# 2nd Prototype Design

**Side note: Rename operations do not count as modifying the file or folder itself, hence the last date modified is not updated in this instance**

## Folders

- Folders are modeled as headers within the EEPROM and are found at the start of the EEPROM
- Hierarchy is modeled as top-down and represented as follows: (E://folder1/folder2/file.txt), where **E://** represents the root directory (**EEPROM**)
- Every folder contains the following elements:

    - Folder name
    - Folder start address
    - Parent folder start address

- The start of a folder contains the header elements written contiguously in the EEPROM in the following order: name, start address, parent start address
- The header of a folder occupies 10 + 2 * 2 = **14 bytes**

## Files

- Files can be of variable length
- Files are plain text
- Each file is partitioned by a start and stop address within the EEPROM
- Files are stored at the end of the EEPROM
- Every file contains a header (not visible to user) with the following attributes:

    - File name
    - File start address
    - File end address
    - Parent folder start address

- The name of the file is restricted to 10 characters (10 bytes)
- The file writing procedure consists of the following:

    a. Compute the size of the text file (number of bytes in the input string).
    b. Set aside 2 bytes (a short int) each for the parent folder start address, parent folder stop address, start address, stop address, date created and date last modified.
    c. Set aside 10 bytes for the file name.
    d. Ask user for filename through serial.

e. Scan file system to see if filename already exists within **current directory**. If it does, ask user if they wish to overwrite file. If it doesn’t exist or if user says yes, continue as normal, otherwise abort.
f. If file doesn’t exist, set date created and date last modified to current date.
g. If file does exist, set date last modified to current date.
h. Compute file size (**16 bytes (for the header) + the # bytes of text data**)
i. Set start address to the next available address for the file (stop address of last file + 1) (if the last file is the file being overwritten, then the start address is the start address of this file (we will shift addresses if file size changes))
j. Write the file data (text) contiguously in EEPROM to the last byte of text
k. Write/update the header elements contiguously in the following order: name, date created, date last modified, start address, stop address, start address of parent folder, stop address of parent folder
l. Shift the addresses of all subsequent folders as needed

## <u>Partitions</u>

- The EEPROM will be partitioned to separate the file headers, folders and files themselves.
- **27,768 bytes** will be allocated to the file partition, **4,000** bytes will be allocated to the file header partition and **1,000** bytes will be allocated to the folder partition
- This will allow us to store a maximum of 71 folders and 250 file headers in all cases
- This means, if on average a file is composed of 300 bytes, we can store 92 files
- The system will be represented in memory as follows:

<u>addresses</u>

| 0 | 999 | 1,000 | 28,767 | 28,768 | 32,767 |
|---|---|---|---|---|---|
| Folder partition | | File partition | | File header partition | |

- Storing it this way makes the system more efficient, as searching the EEPROM for a particular file becomes much faster than not having partitions

## File System Operations

- Read
    - The file system will have to be initialized, along with the contents of the current directory displayed in the GUI
    - In order to read the contents of a selected file, the addresses corresponding to the current file will be accessed
    - The file contents will be displayed on the serial monitor
- Write
    - The text “NOW WRITING FILE” will appear on the serial monitor once the user selects the write option in the operation menu

    - Input will be taken in from the user as pure text
    - The user will press BTN 2 to indicate that they've finished inputting text
    - If the file size exceeds the max capacity of the file partition, or if the file partition cannot accommodate the new file, we prompt the user to call the Organize Memory function. We then check if there is enough free space to write the file. If not, we ask the user to delete old files first, then try again.
    - Writing will occur as previously described
- Organize Memory
    - We call this function once only when the max capacity of the file partition has been reached. This will shift over all the files to close any "free spaces" in between files.
    - We then update the file headers to accommodate for the changes to the file start and end addresses
- Copy/Paste
- Move
- Rename
- Delete
- Format system
    - Write 0's to all addresses of the EEPROM

**Orbit Booster Pack GUI**

Function
- The GUI will allow the user to navigate the file system

Controls
- The potentiometer will be used to scroll the contents of the current directory in order to view all the files and folders in the current level:
- BTN 2 will be used to select a file/folder - this will pop up the operation menu
- BTN 1 will be used to go the parent folder - at root, this won't do anything

Operation Menu
- The following options will be displayed if a folder has been selected:
    - Step into
    - Rename
    - Move
    - Copy
    - Delete

- Likewise, if a file is selected:
    - Read
    - Rename
    - Move
    - Copy

    - Delete

- If copy is selected, the next time the user presses the select button on a folder, the menu will have the following options:
    - Step into
    - Rename
    - Delete
    - Paste here

- Likewise, if a file is selected:
    - Read
    - Rename
    - Delete
    - Paste here

- If move is selected, the next time the user presses the select button on a folder, the menu will have the following options:
    - Step into
    - Rename
    - Delete
    - Move here

- Likewise, if a file is selected:
    - Read
    - Rename
    - Delete
    - Move here

- “Paste here” and “Move here” means pasting the file/folder in the current directory

<u>How it works</u>

- When file system is initialing, the contents of the root folder will be displayed. To go about this, the file header partition will be searched for all the files whose parent is the root folder. These files will be displayed on screen in order as they appear in memory. Next, the folder header partition will be scanned for all folders whose parent folder is the root folder. These folders will be displayed on screen in order as they appear in memory.
- The same will be done for all folders in lower levels
- The addresses of the selected files for the current level will be stored in an array
- The indices of the selected files will be stored in an array. These indices will correspond to the elements of the previous array.
- We will create an additional two arrays representing the same info for the folders
- The four arrays will be used to create a list of all elements in the current directory

- The elements of the directory will be populated according to the folder and file index & address arrays