Technical Paper

Reading Delimited Text Files into SAS®9

THE POWER TO KNOW.

Release Information

Content Version: 1.1July 2015

(This paper replaces TS-673 released in 2009.)

Contents

Introduction

A *delimited file* is a plain text file that contains a separator between the data fields. Reading delimited text files in SAS® 6 was difficult and problematic. For example, to read Comma Separated Value (CSV) files in SAS 6, you had to preprocess the files in order for SAS to recognize that there were missing values between consecutive commas. SAS also treated delimiters that were inside quoted text strings the same as delimiters between strings. In addition, problems occurred when reading PC files on a UNIX machine because UNIX machines and PCs use different end-of-record markers.

Reading delimited text files into SAS®9 is a much easier process because the software has both new and improved options and informats that facilitate the process of reading such text files. This paper is intended for experienced SAS users. It briefly describes the new options and provides examples that illustrate the use of SAS 9.2 informats to read files. The paper also provides information about reading delimited text files with the IMPORT procedure, and it offers a troubleshooting section to help you with common problems.

Options Available for Reading Delimited Text Files

SAS 9.2 has several options available that make it easy to read delimited text files. The following INFILE statement contains several of these options. It is a typical example that reads a tab-delimited UNIX file with a header record and record lengths up to 4 K from a Windows environment:

```
infile 'C:\sasfiles\testfile.csv' dlm='09'x dsd lrecl=4096 truncover
                                  firstobs=2 termstr=LF;
```

All of the available options are described in the next two sections. For syntax and detailed information about these options, see *SAS® 9.2 Language Reference: Dictionary.*

LRECL=System Option

New in SAS 9.2, the LRECL= system option enables you to make the default logical record length larger for all external file operations. A value of **4096** (4 K) can handle most of the data that is used by SAS customers. As a best practice, you should not set this option to very high values (in the 1 MB range or higher) because it can slow system performance significantly. If you need a longer length, **32760** (32 K) is a good upper-end value to use because it creates buffers large enough to handle data without affecting performance.

INFILE Statement Options

- **DELIMITER= option**—Specifies what character (other than the blank default character) to use as the delimiter in files that are being read. Common delimiters include comma (,), vertical pipe (|), semi-colon (;) , and the tab. For example, to specify a vertical pipe as the delimiter, the syntax is DLM='|', as shown here:

  ```
  infile 'C:\mydata\test.dat' dsd dlm='|' lrecl=1024;
  ```

  A tab is specified by its hexadecimal value. For ASCII systems (UNIX, Windows, and Linux), the value is **'09'x**. For EBCDIC systems (z/OS and MVS), the value is **'05'x**. As an example, the syntax to specify a tab delimiter on an ASCII system is DLM='09'x. **Note:** The positioning of the quotation marks and the **x** in hexadecimal values is critical. No space is allowed between the **x** and the quotation marks, as shown in this example:

  ```
  infile 'C:\mydata\test.txt' dsd dlm='09'x truncover;
  ```

- **DLMSTR= option**—Specifies either a character string or the value of a character variable as the delimiter. For example, suppose you want SAS to split records based on finding a character string that contains a tilde, a backslash, and a caret. To do that, use the DLMSTR= option, as follows:

  ```
  infile '/root/user-id/data' dsd dlmstr='~\^' truncover;
  ```

- **DSD (delimiter-sensitive data) option**—Specifies that SAS should treat delimiters within a data value as character data when the delimiters and the data value are enclosed in quotation marks. As a result, SAS does not split the string into multiple variables and the quotation marks are removed before the variable is stored. When the DSD option is specified and SAS encounters consecutive delimiters, the software treats those delimiters as missing values. You can change the default delimiter for the DSD option with the DELIMTER= option.

- **FIRSTOBS= option**—Indicates that SAS should start reading the input file at the record number specified rather than the first record. This option is helpful when reading files that contain a header record, as shown in the following example. You can skip the header by specifying FIRSTOBS=2:

  ```
  infile 'C:\mydata\test.dat' dsd dlm='~' firstobs=2;
  ```

- **LRECL= option**—Specifies the logical record length in bytes. This option is used when the records in a file are longer than 256 bytes (on ASCII platforms). The default input buffer is 256 bytes. Records that exceed this length are truncated when they are read. Setting the LRECL= option to a greater length ensures that the input buffer is long enough for the entire record to be read. This is usually not an issue on EBCDIC platforms because the data control block specifies the logical record length for SAS. As shown in the following example, the LRECL= statement option in an INFILE statement overrides the LRECL= system option, if it is used.

  ```
  infile 'C:\mydata\test.dat' dsd truncover lrecl=4096;
  ```

- **TERMSTR= option**—Specifies what end-of-line character to use for a file. This option is specific to ASCII operating systems and is documented in the SAS 9.2 companion for your operating system. This option is useful when you want to share data files that are created on one operating system with another operating system. For example, if you are working in a UNIX environment and you need to read a file that was created under Windows, use TERMSTR=CRLF. Similarly, if you are in a Windows environment and you need to read a file that was created under UNIX, use TERMSTR=LF.

