROCIMPORT):   changes + 1;
SYMBOLGEN:  Macro variable REP_WITH resolves to ' '
MPRINT(CSVPREP_PROCIMPORT):   a = ' ';
```

Figure 7 Using SYMBOLGEN to see a resolved value.

THE END: REPORT ACTIONS TAKEN

Since this is iterating through the whole file one byte at a time, we will again use a %PUT statement to write a conclusion statement to the log.

```
%put NOTE: File cleaning complete. There were &delete_count. deleted
characters and &change_count. changed characters.;
```

```
NOTE: File cleaning complete. There were 23 deleted characters and 8 changed characters.
```

Figure 8 Example Cleaning Result NOTE in Log File

USE CASE IN PRACTICE: TRAUMA REGISTRY ETL

BACKGROUND

The Marshfield Clinic Health System is a primarily rural health care system in Wisconsin. Originally starting as a single clinic in Marshfield, WI and growing from there has resulted in different trauma registry collection systems, collection periods, and geographical location expansion.

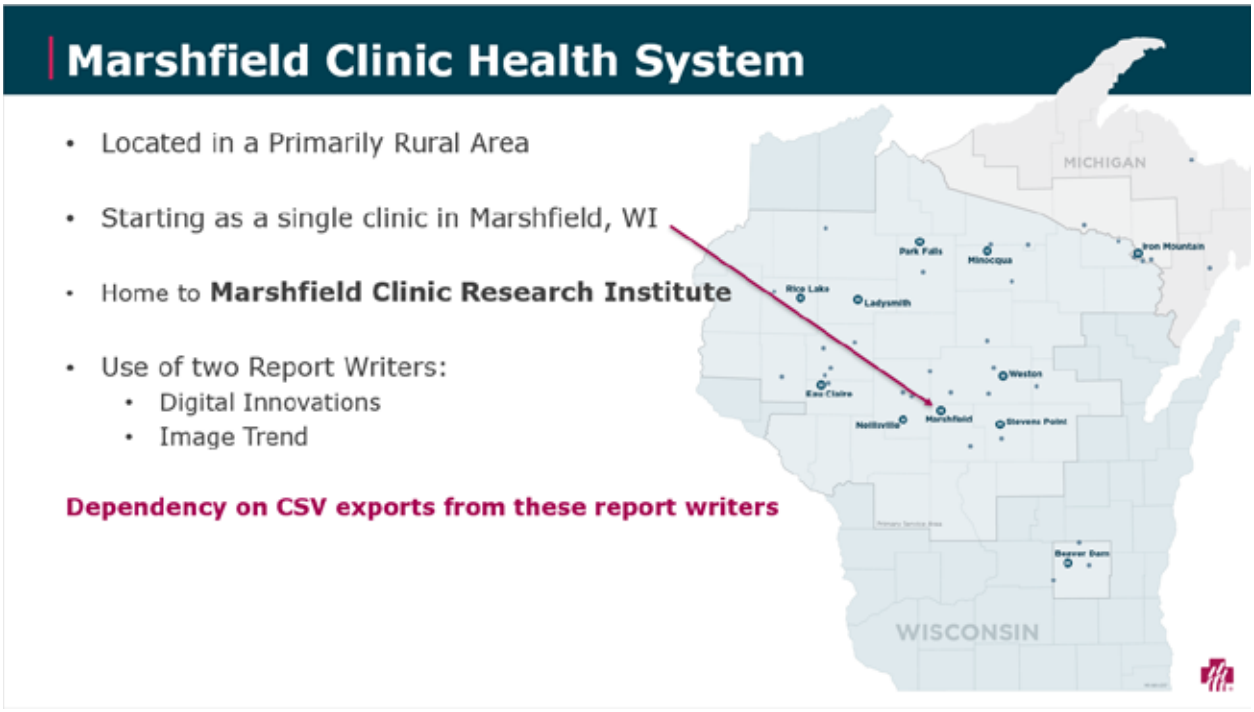


Figure 9 Marshfield Clinic Health System Infographic and Map

For the trauma registry data to be used as a source for trauma research, a combined data model is needed. Dr. Heather Rhodes formed a combined data dictionary encompassing 86 tables and 15,649 distinct fields from available data. The tables cover topics such as Injury Information, Transfer Details, Vitals, and Resuscitation Efforts. Since the data are obtained to do reporting at state and national levels, available data aligns with or exceeds the National Trauma Data Standard (NTDS) published by the American College of Surgeons (ACS) Committee on Trauma (ACS Committee on Trauma, 2025). Reports were developed in DI V7 Hospital Trauma Registry™ and Image Trend™ then exported from their respective systems by an authorized registrar, Heather Berg.

As of June 2025, a total of 22.55 GB of raw data coming out of the report writers was delivered for consideration in the model. These data come from 8 geographical locations, cover up to 28 years (with variance by source system per location), and resulted in a combined ~32,000 unique trauma events/rows of data. As this data source continues to grow, the solution will need to be increasingly considered of resource needs: time, system resources, and storage space. Hence, we have started the process here – with attempts to compress the data to its most compact form without the loss of meaningful data.

METHODS

Each CSV was reviewed using NotePad++ and scanned using a variety of search methods (as mentioned in FIND(), PRXCHANGE(),TRANWRD(), and TRANSLATE() for Finding White Space Characters). The files were found to have space padding entered by the exporter on several fields. There were also a number of fields that were free-text fields created by Rich Text editors in the source system. These contained a variety of complicating factors such as: HTML tags, foreign language alphabetic characters, style-related white space characters, and other characters (like emojis) represented by Unicode. Of the highest trouble were line feed characters causing a similar issue to in Figure 1. To alleviate this issue, each file was run through the macro first with a Line Feed and Carriage Return to

Space reduction with de-duplication on Spaces and Tabs. Afterwards, each field was stripped using the STRIP () function to eliminate leading and trailing blanks caused by having a streak equal one in a location where it could still be eliminated without loss of meaningful data.

A quality assurance comparison was performed for all data electronically and up to 5% of rows from each file. PROC COMPARE was used to compare row to row between raw and clean versions. Frequencies of impacted characters were reviewed using PROC FREQ. PROC SURVEYSELECT was used to select random samples for review, by the row id and reviewed both the CSV itself and in imported SAS datasets to be sure no discrepancies existed. Special attention was paid to reviewing fields identified as having a length greater than fifty, as these were free-text note fields most likely to be impacted.

