Creating, Reading, and Appending Sequential Text Files

A sequential text file is a file where characters are placed in sequence, one right after the other. Sequential text files have no set format other than a beginning and an end. A sequential text file can only be read starting from the beginning of the file. Each character is printed as an ASCII text character. The most common example of a sequential text file is the .txt file saved with Notepad or any other editing tool. Run the following code to see the output of simple PRINT statements.

```
PRINT "Tom Sawyer"
PRINT "Huckleberry Finn"
PRINT "The Prince and the Pauper"
PRINT "A Connecticut Yankee in King Arthur's Court"
PRINT "A Tramp Abroad"
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

When the program is closed, the text is gone. A sequential text file allows that text to be saved to file.

Sequential File Commands and Operations

There are three basic operations in sequential text files.

1. Creating and writing data into a file
2. Reading data from an existing file
3. Appending or adding data into an existing file

Basic File commands and functions include

- OPEN- used to access the file
- OUTPUT - used to write data to the file
- INPUT - used to read data from the file
- LOF() - length of file
- EOF() - end of file
- CLOSE - used to close the file

Special File commands and functions include

- LINE INPUT - extract a full line of data
- INPUTTO$ - extract data up to the assigned deliminator
- INPUT$() - extract a number of bytes of data starting from the beginning

Creating a Sequential Text File

Sequential text files are always created in this fashion

```
OPEN "FileName.ext" for OUTPUT as #1
```

The commands OPEN and OUTPUT are case-insensitive. "FileName.ext" can be any name and extension you choose. The file name can also be a variable.

```
fName$ = "FileName.ext"
OPEN fName$ for OUTPUT as #1
```

#1 is also an arbitrary name. #a, #MyTextFile, #123xyz, or any other combination of numbers and letters will work as well.
Once the file has been opened, data can be written into that file. The data can be literal

```
PRINT #1, "Tom Sawyer"
```

or a variable

```
t$ = "Tom Sawyer"
PRINT #1, t$
```

Finally, when all the data has been written, the file must be closed.

```
CLOSE #1
```

The following code creates the text file "TwainNovels.txt"

```
OPEN "TwainNovels.txt" for OUTPUT as #1
    PRINT #1, "Tom Sawyer"
    PRINT #1, "Huckleberry Finn"
    PRINT #1, "The Prince and the Pauper"
    PRINT #1, "A Connecticut Yankee in King Arthur's Court"
    PRINT #1, "A Tramp Abroad"
CLOSE #1
END
```

Run the above code before proceeding with this tutorial.

Reading a Sequential Text File

Reading a sequential text file also requires the file be OPENed, but for INPUT.

```
OPEN "TwainNovels.txt" for INPUT as #1
    INPUT #1, n1$
    INPUT #1, n2$
    INPUT #1, n3$
    INPUT #1, n4$
    INPUT #1, n5$
CLOSE #1
    PRINT n1$
    PRINT n2$
    PRINT n3$
    PRINT n4$
    PRINT n5$
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

LOF - Length of File

The length, or size, of a sequential text file is determined by the number of characters it contains. Each character occupies one byte. That size is obtained with the LOF() function. The file must first be opened.

```
OPEN "TwainNovels.txt" for INPUT as #1
    Print "LOF = ";LOF(#1)
    INPUT #1, n1$
    INPUT #1, n2$
    INPUT #1, n3$
    INPUT #1, n4$
    INPUT #1, n5$
CLOSE #1
    nChars = 0
    PRINT n1$
    nChars = nChars + Len(n1$)
    PRINT n2$
    nChars = nChars + Len(n2$)
    PRINT n3$
    nChars = nChars + Len(n3$)
    PRINT n4$
    nChars = nChars + Len(n4$)
    PRINT n5$
    nChars = nChars + Len(n5$)
    Print "nChars = ";nChars
END
```

- Output

```
LOF = 118
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
nChars = 108
```

There is an inequality here. LOF equals 118 and nChars equals 108. Where are the other 10 characters?

End-of-Line Markers

Windows marks the end of each record with a carriage return / line feed sequence. At the end of each novel title are these two end-of-line marker characters: Chr$(13) and Chr$(10), two extra characters for each of the five lines. That's ten extra characters.
Sequential text files save all characters in their corresponding ASCII format and display as they do on the keyboard. The letter 'a' is seen as 'a', the number 1 is seen as '1'. Some characters, Control Key, Escape, Enter, to name a few, are invisible characters. They are imbedded in the sequential text file but cannot be seen. The LOF() function incudes these characters when counting the size of the file. The INPUTed variables do not include these end-of-line markers.

