Using Match regex to Import CSV data

By Charles Vass, Technical Services Team Member, 4D Inc.

Technical Note 08-32

Table of Contents

Abstract

The complexities of importing a Comma-Separated Values (CSV) file are greatly simplified by using the **Match regex** command in 4D v11 SQL. This Technical Note presents a 4D method that can be used to parse CSV files, using the Match regex command.

A sample database is included.

Introduction

What is CSV and its many uses? CSV is an acronym for Comma-Separated Values. The best definition I've seen is by Shelley Elmblad of About.com. She explains:

> *CSV is a common file type used to import data from one software application to another, with commas separating the values in each field. CSV is often used to transfer data between databases or spreadsheet tables, and most financial software is a collection of databases or spreadsheets, making CSV a common method of transferring data between financial software or between financial software and spreadsheets or databases.*

The CSV file format is often used to exchange data between disparate applications. The file format, as it is used in Microsoft Excel, has become a pseudo standard throughout the industry, even among non-Microsoft platforms.

As is the case with most exchange formats since XML, CSV files have become somewhat of a legacy format. New applications that wish to include an export format will generally use XML today. In legacy systems though (pre-XML), CSV files had indeed become a de facto industry standard. Just as there are still billions of lines of CoBOL code in use today that need to be maintained, support for a legacy standard such as CSV is likely to be required long after it has stopped being implemented in new designs.

Just to highlight pervasiveness and current relevance to being able to handle a CSV file, there are several iPhone applications that convert CSV data for iPhone presentation.

It is not uncommon for a developer to have a client that regularly receives reports in the CSV format and will need a way to programmatically import these files into a 4D database. Prior to 4D v11 SQL this would mean many, many lines of parsing code that handled all to exceptions from the norm in a CSV file. Employing the 4D v11 SQL command **Match regex** can greatly simplify this task.

What Makes CSV Challenging?

So what makes CSV challenging? The CSV file format!

If all that was ever exported to a CSV file were numbers separated by commas it would be no challenge at all. While no formal specification for CSV exists, RFC 4180 from October 2005 describes a common format and establishes "text/csv" as the MIME type registered with the IANA. Since CSV files existed well before 2005 the RFC is only one special view on CSV files. Below is a comprehensive definition found in numerous sources on the internet.

CSV Rules

While the fields are normally separated by commas, a CSV field contained within quotes can have one or more commas included. A CSV field that contains commas must be contained within quotes. And within those quotes any combination of characters can exist including multiple commas, multiple quotes. In other words, the programmatic challenge is knowing exactly where the closing quote mark is. Here are the rules to follow:

Each record is one line...or is it?

A record separator may consist of a line feed (ASCII/LF=0x0A), or a carriage return and line feed pair (ASCII/CRLF=0x0D 0x0A). However, fields may contain embedded line-breaks (see example below) so a record may span more than one line.

Fields are separated with commas

Example:

```
John,Doe,120 any st.,"Anytown, WW",08123
```

Leading and trailing space-characters adjacent to comma field separators are ignored

So: John , Doe ,... resolves to "John" and "Doe", etc. Space characters can be spaces, or tabs.

Fields with embedded commas must be delimited with double-quote characters

In the above example: "Anytown, WW" had to be delimited in double quotes because it had an embedded comma.

Fields that contain double-quote characters must be surrounded by double-quotes, and the embedded double-quotes must each be represented by a pair of consecutive double quotes

So, John "Da Man" Doe would convert to "John ""Da Man""",Doe

A field that contains embedded line-breaks must be surrounded by double-quotes

So:

```
Field 1:     Conference room 1
Field 2:     John,
                    Please bring the M. Mathers file for review
                    -J.L.
Field 3:     10/18/2002
```

Would convert to:

```
Conference room 1, "John,
Please bring the M. Mathers file for review
-J.L.
",10/18/2002
```

Note that this is a *single* CSV record, even though it takes up more than one line in the CSV file. This works because the line breaks are embedded inside the double quotes of the field.