RESULTS

All 86 raw CSV files were placed in a common directory where they measured at 22.55 GB. For all 86 files to be imported into SAS datasets took 12 hours. After running through the macro there was a dramatic reduction to 1.99 GB (91.15%). There was also a dramatic decrease in the amount of time it took to import the files, from 12 hours to 36 minutes (95% reduction in run time). There were 12 of 86 CSV files that were not able to be effectively imported using PROC IMPORT due to line feed errors. All files were appropriately imported afterwards.

There were 67 tables identified for cleaning. Each containing at least one white space character. Of those, 47 did not have any replacements/changes and needed only deletion of sequential white space characters. Of the 20 tables that had at least one Carriage Return or Line Feed changed to a space, all but two did not import correctly without the use of a DATA step previously.

These CSV files were inundated by white space, with the average number of characters deleted at 328.4 million per file.

Table 1. Summary of Changes by CSV File

N = 67 tables	Min	Mean	Max	StdDev	5th PCT	95th PCT
Number of Characters Deleted	44K	328.4M	1.8B	498.8M	2.7M	1.5B
Number of Characters Changed	0	4.8K	182K	23K	0	15K

A quality assurance review found no differences between raw an imported version. While not meeting the expectation to have one row per trauma event identifier, no data were “lost” in this transaction. Frequency tables of impacted characters were confirmed to only contain the targeted characters and

LIMITATIONS, KNOWN ISSUES, AREAS FOR FUTURE DEVELOPMENT

The main goal of interest was to be able to successfully import the data into SAS for further steps related to more substantial cleaning efforts. Knowing that additional cleaning efforts would ensue, it was decided to only update the CSV files in ways that did not have the potential to change our ability to represent data as entered the source system. In this way, preservation of “ground truth” outweighed inclusion of further cleaning steps.

Another limitation was the size of the data as compared to run time. This is seen as a “first pass” solution that can and should be followed by cleaning that addresses remaining “needs” such as:

- Leading and Trailing White Space Characters: Since the goal was to look for *streaks* there remains the start and end of the string where single whitespace characters could be removed by STRIP () or COMPRESS () functions.
- Multiple Translations: There are times when running it to first convert characters {a, b, c} to a single character {d} may not be the desired approach. It is possible to run the macro multiple

times on the same input/output, but this could be resource consuming. Modification to consider multiple pairings may be needed.

- Special translations: There were times when a series of blanks was intentionally used to indicate a missing value that had meant to be filled in later. For example: Terry arrived at Bayview Hospital with (mother). We determined that removing that series did not change the meaning of the information, but an alternative would be to first run a TRANWRD () to change ‘ ‘ (5 blanks in a row) to ‘_____’ (5 underscores in a row) to better retain ground truth meaning.
- Other Characters: While the present use case is to replace and remove whitespace characters, this could also be used to translate and replace repetition of any provided character. While not explored/assessed, it may be particularly helpful in troubleshooting locations of problematic characters if your replacement is a hex referenced symbol but not found in the data otherwise, maybe © (‘A9’x) for example.

CONCLUSION

When two common place concepts, white space characters and delimited files, come together there are straightforward ways to substantially reduce the amount of noise they create. While whitespace characters may not be easy to see, they are easy to replace and remove. The advantage to managing whitespace characters can be in physical size reduction (reducing of storage capacity needed). It can also increase readability by machines and humans alike.

