Paper 2605-2018

A SAS® Macro That Uses PRX Functions to Verify Delimited Text File Formatting

Paul Genovesi State of Washington/DSHS/RDA

ABSTRACT

The task of importing delimited text files can often seem like a black box process. If the resulting data set contains errors, how do you know? Some errors are easy to spot, such as unspecified extra fields showing up in the data set. Other errors, like data ending up in the wrong field, can be more difficult to identify, especially when they occur well into the resulting data set. Fortunately, every correctly formatted, delimited text file complies with one of two formats[1], each containing its own text pattern. If every cell of the delimited text file matches the text pattern, then the file is said to be in that format. The text-qualified format is the more complex of the two and the one that this paper concentrates on. The non-text-qualified format is very simple and receives only minor mentioning. This paper introduces a SAS® macro that uses SAS PRX functions to verify that a delimited text file is in text-qualified format. If the macro reports that it is, then the file can be imported without error.

INTRODUCTION

SAS Enterprise Guide's Import Data task allows you to import delimited text files either with or without a text qualifier. If you specify a text qualifier, then your file MUST be in text-qualified format in order for your file to be imported without error. If you specify a text-qualifier of <none>, then your file MUST be in non-text-qualified format. But how do you know if your file adheres to one of these formats? You can find out by running the delimited_text_file_verifier macro on your file and choosing the same delimiter and text-qualifier that you choose when importing the file using SAS Enterprise Guide's Import Data task.

TEXT-QUALIFYING RULES

WHEN DOES DATA BECOME TEXT-QUALIFIED?

Text-qualifying is added to SAS dataset data when the dataset is exported to a delimited text file. Within SAS Enterprise Guide, a SAS dataset can be exported to a delimited text file. While viewing your data in the form of a SAS dataset, you will not see any text qualifying or escaping of text-qualifier characters. It is only when you export the SAS dataset to a delimited text file that any text-qualifying is added. This concept is elementary to the vast majority of SAS programmers, but it needs mentioning.

TEXT-QUALIFYING OCCURS BY CELL AND NOT FIELD WITHIN A DELIMITED TEXT FILE

Within a delimited text file, a cell exists at the intersection of a row and a column with the column number determined by counting the active (as opposed to embedded delimiter characters) delimiters in the row. Text qualifying is the enclosing (i.e. surrounding) of the cell's contents with quoting characters such as double quotes or single quotes. For our purposes, we will consider quoting characters to be part of a cell's contents. Contrary to popular belief, text qualifying occurs at the cell level and not the field (i.e. column) level of a delimited text file. This means that a field within the file can contain a mixture of both text-qualified and non-text-qualified cells. When a delimited text file is said to be text-qualified, it means that the file contains at least one text-qualified cell and in order for this file to be correctly imported, it must be imported using this text-qualifier.

RULES FOR NON-TEXT-QUALIFIED FORMAT

For your delimited text file to be in this format, it must not contain any embedded delimiter characters within cell data. In other words, every delimiter character is an active delimiter. Verifying that your file adheres to this format is easy. The power of SAS's prx functions is not even needed. The pattern

[1] Not to be confused with SAS Formats.

associated with this format can best be described as an improvised version of the fence post rule in which fence post sections and not fence posts occur at either end. The delimiters are the fence posts and character strings of zero or more characters are the fence sections. Pattern matching is not needed here since the file contains no text-qualifying and all delimiter characters contained in the file are active delimiters. All that is needed is a count of the number of delimiters existing within each record. If each record contains the correct amount of delimiters, then this file is in non-text-qualified format.

RULES FOR TEXT-QUALIFIED FORMAT

This format is much more complex and much more difficult to verify than non-text-qualified format. This is because the delimited text file can now contain embedded delimiter characters within its cells. To understand the application of the rules for a file in text-qualified format, imagine your data is contained in a SAS dataset that you are about to export to a delimited text file. As previously mentioned, while your data exists in the SAS dataset, it contains no text-qualifying. Text-qualifying, if needed, is added to your data when it is exported to the delimited text file. Here are the rules for how it is added:

1. Any cell containing one or more embedded delimiter characters must be text-qualified using either double quotes or single quotes. The chosen text-qualifier is used for all text-qualifying throughout the file.
2. Any remaining non-text-qualified cells, which contain embedded text-qualifier characters, must also be text qualified and the embedded text-qualifier characters must be escaped with (i.e. preceded by) another text-qualifier character.
3. A cell that contains neither an embedded delimiter character nor an embedded text-qualifier character does not need to be text-qualified but could be text-qualified anyway.

A delimited text file adhering to the above formatting rules contains a pattern that can be verified using pattern matching. Each cell of the file up to and including the active delimiter must match the pattern. As we will see, due to the complexity of this pattern, it cannot be verified by one pattern match using one regular expression. For each cell of the file, verification of the pattern will require a minimum of one and a maximum of four separate pattern matches.

EXAMPLES OF CORRECT CELL CONTENTS FOR A FILE IN TEXT-QUALIFIED FORMAT

In order for a delimited text file to be in “Text-Qualified” format, each cell of the file must adhere to the above 3 rules for this format. Let’s take a look at a few examples of cells adhering to these rules (see Figure 1). For these examples and for examples throughout this paper, the comma is our delimiter and the double quote is our text qualifier. As you can see, each cell is surrounded by 2 commas as it would be in a csv file unless the cell is the first or last cell of the record.

```
1. ,,            (an empty cell)
2. ,"",          (an empty cell that is text-qualified by choice)
3. ,Olympia,     (a non-text-qualified cell)
4. ,Olympia WA,  (a non-text-qualified cell)
5. ,",",         (a text-qualified cell that contains an embedded delimiter
   character)
6. ,"""",        (a text-qualified cell containing one escaped text-
   qualifier character)
7. ,"""""",      (a text-qualified cell containing two escaped text-
   qualifier characters)
8. ,"Olympia",   (a text-qualified cell that is text-qualified by choice)
9. ,"Olympia, WA",    (a text-qualified cell containing an embedded
   delimiter character)
10.      ,""""Olympia"", ""WA"""",     (a text-qualified cell containing an
   embedded delimiter character and 4 escaped text-qualifier characters)
```

Figure 1 - Examples of Correct Cell Contents

THE MACRO'S PATTERN MATCHING PROCESS

PATTERN MATCHING PROGRESSION THROUGH THE FILE

With the text-qualified format there exists a pattern that can be matched going cell by cell across each record of the delimited text file. If we can verify that every delimited cell of our file adheres to the pattern before we import it, then the import will succeed.

The macro first imports the delimited text file into a SAS dataset containing a single character field which contains the entire text file. Pattern matching starts with the first delimited cell of a record and continues matching left-to-right until either all cell matches are successful or one of them fails. If a cell match fails, then we take the remaining unmatched portion of our record and continue matching right-to-left from the opposite end of the record. We do this for each record of the file.

MACRO CODE CONTAINING THE PARSED REGULAR EXPRESSIONS USED TO DETERMINE PATTERN ADHERENCE

As previously stated, the pattern that we are verifying for each cell of the file is much too complex for pattern adherence to be determined using just one pattern match involving one regular expression. We will need to employ a divide-and-conquer approach in which we use at least one and at most four separate pattern matches to arrive at our answer for each cell of the file. The following is the code containing our four separate pattern matches. This is code in which the double quote is being used as the text-qualifier. Code in which the single quote is used as the text qualifier is not shown here but exactly mirrors code using the double quote. Also notice that our delimiter is contained in the macro variable dlm_rgx.

CODE FOR PATTERN MATCH #1

Below is the code for our 1st pattern match:

```
ltr_no_tqs_id   = prxparse('/^([^"]*?)'      ||
                          symget('dlm_rgx') ||
                          '([\d\D]*)$/');
```

`ltr_no_tqs_id stands for left-to-right_no-text-qualifiers_regex-id`

This match will succeed only if our cell contains no text-qualifying (i.e. no double quotes) up to the first delimiter. Notice that the regular expression employs a lazy match that starts at the beginning of the

record text string and extends up to the first delimiter. The match will only succeed if every character up to the delimiter is a non-text-qualifier (i.e. non-double-quote) character. If the match succeeds, then the 1st capture buffer will contain the cell contents and the 2nd capture buffer will contain the remainder of the record. If the match succeeds, then we are finished verifying this cell and we move on to the next cell. If not, then we move on to pattern match #2 for this same cell.

CODE FOR PATTERN MATCH #2

Below is the code for our 2nd pattern match:

```
ltr_tqs_wo_tq_chars_id =
   prxparse('/^'                          ||
            symget('whitespace_lazy')     ||
            '("(?:"")*?[^"]*?"(?:"")*?)' ||
            symget('whitespace_greedy')   ||
            symget('dlm_rgx')             ||
            '([\d\D]*)$/');
```

ltr_tqs_wo_tq_chars_id stands for left-to-right_contains-text-qualifiers_without_embedded_text-qualifier-chars_regex-id

Notice that the regular expression accounts for the existence of inter-field whitespace both before and after the cell's contents. This whitespace could be either (1) tabs and spaces (2) spaces or (3) none. The 3rd argument supplied to the macro (i.e. inter_field_whitespace) must contain one of these 3 choices when running the macro on a text-qualified file.

This match will succeed only if (1) our cell is text-qualified and (2) any embedded, escaped text-qualifier characters (i.e. "") occur immediately within and next to the 2 text-qualifiers and nowhere else in the cell. An example of this would be a cell that has the following contents:

```
"""Olympia, WA"""
```

Also, the embedded, escaped text-qualifier characters must occur in multiples of 2 otherwise, the match for this cell fails. So for example, cells containing the following would both fail to match:

```
 ""Olympia, WA"""  and  """Olympia, WA""
```

Since embedded, escaped text-qualifier characters occur in multiples of 2 characters, we use the following regular expression fragment of:

```
(?:"")*?
```

If pattern match #2 succeeds, then we are finished verifying this cell and we move on to the next cell. If not, then we move on to pattern match #3 for this same cell.

CODE FOR PATTERN MATCH #3

Below is the code for our 3rd and final pattern match:

```
ltr_tqs_w_tq_chars_id =
   prxparse('/^'                                    ||
            symget('whitespace_lazy')               ||
            '("(?:"")*?(?!")[\d\D]*?(?<!")"(?:"")*?)' ||
            symget('whitespace_greedy')             ||
            symget('dlm_rgx')                       ||
            '([\d\D]*)$/');
```

```
  ltr_tqs_w_tq_chars_id stands for left-to-right_text-
qualifiers_with_embedded_text-qualifier-chars_regex-id
```

Pattern match #3 will succeed only if our cell is text-qualified with possible embedded, escaped text-qualifier characters occurring immediately within and next to our text-qualifiers (just the same as in pattern match #2). If pattern match #3 succeeds, then we indicate this by incrementing the variable ltr_ok_fields. If it fails, then this cell has broken the pattern which means this record is not in text-qualified format. Also, left-to-right matching ends and right-to-left matching begins at the opposite end of the record.

Due to the failure of pattern match #2 and the success of pattern match #3, we know that the cell contains at least one embedded text-qualifier character. In order for the cell to be considered to be in text-qualifier format, we must check to see if the cell contains any embedded, unescaped text-qualifier characters. Pattern match #4 will indicate the existence of any of these.

CODE FOR PATTERN MATCH #4

To test for embedded, unescaped text-qualifier characters, we take the contents of capture buffer #1 obtained from the following segment of pattern match #3's regular expression:

```
'("(?:"")*?(?!")[\d\D]*?(?<!")"(?:"")*?)'
```

We then remove the text-qualifiers (i.e. the surrounding double quotes) from the capture buffer contents and test the remaining contents for unescaped text-qualifier characters using this pattern match:

```
check_for_unescaped_tq_chars = prxparse('/(?<!")""(?!")/');
```

For example, if our cell contains the following:

```
"""Olympia"",""WA"""
```

then removing the text-qualifiers will leave us with:

```
""Olympia"",""WA""
```

and this is what we match against the check_for_unescaped_tq_chars pattern.

Pattern match #4 is the one pattern match that we do not want to be successful. If it is successful, then the cell fails this test. A success here means that the contents contain an odd number of consecutive double quotes which means that we have an embedded, unescaped double quote since embedded double quotes must occur in pairs (i.e. multiples of 2) within a text-qualified cell.

Embedded, unescaped text-qualifier characters can cause importing errors especially when they are followed by an embedded delimiter character because this indicates the closing of the text-qualified cell. We need to know if any exist within our text-qualified cell.

