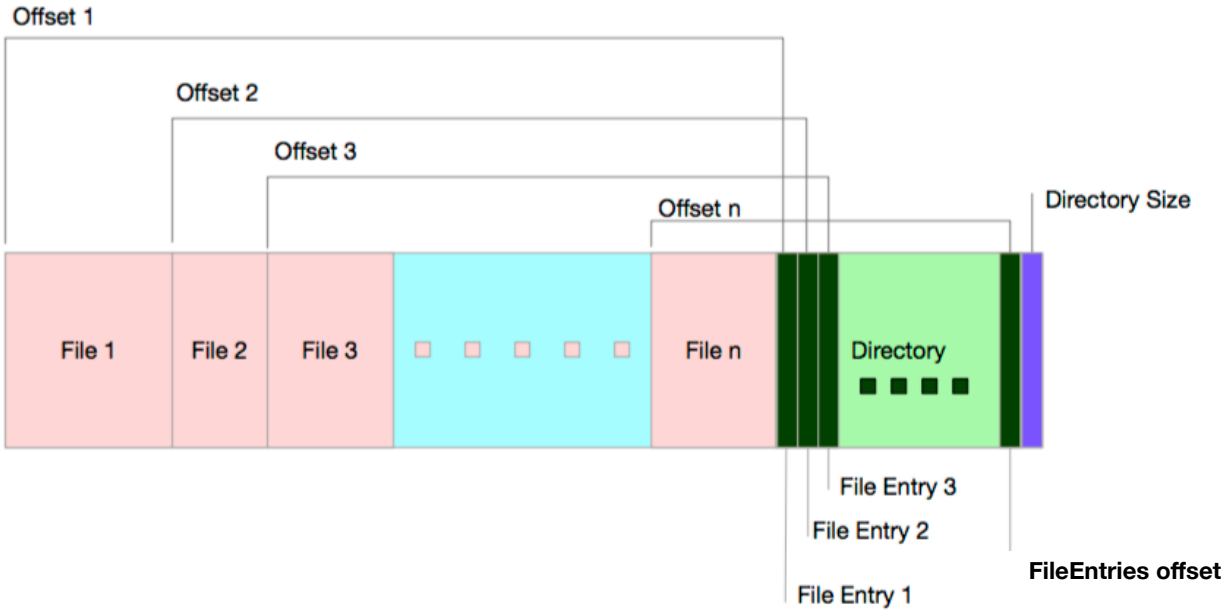
The UnArchiver



EXECUTIVE SUMMARY

Mechanism

The archive file contains : n Files, n FileEntry objects, offset of FileEntry(where FileEntry array begin) and the number of FileEntry Objects.

Archive

1. going over the directory and makes files path copies and save them in List<string>
2. writing these files binary to the target archive file
3. to every file we've written we created instance of FileEntry with all the data and put that data in List<FileEntry>
4. write all FileEntry Objects to the file using Formatter
5. write the offset , where FileEntry begin inside archive file
6. write the number of FileEntry Objects to archive file.

Extract

The reversed process:
1.Read FileEntry offset, and FileEntry Object number from the EOF
2.Read the FileEntry Objects and put them inside List<FileEntry>
3.making the folder for placing the extracted files
4.binary copy and making the files on H.D

Design Summery

For the design of the exercise i used the Following classes:

- [Serializable]FileEntry.cs Class contains the following properties:
    - OffsetStart
    - Size
    - FileName
    - FilePath

- Static Class Archiver.cs : use her methods to archive folder to archive file recursively
    - has a main function in-charge of making the archive file named:
    - Archiver.CreateFromDirectory(string sourceDirectoryName,string destinationArchiveFileName);

- Static Class Extractor.cs : use the various methods to extract archive file completely.
    - has 3 main functions and various helping methods
    - Extractor.GetArchiveInfo(string archiveName); - loads all the data from archive file required for unpacking the archive file into different files and folders
    - Extractor.ExtractToCurrentDir(string archiveName); - gets archive and un-pack it to the current folder
    - Extractor.ExtractSpecificFileToCurrentDir(string archiveName,string fileToExtract); - gets archive file and a specific file to un-pack from the archive, for the simplicity of that method we used LINQ.
- Class Program.cs :
    - private static void Main(string[] args); - runs everything
    - static void RunCommand(string[] args) - parses the user arguments