These easy and advantageous actions have applications in many workplace contexts. The industry application it was developed on was a large-scale project to combine trauma data collected in registries that showed improvements in the ninety percentiles for both final size and speed.

Being that CSV files are amongst the most basic and plentiful interoperability data tools today, the macro %CSVprep_PROCIMPORT can join CSVs everywhere. The macro may be used as a first step to PROC IMPORT (as its name suggests) or in any use of a delimited data file containing problematic or excessive whitespace characters.

ACKNOWLEDGEMENTS

Dr. Heather Rhodes your dedication to trauma research brought this wonderful opportunity to my desk. Your support has been/is essential to the success of this initiative and I know our work will save more lives and lead to better care! The months of work it took to produce the data dictionary deserves my heaviest praise. You rock!

Heather Berg Thank you for always being patient as we ask questions, being thorough in your work, the time you dedicate to this above and beyond your “day job” and supporting trauma research.

This project and attendance at the conference was financially supported in part by the **Raymond Goldbach Endowment**. The endowment fund was started by Marie S. Goldbach to honor the memory of her husband, Raymond Goldbach. (Marshfield Clinic Health System, 2024). They started Marathon Cheese Corporation in Marathon City in 1952. We could not have done this without your generosity and continuous support of clinical research at Marshfield Clinic Research Institute.

RECOMMENDED READING

- *Mozilla Glossary of Web Terms: https://developer.mozilla.org/en-US/docs/Glossary*
- *ASCII Whitespace Character Codes: https://www.ascii-code.com/characters/white-space-characters*
- *SAS Expression Language 2.7: Reference Guide: ISNULL Function: https://documentation.sas.com/doc/en/engelref/2.7/n1fbuh8guwn16an1lfa3zj477bui.htm*

REFERENCES

ACS Committee on Trauma. (2025). *National Trauma Data Standard (NTDS) Data Dictionary*. Retrieved from American College of Surgeons: https://www.facs.org/quality-programs/trauma/quality/national-trauma-data-bank/national-trauma-data-standard/

Marshfield Clinic Health System. (2024, 06 14). *Rhodes named holder of Goldbach research endowment.* Retrieved from Marshfield Clinic Research Institute News: https://marshfieldresearch.org/News/rhodes-named-holder-of-goldbach-research-endowment

Mozilla Foundation. (2025, 07 11). *Glossary - Whitespace*. Retrieved from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Glossary/Whitespace

SAS Institute. (2016, 10 02). *SAS® Visual Data Mining and Machine Learning 8.1*. Retrieved from Using PC Files in Your SAS Session: Delimited Files: https://documentation.sas.com/doc/en/vdmmlcdc/8.1/acpcref/p07k868np8cd18n1teen33x5bym7.htm#:~:text=%2C%20RECFM=%20option.-,CSV%20Files,other%20applications

SAS Institute, & Mincey, M. (2022, 04 18). *Getting started using SASPy® and the SAS® kernel for Jupyter Notebook.* Retrieved from SAS Blogs: https://blogs.sas.com/content/sgf/2022/04/18/saspy-and-the-sas-kernel-for-jupyter-notebook/#:~:text=Using%20the%20SAS%20kernel%20for,notebook%2C%20choose%20New%20%3E%20SAS.

Wooding, N. (2023). Reading Comma Separated Files containing CR/LF embedded in quoted strings. *Southeast SAS Users Group (SESUG) Conference.* Charlotte, NC. Retrieved from https://sesug.org/proceedings/sesug_2023_final_papers/Learning_SAS_I/SESUG2023_Paper_212_Final_PDF.pdf

CONTACT INFORMATION

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

Brooke Ellen Delgoffe, M.S.
Marshfield Clinic Research Institute
715-221-8871 (work)
brooke_delgoffe@hotmail.com
https://marshfieldresearch.org/profiles/11610
https://www.linkedin.com/in/brookedelgoffe/
https://www.researchgate.net/profile/Brooke-Delgoffe
https://redcap.link/DelgoffeContentCenter (QR at Right)

Presentation • Paper • Code

APPENDIX 1

