Indexed Random Access in Native LB

-

Alincon2001

I have always been disappointed with random files as implemented in Liberty Basic. It's true that they can be read and written randomly, and the LB fields statement does allow easy access to data fields. However it is not easy to determine which record is in what position – there is no built-in indexing capability – no built-in way to retrieve records based on key data fields.

There are work-arounds such as reading all records of a file into an array and then searching the array for key data, and the number of the corresponding file record to be retrieved.

Deleting records is another problem. Rewriting the entire file every time one record is deleted is cumbersome and time-consuming. So is keeping track of 'dead' records for re-use by new data.

There are various data-base programs that can be linked to Liberty Basic programs, but the ones I've seen require a lot of calls and a lot of coding, and they can be pricey.

I have devised a way to read files randomly in Liberty Basic without a lot of coding or calls to other software. My plan also includes the ability to easily delete records.

Simply put, files are treated as records. Each file holds only one record, and the file name is made up of the contents of one or two key data fields. The Liberty Basic 'filedialog' command is then used to present a list of the record/files. Just click on any file name to access its record/file independently of all the other record/files. Deleting any record/file can be done right from the file selector, or a user can choose a delete option and the program can delete the record/file using the Liberty Basic 'kill' statement.

Note that the file selector allows *user* sorting on date; allowing more control of file access. A third way *for users* to sort the file selector display is to use three characters of a third data field as the file extension, and click on the 'type' column to present the record/file list in that order.

This record-as-file technique is more viable for smaller amounts of data: up to what would be, in standard usage of the terms, a few hundred 'records' in a single 'file'

The file handling for record-as-file is the same as normal random-access file-handling, except that the record number is always '1'

I was interested to learn in writing demonstration programs for this technique that data can be moved to the fields in the LB 'field' statement while the file is closed. *editor's note: the meaning of this statement is unclear*

To read all of the record/files and produce reports, the Liberty Basic 'files' statement is used. The number of files and their names are easily found. It is then just a matter of looping through 'Open, File, Get statements in a report routine. One additional requirement is a file-open flag because data can not be moved from the fields in the LB 'field' statement if the file is closed.

```
[getFileData]
     dim info$(10, 10)
    files path$, info$(              'get file information
    fc$ = info$(0, 0): fc = val(fc$)   'file count
    return

 [readNaFiles]
    if naFileOpen = 1 then close #naFile
    fileName$=path$+info$(n, 0)
    open  fileName$ for random as #naFile len = 170
    field #naFile,  20 as lastName$, 20 as firstName$, 30 as address$,
 20 as city$, 2 as state$, _
                   9 as zip$, 10 as phone$, 29 as email$, 30 as notes
$
    gettrim #naFile, 1
    naFileOpen = 1
   return
```

The record-as-file technique also allows random access to sequential files! One method is much the same as for random files. Each file still contains only one record, but access to the data fields requires the use of left$ and mid$ statements, or using the Word$ statement, if the data fields are separated with a character such as “|”.

Reading the record/file:

```
filedialog "Choose file", path$, fileName$
 if fileName$ = "" then [exitRead]
open fileName$ for input as #1
 line input #1, naRec$
 close #1
```

Writing the record/file:

```
fileName$=trim$(lastName$)+"_"+trim$(firstName$)+"."+state$
 open fileName$ for output as #7
 print #7,naRec$
 close #7
```

The second method is to carry the record-as-file idea to the next step and use fields as records. In this method, each file actually contains multiple records, but each record holds only the data from one ‘field’, such as last name or address.

To access fields-as-records, each call to input or output data actually has multiple reads or writes – one for each field – but each 'file' still has data for only one entity, and can be accessed via the file selector as described for random files.

Reading:

```
filedialog "Choose file", "c:\libbas\_mylibbas\nameadrs\text2\*.*", fi
leName$
    if fileName$ = "" then [mainLoop]
    open fileName$ for input as #1
    input #1,lastName$
    input #1,firstName$
    input #1,address$
    input #1,city$
    input #1,state$
    input #1,zip$
    input #1,phone$
    input #1,email$
    input #1,notes$
    close #1
```

Writing:

```
fileName$=path$ + trim$(lastName$)+"_"+trim$(firstName$)+"."+state$
    open fileName$ for output as #7
       #7,lastName$
       #7,firstName$
       #7,address$
       #7,city$
       #7,state$
       #7,zip$
       #7,phone$
       #7,email$
       #7,notes$
       close #7
```

To retrieve all data for reporting on either kind of sequential file, the files statement is used just as described for random files to get the number of files and their names, but no special read routine or file-open flag is needed.

Reading all record/files into a two-dimensional array:

```
files path$, info$(
    fc$ = info$(0, 0): fc = val(fc$)
    for cf = 1 to fc
        call readFile, cf
    next
    return

 sub readFile cf
    fileName$=path$+info$(cf, 0)
    open fileName$ for input as #1
        for n = 1 to 9
            input #1,x$
            naRec$(cf,n) = x$
        next n
    close #1
    end sub
```

Reading all records into an array for preparing a report

```
files path$, info$(
    fc$ = info$(0, 0) : fc = val(fc$)
    for cf = 1 to fc
        call readFile, cf
    next
    return

 sub readFile cf
    fileName$="text3\" + info$(cf, 0)
        print fileName$
    open fileName$ for input as #1
        line input #1, naRec$
        naRec$(cf) = naRec$
            print naRec$(cf)
    close #1
    end sub
```