So, to summarize, we could have a cell that passes both pattern match #3 and pattern match #4. This would indicate that the cell is properly text-qualified but it also contains an embedded, unescaped text-qualifier character. This cell would not be considered to be in text-qualified format.

IF LEFT-TO-RIGHT MATCHING FAILS, THEN MATCH RIGHT-TO-LEFT

If left-to-right matching fails at any cell, then we take the unmatched remaining record text string (which still contains the cell that did not match) and match right-to-left starting at the end of the text string. We match using the same 4-pattern-match scenario described above only using regular expressions that are slightly modified for matching right-to-left. If there are no additional incorrect cells contained in the record, then right-to-left matching will eventually fail at the same cell in which left-to-right matching failed.

WHAT IS INTER-FIELD WHITESPACE?

Inter-Field whitespace can exist within a delimited text file whether the text file contains text-qualifying or not. It is the whitespace (i.e. tabs and/or spaces) that can exist between a delimiter and a cell. Here are two examples:

Example #1

Delimiter = comma Inter-Field whitespace = tabs and/or spaces

,<Inter-Field whitespace>”Olympia, WA”<Inter-Field whitespace>,

Note: Inter-Field whitespace exists between the comma and the double quote in 2 locations.

Example #2

Delimiter = comma Inter-Field whitespace = tabs and/or spaces

,<Inter-Field whitespace>Olympia<Inter-Field whitespace>,

Note: Inter-Field whitespace exists between the comma and the cell’s contents (i.e. Olympia) in the 2 locations.

THE DELIMITED_TEXT_FILE_VERIFIER SAS MACRO

The delimited_text_file_verifier SAS macro takes 8 arguments:

```
%macro delimited_text_file_verifier(
    delimiter
   ,text_qualifier
   ,inter_field_whitespace
   ,number_of_fields
   ,first_obs
   ,last_obs
   ,file_lrecl
   ,file_path
  );
```

Required Arguments:

- **delimiter**

  Must be one of the following spelled-out names:

  - AMPERSAND &
  - ASTERISK *
  - AT SYMBOL @
  - BACKSLASH \
  - CARET ^
  - COLON :
  - COMMA ,
  - DOLLAR SIGN $
  - EXCLAMATION POINT !
  - FORWARD SLASH /
  - GRAVE `

- PERCENT SIGN %
- PIPE |
- PLUS SIGN +
- POUND SIGN #
- QUESTION MARK ?
- SEMICOLON ;
- TAB
- TILDE ~

Note: The space character is not supported as a delimiter.

- **text_qualifier**

  Must be one of the following:

  - DOUBLE QUOTE
  - SINGLE QUOTE
  - NONE

- **inter_field_whitespace**

  This is the whitespace that can exist between a delimiter and a cell. For non-text-qualified delimited text files, this must be set to NOT APPLICABLE.

  Must be one of the following:

  - TABS AND SPACES
  - SPACES ONLY
  - NONE
  - NOT APPLICABLE

- **number_of_fields**

  This is the number of fields that the delimited text file contains. Only supply a number here if you are certain of the number of fields. Otherwise, enter the word calc in which case the macro will calculate this value by determining the most common value for the number of error-free parsed fields.

  Must be one of the following:

  - CALC
  - a number

- **first_obs**

  The first record to process in the delimited text file.

  Must be one of the following:

  - a number

- **last_obs**

  This is the last record to process in the delimited text file. Enter the word max if you want to process up to the last record.

Must be one of the following:

- MAX
- a number

- **file_lrecl**

  This value should be greater than the longest record in the delimited text file but not greater than 32,765 (i.e. 32,767 - 2).

  Must be one of the following:

  - a number

- **file_path**

  This is the absolute path for the delimited text file.

A DELIMITED TEXT FILE USED FOR TESTING THE MACRO

A DESCRIPTION OF THE RECORDS CONTAINED IN THE TEXT FILE

As part of macro testing, we use a csv file containing 20 records in the form of test cases (see Figure 2). 10 of these records are in text-qualified format (i.e. the F4 field contains OK), and the other 10 are not (i.e. the F4 field contains ERR). Records that are not in text-qualified format contain cells that either break the pattern or contain embedded, unescaped text-qualifier (i.e. double quote) characters.

Notice that almost all of the inter-field whitespace in the below text file (see Figure 2) exists in the area located between a delimiter and a non-text-qualified field. This inter-field whitespace is used for aligning the fields and making the text file more readable. However, for test cases TC05, TC06, and TC07, there also exists inter-field whitespace located between a delimiter and a text-qualified field. As we will see later, SAS Enterprise Guide's Import Data task has no problem importing a text-qualified file containing inter-field whitespace so long as the option "Generalize import step to run outside SAS Enterprise Guide" is not selected. See Figure 3 for translations of Field 4's Test Case Descriptions.

```
F1_Data,              F2_TC   ,F3_Data,              F4_Test_Case_Description      ,F5_Data
,                     TC01    ,,                     OK--->no_text_qualifying      ,
"",                   TC02    ,"",                   OK--->text_qualifying         ,""
"""",                 TC03    ,"""",                 OK--->text_qualifying         ,""""
",","",",             TC04    ,",","",",             OK--->text_qualifying         ,",","",","
 "F1",                TC05    , "F3" ,               OK--->inter_fld_spaces_F135   , "F5"
    "F1"    ,         TC06    ,"F3",                 OK--->inter_fld_tabs_F15      ,  "F5"
     "F1"     ,       TC07    ,"F3",                 OK--->inter_fld_tbs_spcs_F15  ,   "F5"
""" """" """,         TC08    ,""" """" """,         OK--->escTQc_F135             ,""" """" """
"" """" """,          TC09    ,"" """" """,          ERR-->unescTQc_beg_F135       ,"" """" """
""" """ """,          TC10    ,""" """ """,         ERR-->unescTQc_mid_F135       ,""" """ """
""" """" "",          TC11    ,""" """" "",          ERR-->unescTQc_end_F135       ,""" """" ""
""" """" , """,       TC12    ,""" """" , """,       OK--->escTQc_DLMc_F135        ,""" """" , """
"" """" , """,        TC13    ,"" """" , """,        ERR-->unescTQc_beg_DLMc_F135  ,"" """" , """
""" """ , """,        TC14    ,""" """ , """,        ERR-->unescTQc_mid_DLMc_F135  ,""" """ , """
""" """" , "",        TC15    ,""" """" , "",        ERR-->unescTQc_end_DLMc_F135  ,""" """" , ""
""" , """" """,       TC16    ,""" , """" """,       OK--->DLMc_escTQc_F135        ,""" , """" """
"" , """" """,        TC17    ,"" , """" """,        ERR-->unescTQc_beg_DLMc_F135  ,"" , """" """
""" , """ """,        TC18    ,""" , """ """,        ERR-->unescTQc_mid_DLMc_F135  ,""" , """ """
""" , """" "",        TC19    ,""" , """" "",        ERR-->unescTQc_end_DLMc_F135  ,""" , """" ""
""",                  TC20    ,""",                  ERR-->unescTQc_F135           ,"""
```

Figure 2 - CSV File Containing Test Case Records

F2_TC	F4_Test_Case_Description	Description Translation
TC05	OK--->inter_fld_spaces_F135	Inter-field spaces in fields 1, 3, and 5
TC06	OK--->inter_fld_tabs_F15	Inter-field tabs in fields 1 and 5
TC07	OK--->inter_fld_tbs_spcs_F15	Inter-field tabs and spaces in fields 1, and 5
TC08	OK--->escTQc_F135	Escaped text-qualifier character in fields 1, 3, and 5
TC09	ERR-->unescTQc_beg_F135	Unescaped text-qualifier character in beginning of fields 1, 3, and 5
TC10	ERR-->unescTQc_mid_F135	Unescaped text-qualifier character in middle of fields 1, 3, and 5
TC11	ERR-->unescTQc_end_F135	Unescaped text-qualifier character in end of fields 1, 3, and 5
TC12	OK--->escTQc_DLMc_F135	Escaped text-qualifier character and delimiter character in fields 1, 3, and 5
TC13	ERR-->unescTQc_beg_DLMc_F135	Unescaped text-qualifier character at beginning and delimiter character in fields 1, 3, and 5
TC14	ERR-->unescTQc_mid_DLMc_F135	Unescaped text-qualifier character at middle and delimiter character in fields 1, 3, and 5
TC15	ERR-->unescTQc_end_DLMc_F135	Unescaped text-qualifier character at end and delimiter character in fields 1, 3, and 5
TC16	OK--->DLMc_escTQc_F135	Delimiter character and escaped text-qualifier character in fields 1, 3, and 5
TC17	ERR-->unescTQc_beg_DLMc_F135	Unescaped text-qualifier character at beginning and delimiter character in fields 1, 3, and 5
TC18	ERR-->unescTQc_mid_DLMc_F135	Unescaped text-qualifier character at middle and delimiter character in fields 1, 3, and 5
TC19	ERR-->unescTQc_end_DLMc_F135	Unescaped text-qualifier character at end and delimiter character in fields 1, 3, and 5
TC20	ERR-->unescTQc_F135	Unescaped text qualifier character in fields 1, 3, and 5

Figure 3 - Test Case Description Translations

We then execute the macro on the above text file by calling the macro with the following arguments:

%***delimited_text_file_verifier***(comma,

double quote,

tabs and spaces,

calc,

2,

max,

7000,

H:\SAS_Global\comma_tabs_spaces_dq.csv)

The macro returns the following 4 datasets:

1. all_records – This dataset contains verification information for all of the test case records in our original delimited text file.
2. good_records – This dataset contains verification information for only the test case records in our original delimited text file that are in text-qualified format.
3. bad_records - This dataset contains verification information for only the test case records in our original delimited text file that are **NOT** in text-qualified format.
4. record_stats – This dataset contains the following:
    1) A count of the good records.
    2) A count of the bad records.
    3) A count of the records in which at least one text-qualified cell contains an embedded, unescaped text-qualifier character.

THREE MACRO OUTPUT VERIFICATION RULES FOR TEXT-QUALIFIED RECORDS

For a record to be included in the good_records dataset and thus be considered to be in text-qualified format, it must adhere to the following 3 rules in terms of verification information field values:

1. The value contained in ltr_ok_fields must be equal to the value contained in the macro argument number_of_fields whether the user supplied a number for this argument or the user entered CALC and the macro computed the number. This indicates that the record contains the correct number of fields.
2. The field ltr_remainder must contain a missing value. This indicates that all of the record's cells matched one of the 3 patterns and there is no remainder (i.e. leftover) text.
3. The field ltr_unesc_tq_char_fld_list must contain a missing value. This indicates that none of the record's cells contained any embedded, unescaped text-qualifier characters.

The good_records dataset (see Figure 4) tells us that 10 of our original test case records are in text-qualified format according to the "Three Macro Output Verification Rules for Text-Qualified Records" which are listed above.

	f1	total_ok_fields	ltr_ok_fields	ltr_last_ok_fld_contents	ltr_remainder	ltr_unesc_tq_char_fld_list
1	, TC01 ,, OK...	5	5			
2	"", TC02 ,"", O...	5	5	""		
3	"""", TC03 ,"""", O...	5	5	""""		
4	","","", TC04 ,","","", O...	5	5	","",""		
5	"F1", TC05 ,"F3",...	5	5	"F5"		
6	"F1" , TC06 ,"...	5	5	"F5"		
7	"F1" , TC07 ,"...	5	5	"F5"		
8	""" """ """, TC08 ,""" """ """,...	5	5	""" """ """		
9	""" """, """, TC12 ,""" """, """,...	5	5	""" """, """		
10	""", """ """, TC16 ,""", """ """,...	5	5	""", """ """		

	rtl_ok_fields	rtl_remainder	rtl_last_ok_fld_contents	rtl_unesc_tq_char_fld_list
1	.			
2	.			
3	.			
4	.			
5	.			
6	.			
7	.			
8	.			
9	.			
10	.			