```
/*********************************************
PROGRAM          : Clean CSV for PROC IMPORT.sas
PROGRAMMER       : Brooke Ellen Delgoffe
DATE             : 02-07-2025

PURPOSE: Preps csv files by
-REPLACING characters that will cause import issues.
-DELETING Multiple Subsequent Characters, generally whitespace
     characters for purpose of decreasing size of file when
     padding spaces or tabs exist.


Parameter Definitions:
-----------------------------------------------
IN_CSV                : Input CSV File Path and Name
                DEFAULT: None - Required Input
OUT_CSV               : Output CSV File Path and Name
                      Required to be a different file
                      when DELETE_MULTI_CHARS = Y.
                DEFAULT: IN_CSV Value
REP_WITH         : Character to Replace Problem Characters with
                DEFAULT: Single Space
REP_CHARS        : Characters to Replace (Problem Characters) when
                      within double quotes. These cause new line issues
                      during import.
                DEFAULT: Carriage Return (0D)  Line Feed (0A)
DELETE_MULTI_CHARS: Characters to delete when the character
                      before it was also one of these. Designed to
                      eliminate double spaces, space padding, remove
                      tabbing used for styling, and other whitespace
                      that can be removed to reduce the size of the
                      file without decreasing
                      the content.
                      Works inside or outside of quoted (character
field) data.
                DEFAULT: Space (20)  Tab (09)
                OPTIONS: N (Turn Off) | {List of ANSII Values}

Example Call:
-----------------------------------------------
     *Make New CSV;
%CSVprep_PROCIMPORT(IN_CSV=%bquote(\\mcrf1\MMRhome\delgoffb\SAS\SESUG
2025\TestCSVCleaning.csv)

     ,OUT_CSV=%bquote(\\mcrf1\MMRhome\delgoffb\SAS\SESUG
2025\TestCSVisClean.csv)
                          );
```

