SESUG Paper 124-2019

PROC IMPORT and more.
Or: when PROC IMPORT just doesn't do the job

David B. Horvath, MS, CCP

ABSTRACT

PROC IMPORT comes in handy when quickly trying to load a CSV or similar file. But it does have limitations. Unfortunately, I've run into those limitations and had to work around them. This session will discuss the original CSV specification (early 1980's), how Microsoft Excel violates that specification, how SAS® PROC IMPORT does not follow that specification, and the issues that can result. Simple UNIX tools will be described that can be used to ensure that data hilarities do not occur due to CSV issues. Recommendations will be made to get around some of PROC IMPORT limitations (like field naming, data type determination, limitation in number of fields, separator in data). CSV, TAB, and DLM types will be discussed.

INTRODUCTION

In this paper I will be covering some of the basics of the PROC IMPORT, tips and tricks that can be used with this PROC, and the times you should not use PROC IMPORT (and the code involved).

I am going to focus on code rather than the GUI Import Wizard. PROC IMPORT is a handy tool to ingest (load) data from other sources. You provide the flat file and the SAS Engine creates the dataset. There are three primary formats:

- CSV – Comma Separated Variables
- TAB – Tab (ASCII ‘09x’ EBCDIC ’05’x)
- DLM – You pick it

Particular attention will be paid to quoted strings which receive special treatment in delimited files (doesn’t matter if you’re using PROC IMPORT or INFILE with DSD and DLM).

PROC IMPORT BASICS

PROC IMPORT will scan an input text file and attempt to define the layout for you. Let’s start out with the basic syntax for the PROC:

```
PROC IMPORT
DATAFILE="filename”
OUT=<libref.>SAS data-set <(SAS data-set-options)>
<DBMS=identifier>
<REPLACE> ;
    <data-source-statement(s);>
```

DBMS defines the data source type. CSV, TAB, and DLM options will be discussed here. The other available options could occupy an entire paper themselves; the available options vary with operating system and installed components.

REPLACE will allow the output dataset to be overwritten (replaced). Without this option, an error will be reported if the dataset already exists.

There are a number of optional Data-source-statements:

- DELIMITER is usually only specified with DBMS=DLM and will define the character(s) that separate fields within your file. You can override DBMS=CSV with this option. Multiple values are allowed within a string:
  `DELIMTER='!|'`
- GETNAMES will define the SAS variable names based on a header record in the file ("YES" is the default) and will "sasify" the names – attempt to convert the names into a standard form acceptable to SAS (underlines replace spaces, length, etc.). This will cause DATAROW to default to the value of 2.
- DATAROW=n defines the first source file record that contains data (not skipped). In many ways, DATAROW is similar in function to FIRSTOBS. The default depends on GETNAMES: 1 if "NO", 2 if "YES".
- GUESSINGROWS=n tells the wizard how many records to process while it is attempting determine the layout of the text file you provided – building input and format statements. The default setting is 20 records; if this is not enough for your process, you can set it higher. Either way, there is a tradeoff between time and accuracy – higher count gives greater accuracy

PROC IMPORT EXAMPLE

The best way to understand how this all works is with an example:

Error! Reference source not found. shows an example input CSV file from SAS Documentation.

```
"Africa","Boot","Addis Ababa","12","$29,761","$191,821","$769"
"Asia","Boot","Bangkok","1","$1,996","$9,576","$80"
"Canada","Boot","Calgary","8","$17,720","$63,280","$472"
"Central America/Caribbean","Boot","Kingston","33","$102,372","$393,376","$4,454"
"Eastern Europe","Boot","Budapest","22","$74,102","$317,515","$3,341"
"Middle East","Boot","Al-Khobar","10","$15,062","$44,658","$765"
"Pacific","Boot","Auckland","12","$20,141","$97,919","$962"
"South America","Boot","Bogota","19","$15,312","$35,805","$1,229"
"United States","Boot","Chicago","16","$82,483","$305,061","$3,735"
"Western Europe","Boot","Copenhagen","2","$1,663","$4,657","$129"
```

Input 1. Input example CSV file

The code itself is very simple:

```
proc import datafile="csv1.csv"
    out=shoes dbms=csv replace;
    getnames=no;
run;
```