Figure 4 - The good_records dataset (wrapped for display)

The below bad_records dataset (see Figure 5) tells us that 10 of our original test case records are NOT in text-qualified format according to the "Three Macro Output Verification Rules for Text-Qualified Records" which are listed above.

	f1	total_ok_fields	ltr_ok_fields	ltr_last_ok_fld_contents	ltr_remainder	ltr_unesc_tq_char_fld_list
1	""" """ """, TC09 ,""" """...	0	0		""" """ """, TC...	
2	""" """ """, TC10 ,""" """...	5	5	""" """ """		1,3,5
3	""" """ """, TC11 ,""" """...	0	0		""" """ """, TC...	
4	""" """, """, TC13 ,""" """...	1	0		""" """, """, TC...	
5	""" """, """, TC14 ,""" """...	3	3	""", ERR-->unescTQc_mi...	"""	2,3
6	""" """ """, TC15 ,""" """...	1	0		""" """ """, TC...	
7	""", """ """, TC17 ,""", """...	1	1	""	""" """, TC17...	
8	""", """ """, TC18 ,""" ,...	5	5	""", """ """		1,3,5
9	""", """ """, TC19 ,""" ,...	1	1	""", """ """, TC19 ,"""	""" """, ERR-->u...	
10	""", TC20 ,""",...	2	2	ERR-->unescTQc_F135	"""	

	rtl_ok_fields	rtl_remainder	rtl_last_ok_fld_contents	rtl_unesc_tq_char_fld_list
1	0	,"" """" """", TC...		
2	.			
3	0	,"" """" """", TC...		
4	1	,"" """", """", TC...	"""", ERR-->unescTQc_be...	
5	0	,""		
6	1	,"" """", """", TC...	""	
7	0	,"""" """", TC17...		
8	.			
9	0	,"""" "", ERR-->...		
10	0	,""		

Figure 5 - The bad_records dataset (wrapped for display)

Notice that each record description in the bad_records dataset breaks at least 1 of the above 3 verification rules. A text file record listed in this dataset will cause some degree of error when the text file record is imported. The error could involve something such as data in the wrong field or data in the wrong record or maybe something minor like an embedded, unescaped text-qualifier character that is simply ignored as most of them are.

Unfortunately, the original csv file containing all 20 test cases cannot be imported because the records containing errors have a tendency to sometimes pull in data from the record that follows which ends up polluting both test cases. So records not in text-qualified format need to be imported in a delimited text file in which they are the only data record. The next section contains delimited text files containing one header record and one data record per file.

COMPARING THE TEXT FILE'S MACRO VERIFICATION OUTPUT TO ITS IMPORT-RESULTING DATASET

Running the macro on a delimited text file will tell you whether or not the file is in text-qualified format and, if not, then the records and their cells that are keeping the file from being in this format. If the macro verification output indicates that your file is in this format, then you will receive no errors when importing this file using a text-qualifier as part of SAS Enterprise Guide's Import Data task.

USING INDIVIDUAL TEST CASES FOR THE COMPARISON

We will now take a look at some test case records chosen from the above text file containing the 20 test case records. Some of our chosen records are in text-qualified format and some contain errors and are not in text-qualified format.

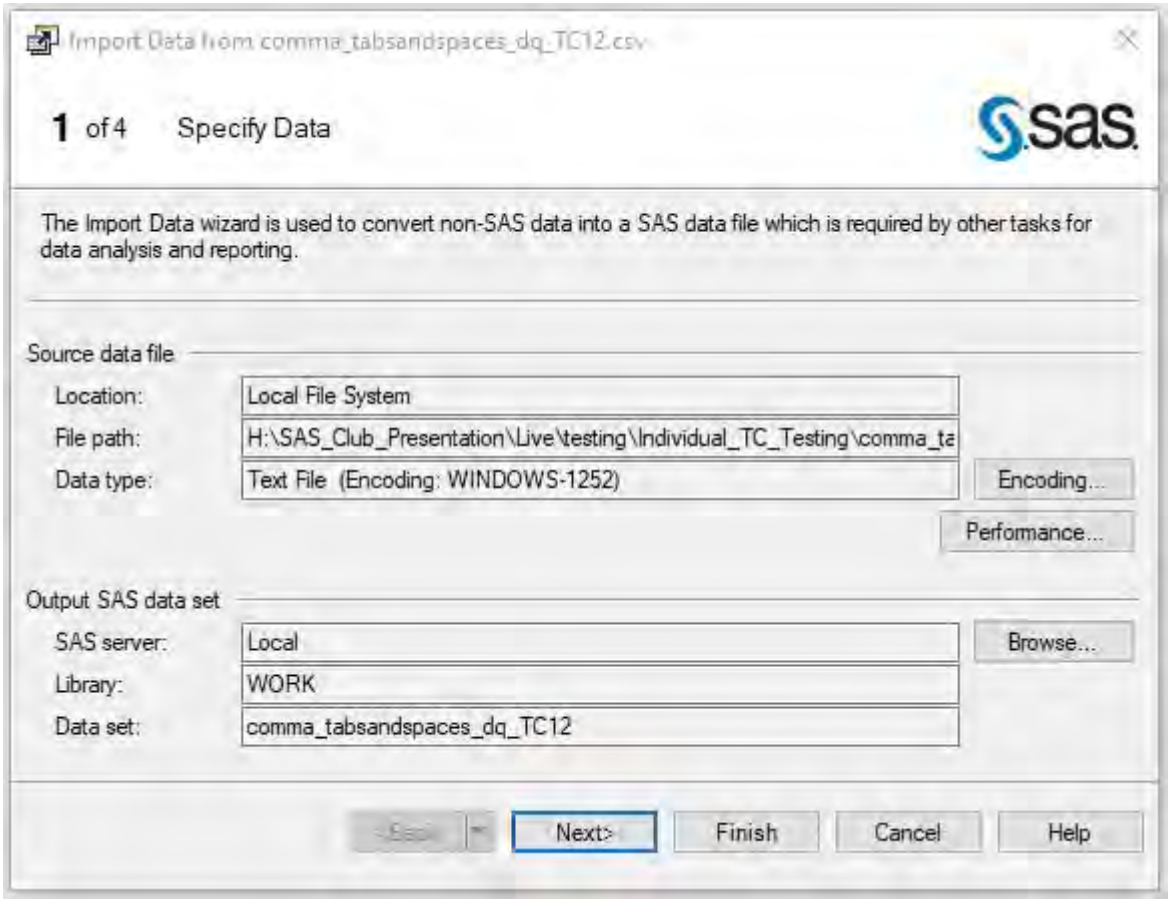
Note: As previously mentioned, the records containing errors (and hence not in text-qualified format) must be individually imported from text files in which they are the only data record in addition to the header record. This is necessary because the error contained in one record will often cause the import of this record to pull in data from the ensuing record which ends up polluting both test cases.

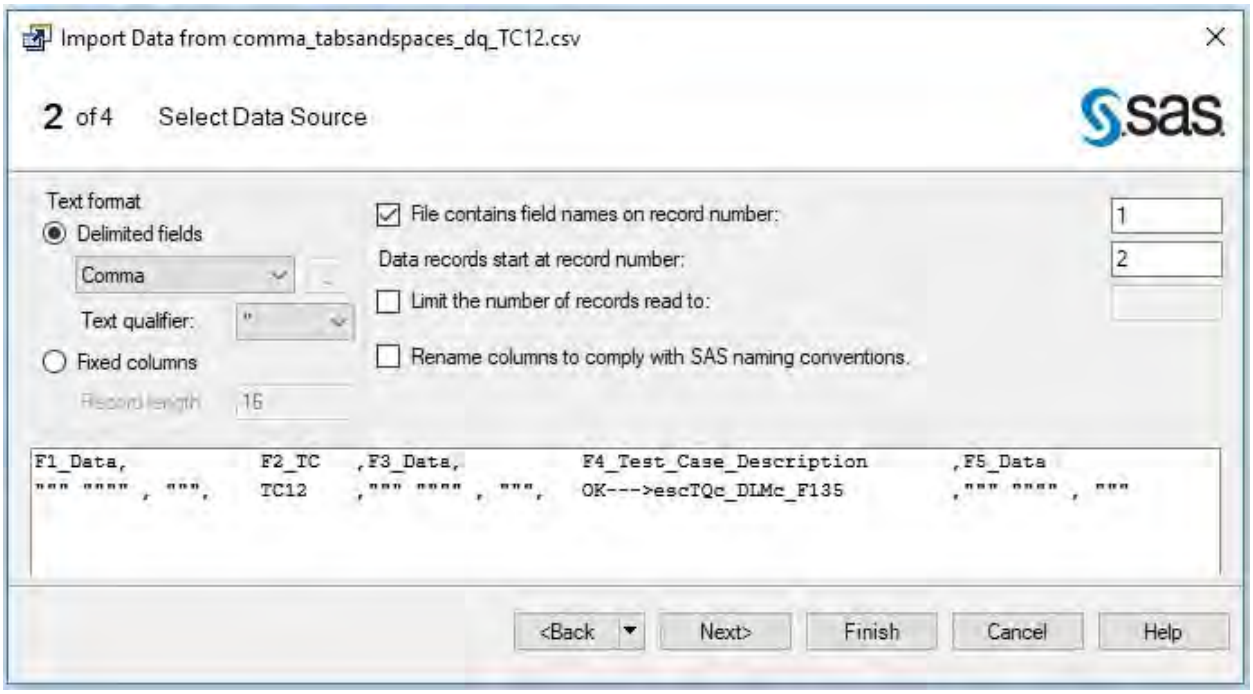
For each test case, we have the following:

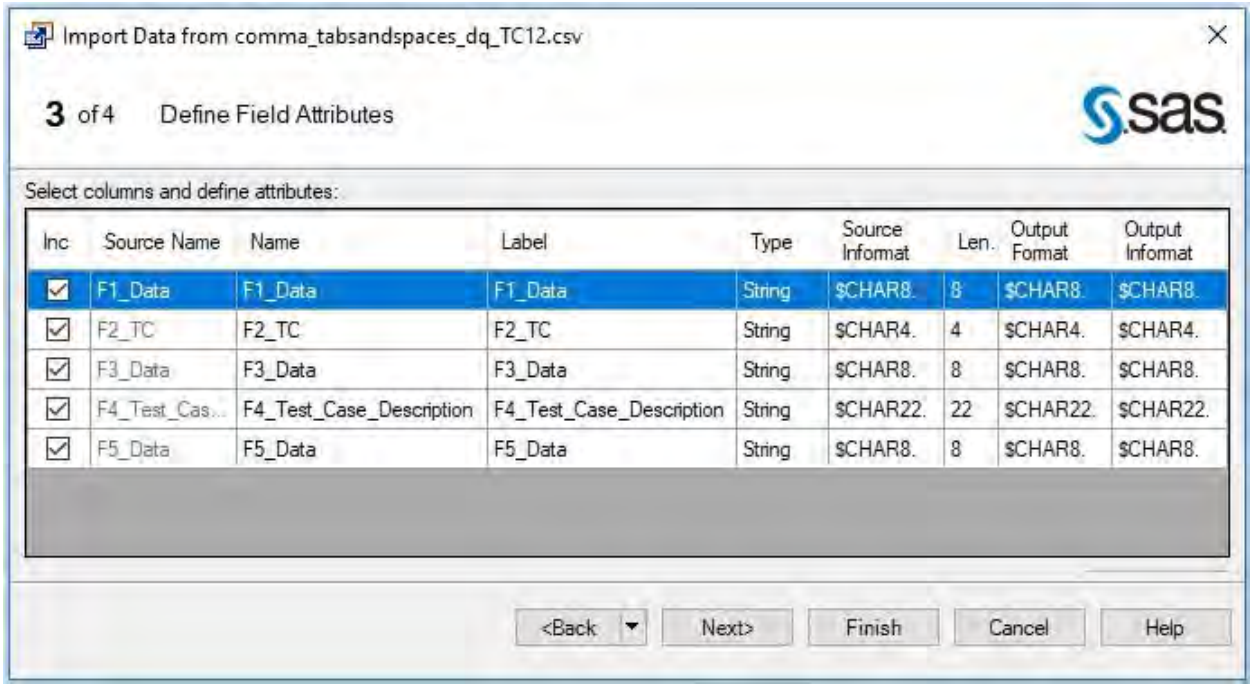
1. The delimited text file containing:
    - the header record
    - the data record
2. The macro verification output (wrapped to fit into this document)
3. The SAS dataset created by importing the delimited text file using SAS Enterprise Guide's Import Data task
    - Within the Import Data task selection windows, the selection "Generalize import step to run outside SAS Enterprise Guide" (contained in the task's Advanced Options window) is left unchecked. This insures that any inter-field whitespace in the form of spaces and/or tabs is stripped off and not made part of the imported data.