```
      *Replace in File;
%CSVprep_PROCIMPORT (IN_CSV=%bquote(\\mcrf1\MMRhome\delgoffb\SAS\SESUG
2025\TestCSVisClean.csv));

Notes / Known Issues
-----------------------------------------------
*!!!  WARNING: This makes changes to CSV files by default    !!!*
Check settings carefully to avoid making undesired changes to input
CSVs


*********************************************/;;


%macro CSVprep_PROCIMPORT (
      IN_CSV=%bquote()  /* *REQUIRED* Input CSV File Path and Name*/
      ,OUT_CSV=IN_CSV  /* Output CSV File Path and Name*/
      ,rep_with=' ' /*Default Character to Replace Problem Characters
with*/
      ,rep_chars = %str('0D'x,'0A'x)   /*Characters to Replace CR (0D)
or LF (0A)*/
      ,delete_multi_chars = %str('20'x,'09'x) /*Characters to Delete
when there are multiple in a row. Space (20) or Tab (09) */
                        );;

/************************/
/*    File Pointers   */
/************************/
*Use filename statements to give a name to the files to be read,
modified, or created;
filename csvfile "&IN_CSV.";
%if &OUT_CSV. = IN_CSV %then %do;
      filename csvfile2 "&IN_CSV.";
/*****************************************************
*                                                    *
*     WARNING: This can make changes to the CSV files*
*                      or make new csv files.        *
*           Place Warning in Log if making changes to*
*                the file to be sure that is known.  *
*****************************************************/
      %let REPLACE_IN_FILE = Y;
      %put WARNING: Changes will be made to the input csv. This cannot
be undone and in rare circumstances may cause unforeseen damage to the
file.;
      %end;
%else %do;
      filename csvfile2 "&OUT_CSV.";
      %let REPLACE_IN_FILE = N;
      %put NOTE: A new csv will be created with the requested changes.;
      %end;

*Initiate Macro variables to count modifications;
```

```
%let delete_count = 0;
%let change_count = 0;

      data _null_;
        /* RECFM=N reads the file in binary format. The file consists
of a stream of bytes with no record boundaries. */

        /*SHAREBUFFERS specifies that the FILE statement and the INFILE
statement share the same buffer. This can only be used if input and
output file are the same file. They can only be the same if NOT
changing the size of the file through addition or deletion of
characters. */
        infile csvfile recfm=n
            %if &REPLACE_IN_FILE.=Y %then %do; SHAREBUFFERS %end;
                  ;;
        file csvfile2 recfm=n;

        /* OPEN is a flag variable used to determine if the CR/LF is
within double quotes or not. Retain this value. */
/* STREAK is a count variable that counts how many of a character are
in a row. */
        retain
                  open 0
                  streak 0
                  deletes 0
                  changes 0
            ;
      /*Input all bytes as character*/
        input a $char1.;


        /* If the character is a double quote, set OPEN to its opposite
value. */
        if a = '"' then open = ^(open);

              /* If the character is after an open double quote,
replace the byte with the appropriate value. */
      if open then do;
            if a in(&rep_chars.) then do;
            changes + 1;
            a = &rep_with.;
            call symput("change_count", strip(put(changes,best.)));
            end;
        end; *End in Quote Only Loop;

        /* Track Sequential Chars - If the character is in the list and
the last one was too*/
        if a in(&delete_multi_chars.) then streak+1; else streak=0;

      %if &REPLACE_IN_FILE. ne Y %then %do;
        %if &delete_multi_chars. ne N %then %do;
       if streak le 1 then put a $char1.;
```

```
       else if streak ge 2 then do;
            deletes + 1;
            call symput("delete_count", strip(put(deletes,best.)));
            delete;
            end;
         %end;
         %else %do;
            put a $char1.;
         %end;
      %end;
run;

%put NOTE: File cleaning complete. There were &delete_count. deleted
characters and &change_count. changed characters.;

*Remove file references;
filename csvfile clear;
filename csvfile2 clear;

%mend;
```