**CleanSync**

**Developer’s Guide**

**CS3215 Team9: 0110**

1. Product description: ------------------------------------------------------------- page 3

1.1 CleanSync -------------------------------------------------------------------- page 3

1.2 Features ----------------------------------------------------------------------- page 3

2. Technical description: ----------------------------------------------------------- page 4

2.1.1 [CompareLogic](http://code.google.com/p/cleansync/wiki/CompareLogic) ---------------------------------------------------------------- page 4

2.1.2 [ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult) ------------------------------------------------------------ page 6

2.1.3 [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) -------------------------------------------------------------- page 8

2.1.4 [ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult) ------------------------------------------------------------ page 11

2.1.5 Conflict ------------------------------------------------------------------------- page 13

2.1.6 [ConflictHandler](http://code.google.com/p/cleansync/wiki/ConflictHandler) ---------------------------------------------------------------- page 15

2.1.7 Differences --------------------------------------------------------------------- page 16

2.1.8 [DataInputOutput](http://code.google.com/p/cleansync/wiki/DataInputOutput) --------------------------------------------------------------- page 18

2.1.9 [JobDefinition](http://code.google.com/p/cleansync/wiki/JobDefinition) ------------------------------------------------------------------- page 19

2.1.10 [JobLogic](http://code.google.com/p/cleansync/wiki/JobLogic) ----------------------------------------------------------------------- page 22

2.1.11 [LogFile](http://code.google.com/p/cleansync/wiki/LogFile) ------------------------------------------------------------------------- page 23

2.1.12 MainLogic --------------------------------------------------------------------- page 25

2.1.13 [ReadAndWrite](http://code.google.com/p/cleansync/wiki/ReadAndWrite) ---------------------------------------------------------------- page 27

2.1.14 [SyncLogic](http://code.google.com/p/cleansync/wiki/SyncLogic) ------------------------------------------------------------------------------------- page 30

2.2 Sequence Diagrams ------------------------------------------------------------ page 34

2.2.1 Create job --------------------------------------------------------------------- page 34

2.2.2 Accept job --------------------------------------------------------------------- page 35

2.2.3 Accept & sync ---------------------------------------------------------------- page 36

2.2.4 FirstSetup ---------------------------------------------------------------------- page 37

2.2.5 NormalSync ------------------------------------------------------------------- page 38

**1. Product description:**

**1.1 CleanSync**

CleanSync is a one stop user-friendly and file synchronization software that facilitates the daily back-up and synchronization needs of Users who have to bring home the files from the office to continue their work at home and thus have to ensure that the two sets of files at both locations are synchronized. Other than conventional synchronization between 2 folders, CleanSync utilizes our new technology, Clean Synchronization, which allows users to sync two computers using a third USB device, while keeping disk space usage on the external device, to a minimum. With Clean Synchronization, users can synchronize between workstations using an external device without keeping track of two separate synchronization jobs of the external drive to each of the computers separately.

**1.2 Features**

Clean Synchronization: Utilizing clean synchronization, users can synchronize between two computers with a using a third USB device. Only files that are required to be updated will be stored on the USB device, keeping disk usage low. With Clean Synchronization, users will be able to synchronize between two folders in two computers using a third device that has a maximum disk size that is less than that of the folders.