Note: *In Excel, leading spaces between the comma used for a field separator and the double quote will sometimes cause fields to be read in as unquoted fields, even though the first non-space character is a double quote. To avoid this quirk, simply remove all leading spaces after the field-separator comma and before the double quote character in your CSV export files.*

Note: *It is not the role of the CSV parser to determine the end of a record. Its role is to parse the fields within a record. It expects to receive one complete record and ignores line break characters contained within the record.*

Fields with leading or trailing spaces must be delimited with double-quote characters

So to preserve the leading and trailing spaces around the last name above:

```
John ,"  Doe  ",...
```

Note: *Some applications will insist on helping you by removing leading and trailing spaces from all fields regardless of whether the CSV used quotes to preserve them. They may also insist on removing leading zeros from all fields regardless of whether you need them. One such application is Excel.*

Fields may always be delimited with double quotes

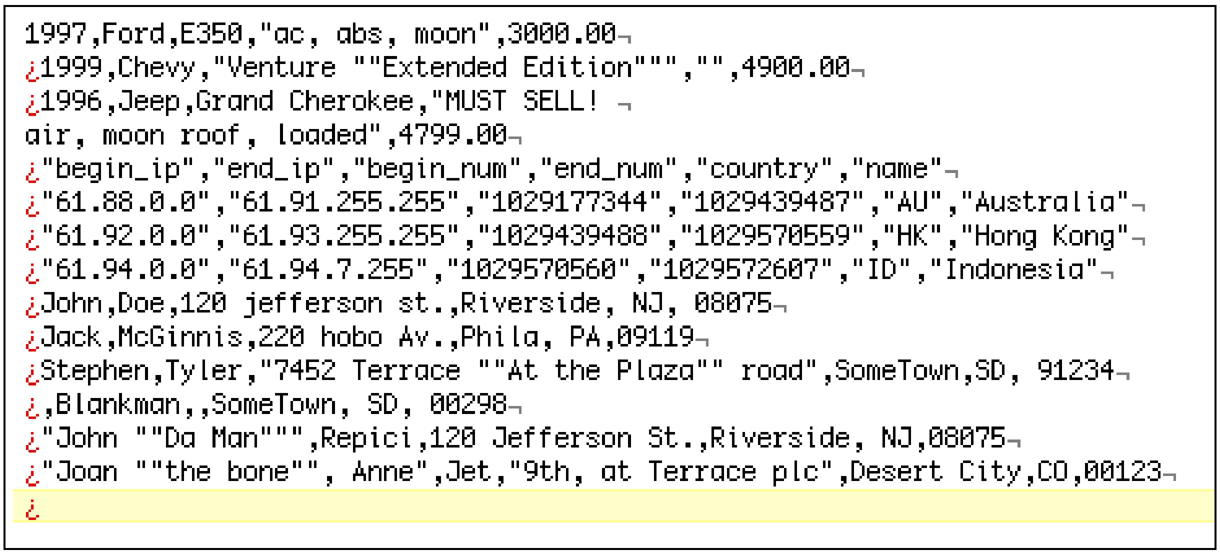
The delimiters will always be discarded.

The first record in a CSV file may be a header record containing column (field) names

There is no mechanism for automatically discerning if the first record is a header row, so in the general case, this will have to be provided by an outside process (such as prompting the user). The header row is encoded just like any other CSV record in accordance with the rules above. A header row for the multi-line example above, might be:

```
Location, Notes, "Start Date", ...
```

The image shown below should give you a visual appreciation for the valid variances described above:



```
1997,Ford,E350,"ac, abs, moon",3000.00¬
¿1999,Chevy,"Venture ""Extended Edition""","",4900.00¬
¿1996,Jeep,Grand Cherokee,"MUST SELL! ¬
air, moon roof, loaded",4799.00¬
¿"begin_ip","end_ip","begin_num","end_num","country","name"¬
¿"61.88.0.0","61.91.255.255","1029177344","1029439487","AU","Australia"¬
¿"61.92.0.0","61.93.255.255","1029439488","1029570559","HK","Hong Kong"¬
¿"61.94.0.0","61.94.7.255","1029570560","1029572607","ID","Indonesia"¬
¿John,Doe,120 jefferson st.,Riverside, NJ, 08075¬
¿Jack,McGinnis,220 hobo Av.,Phila, PA,09119¬
¿Stephen,Tyler,"7452 Terrace ""At the Plaza"" road",SomeTown,SD, 91234¬
¿,Blankman,,SomeTown, SD, 00298¬
¿"John ""Da Man""",Repici,120 Jefferson St.,Riverside, NJ,08075¬
¿"Joan ""the bone"", Anne",Jet,"9th, at Terrace plc",Desert City,CO,00123¬
¿
```

It shows numerous variations of strings that fully meet the CSV definitions shown above. The inverted question mark (¿) represents the new line character, ascii 0x0A. Study it closely, you will notice such things as quotes inserted inside quoted fields, a record on more than one line, empty fields, quoted fields containing commas.

