**Synchroniser**

The application *Synchroniser.py* creates a copy of the working (*Source*) directory into a *Replica* directory. It periodically checks for changes in the *Source* directory and synchronizes it in the *Replica* directory.

**How To Run**

1. First create the *Source* directory/folder. The directory in which you will be creating all your files and folders.
   1. The *Source* directory can also be a directory that you are already working on.
   2. It can also be an empty directory where you wil be adding all the files.
2. Create a *Replica* directory. The directory where you want all the data to be saved.
   1. The *Replica* directory should be created new and should be empty.
   2. If a pre-existing directory is assigned to be a *Replica* directory, its content will be deleted and updated with *Source* directory contents

**Upon Start Up**

Upon running the *Synchroniser.py* script several command line input messages pop up. Each described as below:

1. Enter the full path of the *Source* directory
2. Enter the full path of the *Replica* directory
3. Enter the full path of the *Logs.log* file without the exension *Logs.log*
   1. The name of the logging file is set to *Logs.log* by default and cannot be changed at the moment.
   2. Ex: If the full path of *Logs.log* is *C:\Users\neelg\source\repos\python\Synchroniser\Synchroniser\Logs.log* enter *C:\Users\neelg\source\repos\python\Synchroniser\Synchroniser*
4. Enter the frequency of how often you want the synchronization between *Source* and *Replica* to happen.
   1. The input frequency should be an integer. The units are set to seconds.
   2. Ex: If you enter *20* then synchronization first starts after *20* seconds and periodically repeats every *20* seconds

**Dependencies**

All the imports used are internal python libraries, therefore installation of any external library is not required.

1. It is advisable to use *Python 3.10* for stable operation as the python libraries used are from version: *3.10*
2. The code should run on both Windows and Linux without any modifications.

**About the Code**

The functionality is split into three classes as follows:

1. Class StartUp() : This class initiates the start-up procedure and takes all the input from the command lines and saves then as class members. The class members are:
   1. Full source directory path
   2. Full replica directory path
   3. Full Logs.log file path
   4. Frequency of periodic check
2. Class HashDigest(): This class takes the file path and creates a MD5 hexhash for that file. The class allows for the flexibility to add other hash algorithm such as SHA256 with minimal change the main file management code.
3. Class SourceToReplicaManagement(): This implements a breadth first search type algorithm to expand through the files in each directory and the nested directories with directories. The flow is as follows:
   1. Function checkSource(): first checks the Source directory. All the items that are files are first processed (check if needed to be created/modified in replica directory). All the items that are directories are stored in a list with its full path.
   2. Function beginBFS(): is where the main breadth first search algorithm is implemented. It is recursive function which goes through all the directories from the checkSource() and process all the files in those directories (check if needed to be created/modified in replica directory). It then stores all the directories that might be nested from the previous directory and stores it in a list. A recursive call is then made to beginBFS() on newly created list of directories. This goes on until the last nested directory is reached and all the files processed. The path of the directories are stored in the class member *globalDirList.*

Text

Description automatically generated

When the size of the class member *globalDirList* stop increasing and the value of class member *level*  increases to match the size of the *globalDirList* the recursion is stopped and marks the end of syncronization code.

* 1. Function fileLevelModification(): checks if the files needs to created/modified in the replica directory. The modification condition is raised when the MD5Hash of the source and replica files do not match and copy of the source file is initiated.

Text

Description automatically generated

* 1. Function removalProcedure(): Compares the items of the replica directory with the contents of the source directory. If a file is removed from the source directory then it is also removed from the replica directory. If a directory was deleted from the source then the entire directory tree is also deleted from the replica.

Text

Description automatically generated