EOF - End of File

Liberty BASIC uses a special End of File function, EOF(), to determine if the End of File has been reached. If the End of File has been reached, EOF returns 1 (True). If the End of File has not been reached, EOF returns 0 (False). EOF is useful for retrieving information when the file contains an unknown amount of data.

```
OPEN "TwainNovels.txt" for INPUT as #1
    WHILE EOF(#1) = 0
        INPUT #1, n$
        PRINT n$
    WEND
CLOSE #1
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

WHILE EOF(#1) = 0 translates to *While the End of File is false (hasn't been reached)*

This could also be coded as

```
OPEN "TwainNovels.txt" for INPUT as #1
```

```
    WHILE EOF(#1) <> 1
        INPUT #1, n$
        PRINT n$
    WEND
CLOSE #1
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

WHILE EOF(#1) 1 translates to *While the End of File is not true (hasn't been reached)*

A third way is to use the NOT() operator:
WHILE NOT(EOF(#1)) translates to *While not at the End of File*

In either case, the code continues to input data until the End of File has been reached.

Counting the Data

Sometimes the number of data needs to be known. This is especially true if an array is used to capture the data. If a sequential text file contains an unknown number of data, use a counter to determine that exact number. Increase the counter by one with each data read.

```
ct = 0
OPEN "TwainNovels.txt" for INPUT as #1
   DO
        ct = ct + 1
        INPUT #1, n$
        PRINT ct, n$
    LOOP WHILE EOF(#1) = 0
CLOSE #1
PRINT "There are ";ct;" items of data in this text file."
END
```

- Output

```
1             Tom Sawyer
2             Huckleberry Finn
```

```
3             The Prince and the Pauper
4             A Connecticut Yankee in King Arthur's Court
5             A Tramp Abroad
There are 5 items of data in this text file.
```

Once the number of data is known, DIM the array, OPEN the file again, and INPUT the data into the array. Be sure to CLOSE the file first so the INPUT begans with the first data.

```
nNovels = 5 ' The number obtained by the counter
Dim TwainNovel$(nNovels)
OPEN "TwainNovels.txt" for INPUT as #1
    FOR i = 1 to nNovels
        INPUT #1, TwainNovel$(i)
    NEXT i
CLOSE #1
FOR i = 1 to nNovels
    PRINT TwainNovel$(i)
NEXT i
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

Recapping

To read a known number of data from a sequential file

- OPEN the file
- use a FOR NEXT loop to INPUT data
- CLOSE the file

To read an unknown number of data from a sequential text file

- OPEN the file
- use a WHILE WEND to INPUT data until the EOF is reached
- CLOSE the file

To count the number of data in a sequential text file

- OPEN the file
- use a counter
- use a DO LOOP to INPUT data until EOF is reached
- CLOSE the file
- OPEN the file again
- use a FOR NEXT loop to INPUT data
- CLOSE the file again

Comma Deliminators

It's not just End of Line markers that separate data; a comma will also separate data. Deliminator is another word for separator. Run this code to create a new file.

```
OPEN "SevenDwarves.txt" for OUTPUT as #1
    PRINT #1, "Happy, Sleepy, Bashful, Grumpy, Sneezey, Doc, Dopey"
CLOSE #1
END
```

Next, read the file.

```
OPEN "SevenDwarves.txt" for INPUT as #1
    INPUT #1, dwarf$
    PRINT dwarf$
CLOSE #1
END
```

- Output

```
Happy
```

Where are the rest of the dwarves? INPUT will only grab data up to, but not including, the comma. Use either a FOR NEXT loop if you know the number of data to INPUT

```
' Number of data is known
OPEN "SevenDwarves.txt" for INPUT as #1
    FOR i = 1 to 7
        INPUT #1, dwarf$
        PRINT dwarf$
    NEXT i
CLOSE #1
```

```
END
```

- Output

```
Happy
Sleepy
Bashful
Grumpy
Sneezey
Doc
Dopey
```

or use a WHILE WEND loop if you don't.

```
' Number of data unknown
OPEN "SevenDwarves.txt" for INPUT as #1
    WHILE EOF(#1) = 0
        INPUT #1, dwarf$
        PRINT dwarf$
    WEND
CLOSE #1
END
```

- Output

```
Happy
Sleepy
Bashful
Grumpy
Sneezey
Doc
Dopey
```

It isn't always desirable to deliminate data with commas. To place the entire line of data, including any commas, in a variable, use LINE INPUT.

Line Input

LINE INPUT will extract the full line of data, up to the End of Line markers. LINE INPUT ignores commas as deliminators.

```
OPEN "SevenDwarves.txt" for INPUT as #1
```

```
    LINE INPUT #1, dwarf$
    PRINT dwarf$
CLOSE #1
END
```

- Output

```
Happy, Sleepy, Bashful, Grumpy, Sneezey, Doc, Dopey
```

Assigning a Unique Deliminator

Actually, any character can be assigned as the deliminator. The assigned deliminator is substituted for the usual comma deliminator. The command for assigning a unique deliminator is INPUTTO$(). The following code separates the SevenDwarves data at the *r*.

