Parsing Obnoxious Text Files

Whil Hentzen

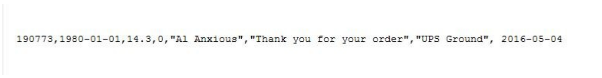
As we are increasingly asked to integrate our VFP apps with others systems, we can run into some obnoxious situations. One of those is a text file that while technically follows the specified format, still contains garbage that makes parsing harder.

Calvin Hsia demonstrated the improvements in VFP's amazing sting handling functions at one of the last Microsoft DevCons. "Want to see me parse 100,000 lines of text?" He hits a key, looks up, and asks, "Want to see me do it again?

Speed isn't all it takes, though.

Data dumps from some database that contain freeform text fields are one such beast. Lately I've been regularly receiving comma delimited dump that had additional carriage return/line feed combinations in the free form text fields, and it got me to scratching my noggin for a bit.

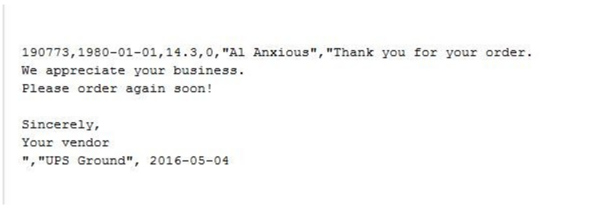
A normal comma delimited file looks like **Figure 1**.



```
190773,1980-01-01,14.3,0,"Al Anxious","Thank you for your order","UPS Ground", 2016-05-04
```

Figure 1. A regular comma delimited file.

A comma delimited file with garbage in the text fields looks like **Figure 2**.



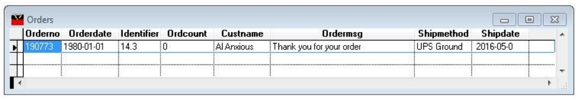
```
190773,1980-01-01,14.3,0,"Al Anxious","Thank you for your order.
We appreciate your business.
Please order again soon!

Sincerely,
Your vendor
","UPS Ground", 2016-05-04
```

Figure 2. A comma delimited file with CR/LF characters in a delimited text field.

Normally, you import a command delimited file with a standard

```
append from <filename> delimited
```

command, which results in a DBF that looks like **Figure 3**.



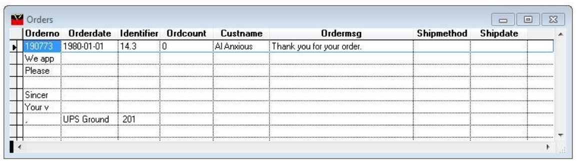
Orders

Orderno	Orderdate	Identifier	Ordcount	Custname	Ordermsg	Shipmethod	Shipdate
190773	1980-01-01	14.3	0	Al Anxious	Thank you for your order	UPS Ground	2016-05-0

Figure 3. A regular delimited file.

Nice and clean. The hardest work is making sure the fields in the text file match up with the fields in your DBF, as all too often, the provider of the data dump wasn't meticulous about defining the layout.

Importing the file with the extra garbage, however, produces more garbage, as shown in **Figure 4**.



Orders

Orderno	Orderdate	Identifier	Ordcount	Custname	Ordermsg	Shipmethod	Shipdate
190773	1980-01-01	14.3	0	Al Anxious	Thank you for your order.		
We app							
Please							
Sincer							
Your v							
,	UPS Ground	201					

Figure 4. An imported file with CR/LF.

Initial inclination was to replace each CRLF with a diff char, but it wouldn't work - the 'good' CR/LF at the end of each legitimate line would also be identified and replaced.

I considered trying to count fields, but that seemed prone to error.

Then I realized that the errant CR/LF were always in strings surrounded by quotes. So I walked through the record a char at a time, counting the number of quotes, and replaced the CR or LF with a space ONLY if the # of quotes that had been encountered was odd (and thus in the middle of a field.)

```
* parse.prg
* parse order file, remove all CR or LF
* inside double quotes

local x, lil, i, thisChar, numDQ

x=filetostr('garbage.csv')
lil = len(x)
numDQ = 0
for i = 1 to lil
  * update wait window once in a while
  if mod(i,10000) = 0
  wait window nowait lil-i
  endif
  thisChar = substr(x,i,1)
  if asc(thisChar) = 34 && Double Quote
    * determine if we're inside a string
    if numDQ = 0
    * we are now inside the DQ string
    numDQ = 1
    else
    * we are finishing up with a DQ string
    numDQ = 0
    endif
  else
    * if numDQ = 1, we are inside a string,
    * so check if this char is 10/13,
    *   if we're inside,
    *     replace with a space (32)
    *   if we're not inside, do nothing
    if numDQ=1
      if asc(thisChar) = 10 ;
        or asc(thisChar) = 13
        x=stuff(x,i,1,chr(32))
      else
        * this char isn't a CR or LF, so
        * don't do anything
      endif
    else
      * not inside, so
      * don't do anything
    endif
  endif
next
strtofile(x,'garbageNOT.csv')
```

If the CR/LF characters need to be maintained, they could be replaced by character strings more easily identifiable than simple spaces. For example, the “<br>” string would be logical and easy to find. Once a field containing CR/LF placeholders was imported into a DBF's memo field, the <br> strings could be replaced with the original CR/LF characters.

Author Profile

Whil Hentzen is an independent software developer based in Milwaukee, Wisconsin (as opposed to Milwaukee, Minnesota, as many people think.) His writing has killed many trees over the years, but none since 2007. He has realized he really sort of misses it. You can reach him at whil@whilhentzen.com