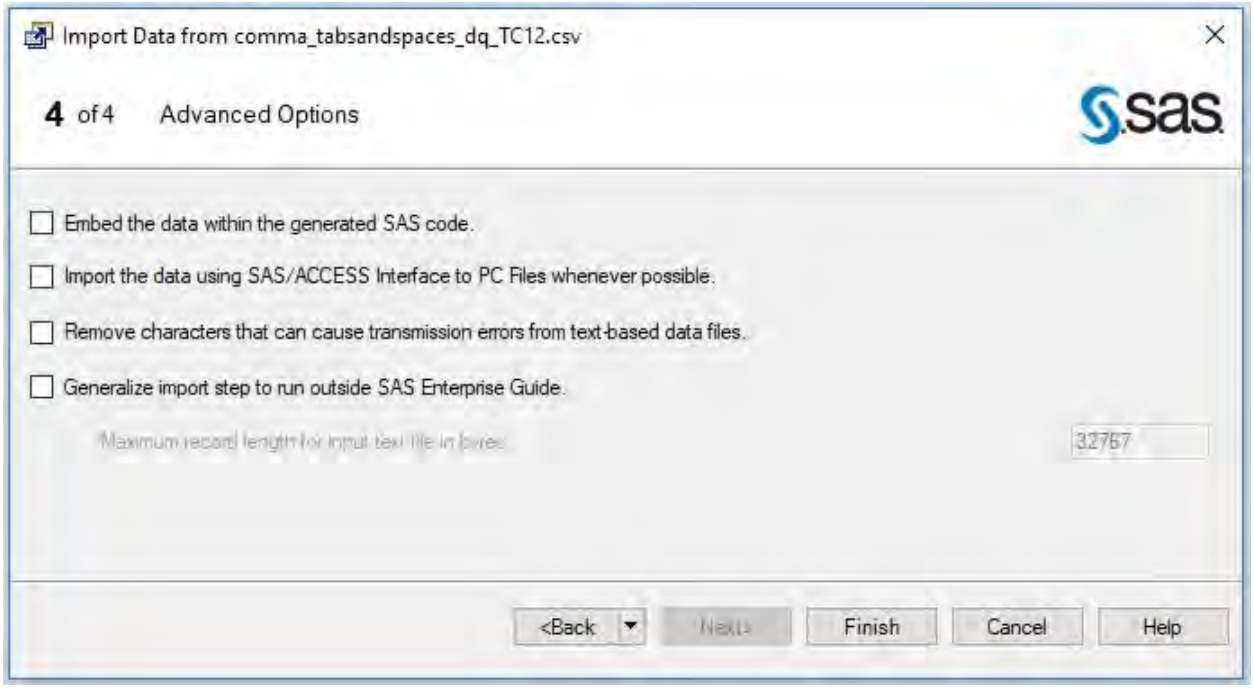
SELECTION WINDOWS USED FOR IMPORTING THE TEST CASE'S DELIMITED TEXT FILE USING SAS ENTERPRISE GUIDE'S IMPORT DATA TASK

Listed below are the selection windows that we use for importing the delimited text file for test case TC_12. We use the same option selections for all of the delimited text file importing occurring in this paper unless otherwise noted.




Import Data from comma_tabsandspaces_dq_TC12.csv
2 of 4 Select Data Source
sas
Text format
Delimited fields
Comma
Text qualifier:
Fixed columns
Record length
16
File contains field names on record number:
1
Data records start at record number:
2
Limit the number of records read to:
Rename columns to comply with SAS naming conventions.
F1_Data, F2_TC ,F3_Data, F4_Test_Case_Description ,F5_Data
""" """" , """, TC12 ,""" """" , """, OK--->escTQc_DLMc_F135 ,""" """" , """
<Back
Next>
Finish
Cancel
Help

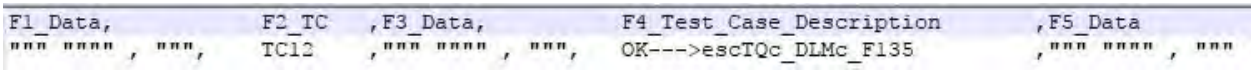

Import Data from comma_tabsandspaces_dq_TC12.csv
3 of 4 Define Field Attributes
sas
Select columns and define attributes:
Inc
Source Name
Name
Label
Type
Source Informat
Len.
Output Format
Output Informat
F1_Data F1_Data F1_Data String $CHAR8. 8 $CHAR8. $CHAR8.
F2_TC F2_TC F2_TC String $CHAR4. 4 $CHAR4. $CHAR4.
F3_Data F3_Data F3_Data String $CHAR8. 8 $CHAR8. $CHAR8.
F4_Test_Cas... F4_Test_Case_Description F4_Test_Case_Description String $CHAR22. 22 $CHAR22. $CHAR22.
F5_Data F5_Data F5_Data String $CHAR8. 8 $CHAR8. $CHAR8.
<Back
Next>
Finish
Cancel
Help



- Notice that the box for “Generalize import step…” in the last selection window is left unchecked.

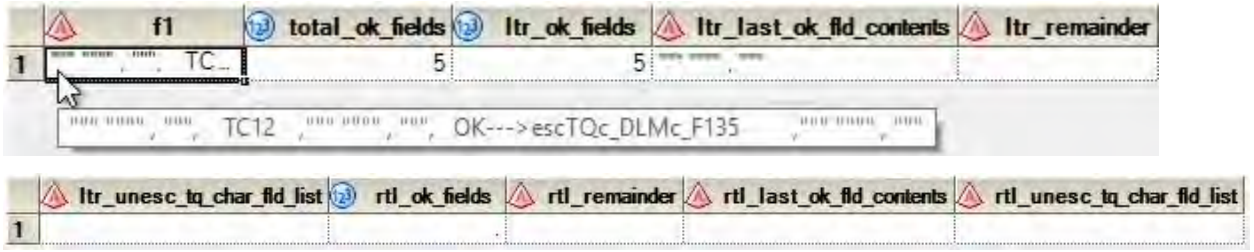
TEST CASE TC_12:

1. Text File Contents:



- The record contained in this delimited text file is in text-qualified format.

2. Macro verification output for the text file (wrapped):



- The macro output contained in the above SAS dataset verifies that the record is indeed in text-qualified format.
- The above macro output satisfies the following 3 macro verification rules:
  - The variable ltr_ok_fields contains the correct value for the number of fields that the text file contains.
  - The variable ltr_remainder contains a value of missing meaning that the pattern matching consumed the entire record and there was nothing left over.
  - The variable ltr_unesc_tq_char_fld_list contains a value of missing meaning that none of the 5 fields contained any embedded, unescaped text-qualifier (i.e. double quote) characters.

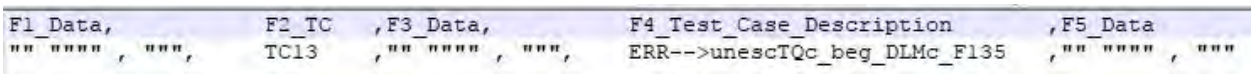
3. SAS Dataset created by importing the text file:

	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1	"",,	TC12	"",,	OK-->escTQc_DLMc_F135	"",,

- The above SAS dataset contains the results of importing the text file using SAS Enterprise Guide's Import Data task using the selections contained in the above 4 selection windows.
- As you can see the data contained in the dataset is error-free.

TEST CASE TC_13:

1. Text File Contents:



```
F1_Data,          F2_TC   ,F3_Data,          F4_Test_Case_Description       ,F5_Data
"" """" , """,    TC13    ,"" """" , """,    ERR-->unescTQc_beg_DLMc_F135   ,"" """" , """
```

- This record contains errors such as having an embedded, unescaped text-qualifier character near the beginning of cells 1, 3, and 5.

2. Macro verification output for the text file (wrapped):

	f1	total_ok_fields	ltr_ok_fields	ltr_last_ok_fld_contents
1	"" """" , """, TC...	1	0	

"" """" , """, TC13 ,"" """" , """, ERR-->unescTQc_beg_DLMc_F135 ,"" """" , """

	ltr_remainder	ltr_unesc_tq_char_fld_list	rtl_ok_fields	rtl_remainder
1	"" """" , """, TC...		1	,"" """, "", TC13 ,"" """

"" """" , """, TC13 ,"" """" , """, ERR-->unescTQc_beg_DLMc_F135 ,"" """" , """

	rtl_last_ok_fld_contents	rtl_unesc_tq_char_fld_list
1	""", ERR-->unescTQc_be...	

""", ERR-->unescTQc_beg_DLMc_F135 ,"""""""" , """

- The variable ltr_ok_fields has a value of 0 indicating that cell #1's pattern matching failed.
  - The matching failed because cell #1 begins with an even number of double quotes (i.e. the text-qualifier and the embedded, unescaped text-qualifier character). A text-qualified cell must begin and end with an ODD number of double quotes if the double quote is the text-qualifier.
  - AS a result of cell #1's failed matching, the macro terminates subsequent left-to-right matching and begins right-to-left matching using the string contained in the variable ltr_remainder.
- Whenever the variable ltr_ok_fields contains a value of 0, then it means that matching has failed at the first cell and the variable ltr_remainder will contain the entire record (as seen above).

3. SAS Dataset created by importing the text file:

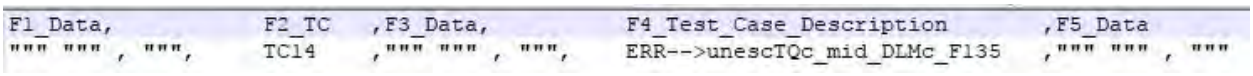
	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1	"",,	TC13	"",,	ERR-->unescTQc_beg_DLMc_F135	"",,

- The Import Data task ignores the embedded, unescaped text-qualifier character near the beginning of fields 1, 3, and 5, but other than that, the data looks perfect! So how could the macro be so seemingly wrong in its prediction that this record is not in text-qualified format? The answer is that the macro was not wrong at all! The pattern was broken in cell #1, so no further

left-to-right matching takes place. Sometimes, a pattern-breaking cell causes huge importing errors when the record is imported as we will see in the next 2 test cases, and sometimes the errors are quite small as in this test case. The macro cannot predict the size of the import error caused by a pattern-breaking cell. It will simply report the cell that broke the pattern.

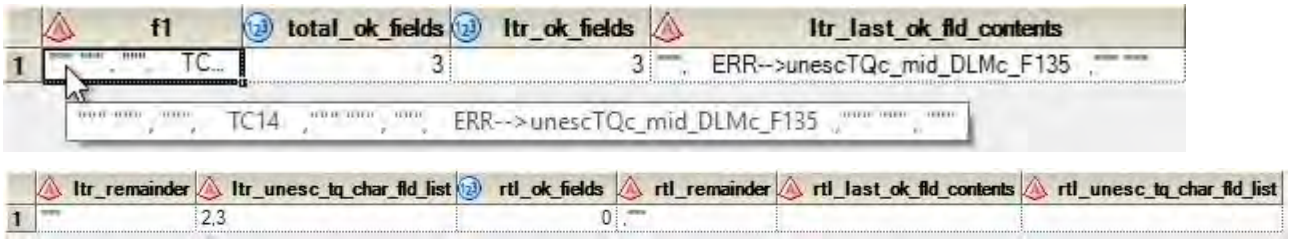
TEST CASE TC_14:

1. Text File Contents:



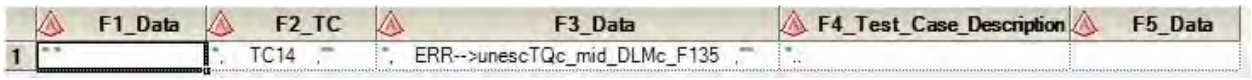
- This record contains an embedded, unescaped text-qualifier character near the middle of cells 1, 3, and 5.

2. Macro verification output for the text file (wrapped):



- The variable ltr_ok_fields indicates that 3 cells were in text-qualified format but this delimited text file contains 5 data cells so this value is incorrect and the contents of these 3 matched cells are incorrect also.
- The variable ltr_remainder does not contain a value of missing as it should.
- The variable ltr_unesc_tq_char_fld_list indicates that fields 2 and 3 both contain embedded, unescaped text-qualifier characters, but realize that this matched field 2 contains part of cell #3 and the matched field 3 contains part of cell #5.
- Any one of the above 3 bullet points would be enough to prove that the record is not in text-qualified format.
- This test case is an example of an embedded, unescaped text-qualifier character causing pattern matching to prematurely match the partial contents of cell #1. In the next test case (test case TC_15), we will see an example of an embedded, unescaped text-qualifier character causing pattern matching to extend beyond the contents of cell #1.