  The following INFILE statement illustrates reading a MAC file from a PC:

  ```
  infile 'C:\mydata\test.dat' dsd dlm='|' termstr=cr;
  ```

- **TRUNCOVER option**—Specifies that SAS should use the available input data in the current record only to populate as many variables as possible. By default, if the INPUT statement reads past the end of a record without finding enough data to populate the variables listed, it continues to read data from the next record. This action is called *flowover*. When you use the TRUNCOVER option, as shown in the following example, SAS does not proceed to the next record for more data to populate the variables:

  ```
  infile 'C:\mydata\test.dat' dsd dlm='|' truncover;
  ```

Using Informats with Data that Contains Delimiters

If your data is longer than the default length, you need to use informats. For example, date or time values, names, and addresses can be longer than eight characters. In such cases, you either need to add an INFORMAT statement to the DATA step or add informats directly in the INPUT statement. However, when the informats are used in the INPUT statement, care must be taken to honor the function of the delimiter to prevent read errors. If you add informats in an INPUT statement, you must add a colon (:) in front of the informat, as shown in this example:

```
data a;
   infile 'C:\sas\example2.csv' dlm='09'X dsd truncover;
   input fname :$20. lname :$30. address1 :$50. address2 :$50. city :$40.
   state :$2. zip phone :$12.;
run;
```

The colon indicates that SAS should read from the first character after the current delimiter to the number of characters specified in the informat or to the next delimiter, whichever comes first. The colon also positions the input pointer for the next variable, if there is one.

When a file contains numeric variables or character variables with a length of eight characters or less, as shown in the following example, you do not need informats to read in the data:

```
data a;
   infile 'C:\sas\example1.csv' dlm='|' dsd truncover;
   input fname $ lname $ age;
run;
```

In this example, the variables FNAME and LNAME are character variables and AGE is a numeric variable. All three variables have a length of 8 bytes, the default length, which is evident because no informats are specified.

Reading Delimited Text Files with the IMPORT Procedure

You can also use PROC IMPORT to read files that are in the proper format for the operating system on which SAS is running or to generate a significant portion of the DATA step that you need if the file is not properly structured.

When you use PROC IMPORT to read a CSV, tab, or other character-delimited file, the procedure does the following:

- scans the first 20 records
- collects the variable names from the first row
- scans the remaining 19 rows and determines the variable types
- assigns an informat and a format to each variable
- creates an INPUT statement
- submits all of the code to the DATA step compiler, which, in turn, executes the code

The following is a simple IMPORT procedure that reads files that have a tilde (~) as the character delimiter:

```
proc import datafile=’c:\mydata\test.fil’ dbms=dlm out=work.test replace;
   delimiter=”~”;
   getnames=yes;
   guessingrows=500;
run;
```

During this procedure, you should wait for the importing process to finish. Then press F4 to recall the generated DATA step. At this point, you can add or remove options from the INFILE statement and tailor the INFORMAT, FORMAT, and INPUT statements to your data.

If you use this method and you modify an informat, you should also modify the format for that same variable. The informat and format for a given variable also must be of the same type (either character or numeric). In addition, if the type is character, the assigned format should be as long as the variable to avoid truncation when the data is displayed. For example, if a character variable is 400 bytes long but has a format of $char50, only the first 50 characters are shown when the data is displayed.

By default, PROC IMPORT expects the variable names to appear in the first record (or row). PROC IMPORT scans the first 20 rows to count the variables and it attempts to determine the proper informat and format for each variable. You can use the following options to do the following: modify which variable names SAS extracts; indicate at which row SAS should begin reading; and indicate how many rows SAS should scan for variables:

- **GETNAMES= YES | NO**—Setting GETNAME=YES extracts the variable names from the first record of the file. SAS scans the first record of the file at this time. Setting GETNAME=NO indicates that SAS should create simple names for each variable found. The variables will include the name VAR and sequential numbers; for example, VAR1-VAR*n*, where *n* is the number of columns found in the file.
- **DATAROW=*row-number***—Specifies the row number where SAS should begin reading the data to determine type and form.
- **GUESSINGROWS=*number-of-rows-to-scan***—Specifies how many records SAS should scan for the type and length of the variables. The default value is **20**, although the range can be any number from **1** to **32767**.