How Does 4D's Match regex Command Solve the Problem?

The **Match regex** command can be used to check the conformity of a character string with respect to a set of synthesized rules by means of a meta-language called “regular expression” or “rational expression.” The “regex” abbreviation is commonly used to indicate these types of notations.

The key to parsing CSV is that the protocol produces a defined pattern, fields in a record separated by commas. Regex is all about matching patterns in a text string. If you can define the pattern, in the language of Regex, it can find it.

Using 4D's **Match regex** a CSV record can be defined to contain three possible regex patterns. Those three patterns are:

- a string of characters beginning and ending with quote marks, optionally followed by a comma
- a string of characters not beginning and ending with quote marks, optionally followed by a comma
- an empty field also optionally followed by a comma.

Using **Match regex** reduces the number of lines of string manipulation code. Determining where a CSV field begins and ends without using **Match regex** is a real coding exercise in 4D. A comprehensive solution will typically take a dozen or more lines of code. After defining the search pattern one time, the job of identifying the starting point and length of the field is reduced to one line of code using **Match regex**.

Introduction to, Listing, and Explanation of the 4D Code

There are numerous adaptations on the internet of a Java class called CSVRE from Chapter 7 of *Mastering Regular Expressions* (p. 205, first ed.) The CSVRE method shown below uses a simplified pattern from that class.

The simplification in this TN version of CSVRE is that it will not attempt to account for back-to-back double-quotes ("") contained within a quoted string in the regex pattern. Instead the method replaces all occurrences of "" with a unique string that can be easily replaced after a **Match regex** success with a single double-quote.

As mentioned before, there will be three alternations of patterns in the pattern string:

- The first pattern (\"([^\"]+?)\",?) looks for fields beginning with a quote and ending with a quote followed by an optional comma.
- The second pattern (([^,]+),?) looks for an unquoted field and ending with an optional comma.
- The third pattern (,) looks for empty fields.

A close study of the patterns will note that the leading comma is not in the pattern. That is because **Match regex** is always given the position of what would be the first character following a delimiting comma that starts a field, $Ndx:=$POS+$LEN.

The method is meant to be called from within a parsing loop. Here is an example of a 4D method used to parse a CSV file:

```
C_LONGINT($Ndx;$SOA;$RIS)
C_TIME($DocRef_H)
C_STRING(2;$RecDelim_A2)
C_TEXT($CSV_Rcd_T)

 `===================== Initialize and Setup ==============================

$DocRef_H:=Open document("CSV_Test.txt")

 `======================= Method Actions =================================

If (OK=1)
   $RecDelim_A2:="\r\n"
   $RIS:=0
   ARRAY TEXT($CSV_Flds_aT;0)
   RECEIVE PACKET($DocRef_H;$CSV_Rcd_T;$RecDelim_A2)
   While (OK=1)
         `//  If the first record in a header, handle the exception here
         //
         $RIS:=$RIS+1

         `// $Ndx is the number of fields returned
         $Ndx:=CSVRE ($CSV_Rcd_T;->$CSV_Flds_aT)
         If ($Ndx>0)
               `//  Now save the returned fields into arrays or fields
               `//
               `<  Do Something with returned fields here >

         End if
         `//  Get the next record. If the end of the file is reached,
         `//  OK will be set to zero
         `//
         RECEIVE PACKET($DocRef_H;$CSV_Rcd_T;$RecDelim_A2)

   End while

   `======================= Clean up and Exit ==========================

   CLOSE DOCUMENT($DocRef_H)

End if
```

Note that the CSVRE method is passed a string containing a single record or row and it parses each field into a text array.