3. SAS Dataset created by importing the text file:



- The above import-resulting SAS dataset mirrors the macro verification output.
- In cell #1 of the text file, the embedded, unescaped double quote is followed by a space and then an embedded comma delimiter. This sequence of characters causes the Import Data task to close the importing of cell #1 at this embedded delimiter. Notice that the remainder of cell #1 is imported into field #2. The ripple effect continues with cell #4 and part of cell #5 being imported into field #3, the remaining part of cell #5 being imported into field #4, and field #5 ending up containing nothing.
- This test case is a good example of an embedded, unescaped text-qualifier character causing havoc due to its proximity to an embedded delimiter character.

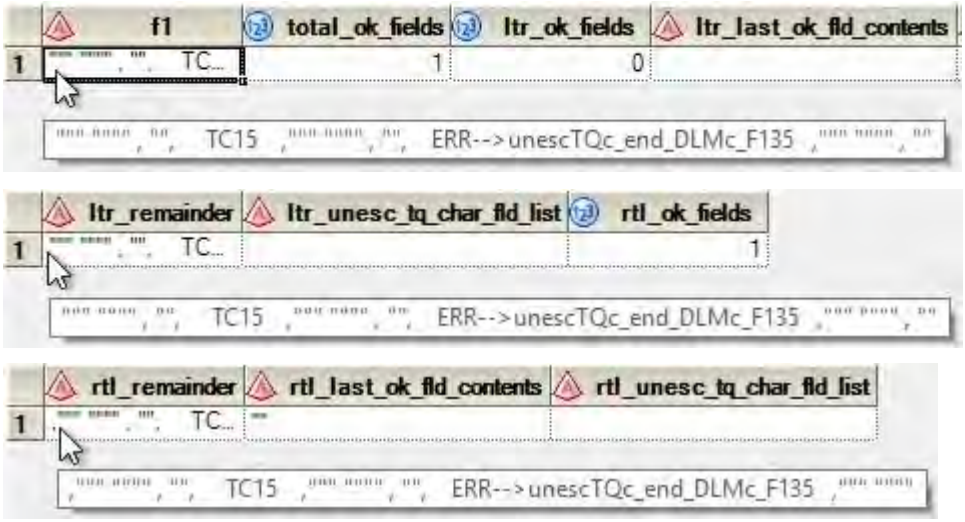
TEST CASE TC_15:

1. Text File Contents:

```
F1_Data,            F2_TC    ,F3_Data,            F4_Test_Case_Description        ,F5_Data
""" """" , "",      TC15     ,""" """" , "",     ERR-->unescTQc_end_DLMc_F135    ,""" """" , ""
```

   - This record contains an embedded, unescaped text-qualifier character near the end of cells 1, 3, and 5.

2. Macro verification output for the text file (wrapped):



   - The variable ltr_ok_fields contains a value of 0 indicating that cell #1's match failed. As we have seen before, if cell #1's match failed, then the variable ltr_remainder will contain the entire record.
   - The unescaped text-qualifier character (i.e. 2nd to last character in cell #1) near the end of cell #1 causes pattern matching to proceed beyond the end of cell #1 because the unescaped text-qualifier character actually escapes the text-qualifier that follows it.
   - Cell #1's match failed because nowhere in the entire record does a closing delimiter for cell #1 exist.
   - This test case is an example of an embedded, unescaped text-qualifier character causing the pattern matching to extend beyond the limits of cell #1 as opposed to the preceding test case TC_14 where an embedded, unescaped text-qualifier character causes pattern matching to match prematurely before the end of cell #1.

3. SAS Dataset created by importing the text file:

   - As you can see, this is a huge importing error. The entire record has been sucked into the first field.
   - This importing error occurs because the Import Data task interprets the 2 double quotes near the end of cell #1 as an escaped double quote pair and not the end of cell #1. The same goes for cell #3 and cell #5. All of this contributes to the entire record being sucked into the F1_Data field.
   - This is yet another example of an embedded, unescaped text-qualifier character being within spaces and/or tabs of an embedded delimiter character and its disastrous effect on the importing of delimited text files.

INTER-FIELD WHITESPACE AND HOW IT RELATES TO SAS ENTERPRISE GUIDE'S IMPORT DATA TASK

A DELIMITED TEXT FILE CONTAINING INTER-FIELD WHITESPACE

I cannot answer the question of how inter-field whitespace makes its way into a text-qualified delimited text file. What I can say is that SAS Enterprise Guide's Import Data task knows how to deal with it as long as you don't select "Generalize import step to run outside SAS Enterprise Guide" in the last (i.e. 4th) selection window of the Import Data task process. For example, take a look at the following text-qualified delimited text file (see Figure 6) which contains records that are all in text-qualified format:

```
F1_Data,          F2_TC   ,F3_Data,          F4_Test_Case_Description      ,F5_Data
,                 TC01    ,,                OK--->no_text_qualifying      ,
"",               TC02    ,"",              OK--->text_qualifying         ,""
"""",             TC03    ,"""",            OK--->text_qualifying         ,""""
",",",",          TC04    ,",",",",         OK--->text_qualifying         ,",",","
 "F1",            TC05    , "F3" ,          OK--->inter_fld_spaces_F135   , "F5"
    "F1"    ,     TC06    ,"F3",            OK--->inter_fld_tabs_F15      ,  "F5"
     "F1"     ,   TC07    ,"F3",            OK--->inter_fld_tbs_spcs_F15  ,   "F5"
""" """" """,     TC08    ,""" """" """,    OK--->escTQc_F135             ,""" """" """
```

Figure 6 - Records in Text-Qualified Format and Containing Inter-Field Whitespace

Notice that Figure 6's text file has inter-field whitespace occurring throughout. As a reminder, inter-field whitespace is defined as tabs and/or spaces that exist between a delimiter and the cell.

NOT SELECTING "GENERALIZE IMPORT STEP…" AND ITS EFFECT ON IMPORTING A TEXT FILE CONTAINING INTER-FIELD WHITESPACE

If we use SAS Enterprise Guide's Import Data task and leave the "Generalize import step…" option unselected, then the resulting imported dataset will look like that in Figure 7:

	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1		TC01		OK--->no_text_qualifying	
2		TC02		OK--->text_qualifying	
3	"	TC03	"	OK--->text_qualifying	"
4	,"	TC04	,"	OK--->text_qualifying	,"
5	F1	TC05	F3	OK--->inter_fld_spaces_F135	F5
6	F1	TC06	F3	OK--->inter_fld_tabs_F15	F5
7	F1	TC07	F3	OK--->inter_fld_tbs_spcs_F15	F5
8	"""	TC08	"""	OK--->escTQc_F135	"""

Figure 7 - Dataset Resulting from Running Import Data Task and NOT Selecting "Generalize import step" Option

The dataset contained in Figure 7 is a perfect import of the delimited text file. By not selecting the option, the Import Data task has no problem dealing with the inter-field whitespace contained in the text file.

SELECTING "GENERALIZE IMPORT STEP…" AND ITS EFFECT ON IMPORTING A TEXT FILE CONTAINING INTER-FIELD WHITESPACE

If you select the option "Generalize import step…", then the resulting imported dataset will look like that in Figure 8:

	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1				OK--->no_te	
2				OK--->text_qu	
3	"		"	OK--->text_qual	"
4	,",		,",	OK--->text_qualif	,",
5	F1		F3	OK--->inter_fld_s	F5
6	"F1"_		F3	OK--->inter_fld	"F5"_
7	"F1"		F3	OK--->inter_fld	"F5"
8	"""		"""	OK--->escTQc_F135	"""

Figure 8 - Dataset Resulting from Running Import Data Task and Selecting "Generalize import step..." Option

Notice with the dataset in Figure 8 that the field F1_Data contains cells in which the text-qualifiers have not been removed and inter-field whitespace ends up in the cell data in field F2_TC and causes the important part of the cell data to get truncated.

SELECTING “GENERALIZE IMPORT STEP…” AND ITS EFFECT ON IMPORTING A TEXT FILE NOT CONTAINING INTER-FIELD WHITESPACE

The delimited text file in Figure 9 contains no inter-field whitespace:

```
F1_Data,F2_TC,F3_Data,F4_Test_Case_Description,F5_Data
,TC01,,OK--->no_text_qualifying,
"",TC02,"",OK--->text_qualifying,""
"""",TC03,"""",OK--->text_qualifying,""""
",","",",TC04,",","",",OK--->text_qualifying,",","",","
"F1",TC05,"F3",OK--->inter_fld_spaces_F135,"F5"
"F1",TC06,"F3",OK--->inter_fld_tabs_F15,"F5"
"F1",TC07,"F3",OK--->inter_fld_tbs_spcs_F15,"F5"
"""""""""",TC08,"""""""""",OK--->escTQc_F135,""""""""""
"""""""",""",TC12,"""""""",""",OK--->escTQc_DLMc_F135,"""""""","""
""",""""""",TC16,""",""""""",OK--->DLMc_escTQc_F135,""",""""""
```

Figure 9 - A Delimited Text File Containing no Inter-Field Whitespace

If we import the text file in Figure 9 and select the “Generalize import step” option, then the dataset in Figure 10 is created:

	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1		TC01		OK--->no_text_qualifying	
2		TC02		OK--->text_qualifying	
3	"	TC03	"	OK--->text_qualifying	"
4	,",	TC04	,",	OK--->text_qualifying	,",
5	F1	TC05	F3	OK--->inter_fld_spaces_F135	F5
6	F1	TC06	F3	OK--->inter_fld_tabs_F15	F5
7	F1	TC07	F3	OK--->inter_fld_tbs_spcs_F15	F5
8	""""""""	TC08	""""""""	OK--->escTQc_F135	""""""""
9	"""",	TC12	"""",	OK--->escTQc_DLMc_F135	"""",
10	","""	TC16	","""	OK--->DLMc_escTQc_F135	","""

Figure 10 - A Successful Import Using the Selection "Generalize import step..."

We can see from the dataset in Figure 10 that the import contains no errors due to the fact that the text file contained no inter-field whitespace.

WHEN TO SELECT THE "GENERALIZE IMPORT STEP…" OPTION AND WHEN NOT TO

When using SAS Enterprise Guide's Import Data task to import delimited text files, it is highly recommended that you do not select the "Generalize import step…" option unless both of the following occur:

1. You are sure that your delimited text file contains no inter-field whitespace in the form of spaces and/or tabs.
2. You need to copy the code that the Import Data task creates in the background in order to run this code outside of SAS Enterprise Guide.

INTER-FIELD WHITESPACE AND PROC IMPORT

Let's take a look at how PROC IMPORT handles inter-field whitespace. We will use the following code:

```
PROC IMPORT OUT= WORK.test
            DATAFILE= "H:\comma_tabsandspaces_dq_proc_import.csv"
            REPLACE;
     GETNAMES=YES;
     DATAROW=2;
     GUESSINGROWS=MAX;
RUN;
```

to import the following CSV file:

```
F1_Data,            F2_TC   ,F3_Data,F4_Test_Case_Description          ,F5_Data
 "F1",              TC05    , "F3" , OK--->inter_fld_spaces_F135       , "F5"
    "F1"     ,      TC06    ,"F3",   OK--->inter_fld_tabs_F145        ,   "F5"
     "F1"      ,    TC07    ,"F3",    OK--->inter_fld_tbs_spcs_F145   ,   "F5"
```

Here is the import-resulting dataset:

	F1_Data	F2_TC	F3_Data	F4_Test_Case_Description	F5_Data
1	F1	TC05	F3	OK--->inter_fld_spaces_F135	F5
2	"F1"	TC06	F3	OK--->inter_fld_tabs_F145	"F5"
3	"F1"	TC07	F3	OK--->inter_fld_tbs_spcs_F145	"F5"

As you can see, PROC IMPORT does a good job except when the inter-field whitespace contains tabs as is the case with test cases TC06 and TC07. You can see in fields F1_Data and F5_Data of these test cases that both the tabs and the double quote text-qualifiers become part of the data.

CONCLUSION

The delimited_text_file_verifier SAS macro will verify whether your delimited text file is in non-text-qualified or text-qualified format. Contained within each format is a text pattern that a delimited text file must adhere to if it is considered to be in one of the 2 formats. For text-qualified format, the macro uses SAS PRX functions to perform pattern matching on each cell of each record of the file. Verification involves the calculation of the following 3 values for each record:

1. The number of correctly-formatted cells going left-to-right across the record.
    - This value must equal the number of fields in the text file.
2. The text string containing the remaining unmatched record.
    - This value must be equal to missing.
3. The number of cells containing embedded, unescaped text-qualifier characters.
    - This value must be equal to missing.

