December, 2003

Advisor Answers

Appending memo fields

VFP 8/7/6

Q: I am trying to append data from a .csv file into a VFP free table using VFP 6 SP5 as follows:

```
APPEND FROM (lcImportFileName) TYPE CSV
```

The data is imported correctly except a comment field of indefinite length that I would like brought into a memo field in my table. The memo field is just left blank and the comment field in the .csv file is ignored. Can you help?

–Brian McCashin (via Advisor.COM)

A: When you're moving data from one table to another, APPEND FROM and COPY TO handle memo fields with no problems. But when you're working with text data (CSV or any of the other text types), memo fields aren't supported. The Help topic for APPEND FROM doesn't address this issue, though help for COPY TO does mention it.

If you can't use APPEND FROM, how can you read this data? Basically, all the solutions come down to reading in the file, parsing it, and storing the data. VFP offers a number of choices for reading text files and parsing them. You could use the low-level file functions to read the data or throw the entire file into a memo field and then use MLINE() to read it, but I prefer the FILETOSTR() function introduced in VFP 6. That function lets you read the entire contents of a file into a single variable.

Once you've read the data in, you have several options for breaking it into records and fields. My favorite way to parse data is the ALINES() function, which breaks data up into lines, putting each in an array element. (Starting in VFP 7, you can specify the line separator characters, which means you can parse pretty much anything with ALINES().) ALINES() has one serious limitation—since it fills an array, it only works if there no more than 65,000 lines in the data being parsed. (On the good news front, Microsoft has already announced that the next version of VFP, likely to ship late in 2004, will lift the 65,000 element limit for arrays.)

A CSV file contains a list of fields in the first row. That list is useful for converting data to the right type, as well as to know which fields we have data for. The list of fields is comma-separated, so in VFP 7 and later, ALINES() can parse it directly. In VFP 6, you need to replace the commas with CHR(13)+CHR(10) before applying ALINES().

After the first row, each row in a CSV file contains the data for one record, with fields separated by commas. Character and memo fields are surrounded with quotation marks. Here's an example:

```
"abc","Here's some text.",123.45,987654,12/31/2002,01/01/2002
00:00:00,37.0000,19,"abcdef",T
```

Note that date, datetime and logical fields have no delimiters surrounding them, as they do when you write them as constants in VFP. (That is, dates and datetimes are normally surrounded by curly braces, and logical values have periods, like .T., but in CSV files, there are no curly braces or periods around such data.)

The main challenge in parsing this sort of data is converting values to the right type so they can be inserted into the appropriate field. I'm assuming that since you were using APPEND FROM, the table you want to put the data in already exists. Since the CSV file contains the field names, it's easy to look up each field and convert the data from character to the appropriate type.

There's one other issue with dates. You need to make sure that SET DATE is the same when processing the date as when it was created.

Here's the function I wrote to do the whole task. It accepts two parameters, the name of the CSV file and the alias for an open table or cursor in which to put the data. This function is included on this month's Professional Resource Disk as AppendCSV.PRG.

```
* FUNCTION AppendCSV
* Append data in CSV format to specified work area,
* including memo field.
* Limitations:
* 1) This code does NOT support General fields and will
* not work properly if the specified table/cursor has
* General fields anywhere other than the end of each
* record.
* 2) This code will fail if the number of records to
* be appended is greater than 65,000.
LPARAMETERS cCSVFile, cAlias
   * cCSVFile = the file containing the data
   * cAlias = the alias where the data should be placed

LOCAL nOldSelect, nOldStrict
```

```
LOCAL cInputValue, aRecords[1], cFldList, cValList
LOCAL nFieldCount, nField, nRow, cString
LOCAL nCommaPos, nOccur

nOldSelect = SELECT()
SELECT (cAlias)

nOldStrict=SET("Strictdate")
SET STRICTDATE TO 0

cInputValue=FILETOSTR("test.txt")
ALINES(aRecords,cInputValue)

cFldList = aRecords[1]
cValList = ""
IF VERSION(5)>=700
   nFieldCount = ALINES(aFieldlist, cFldList,",")
ELSE
   * For VFP 6 pre-process the line
   LOCAL cFldListLines
   cFldListLines = STRTRAN(cFldList,",",CHR(13)+CHR(10))
   nFieldCount = ALINES(aFieldList, cFldListLines)
ENDIF

* Build expressions to convert each field
FOR nField = 1 TO nFieldCount
   DO CASE
   CASE INLIST(TYPE(FIELD(nField)), ;
               "C", "N", "I", "Y", "B")
      cValList = cValList + "eval(aDataValues[ " + ;
                 TRANSFORM(nField) + "]), "
   CASE TYPE(FIELD(nField))="M"
      cValList = cValList + "SUBSTR(aDataValues[ " + ;
                 TRANSFORM(nField) + "], 2, ;
                 LEN(aDataValues[ " + ;
                 TRANSFORM(nField) + "])-2), "
   CASE INLIST(TYPE(FIELD(nField)), "D", "T")
      cValList = cValList + "eval('{' + aDataValues[ " ;
                 + TRANSFORM(nField) + "]+'}'), "
   CASE INLIST(TYPE(FIELD(nField)),"L")
      cValList = cValList + "eval('.' + aDataValues[ " ;
                 + TRANSFORM(nField) + "]+'.'), "
   ENDCASE
ENDFOR

* Drop last comma
cValList = LEFT(cValList, LEN(cValList)-2)

FOR nRow = 2 TO ALEN(aRecords,1)
   IF OCCURS(",",aRecords[nRow]) > nFieldCount-1
      * Embedded commas so parse this one by hand
      cString = aRecords[nRow]
      nCommaPos = AT(",", cString)
      nOccur = 1
      nField = 1
```

```
      DO WHILE nCommaPos<>0
         IF MOD(OCCURS('"',LEFT(cString,nCommaPos-1)),;
                2)=0
            aDataValues[nField] = ;
                LEFT(cString,nCommaPos-1)
            nField = nField + 1
            cString = SUBSTR(cString, nCommaPos+1)
            nCommaPos = AT(",",cString)
            nOccur=1
         ELSE
            nOccur = nOccur + 1
            nCommaPos = AT(",", cString, nOccur)
         ENDIF
      ENDDO
   ELSE
      * No embedded commas, so just parse with ALINES()
      IF VERSION(5)>=700
         ALINES(aDataValues,aRecords[nRow],",")
      ELSE
         * Preprocess
         cRecord = ;
           STRTRAN(aRecords[nRow], ",", CHR(13)+CHR(10))
         ALINES(aDataValues, cRecord)
      ENDIF
   ENDIF

   INSERT INTO (cAlias) (&cFldList) ;
      VALUES (&cValList)

ENDFOR

* Clean up
SELECT (nOldSelect)
SET STRICTDATE TO nOldStrict

RETURN
```

The basic plan for this function is to read in the file, break it into lines and process one line (record) at a time. Each line is broken up into individual data items, and then inserted into the table.

There are two complex sections. The first of them (the FOR nField loop) parses the list of field names and builds the expressions needed to convert the data for insertion.

The other complex code (the IF-ELSE inside the FOR nRow loop) breaks a line up into values for the individual fields. If the data doesn't contain any commas, this is simple; just use ALINES(). However, if there are any commas in the record (that is, in a character or memo field), ALINES() won't break it up correctly. In that case, you have to pull out one field at a time, using a variety of cues to figure out where one field ends and the next begins.

It would be handy (and probably faster) if APPEND FROM could import memo data. Since it can't, it's good to know we have the tools available to do the job.

–Tamar