The method expects two arguments, $1 is the CSV string to be parsed and $2 is a pointer to a text or string array to receive the parsed fields.

The method has a number of "defensive" coding techniques included. It first assures a good parameter count, the second checks to make sure a pointer to a text or string array was passed, the third is a test for an empty string string before a test.

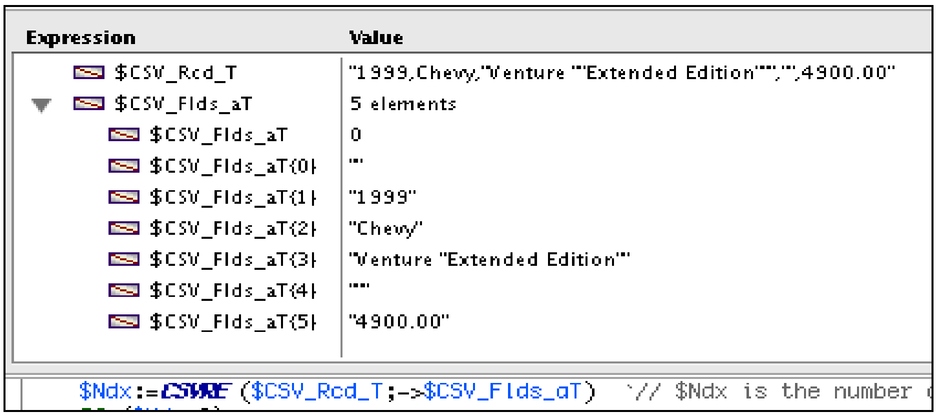
Here is the CSVRE method:

```
 `===================== Declare Variables =================================
     `method_parameters_declarations
C_TEXT($CSV_Rcd_T;$1)
C_POINTER($CSV_Flds_atP;$2)
  `-----------------------------------------------------------------------
  `method_wide_constants_declarations
  `-----------------------------------------------------------------------
  `local_variable_declarations
C_LONGINT($Ndx;$SOA;$RIS;$POS;$LEN)
C_TEXT($CSV_Pattern_T)
C_BOOLEAN($Break_B)
 `====================== Initialize and Setup ==============================
If (Count parameters>1)
   $CSV_Rcd_T:=$1
   $Ndx:=Type($2->)
   If (($Ndx=Text array ) | ($Ndx=String array ))
         $CSV_Flds_atP:=$2
         $RIS:=50
         ARRAY TEXT($CSV_Flds_aT;$RIS)

         $CSV_Pattern_T:="\"([^\"]+?)\",?|([^,]+),?|,"
         $Ndx:=1
         $SOA:=0
         $Break_B:=False
 `======================== Method Actions ===============================
         Repeat
               $CSV_Rcd_T:=Replace string($CSV_Rcd_T;"\"\"";"..||..")
               If (Match regex($CSV_Pattern_T;$CSV_Rcd_T;$Ndx;$POS;$LEN))
                     $Ndx:=$POS+$LEN
                     $SOA:=$SOA+1
                     ...
                     $CSV_Flds_aT{$SOA}:=Substring($CSV_Rcd_T;$POS;$LEN)
                     If ($LEN>0)
                           `//  If the ending is a comma, delete it
                           ...
                           `//  If the leading char is a “, delete it
                           `//
                           ...
                           `//  If the ending char is a “, delete it
                           `//
                           ...
                           `//  Replace all occurrences of unique string
                           `//  with a single “.
                           ...
                           `//  Trim any leading or trailing white-space.
                           ...
                     End if
               Else
                     $Break_B:=True
               End if
         Until ($Break_B)
`======================= Clean up and Exit ==============================
         ...
         COPY ARRAY($CSV_Flds_aT;$CSV_Flds_atP->)
   Else
         ...
    End if
Else
   ...
End if
```