If a record contains the correct values for all 3, then the record's verification information will exist in the good_records dataset. Otherwise, it will exist in the bad_records dataset. If all of your delimited text file's records are listed in the good_records dataset, then your file is in text-qualified format and will be able to be successfully imported using SAS Enterprise Guide's Import Data task.

CONTACT INFORMATION

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

Paul Genovesi
State of Washington/Department of Social and Health Services/Research and Data Analysis Division
pcg7285g@gmail.com

```
/*-- This program header template borrowed from Don Henderson----------------
  DEVELOPED BY : Paul Genovesi
  PURPOSE      : This macro determines whether a delimited text file is in one
                 of the following two formats:
                 1.  Text-qualified
                 2.  Non-text-qualified
|-------------------------------------------------------------------------|
|  MAINTENANCE HISTORY                                                    |
|-------------------------------------------------------------------------|
|  DATE    |     BY    | DESCRIPTION                                      |
|----------|-----------|--------------------------------------------------|
| Mar 2018 |  Paul G   | Initial Creation
|-------------------------------------------------------------------------*/
%macro delimited_text_file_verifier(
        delimiter              /* Must be one of the delimiters contained in
                                  the macro variable dlm_list whose
                                  pipe-delimited contents are listed
                                  below.                          */

       ,text_qualifier         /* Must be one of the following:
                                  1. double quote
                                  2. single quote
                                  3. none                 */

       ,inter_field_whitespace  /* This is the whitespace that can exist
                                  between a text qualifier and a delimiter
                                  within a text-qualified delimited text
                                  file.
                                  Must be one of the following:
                                    1. tabs and spaces
                                    2. spaces only
                                    3. none
                                    4. not applicable               */

       ,number_of_fields  /* This is the number of fields that you are sure
                             the delimited text file contains.  Only supply
                             a number here if you are certain of the number
                             of fields.  Otherwise, enter the word calc in
                             which case the macro will calculate this value
                             by determining the most common value for the
                             number of error-free parsed fields.
                             Must be one of the following:
                               1. calc
                               2. <number>                    */

       ,first_obs         /* The first record to process in the delimited
                             text file.
                             Must be one of the following:
                               1. <number>                    */

       ,last_obs          /* The last record to process in the delimited
                             text file.
                             Must be one of the following:
```

```
                    1. max
                    2. <number>                    */

    ,file_lrecl      /* This value should be greater than the longest
                    record in the delimited text file.
                    Must be one of the following:
                    1. <number>                    */

    ,file_path       /* Absolute path for the delimited text file. */
    );

%let delimiter            =%cmpres(%upcase(&delimiter));
%let text_qualifier       =%cmpres(%upcase(&text_qualifier));
%let inter_field_whitespace=%cmpres(%upcase(&inter_field_whitespace));
%let number_of_fields     =%upcase(&number_of_fields);
%let last_obs             =%upcase(&last_obs);

%put delimiter            =:&delimiter:;
%put text_qualifier       =:&text_qualifier:;
%put inter_field_whitespace =:&inter_field_whitespace:;
%put number_of_fields     =:&number_of_fields:;
%put first_obs            =:&first_obs:;
%put last_obs             =:&last_obs:;
%put file_lrecl           =:&file_lrecl:;
%put file_path            =:&file_path:;

%local dlm_list  text_qualifier_list  inter_field_whitespace_list
      error_msg;

%local most_common_ok_fields_value  max_occurrances; * hash-related;

%let dlm_list=&dlm_list.|AMPERSAND;
%let dlm_list=&dlm_list.|ASTERISK;
%let dlm_list=&dlm_list.|AT SYMBOL;
%let dlm_list=&dlm_list.|BACKSLASH;
%let dlm_list=&dlm_list.|CARET;
%let dlm_list=&dlm_list.|COLON;
%let dlm_list=&dlm_list.|COMMA;
%let dlm_list=&dlm_list.|DOLLAR SIGN;
%let dlm_list=&dlm_list.|EXCLAMATION POINT;
%let dlm_list=&dlm_list.|FORWARD SLASH;
%let dlm_list=&dlm_list.|GRAVE;
%let dlm_list=&dlm_list.|PERCENT SIGN;
%let dlm_list=&dlm_list.|PIPE;
%let dlm_list=&dlm_list.|PLUS SIGN;
%let dlm_list=&dlm_list.|POUND SIGN;
%let dlm_list=&dlm_list.|QUESTION MARK;
%let dlm_list=&dlm_list.|SEMICOLON;
%let dlm_list=&dlm_list.|TAB;
%let dlm_list=&dlm_list.|TILDE;
%let dlm_list=&dlm_list.|;
%*put dlm_list=:&dlm_list:;

%let text_qualifier_list=&text_qualifier_list.|DOUBLE QUOTE;
```

```
%let text_qualifier_list=&text_qualifier_list.|SINGLE QUOTE;
%let text_qualifier_list=&text_qualifier_list.|NONE;
%let text_qualifier_list=&text_qualifier_list.|;

%let inter_field_whitespace_list=&inter_field_whitespace_list.|TABS AND SPACES;
%let inter_field_whitespace_list=&inter_field_whitespace_list.|SPACES ONLY;
%let inter_field_whitespace_list=&inter_field_whitespace_list.|NONE;
%let inter_field_whitespace_list=&inter_field_whitespace_list.|NOT APPLICABLE;
%let inter_field_whitespace_list=&inter_field_whitespace_list.|;

%if  %str(&delimiter)             eq %str() or
     %str(&text_qualifier)        eq %str() or
     %str(&inter_field_whitespace) eq %str() or
     %str(&number_of_fields)      eq %str() or
     %str(&first_obs)             eq %str() or
     %str(&last_obs)              eq %str() or
     %str(&file_lrecl)            eq %str() or
     %str(&file_path)             eq %str()    %then %do;
   %let error_msg=&error_msg%str(ERROR: One of the arguments to the macro );
   %let error_msg=&error_msg%str(delimited_text_file_verifier is missing );
   %let error_msg=&error_msg%str(a value.);
   %put &error_msg;
%end;
%else %if %index(&dlm_list, %str(|&delimiter|)) eq %str(0) %then %do;
   %let error_msg=&error_msg%str(ERROR: The 1st argument to the macro );
   %let error_msg=&error_msg%str(delimited_text_file_verifier must contain );
   %let error_msg=&error_msg%str(a delimiter existing in the delimiter );
   %let error_msg=&error_msg%str(list.);
   %put &error_msg;
%end;
%else %if %index(&text_qualifier_list, %str(|&text_qualifier|))
          eq
         %str(0) %then %do;
   %let error_msg=&error_msg%str(ERROR: The 2nd argument to the macro );
   %let error_msg=&error_msg%str(delimited_text_file_verifier must contain );
   %let error_msg=&error_msg%str(a text qualifier existing in the text );
   %let error_msg=&error_msg%str(qualifier list.);
   %put &error_msg;
%end;
%else %if %index(&inter_field_whitespace_list,
               %str(|&inter_field_whitespace|))
          eq
         %str(0) %then %do;
   %let error_msg=&error_msg%str(ERROR: The 3rd argument to the macro );
   %let error_msg=&error_msg%str(delimited_text_file_verifier must contain );
   %let error_msg=&error_msg%str(a selection existing in the inter-field );
   %let error_msg=&error_msg%str(whitespace list.);
   %put &error_msg;
%end;
%else %if %str(&number_of_fields) ne %str(CALC)
          and
         %sysfunc(prxmatch(/^[1-9]\d*$/,&number_of_fields)) eq %str(0)
      %then %do;
   %let error_msg=&error_msg%str(ERROR: The 4th argument to the macro );
```

```
  %let error_msg=&error_msg%str(delimited_text_file_verifier must );
  %let error_msg=&error_msg%str(contain either the word calc or a number );
  %let error_msg=&error_msg%str(greater than 0.);
  %put &error_msg;
%end;
%else %if %sysfunc(prxmatch(/^[1-9]\d*$/,&first_obs)) eq %str(0) %then %do;
  %let error_msg=&error_msg%str(ERROR: The 5th argument to the macro );
  %let error_msg=&error_msg%str(delimited_text_file_verifier must );
  %let error_msg=&error_msg%str(contain a number greater than 0.);
  %put &error_msg;
%end;
%else %if %str(&last_obs) ne %str(MAX)
         and
        %sysfunc(prxmatch(/^[1-9]\d*$/,&last_obs)) eq %str(0)
     %then %do;
  %let error_msg=&error_msg%str(ERROR: The 6th argument to the macro );
  %let error_msg=&error_msg%str(delimited_text_file_verifier must );
  %let error_msg=&error_msg%str(contain either the word max or a number );
  %let error_msg=&error_msg%str(greater than 0.);
  %put &error_msg;
%end;
%else %if %sysfunc(prxmatch(/^[1-9]\d*$/,&file_lrecl)) eq %str(0) %then %do;
  %let error_msg=&error_msg%str(ERROR: The 7th argument to the macro );
  %let error_msg=&error_msg%str(delimited_text_file_verifier must );
  %let error_msg=&error_msg%str(contain a number greater than 0.);
  %put &error_msg;
%end;
%else %if %sysfunc(fileexist(&file_path)) eq %str(0) %then %do;
  %let error_msg=&error_msg%str(ERROR: The 8th argument to the macro );
  %let error_msg=&error_msg%str(delimited_text_file_verifier contains a );
  %let error_msg=&error_msg%str(file that does not exist.);
  %put &error_msg;
%end;
%else %if %str(&text_qualifier) ne %str(NONE)
        and
        %str(&inter_field_whitespace) eq %str(NOT APPLICABLE) %then %do;
  %let error_msg=&error_msg%str(ERROR: For the delimited_text_file_verifier );
  %let error_msg=&error_msg%str(macro, if the text_qualifier argument does );
  %let error_msg=&error_msg%str(not equal NONE, then the );
  %let error_msg=&error_msg%str(inter_field_whitespace argument must not );
  %let error_msg=&error_msg%str(equal NOT APPLICABLE.);
  %put &error_msg;
%end;
%else %if %str(&text_qualifier) eq %str(NONE)
        and
        %str(&inter_field_whitespace) ne %str(NOT APPLICABLE) %then %do;
  %let error_msg=&error_msg%str(ERROR: For the delimited_text_file_verifier );
  %let error_msg=&error_msg%str(macro, if the text_qualifier argument );
  %let error_msg=&error_msg%str(equals NONE, then the );
  %let error_msg=&error_msg%str(inter_field_whitespace argument must );
  %let error_msg=&error_msg%str(equal NOT APPLICABLE.);
  %put &error_msg;
%end;
%else %if %str(&delimiter) eq %str(TAB)
```

```
        and
        %str(&inter_field_whitespace) eq %str(TABS AND SPACES) %then %do;
    %let error_msg=&error_msg%str(ERROR: For the delimited_text_file_verifier );
    %let error_msg=&error_msg%str(macro, if TAB is the delimiter then );
    %let error_msg=&error_msg%str(inter_field_whitespace must not equal );
    %let error_msg=&error_msg%str(TABS AND SPACES.);
    %put &error_msg;
  %end;
  %else %do;
    * Import the delimited text file into a one-field dataset.;
    DATA one_field_dataset;
      length current_record_length 8
           longest_record_length 8;
      length f1 $ &file_lrecl;
      keep f1;
      retain longest_record_length 0;
