1st 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

***We will use Epoch time to represent the date in 2 bytes**

Folders

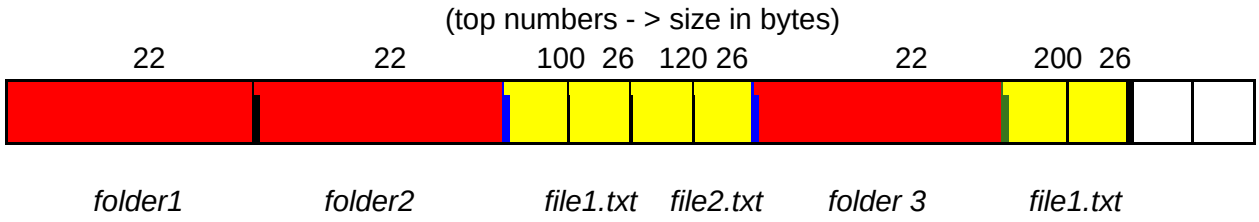
- Folders are essentially directory dividers
- Folders are modeled as partitions within the EEPROM (they contain smaller subpartitions -> files/subfolders)
- Hierarchy is modeled as top-down and are represented as follows: (E://folder1/folder2/file.txt), where **E://** represents the root directory (**EEPROM**)
- Every folder has a header containing the following elements:
    - Folder name
    - Date created
    - Start address
    - Stop address
- The start of a folder contains the header elements written contiguously in the EEPROM in the following order: name, date created, start address, stop address
- The header of a folder occupies 10 + 4 + 2 * 2 = **18 bytes**

Files

- Files can be of variable length
- Files are plain text
- Files are modeled as subpartitions within folder partitions in the EEPROM
- Each file is partitioned by a start and stop address within the EEPROM
- Every file contains a header (not visible to user) with the following elements:
    - Name
    - Date created
    - Date last modified
    - Start address
    - Stop address
- The name of the file is restricted to 10 characters (10 bytes)
- The file writing procedure will consist 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 start address, stop address, date created and date last modified.
c. Set aside 10 bytes for the file name.
d. Ask user for filename.
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 (**18 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 within current directory + 1, else stop address of current directory folder + 1) - (if the last file is the file being overwritten, then the start address is the start address of this file)
j. Write the file data (text) contiguously in EEPROM to the last byte of text
k. Write/updated the header elements contiguously in the following order: name, date created, date last modified, start address, stop address
l. Update the addresses of all affected elements (**TBD)**

Example system as seen in EEPROM:



- Thickest black delimits folder1, blue delimits folder 2 and green delimits folder 3
- Thick black delimits file-file boundaries
- Thin black delimits file header - file boundaries
- **Note: files can have same name if in different directories**
- Word description of system:
  - at root level (**E:*//***), only folder1 exists.
  - within folder1, folder2 and folder3 exist.
  - within folder2, file1.txt and file2.txt exist.
  - within folder3, file1.txt exists

Problem:

How do we implement folder delimiters?

Storage Ideas:

Create contiguous files and folder. Write folders to the lowest memory address available; write files to the highest memory address available. Assign a byte to each file which allows it to reference which folder(s) it's in. **Folders do not represent a range of available memory.** This obviates any excessive shifting of data.