**2. Technical Decription**

|  |  |
| --- | --- |
| |  | | --- | |  |   **2.1.1 CompareLogic** Compare Logic handle comparison between two folders, given two root folders, it is able to tell what are the changes: "modified", "deleted" or "created". And return a comprehensive result to callers.Methods:public functions:public Differences ConvertFolderMetaToDifferences(FolderMeta folderMeta)public Differences CompareDirectories(FolderMeta newTree, FolderMeta oldTree)public List<Conflicts> ComparePCwithUSB(Differences USBFoldersAndFiles, Differences PCFoldersAndFiles)public FolderMeta ConvertDifferencesToTreeStructure(FolderMeta root, Differences differences)private functions:private void CompareDirectories(FolderMeta newTree, FolderMeta oldTree, Differences differences)private void CompareFiles(FolderMeta newFolder, FolderMetav oldFolder, Differences differences)private void CompareFolders(FolderMeta newFolder, FolderMeta oldFolder, Differences differences)private FolderMeta checkList(FolderMeta folderMeta, List<FolderMeta> folderList)private FileMeta checkList(FileMeta fileMeta, List<FileMeta> fileList)private void RemoveFile(FileMeta fileToBeRemoved, List<FileMeta> fileList)private void RemoveFolder(FolderMeta folderToBeRemoved, List<FolderMeta> folderList)private bool CheckFileInList(FileMeta fileToBeChecked, List<FileMeta> fileList)private bool CheckFolderInList(FolderMeta folderToBeChecked, List<FolderMeta> folderList)private void ConvertSubFolders[?](http://code.google.com/p/cleansync/w/edit/ConvertSubFolders)(FolderMeta root, Differences differences, List<FolderMeta> subFolders)private void AddDeletedFilesToRootFileList(FolderMeta root, List<FileMeta> deletedFileList)private void RemoveUntouchedFilesFromRootFileList(FolderMeta root, Differences differences, List<FileMeta> subFiles)private static void ClearRootUnTouchedFolders(FolderMeta root)private static void ClearRootUnTouchedFiles(FolderMeta root)private void ClearFileList(FolderMeta root, List<FileMeta> fileList, [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta).Type type)private void ClearFolderList(FolderMeta root, List<FolderMeta> folderList, [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta).Type type)private bool checkFileExistence(FileMeta file, Differences differences)private bool checkFolderExistence(FolderMeta folder, Differences differences) |

|  |  |
| --- | --- |
| |  | | --- | |  |  2.1.2 ComparisonResultComparison Result contains the result through compare logic, which maintains three different results: Differences on PC, difference on removable devices and lastly the conflict between the two above.Attributes:public Differences USBDifferences This is the difference on USB relative to PC, of type Differences. public Differences PCDifferences This is the difference on PC relative to USB, of type Differences. public List<Conflicts> conflictList This contains conflicts of PC and USB, which maintained inside a list of Conflicts. Constructors:public [ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult)(Differences USBDifferences, Differences PCDifferences, List<Conflicts> conflictList)Methods:public List<string> ConvertComparisonResultToListOfString([ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult) comparisonResult)private string ConflictlistToString(List<Conflicts> conflictList) |
| **2.1.3 ComponentMeta** This is the basic file and folder metadata definition extends form the basic metadata definition "[ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta)". It provides the basic information about each files, also, give the structure of the folders using tree structure.Attributes:public enum Type{ New,Modified,Deleted,NotTouched } Defines the type (or status) of one file(folder). New: first createdModified: being modified by other sideDeleted: Deleted by other sideNotTouched: No change on both sidespublic string Name Name of the file/folder metadata public string Path Relative path based on the root folder selected. Eg:"D:\temp\temp1\a.txt", if we selected "D:\temp" as the root, then path will be "temp1\a.txt". public string AbsolutePath Stores the absolutePath of the file/folder metadata public string rootDir Give the root folder selected by user Constructor:public [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta)(string path, string rootDir)Methods:public static bool operator >([ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) first, [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) second) Compare if the first component metadata is larger than second. public static bool operator <([ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) first, [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) second) Compare if the first component metadata is smaller than second. FileMeta **An instance extends from**[**ComponentMeta**](http://code.google.com/p/cleansync/wiki/ComponentMeta)**which specifies the file informations.** Attributes:public Type FileType Type of the file: using the Type in the [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) enum. public DateTime LastModifiedTime Get the last modified time for the file. public DateTime CreationTime Get the creation time of the file. public long Size Get the size of the file. Constructor:public FileMeta(string path, string rootDir) : base(path, rootDir) Inheritated from the ComponentMata. public FileMeta(FileMeta file) : base(file.AbsolutePath,file.rootDir) Using the given FileMeta to construct new FileMeta object. Methods:public static int ConvertToKiloByte(FileInfo fileInfo) Return file size in K bytes. public string getString() Provide a string representation of the file metadata. FolderMeta **An instance extends from**[**ComponentMeta**](http://code.google.com/p/cleansync/wiki/ComponentMeta)**which specifies the folder informations.** Attributes:public Type FolderType Type of the folder: using the Type in the [ComponentMeta](http://code.google.com/p/cleansync/wiki/ComponentMeta) enum. public List<FolderMeta> folders List of folders inside one folder. public List<FileMeta> files List of files inside one folder. public long Size Size of the folder. Constructor:public FolderMeta(string path, string rootDir) Inheritated from the ComponentMata. public FolderMeta(FolderMeta root):base(root.AbsolutePath,root.rootDir) Given one folder metadata to generate its own metadata. Methods:public void AddFile(FileMeta file) Add one file to the file list. public void AddFolder(FolderMeta folder) Add one folder to the folder list. public IEnumerator<FolderMeta> GetFolders() Get all the folders in the folder metadata. public IEnumerator<FileMeta> GetFiles() Get all the files in the folder metadata. public String getString() Return a string representation of the folder metadata. |