```
OPEN "SevenDwarves.txt" for INPUT as #1
    For i = 1 to 2
        n$ = INPUTTO$(#1, "r")
        PRINT n$
    Next i
CLOSE #1
END
```

- Output

```
Happy, Sleepy, Bashful, G
umpy, Sneezey, Doc, Dopey
```

The unique deliminator, in this case the *r*, is lost, just as the comma is lost with INPUT. Note, also, that assigning a unique deliminator does not prevent the usual End of Line marker, Chr$(13);Chr$(10), from forcing a new line.

```
OPEN "TwainNovels.txt" for INPUT as #1
    WHILE EOF(#1) = 0
        n$ = INPUTTO$(#1, "r")
        PRINT n$
    WEND
CLOSE #1
END
```

- Output

```
Tom Sawye

Hucklebe

y Finn
The P
ince and the Paupe

A Connecticut Yankee in King A
thu
's Cou
t
A T
amp Ab
oad
```

The blank lines are the result of the Chr$(13);Chr$(10) End of Line markers.

Placing the Entire Contents of the File into One String Variable

Partial or full contents of the file can be read using the INPUT$() function. Contents of the file, from the beginning to a defined number of bytes can be placed in a string variable.

```
OPEN "TwainNovels.txt" for INPUT as #1
    n$ = INPUT$(#1, 15)
CLOSE #1
PRINT n$
END
```

- Output

```
Tom Sawyer
Huc
```

Only 13 characters are visible. The other 2 bytes are the invisible characters, Chr$(13) and Chr$(10), that force a new line at the end of Tom Sawyer.

```
OPEN "TwainNovels.txt" for INPUT as #1
    n$ = INPUT$(#1, 15)
CLOSE #1
PRINT n$
FOR i = 1 to LEN(n$)
```

```
    PRINT i, MID$(n$, i, 1), ASC(MID$(n$, i, 1))
NEXT i
END
```

- Output

```
Tom Sawyer
Huc
1               T               84
2               o               111
3               m               109
4                               32
5               S               83
6               a               97
7               w               119
8               y               121
9               e               101
10              r               114
11                              13
12                              10
13              H               72
14              u               117
15              c               99
```

The entire contents of the file may be read and stored in one string variable. This may be useful for text files containing narrative paragraphs, or for later parsing of unknown data. Whatever the reason, the entire contents is defined with the LOF() function.

```
OPEN "TwainNovels.txt" for INPUT as #1
    n$ = INPUT$(#1, LOF(#1))
CLOSE #1
PRINT n$
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

Recapping

- Read a simple sequential text file with the INPUT command
- Read an entire line of data, including commas, with the LINE INPUT command
- Read the entire file as one string variable with the INPUT$() and LOF() functions

Appending a Sequential Text File

PRINTing to a sequential text file that's been OPENed for OUTPUT will not place the new text at the end of the file. It will, in fact, overwrite the file.

```
OPEN "TwainNovels.txt" for OUTPUT as #1
    PRINT #1, "The Guilded Age"
CLOSE #1
OPEN "TwainNovels.txt" for INPUT as #1
    n$ = INPUT$(#1, LOF(#1))
CLOSE #1
PRINT n$
END
```

- Ouput

```
The Guilded Age
```

"The Guilded Age" was not added to the contents of TwainNovels.txt, but, instead, replaced the contents of TwainNovels.txt. To add information to an existing sequential text file, use the APPEND command. First, rebuild the original TwainNovels.txt file.

```
OPEN "TwainNovels.txt" for OUTPUT as #1
    PRINT #1, "Tom Sawyer"
    PRINT #1, "Huckleberry Finn"
    PRINT #1, "The Prince and the Pauper"
    PRINT #1, "A Connecticut Yankee in King Arthur's Court"
    PRINT #1, "A Tramp Abroad"
CLOSE #1
```

Read the contents of the file.

```
OPEN "TwainNovels.txt" for INPUT as #1
    FOR i = 1 to 5
        INPUT #1, n$
        PRINT n$
    NEXT i
CLOSE #1
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
```

Now, OPEN the file for APPEND to add more content.

```
OPEN "TwainNovels.txt" for APPEND as #1
    PRINT #1, "The Guilded Age"
CLOSE #1
OPEN "TwainNovels.txt" for INPUT as #1
    FOR i = 1 to 6
        INPUT #1, n$
        PRINT n$
    NEXT i
CLOSE #1
END
```

- Output

```
Tom Sawyer
Huckleberry Finn
The Prince and the Pauper
A Connecticut Yankee in King Arthur's Court
A Tramp Abroad
The Guilded Age
```

Recapping

- To create a new sequential text file, use OPEN for OUTPUT
- To read a sequential text file, use OPEN for INPUT
- To add content to an existing text file, use OPEN for APPEND

Common Errors with Sequential Text File Programming

Liberty BASIC will halt execution with an error message if any problems are encountered during opening, reading, and writing sequential text files. The following code will present no error if TwainNovels.txt remains present in the relevant Liberty BASIC directory.

```
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovels.txt" for INPUT as #1
        FOR i = 1 to 5
            INPUT #1, n$
            PRINT n$
        NEXT i
    CLOSE #1
END

[ErrorDeBug]
    PRINT Err
END
```

Errors will occur if the file can't be found, can't be opened, can't be written to, or attempts are made to extract data that isn't there.