Error! Reference source not found. shows some of the SAS code generated as a result – only handling the first variable due to space limitations.

```
7          /*********************************************************************
8          *   PRODUCT:   SAS
9          *   VERSION:   9.2
10         *   CREATOR:   External File Interface
11         *   DATE:      10JUN14
12         *   DESC:      Generated SAS Datastep Code
13         *   TEMPLATE SOURCE:  (None Specified.)
14         *********************************************************************/
15            data WORK.SHOES                                          ;
16            %let _EFIERR_ = 0; /* set the ERROR detection macro variable */
17            infile 'csv1.csv' delimiter = ',' MISSOVER DSD lrecl=32767 ;
18               informat VAR1 $34. ;
```

```
...
25                format VAR1 $34. ;
...
32             input
33                          VAR1 $
...
40             ;
41             if _ERROR_ then call symputx('_EFIERR_',1);  /* set ERROR detection
macro variable */
42             run;
```

Output 2. Subset of generated SAS Code

Error! Reference source not found. shows the contents of the resulting SAS Dataset.

```
Obs    VAR1                                  VAR2      VAR3             VAR4
VAR5           VAR6           VAR7
  1    Africa                                Boot      Addis Ababa      12
$29,761        $191,821       $769
  2    Asia                                  Boot      Bangkok          1
$1,996         $9,576         $80
  3    Canada                                Boot      Calgary          8
$17,720        $63,280        $472
  4    Central America/Caribbean             Boot      Kingston         33
$102,372       $393,376       $4,454
  5    Eastern Europe                        Boot      Budapest         22
$74,102        $317,515       $3,341
...
 10    Western Europe                        Boot      Copenhagen       2
$1,663         $4,657         $129
```

Output 2. Subset of the resulting SAS Dataset

Since GETNAMES was set to “NO”, the first record in the file was treated as a data record and simple variable names were generated (VAR1, VAR2, …). If the file did contain column names (the cause with most generated CSV files) and GETNAMES was set to “YES”, then the variable names would be more meaningful.

THE CSV SPECIFICATION AND ISSUES

“CSV” stands for Comma Separated Values; when it first came out, there was no formal specification. As a result there were different interpretations. Historically, the data layout can be traced to the FORTRAN 77 “list directed” or “free form input” specification.

The first mention of CSV processing I could find in SAS was from TS-673 (version 6). However, processing included some flaws:

- Did not handle empty fields (two delimiters in a row)
- Did not handle quoted strings containing delimiter
-

In addition, each vendor implemented their own “standard” which might or might not support quoted strings. As a matter of fact, Excel did not quote strings for a long time. While that is not normally a problem, if your data contains the delimiter (a comma in this situation), it would appear as two fields rather than one. I’ve encountered programmatic CSV creation that did not quote strings because the developer did not know any better.

There is now a definitive reference now: RFC 4180 published by the Internet Engineering Task Force (IETF) in 2005. The specified format is basically:

Optional Header (usually column names):

```
Field_name,field_name,field_name
```

Followed by Datalines:

```
aaa,bbb,ccc
zzz,yyy,xxx
```

Optionally, strings are contained within single or double quotes

```
"aaa","bbb","ccc"
```

If a quoted string contains the quote character, it must be escaped:

- "Af"r,ica" Becomes “Af”r and ica” fields
- "Af""r,ica" Becomes “Af”r,ica” single field

Each line should have a line terminator. The handling of quotes and formatting really only matters if you have issues importing.

PROBLEMS CAUSED BY MISSING QUOTATIONS

There are a number of problems you can encounter with the CSV. When a field contains the delimiter, the data can cause improper loading. The consistency of data really determines what happens. For instance, if the data is consistent, the results will be consistent (in this case, you’ll have two fields):

```
Applicant, Instructor
Resources, Human
Management, Fiscal
```

Unfortunately, if the data is not consistent, the results will not be either:

```
Applicant, Instructor
Resources, Human
Training
```

You can program around having two fields, but it is much more difficult to handle situations where the number of fields changes. In addition, when there are inconsistencies between files, you may have to change your code for those files – something difficult when dealing with locked down production processes. Some issues I’ve encountered include:

- Inconsistent number of fields
- Column names exist in some files (but not all)
- Page titles

RESOLVING ISSUES WITH CSV FILES

Personally, I advise against using real CSV files with comma delimiters. I recommend the use of tab delimited files in place, with the use of other delimiters (like the pipe character “|”).