Troubleshooting Guide

This section provides guidance if you have problems correctly reading a file. Whenever you have trouble correctly reading a file, check the SAS log because it often provides clues to solving the problem.

Problem 1

You see the following notes together in the SAS log:

```
NOTE: The maximum record length was 256.
One or more lines were truncated.
```

These notes do not indicate an error. However, a problem does exist. The notes indicate that you are using the default LRECL= value of **256**, but some of your data exceeds 256 characters. As a result, any characters in column 257 or greater will be lost.

Solution

Increase the value of the LRECL= option to at least the length of the longest record in the file that is being read.

Problem 2

The SAS log contains the following message and all the data in the data set appears to be read:

```
One or more lines were truncated.
```

Based on the information in Problem 1, this seems to indicate a problem. However, it most likely is not a problem. This truncation message occurs when the following conditions are true:

- Maximum record length is less than the active logical record length.
- All of the data is in the data set.
- You are using a value for the FIRSTOBS= option that is higher than **1**.

This message indicates that the FIRSTOBS= option has skipped a line because the line is longer than the LRECL= value. SAS recognizes that a line is longer than the LRECL= value, but it does not recognize that the line has been skipped by the FIRSTOBS= option.

Problem 3

Your last variable is numeric, and it is missing in every observation.

The most frequent cause for a missing variable is reading a Windows file on a UNIX platform. Windows uses a carriage return (<CR>) and a line feed (<LF>) to signal the end of a record. UNIX platforms only use a line feed. The UNIX system reads the carriage return as data. Because it is not numeric data, an invalid data message is generated, as shown in the following output:

Sample Log:

```
NOTE: Invalid data for d in line 1 7-8.
RULE:     ----+----1----+----2----+----3
1   CHAR  1,2,3,4. 8
    ZONE  32323230
    NUMR  1C2C3C4D
a=1 b=2 c=3 d=. _ERROR_=1 _N_=1
```

The message, the ruler, and the **CHAR/ZONE/NUMR** group show a period at the end of your number. Below the period in the **ZONE** line is a **0**; below that in the **NUMR** line is a **D**. These are elements of '0D'x (the hexadecimal representation of a carriage return), but the system is reading the components of the hexadecimal value as actual data.

Solution

If you have SAS®9, use TERMSTR=CRLF in your INFILE statement.

For releases before SAS®9, either convert the file to the proper UNIX structure or add the carriage return (<CR>) to the list of delimiters. If you are reading a CSV file, specify the carriage return as DL DLM='2C0D'x. If you are reading a tab-delimited file, use DLM='090D'x. If you have a different delimiter, contact SAS Technical Support for help.

Problem 4

When you read a CSV file, the records are split inside of a quoted string. For example, sometimes a long comment field abruptly stops and continues on the next row of the file that you are reading. You can see it if you open the file in Windows Notepad on the PC or in a plain-text editor under UNIX.

This problem happens most commonly in files that are created from Microsoft Excel worksheets. In Excel, the column contains soft returns to help in formatting. When you save the worksheet as a CSV file, the soft returns incorrectly appear to SAS as end-of-record markers. This, in turn, causes the records to be split incorrectly.

Solution

In SAS®9 under Windows, you can solve the problem by using the option TERMSTR=CRLF. This setting indicates that SAS should accept only the complete return and line feed as an end-of-record marker.

For releases before SAS®9, run the following program. It converts any line-feed character between double quotation marks into a space to enable SAS to read the file correctly. In this program, the INFILE and FILE statement do refer to the exact same file. The SHAREBUFFERS option enables the external file to be updated in place, removing the offending characters.

Note: If you cannot easily re-create or restore your CVS file, be sure to make a copy of the file before updating it.

```
data _null_;
   infile 'C:\_today\mike.csv' recfm=n sharebuffers;
   file 'C:\_today\mike.csv' recfm=n;
   input a $char1.;
   retain open 0;

    /* This statement toggles the open flag. */
   if a='"' then open=not open;

   if a='0A'x and open then put ' ';
run;
```

The program reads the file one byte at a time and replaces the line-feed character, as needed.

THE POWER TO KNOW.

To contact your local SAS office, please visit: sas.com/offices