- File Doesn't Exist (Err = 62)

can happen when the file doesn't exist, when the file name is simply misspelled or when the path to that file is wrong. This is a common occurance when a path is *hardcoded* rather than *relevant*.

  - Example

```
'   TwainNovels is misspelled TwainNovel
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovel.txt" for INPUT as #1
        FOR i = 1 to 5
            INPUT #1, n$
            PRINT n$
        NEXT i
    CLOSE #1
END

[ErrorDeBug]
    PRINT Err
END
```

Liberty BASIC closes, giving the warning

```
Runtime Error: OS Error: The system cannot find the file specified.
(see error.log for more information)
```

- File Already Open (Err = 0)

occurs when an attempt is made to OPEN a file that is already OPEN.

- Example

```
'   TwainNovels.txt is opened twice
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovels.txt" for INPUT as #1
    OPEN "TwainNovels.txt" for INPUT as #1
        FOR i = 1 to 5
            INPUT #1, n$
            PRINT n$
        NEXT i
    CLOSE #1
END

[ErrorDeBug]
    PRINT Err
END
```

Liberty BASIC first gives a notice that the program has ended and all open windows and files have been closed

```
Please Note:
#1
These handles closed by Liberty BASIC.
Please add the appropriate CLOSE commands.
```

then a warning is given regarding the error.

```
Warning
Runtime Error: Handle #1 already in use
(see error.log for more information)
```

Note that there is not an Error number associated with this error.

- Input Past End of File (Err = 62)

occurs when attempts are made to extract more data than what the file holds.

- Example

```
'   Run out of data
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovels.txt" for INPUT as #1
        FOR i = 1 to 20
            INPUT #1, n$
            PRINT n$
        NEXT i
    CLOSE #1
END

[ErrorDeBug]
    PRINT Err
END
```

The usual Liberty BASIC notice that the program has ended and all open windows and files have been closed is given

```
Please Note:
#1
These handles closed by Liberty BASIC.
Please add the appropriate CLOSE commands.
```

followed by a warning regarding the error.

```
Warning
Runtime Error: Input past end of file: #1
(see error.log for more information)
```

- Attempting to INPUT data from a file OPENed for OUTPUT (Err = 62)

Liberty BASIC creates a new file whenever that file is OPENed for OUTPUT. Attempts to then INPUT data will result in the same error as attempting to extract more data than what the file holds.

    - Example

```
'   Input from file opened for Output
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovel.txt" for OUTPUT as #1
        FOR i = 1 to 5
            INPUT #1, n$
            PRINT n$
        NEXT i
    CLOSE #1
```

```
END

[ErrorDeBug]
    PRINT Err
END
```

Liberty BASIC will close all windows, files, and the program itself

```
Please Note:
#1
These handles closed by Liberty BASIC.
Please add the appropriate CLOSE commands.
```

and then give the error warning.

```
Warning
Runtime Error: Input past end of file: #1
(see error.log for more information)
```

- Writing to a File OPENed for INPUT (Err = 0)

results in a serious error, from which Liberty BASIC cannot recover. It is *NOT RECOMMENDED* that you run the following code, as you *WILL* have to use the *TASK MANAGER* to recover and end the program.

    - Example

```
    ON ERROR GOTO [ErrorDeBug]
    OPEN "TwainNovels.txt" for INPUT as #1
        PRINT #1, "A Guilded Age"
    CLOSE #1
END

[ErrorDeBug]
    PRINT Err
END
```

results in this warning

```
Runtime Error: FileSystemAccessDenied
(see error.log for more information)
```

Again, you *WILL* have to use the *TASK MANAGER* to recover from this error.

For more information on errors and debugging your program, see Fast FAQ.

Advantages and Disadvantages of a Sequential Text File

Sequential text files can be used to store any type of data. The structure is simple and the contents are easily obtainable. Sequential text files must be read starting from the beginning. As files increase in size, it may become time consuming to search for information toward the end of the file. Any change to the file necessitates a rewriting of the entire file to disk. Still, for small amounts of data, sequential text files offer the most economical option.

More on Files

An Introduction to Working with Files