If you receive a file that is not in the proper format, you have a few choices to make it correct:

- Ask for a new version properly formatted
- Manually edit the file
- Use some UNIX and SAS tricks

As a result, I am going to present a few tips and tricks.

MODIFYING GENERATED CODE

My favorite trick is to let PROC IMPORT write the initial code for a particular file and then copy/paste/edit the resulting DATA step into the program. That means I don’t have to type in all the variable names or

formats – I freely admit to being lazy. But that means I can also alter variable names, formats, and data types as appropriate. While the wizard is pretty good at selecting formats, it only looks at a small subset of records to make those determinations.

A good example of this is date fields. Excel will let you enter '14-JUL' as a date, treat it as July 14 2017 (the year entered) for calculation purposes, but write it out to a CSV as '14-JUL'. As a result, the PROC IMPORT wizard may not interpret the field correctly or could treat it as a date which will not be input correctly. As a programmer, I can input the field as a character string and programmatically convert '14-JUL' to a correct date in a result field.

I can also perform additional data validation – in addition to the date corrections, you could validate amount fields that may contain non-numeric characters and handling appropriately.

In addition, you can make use of more INFILE options like DLMSTR, DLMSOPT, END, and others to better handle the input.

One little trick to convert the Wizard's generated code in the log makes use of the UNIX tool to cut columns out of a file:

```
cut -c 10- csv3.log > data.txt
```

Will eliminate row numbers and formatting in log columns 1-9.

WORKING WITH A DELIMITED FILE NOT QUOTE ESCAPED

This problem was originally reported by Don Stanley bchr0001@GMAIL.COM

This is the case where quotes are included in the record and are just data:

```
AAAA.BBBB.'I.CCCC'.0.DL
```

Note that the field delimiter is the period (".").

In this case, both PROC IMPORT and INFILE DLM DSD wants to treat the string `'I.CCCC‘` as one field, but in reality, it is two fields. In this case, DSD was included because of the possibility of empty fields. But DSD also invokes the quoted field handling.

There are multiple solutions – within SAS or external using UNIX/Linux commands.

SAS Solution (I lost the contributor name)

```
infile custdata dlm='.' ; /* DSD not needed */

input @ ;
do while(index(_infile_,'..')>0) ;
  _infile_ = tranwrd(_infile_,'..','. .') ;
end ; /* period-period to period-space-period */

input << delimited fields >> ;
```

Because the _infile_ variable is limited to 32,767 characters, this solution will not work in call cases (failing with large records). In those cases, you certainly can use the UNIX/Linux Shell Solution

UNIX/Linux Shell Solution

The following command will use a small program written in the awk scripting language to convert two periods in sequence to period-space-period. As a result, you will no longer have two delimiters together and can avoid the use of the DSD statement.

```
awk '{gsub("..", ". .");print $0;}' INPUTFILE > OUTPUTFILE
```

Depending on the UNIX/Linux implementation, the command may be awk, nawk, or even gawk. This will work on essentially unlimited length records. You can also execute this command under Microsoft Windows using CYGWIN or the Windows 10 Bash Shell.

SOME ADDITIONAL UNIX/LINUX TIPS

UNIX/Linux provides a number of commands to help you see what is contained in the input file. These come in handy when you are trying to figure out exactly what the file looks like or why you care not able to ingest the data as you expect.

The `head` command will show the first few rows. This is very handy when trying to answer questions like "Does the file have extra lines at the top (like a title line)?"

The `tail` command will show the last few rows and is handy when trying to answer questions like "Do I have all the data?" and "Are the records consistent beginning to end?"

The `wc -l` (word count, line) command will show how many lines in total. That way you can easily determine if the file is the right size!

You can always edit the file in your preferred editor (like `vi` or `notepad+`) – that is, unless the file is so large that it will not load.

The `od -c -x` (object dump, character and hex) command is useful to see format and contents of file (including unprintable characters).