      infile "&file_path"  LRECL=&file_lrecl  firstobs=&first_obs
                    obs=&last_obs
                    end=last;
      input @;
      f1 = _infile_;

      current_record_length = lengthn(f1);
      if current_record_length > longest_record_length then
        longest_record_length = current_record_length;
      if last then do;
        longest_record_length = longest_record_length + 2;
        call symputx('longest_record_length',
                  put(longest_record_length, 5.));
      end;
    RUN;
    * The following macro variables are created here:
      NOTE:  Some text-qualified-related macro variables are created but
           then not used because the text_qualifier macro variable has
           a value of NONE.
      1. dlm_lit - the literal value for the delimiter.
      2. dlm_rgx - the regex pattern for the delimiter.
      3. whitespace_greedy - the regex pattern for greedy white space.
      4. whitespace_lazy   - the regex pattern for lazy white space.
      5. escapables - a list of perl metachars that need escaping when used
                  within a regex. ;
    * HEX regex values used below:   \x20 = space   \x09 = tab;
    %local  escapables  whitespace_greedy  whitespace_lazy  dlm_lit  dlm_rgx;
    %let escapables=%str(^$.|*+?\/);
    %if %str(&delimiter) eq %str(TAB) %then %do;
      %let whitespace_greedy=%str(\x20*); * \x20 = space;
      %let whitespace_lazy  =%str(\x20*?);
      %let dlm_lit=;  * When a tab, this macro variable is used later. ;
      %let dlm_rgx=%str(\x09); * \x09 = tab;
    %end;
    %else %if %str(&delimiter) eq %str(AMPERSAND)
           or
           %str(&delimiter) eq %str(PERCENT SIGN) %then %do;
      %if %str(&inter_field_whitespace) eq %str(SPACES ONLY) %then %do;
```

```
    %let whitespace_greedy=%str(\x20*); * \x20 = space;
    %let whitespace_lazy  =%str(\x20*?);
  %end;
  %else %if %str(&inter_field_whitespace) eq %str(TABS AND SPACES) %then %do;
    %let whitespace_greedy=%str([\x09\x20]*);  * \x09 = tab;
    %let whitespace_lazy  =%str([\x09\x20]*?);
  %end;
  %else %do;
    %let whitespace_greedy=%str();
    %let whitespace_lazy  =%str();
  %end;

  %if %str(&delimiter) eq %str(AMPERSAND) %then %do;
    %let dlm_lit=%nrbquote(&);
    %let dlm_rgx=%str(\x26);
  %end;
  %else %do;
    %let dlm_lit=%nrbquote(%);
    %let dlm_rgx=%str(\x25);
  %end;
%end;
%else %do;
  %if %str(&inter_field_whitespace) eq %str(SPACES ONLY) %then %do;
    %let whitespace_greedy=%str(\x20*); * \x20 = space;
    %let whitespace_lazy  =%str(\x20*?);
  %end;
  %else %if %str(&inter_field_whitespace) eq %str(TABS AND SPACES) %then %do;
    %let whitespace_greedy=%str([\x09\x20]*);  * \x09 = tab;
    %let whitespace_lazy  =%str([\x09\x20]*?);
  %end;
  %else %do;
    %let whitespace_greedy=%str();
    %let whitespace_lazy  =%str();
  %end;

  %if %str(&delimiter)       eq %str(ASTERISK) %then
    %let dlm_lit=%str(*);
  %else %if %str(&delimiter) eq %str(BACKSLASH) %then
    %let dlm_lit=%str(\);
  %else %if %str(&delimiter) eq %str(CARET) %then
    %let dlm_lit=%str(^);
  %else %if %str(&delimiter) eq %str(DOLLAR SIGN) %then
    %let dlm_lit=%str($);
  %else %if %str(&delimiter) eq %str(FORWARD SLASH) %then
    %let dlm_lit=%str(/);
  %else %if %str(&delimiter) eq %str(PIPE) %then
    %let dlm_lit=%str(|);
  %else %if %str(&delimiter) eq %str(PLUS SIGN) %then
    %let dlm_lit=%str(+);
  %else %if %str(&delimiter) eq %str(QUESTION MARK) %then
    %let dlm_lit=%str(?);
  %else %if %str(&delimiter) eq %str(AT SYMBOL) %then
    %let dlm_lit=%str(@);
  %else %if %str(&delimiter) eq %str(COLON) %then
```

```
    %let dlm_lit=%str(:);
  %else %if %str(&delimiter) eq %str(COMMA) %then
    %let dlm_lit=%str(,);
  %else %if %str(&delimiter) eq %str(EXCLAMATION POINT) %then
    %let dlm_lit=%str(!);
  %else %if %str(&delimiter) eq %str(GRAVE) %then
    %let dlm_lit=%str(`);
  %else %if %str(&delimiter) eq %str(POUND SIGN) %then
    %let dlm_lit=%str(#);
  %else %if %str(&delimiter) eq %str(SEMICOLON) %then
    %let dlm_lit=%str(;);
  %else %if %str(&delimiter) eq %str(TILDE) %then
    %let dlm_lit=%str(~);

  %if %index(&escapables,&dlm_lit) %then
    %let dlm_rgx=%str(\&dlm_lit);
  %else
    %let dlm_rgx=%str(&dlm_lit);
%end;
%put delimiter        =:&delimiter:;
%put dlm_lit          =:&dlm_lit:;
%put dlm_rgx          =:&dlm_rgx:;
%put whitespace_greedy=:&whitespace_greedy:;
%put whitespace_lazy  =:&whitespace_lazy:;
%put escapables       =:&escapables:;
%put longest_record_length=:&longest_record_length:;

%if %str(&text_qualifier) eq %str(NONE) %then %do;
  data all_records;
    length most_common_num_fields_value  8;  * hash-related;
    length max_occurrances               8;  * hash-related;
    length f1_new                    $  &longest_record_length
         number_of_fields             8
         ;
    set one_field_dataset (obs=max) end=last;
    drop f1;
    drop most_common_num_fields_value
        max_occurrances
        num_fields_value
        num_occurrances
        rc
        ;  * Hash-related;
    rename f1_new=f1;
    if _n_ eq 1 then do;
      * Hash-related Begin;
      declare hash num_fields_determiner(ordered: 'y');
      declare hiter i_num_fields_determiner('num_fields_determiner');
      num_fields_determiner.defineKey('num_fields_value');
      num_fields_determiner.defineData('num_fields_value',
                                       'num_occurrances');
      num_fields_determiner.defineDone();
      call missing(num_fields_value,      num_occurrances);
      * Hash-related End;
    end;
```

```
  f1_new = trimn(f1);
  number_of_fields = lengthn(prxchange('s/[^'              ||
                                symget('dlm_rgx')  ||
                               ']//', -1, f1_new)
                              ) + 1;
  * Hash-related Begin;
  num_fields_value = number_of_fields;
  if num_fields_determiner.find() ne 0 then do;
    num_occurrances = 1;
    num_fields_determiner.add();
  end;
  else do;
    num_occurrances = sum(num_occurrances, 1);
    num_fields_determiner.replace();
  end;
  * Hash-related End;
  if last then do;
    * Hash-related Begin;
    rc = i_num_fields_determiner.first();
    do while (rc = 0);
      if num_occurrances gt max_occurrances then do;
        most_common_num_fields_value = num_fields_value;
        max_occurrances              = num_occurrances;
      end;
      rc = i_num_fields_determiner.next();
    end;
    call symputx('most_common_num_fields_value',
                 most_common_num_fields_value);
    call symputx('max_occurrances',  max_occurrances);
    * Hash-related End;
  end;
run;
%put tqNONE_most_common_num_fields_value =:&most_common_num_fields_value:;
%put tqNONE_max_occurrances              =:&max_occurrances:;
data good_records (drop=good_rec_count  bad_rec_count)
    bad_records  (drop=good_rec_count  bad_rec_count)
    record_stats (keep=good_rec_count  bad_rec_count)
    ;
  set all_records end=last;
  retain good_rec_count  0
       bad_rec_count   0
       most_common_num_fields_value 0
       ;
  if _n_ eq 1 then do;
    most_common_num_fields_value =
      symget('most_common_num_fields_value');
  end;
  if number_of_fields
    eq
    most_common_num_fields_value then do;
      good_rec_count = good_rec_count + 1;
      output good_records;
  end;
  else do;
```

```
        bad_rec_count = bad_rec_count + 1;
        output bad_records;
    end;
    if last then
      output record_stats;
  run;
%end;
%else %do;  %* text_qualifier equals either DOUBLE QUOTE or SINGLE QUOTE ;
  data all_records;
    length most_common_ok_fields_value 8;  * hash-related;
    length max_occurrances             8;  * hash-related;
    length f1_new                   $  &longest_record_length
        total_ok_fields              8
        ltr_ok_fld_count             8
        ltr_last_ok_fld_contents    $  &longest_record_length
        ltr_remainder               $  &longest_record_length
        ltr_keep_going               8
        ltr_unesc_tq_char_fld_list $  200
        rtl_ok_fld_count             8
        rtl_remainder               $  &longest_record_length
        rtl_last_ok_fld_contents    $  &longest_record_length
        rtl_keep_going               8
        rtl_unesc_tq_char_fld_list $  200
        field_wo_tqs                $  &longest_record_length
        text_qualifier              $  12
        dlm_name                    $  20
        ;
    set one_field_dataset (obs=max) end=last;
    drop
      ltr_keep_going
      rtl_keep_going
      ltr_no_tqs_id
      ltr_tqs_wo_tq_chars_id
      ltr_tqs_w_tq_chars_id
      ltr_file_wo_tqs_id
      rtl_no_tqs_id
      rtl_tqs_wo_tq_chars_id
      rtl_tqs_w_tq_chars_id
      rtl_file_wo_tqs_id
      remove_a_fields_tqs
      check_for_unescaped_tq_chars
      field_wo_tqs
      f1
      text_qualifier
      dlm_name
      ;
    drop most_common_ok_fields_value
      max_occurrances
      ok_fields_value
      num_occurrances
      rc
      ;  * Hash-related;
    rename ltr_ok_fld_count=ltr_ok_fields
        rtl_ok_fld_count=rtl_ok_fields
```

```
      f1_new=f1
      ;
if _n_ eq 1 then do;
  * Hash-related Begin;
  declare hash num_fields_determiner(ordered: 'y');
  declare hiter i_num_fields_determiner('num_fields_determiner');
  num_fields_determiner.defineKey('ok_fields_value');
  num_fields_determiner.defineData('ok_fields_value',
                                   'num_occurrances');
  num_fields_determiner.defineDone();
  call missing(ok_fields_value,      num_occurrances);
  * Hash-related End;
  text_qualifier = symget('text_qualifier');
  dlm_name       = symget('delimiter');
  retain ltr_no_tqs_id
         ltr_tqs_wo_tq_chars_id
         ltr_tqs_w_tq_chars_id
         ltr_file_wo_tqs_id
         rtl_no_tqs_id
         rtl_tqs_wo_tq_chars_id
         rtl_tqs_w_tq_chars_id
         rtl_file_wo_tqs_id
         remove_a_fields_tqs
         check_for_unescaped_tq_chars
         text_qualifier
         dlm_name
         ;
  if text_qualifier eq "DOUBLE QUOTE" then do;
    ltr_no_tqs_id   = prxparse('/^([^"]*?)'   ||
                       symget('dlm_rgx')  ||
                       '([\d\D]*)$/');
    rtl_no_tqs_id   = prxparse('/^([\d\D]*)'  ||
                       symget('dlm_rgx')  ||
                       '([^"]*)$/');
    ltr_tqs_wo_tq_chars_id = prxparse('/^' ||
                              symget('whitespace_lazy')    ||
                              '("(?:"")*?[^"]*?"(?:"")*?)' ||
                              symget('whitespace_lazy')    ||
                              symget('dlm_rgx')  ||
                              '([\d\D]*)$/');
    rtl_tqs_wo_tq_chars_id = prxparse('/^([\d\D]*)'               ||
                              symget('dlm_rgx')  ||
                              symget('whitespace_lazy')    ||
                              '("(?:"")*?[^"]*?"(?:"")*?)' ||
                              symget('whitespace_greedy')  ||
                              '$/');

    ltr_tqs_w_tq_chars_id
      = prxparse('/^'                              ||
             symget('whitespace_lazy')               ||
             '("(?:"")*?(?!")[\d\D]*?(?<!")"(?:"")*?)' ||
             symget('whitespace_lazy')               ||
             symget('dlm_rgx')  ||
             '([\d\D]*)$/');
```

```
  rtl_tqs_w_tq_chars_id =
    prxparse('/^([\d\D]*)'                          ||
             symget('dlm_rgx')                      ||
             symget('whitespace_lazy')              ||