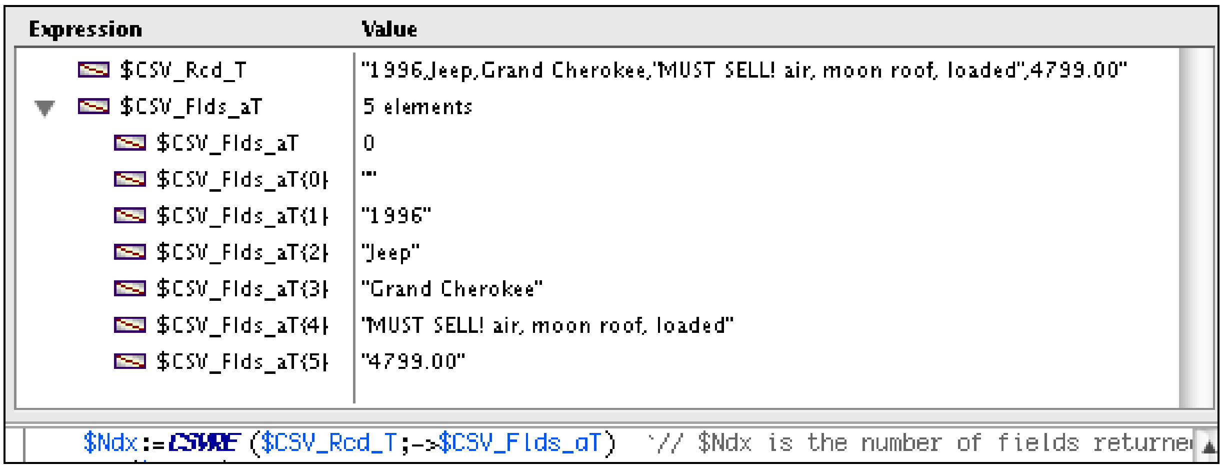
Note that the method uses standard 4D code to accomplish a number of the remaining cleanup chores of parsing a CSV record. It removes any trailing comma, removes any leading and trailing single quote, and removes any leading or trailing white space. In the included database you will note that in the method "StringTrim". It is a comprehensive method that cleans whitespace as defined in the ANSI Standard C **isspace** function (space, form feed, new line, carriage return, horizontal tab or vertical tab) plus the non-breaking space character (ascii 202), and not just the space character, ascii 32.

Below are a couple of screenshots of the CSVRE in action, from the sample database, in the debugger:



Expression	Value
$CSV_Rcd_T	"1999,Chevy,"Venture ""Extended Edition""","",4900.00"
$CSV_Flds_aT	5 elements
$CSV_Flds_aT	0
$CSV_Flds_aT{0}	""
$CSV_Flds_aT{1}	"1999"
$CSV_Flds_aT{2}	"Chevy"
$CSV_Flds_aT{3}	"Venture "Extended Edition""
$CSV_Flds_aT{4}	"""
$CSV_Flds_aT{5}	"4900.00"

```
$Ndx:=CSVRE ($CSV_Rcd_T;->$CSV_Flds_aT)  `// $Ndx is the number
```



Expression	Value
$CSV_Rcd_T	"1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00"
$CSV_Flds_aT	5 elements
$CSV_Flds_aT	0
$CSV_Flds_aT{0}	""
$CSV_Flds_aT{1}	"1996"
$CSV_Flds_aT{2}	"Jeep"
$CSV_Flds_aT{3}	"Grand Cherokee"
$CSV_Flds_aT{4}	"MUST SELL! air, moon roof, loaded"
$CSV_Flds_aT{5}	"4799.00"

```
$Ndx:=CSVRE ($CSV_Rcd_T;->$CSV_Flds_aT)  `// $Ndx is the number of fields returne
```

Conclusion

This technical note described the general challenge of importing a CSV file and a solution using the new **Match regex** command in 4D v11 SQL. This information should assist the 4D developer in the simplification of the task of importing a file that has been saved in the CSV format.

Related Resources

- Technical Note 07-47, *Introduction to Regular Expressions in 4D v11 SQL*
- ICU Users Guide:
  http://www.icu-project.org/userguide/regexp.html
- Regular-Expressions.info:
  http://www.regular-expressions.info/tutorial.html
- evolt.org - Regular Expression Basics:
  http://www.evolt.org/article/rating/20/22700/
- Regular expression:
  http://en.wikipedia.org/wiki/Regular_expression
- GREP: GREP is and acronym for Global Regular Expression Parsing (a utility from the UNIX world). There are many resources online for GREP that can help with understanding Regular Expresions.