What Does My File Look Like? Another awk Program

A common question with input files involves the maximum record length and field count based on provided delimiters. An awk program will process large volumes of data in short time periods.

```
BEGIN{   FS=",";   mNF=0;   mRL=0; }
{
   if (NF > mNF) mNF=NF;
   RL=length($0);
   if (RL > mRL) mRL=RL;
}
END{   print "max NF " mNF " max RECL " mRL;}
```

"Max NF" is highest number of fields on any record (does not handle quote escaping) and "Max RECL" is the longest record length in the file. This code can be saved to a file and executed on your data.

```
awk –f csv_check.awk INPUTFILE.csv
```

DROPPING TITLES

A common problem are title record(s) in a file before the column headings. You have a choice – you can edit the input file or you can set GETNAMES=NO and DATAROW=3. However, with GETNAMES/DATAROW, you will get default variable names (VAR1, VAR2, …) rather than the column headings. My preference is to edit the data file using UNIX/Linux commands (because I can put this in a script).

Error! Reference source not found. shows an example input CSV with a header.

```
This is a title line and should be here
country,shoe,city,count,average salary,max salary,average
"Africa","Boot","Addis Ababa","12","$29,761","$191,821","$769"
"Asia","Boot","Bangkok","1","$1,996","$9,576","$80"
"Canada","Boot","Calgary","8","$17,720","$63,280","$472"
```

Input 2. Input example CSV file

Error! Reference source not found. shows the results of executing PROC IMPORT on this data.

```
Number of names found is less than number of variables found.
Name This is a title line and should be here truncated to
This_is_a_title_line_and_should.
Problems were detected with provided names.  See LOG.
```

Output 3. PROC IMPORT Results

How to get rid of the first line since DATAROW= won't help:

`wc -l INPUT` – returns value 5 (5 lines)

`tail -4 INPUT > OUTPUT` – saves last 4 lines

DROPPING JUNK RECORDS AT THE END

Another common problem are junk record(s) at the end of a file. Unfortunately, OBS= will not work to ignore those records. You could check _N_ to force the datastep to complete early or (again) you can edit the data file using UNIX/Linux commands (which I can put this in a script).

Error! Reference source not found. shows an example input CSV with junk data.

```
country,shoe,city,count,average salary,max salary,average
"Africa","Boot","Addis Ababa","12","$29,761","$191,821","$769"
"Asia","Boot","Bangkok","1","$1,996","$9,576","$80"
"Canada","Boot","Calgary","8","$17,720","$63,280","$472"
,,,,,,
```

Input 3. Input example CSV file

Error! Reference source not found. shows the results of executing PROC IMPORT on this data.

Obs	country	shoe	city	count	average_ salary	max_salary	average
1	Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
2	Asia	Boot	Bangkok	1	$1,996	$9,576	$80
3	Canada	Boot	Calgary	8	$17,720	$63,280	$472
4							

Output 4. PROC IMPORT Results

How to get rid of the last line since DATAROW= won't help:

`wc -l INPUT` – returns value 5 (5 lines)

`head -4 INPUT > OUTPUT` – saves first 4 lines

CONCLUSION

On a personal note, I seem to learn a great deal when working on presentations, new classes, and papers like this. Prior to creating the presentation and this paper, I was treating PROC IMPORT like it was

still V6 when it didn't handle quoted strings properly. In creating examples, I got to learn some new habits and will trust the wizard more.

In any command examples, the single and double quotation marks should be simple, not the "smart quotes" forced my Microsoft. The same applies to dashes or minus signs – they should not be "em dashes" (- versus –).

Remember that the PROC IMPORT wizard will make your life easier but is far from perfect. By starting with the wizard, you can expand the code to handle your specific case.

REFERENCES

McQuown, Gary. "PROC IMPORT with a Twist", Coder's Corner Proceedings of the SUGI 30. Available at http://www2.sas.com/proceedings/sugi30/038-30.pdf

SAS, Inc, "Base SAS ® 9.2 Procedures Guide: Importing a Comma-Delimited File with CSV Extension", Available at http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000314361.htm

Shafranovich, Y. "Common Format and MIME Type for Comma-Separated Values (CSV) Files", Copyright (C) The Internet Society (2005). Available at http://tools.ietf.org/html/rfc4180

Wikipedia, "Comma-separated variables", available at http://en.wikipedia.org/wiki/Comma-separated_values (for good background)

ACKNOWLEDGMENTS

I want to thank the organizers of this great conference, my employer for their willingness to allow me to expand my horizons through events like this, and, last but certainly not least, my spouse Mary who doesn't complain when I spend so much time at the keyboard working on documents like this.

CONTACT INFORMATION

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

David B. Horvath, CCP
+1-610-859-8826
dhorvath@cobs.com
http://www.cobs.com