             '("(?:"")*?(?!")[\d\D]*?(?<!")"(?:"")*?)' ||
             symget('whitespace_greedy')            ||
             '$/');
  remove_a_fields_tqs   = prxparse('/^'                     ||
                           symget('whitespace_greedy') ||
                           '"([\d\D]*)"'            ||
                           symget('whitespace_greedy') ||
                           '$/');
  check_for_unescaped_tq_chars = prxparse('/(?<!")""*(?!")/');
end;
else if text_qualifier eq "SINGLE QUOTE" then do;
  ltr_no_tqs_id   = prxparse("/^([^']*?)"   ||
                    symget('dlm_rgx')  ||
                    "([\d\D]*)$/");
  rtl_no_tqs_id   = prxparse("/^([\d\D]*)"  ||
                    symget('dlm_rgx')  ||
                    "([^']*)$/");
  ltr_tqs_wo_tq_chars_id = prxparse("/^"                    ||
                           symget('whitespace_lazy')    ||
                           "('(?:")*?[^']*?'(?:")*?)" ||
                           symget('whitespace_lazy')  ||
                           symget('dlm_rgx')  ||
                           "([\d\D]*)$/");
  rtl_tqs_wo_tq_chars_id = prxparse("/^([\d\D]*)"           ||
                           symget('dlm_rgx')  ||
                           symget('whitespace_lazy')    ||
                           "('(?:")*?[^']*?'(?:")*?)" ||
                           symget('whitespace_greedy')  ||
                           "$/");

  ltr_tqs_w_tq_chars_id =
    prxparse("/^"                               ||
             symget('whitespace_lazy')          ||
             "('(?:")*?(?!')[\d\D]*?(?<!')'(?:")*?)" ||
             symget('whitespace_lazy')          ||
             symget('dlm_rgx')                  ||
             "([\d\D]*)$/");
  rtl_tqs_w_tq_chars_id =
    prxparse("/^([\d\D]*)"                      ||
             symget('dlm_rgx')                  ||
             symget('whitespace_lazy')          ||
             "('(?:")*?(?!')[\d\D]*?(?<!')'(?:")*?)" ||
             symget('whitespace_greedy')        ||
             "$/");
  remove_a_fields_tqs =
    prxparse("/^"                   ||
             symget('whitespace_greedy') ||
             '"([\d\D]*)"'          ||
             symget('whitespace_greedy') ||
             "$/");
```

```
    check_for_unescaped_tq_chars = prxparse("/(?<!')(')*(?!')/");
  end;
  else if strip(text_qualifier) eq "NONE" then do;
    ltr_file_wo_tqs_id   = prxparse('/^([\d\D]*?)'     ||
                           symget('dlm_rgx')  ||
                           '([\d\D]*)$/');
    rtl_file_wo_tqs_id   = prxparse('/^([\d\D]*)'      ||
                           symget('dlm_rgx')  ||
                           '([\d\D]*)$/');
  end;
end;
f1_new = trimn(f1);
ltr_ok_fld_count = 0;
ltr_keep_going = 1;
if dlm_name eq 'TAB' then do;
  ltr_remainder  = trimn(f1_new) || '09'x;
end;
else do;
  ltr_remainder  = trimn(f1_new) || symget('dlm_lit');
end;
ltr_unesc_tq_char_fld_list = "";
do while (ltr_keep_going);
  if strip(text_qualifier) ne "NONE" then do;
    if prxmatch(ltr_no_tqs_id, ltr_remainder) then do;
      ltr_last_ok_fld_contents =
        strip(prxposn(ltr_no_tqs_id, 1, ltr_remainder));
      ltr_remainder =
        strip(prxposn(ltr_no_tqs_id, 2, ltr_remainder));
      ltr_ok_fld_count = ltr_ok_fld_count + 1;
    end;
    else if prxmatch(ltr_tqs_wo_tq_chars_id, ltr_remainder) then do;
      ltr_last_ok_fld_contents =
        strip(prxposn(ltr_tqs_wo_tq_chars_id, 1, ltr_remainder));
      ltr_remainder =
        strip(prxposn(ltr_tqs_wo_tq_chars_id, 2, ltr_remainder));
      ltr_ok_fld_count = ltr_ok_fld_count + 1;
    end;
    else if prxmatch(ltr_tqs_w_tq_chars_id, ltr_remainder) then do;
      ltr_last_ok_fld_contents =
        strip(prxposn(ltr_tqs_w_tq_chars_id, 1, ltr_remainder));
      ltr_remainder =
        strip(prxposn(ltr_tqs_w_tq_chars_id, 2, ltr_remainder));
      ltr_ok_fld_count = ltr_ok_fld_count + 1;
      if prxmatch(remove_a_fields_tqs,
              strip(ltr_last_ok_fld_contents)) then do;
        field_wo_tqs = strip(prxposn(remove_a_fields_tqs, 1,
                               ltr_last_ok_fld_contents));
        if prxmatch(check_for_unescaped_tq_chars,
                field_wo_tqs) then do;
          ltr_unesc_tq_char_fld_list =
            strip(ltr_unesc_tq_char_fld_list) ||
            "," || strip(put(ltr_ok_fld_count, 4.));
        end;
      end;
```

```
      end;
      else
        ltr_keep_going = 0;
    end;
    else do;
      if prxmatch(ltr_file_wo_tqs_id, ltr_remainder) then do;
        ltr_last_ok_fld_contents =
          strip(prxposn(ltr_file_wo_tqs_id, 1, ltr_remainder));
        ltr_remainder =
          strip(prxposn(ltr_file_wo_tqs_id, 2, ltr_remainder));
        ltr_ok_fld_count = ltr_ok_fld_count + 1;
      end;
      else
        ltr_keep_going = 0;
    end;
  end;
  if missing(ltr_remainder)
    and
    missing(ltr_unesc_tq_char_fld_list) then do;
      * NOTE: We add our value to the hash only if there is no ltr
            remainder and no unescaped tq chars. In other words,
            this record experienced no formatting issues. ;
      * Hash-related Begin;
      ok_fields_value = ltr_ok_fld_count;
      if num_fields_determiner.find() ne 0 then do;
        num_occurrances = 1;
        num_fields_determiner.add();
      end;
      else do;
        num_occurrances = sum(num_occurrances, 1);
        num_fields_determiner.replace();
      end;
      * Hash-related End;
  end;
  * Below reads, if variable is not missing, then remove the leading
    comma from the variable. ;
  if ^missing(ltr_unesc_tq_char_fld_list) then do;
    ltr_unesc_tq_char_fld_list =
      prxchange('s/^\s*,\s*([\d\D]*)\s*$/$1/', 1,
              ltr_unesc_tq_char_fld_list);
  end;

  rtl_unesc_tq_char_fld_list = "";
  * Below reads, if there exists remainder after processing from
    left-to-right, then begin processing the remainder from
    right-to-left. ;
  if ^missing(ltr_remainder) then do;
    * Below removes the delimiter from the end of the string. ;
    ltr_remainder = prxchange('s/^([\d\D]*)'     ||
                        symget('dlm_rgx')  ||
                        '\s*$/$1/', 1, ltr_remainder);
    * Prepend with the delimiter prior to right-to-left parsing. ;
    if dlm_name eq 'TAB' then do;
      rtl_remainder = '09'x || ltr_remainder;
```

```
end;
else do;
  rtl_remainder = symget('dlm_lit') || ltr_remainder;
end;

rtl_ok_fld_count = 0;
rtl_keep_going = 1;
do while (rtl_keep_going);
  if text_qualifier ne "NONE" then do;
    if prxmatch(rtl_no_tqs_id, rtl_remainder) then do;
      rtl_last_ok_fld_contents
        = strip(prxposn(rtl_no_tqs_id, 2, rtl_remainder));
      rtl_remainder
        = strip(prxposn(rtl_no_tqs_id, 1, rtl_remainder));
      rtl_ok_fld_count = rtl_ok_fld_count + 1;
    end;
    else if prxmatch(rtl_tqs_wo_tq_chars_id, rtl_remainder) then do;
      rtl_last_ok_fld_contents
        = strip(prxposn(rtl_tqs_wo_tq_chars_id, 2, rtl_remainder));
      rtl_remainder
        = strip(prxposn(rtl_tqs_wo_tq_chars_id, 1, rtl_remainder));
      rtl_ok_fld_count = rtl_ok_fld_count + 1;
    end;
    else if prxmatch(rtl_tqs_w_tq_chars_id, rtl_remainder) then do;
      rtl_last_ok_fld_contents
        = strip(prxposn(rtl_tqs_w_tq_chars_id, 2, rtl_remainder));
      rtl_remainder
        = strip(prxposn(rtl_tqs_w_tq_chars_id, 1, rtl_remainder));
      rtl_ok_fld_count = rtl_ok_fld_count + 1;
      if prxmatch(remove_a_fields_tqs,
                  rtl_last_ok_fld_contents) then do;
        field_wo_tqs = strip(prxposn(remove_a_fields_tqs, 1,
                                     rtl_last_ok_fld_contents));
        if prxmatch(check_for_unescaped_tq_chars,
                    field_wo_tqs) then do;
          rtl_unesc_tq_char_fld_list =
            strip(rtl_unesc_tq_char_fld_list) ||
            "," || strip(put(rtl_ok_fld_count, 4.));
        end;
      end;
    end;
    else
      rtl_keep_going = 0;
  end;
  else do;
    if prxmatch(rtl_file_wo_tqs_id, rtl_remainder) then do;
      rtl_last_ok_fld_contents
        = strip(prxposn(rtl_file_wo_tqs_id, 1, rtl_remainder));
      rtl_remainder
        = strip(prxposn(rtl_file_wo_tqs_id, 2, rtl_remainder));
      rtl_ok_fld_count = rtl_ok_fld_count + 1;
    end;
    else
      rtl_keep_going = 0;
```

```
      end;
    end;
  end;
  * Below reads, if variable is not missing, then remove the leading
    comma from the variable. ;
  if ^missing(rtl_unesc_tq_char_fld_list) then do;
    rtl_unesc_tq_char_fld_list =
      prxchange('s/^\s*,\s*([\d\D]*)\s*$/$1/', 1,
              rtl_unesc_tq_char_fld_list);
  end;
  total_ok_fields = sum(ltr_ok_fld_count,
                  rtl_ok_fld_count);
  output;
  if last then do;
    * Hash-related Begin;
    rc = i_num_fields_determiner.first();
    do while (rc = 0);
      if num_occurrances gt max_occurrances then do;
        most_common_ok_fields_value = ok_fields_value;
        max_occurrances           = num_occurrances;
      end;
      rc = i_num_fields_determiner.next();
    end;
    call symputx('most_common_ok_fields_value', most_common_ok_fields_value);
    call symputx('max_occurrances',           max_occurrances);
    * Hash-related End;
  end;
run;
%put most_common_ok_fields_value=:&most_common_ok_fields_value::;
%put Max_occurrances          =:&max_occurrances::;

%if %str(&number_of_fields) eq %str(CALC) %then
  %let number_of_fields =&most_common_ok_fields_value;
%put number_of_fields=:&number_of_fields::;
%*let number_of_fields =&most_common_ok_fields_value;

%if %str(&text_qualifier) ne %str(NONE) %then %do;
  data good_records (drop=good_rec_count  bad_rec_count  unesc_tqc_count)
       bad_records  (drop=good_rec_count  bad_rec_count  unesc_tqc_count)
       record_stats (keep=good_rec_count  bad_rec_count  unesc_tqc_count)
       ;
    set all_records end=last;
    retain good_rec_count  0
           bad_rec_count   0
           unesc_tqc_count 0
           ;
    if missing(ltr_remainder)
      and
      ltr_ok_fields eq symget('number_of_fields')
      and
      missing(ltr_unesc_tq_char_fld_list) then do;
        good_rec_count = good_rec_count + 1;
        output good_records;
    end;
```

```
          else do;
               bad_rec_count = bad_rec_count + 1;
               output bad_records;
          end;
          if ^missing(ltr_unesc_tq_char_fld_list) then
            unesc_tqc_count = unesc_tqc_count + 1;
          if last then
            output record_stats;
        run;
      %end;
      %else %do;
        data good_records (drop=good_rec_count  bad_rec_count)
             bad_records  (drop=good_rec_count  bad_rec_count)
             record_stats (keep=good_rec_count  bad_rec_count)
             ;
          set all_records end=last;
          retain good_rec_count  0
                 bad_rec_count   0
                 ;
          if missing(ltr_remainder)
            and
            ltr_ok_fields eq symget('number_of_fields') then do;
               good_rec_count = good_rec_count + 1;
               output good_records;
          end;
          else do;
               bad_rec_count = bad_rec_count + 1;
               output bad_records;
          end;
          if last then
            output record_stats;
        run;
      %end;
    %end;
    proc delete data=one_field_dataset;
    run;
  %end;
  %if %length(&error_msg) ne  %str(0) %then %do;
    %abort cancel;
  %end;
  %else %do;
    proc print data=RECORD_STATS;
    run;
  %end;
%mend delimited_text_file_verifier;
```