**2.1.4 ComparisonResult**

## Comparison Result contains the result through compare logic, which maintains three different results: Differences on PC, difference on removable devices and lastly the conflict between the two above.

**Attributes:**

**public Differences USBDifferences**

This is the difference on USB relative to PC, of type Differences.

**public Differences PCDifferences**

This is the difference on PC relative to USB, of type Differences.

**public List<Conflicts> conflictList**

This contains conflicts of PC and USB, which maintained inside a list of Conflicts.

**Constructors:**

**public**[**ComparisonResult**](http://code.google.com/p/cleansync/wiki/ComparisonResult)**(Differences USBDifferences, Differences PCDifferences, List<Conflicts> conflictList)**

**Methods:**

**public List<string> ConvertComparisonResultToListOfString(**[**ComparisonResult**](http://code.google.com/p/cleansync/wiki/ComparisonResult)**comparisonResult)**

**private string ConflictlistToString**[**?**](http://code.google.com/p/cleansync/w/edit/ConflictlistToString)**(List<Conflicts> conflictList)**

# 2.1.5 Conflict

### Defines the basic object of conflicts between files.

## Attributes:

## public enum FolderFileType{FolderConflict,FileConflict }

Defines the conflict is between files or between folders.

**public enum ConflictType{New,Modified,Deleted}**

Defindes the confilct type between pairs, whether it is

1. Newly created

2. Being Modified

3. Being Deleted

**public FolderMeta CurrentPCFolder**

Folder metadata of the current PC folder being synchronized.

**public FolderMeta USBFolder**

Folder metadata of the current USB folder being synchronized.

**public FileMeta CurrentPCFile**

File metadata on current PC file being synchronized.

**public FileMeta USBFile**

File metadata on current USB file being synchronized.

**public FolderFileType FolderOrFileConflictType**

Defines whether it is a folder conflict or file conflict.

**public ConflictType PCFolderFileType**

Defines whether it is "new" or "deleted" of conflict on PC side.

**public ConflictType USBFolderFileType**

Defines whether it is "new" or "deleted" of conflict on USB side.

## Constructors:

## public Conflicts(FolderMeta currentPCFolder, FolderMeta USBFolder, ConflictType PCFolderFileType, ConflictType USBFolderFileType)

Defined for folder conflicts.

**public Conflicts(FileMeta currentPCFile, FileMeta USBFile,ConflictType PCFolderFileType, ConflictType USBFolderFileType)**

Defined for file conflicts.

## Methods:

## public override string ToString()

Return a string representation of the conflicts.

|  |
| --- |
| 2.1.6 ConflictHandlerGiven one conflict the Conflict will handle different user inputs: Whether to keep a copy on the removable device or on the PC, which the file/folder information will be provided as a reference.Constructor:public [ConflictHandler](http://code.google.com/p/cleansync/wiki/ConflictHandler)()Methods:public method:public [ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult) handleConflicts([ComparisonResult](http://code.google.com/p/cleansync/wiki/ComparisonResult) comparisonResult, int userChoice)private method:private void handleFolderConflict(Differences differencesToBeUpdated , Conflicts folderConflict, int USBorPCIndex)private void handleFileConflict(Differences differencesToBeUpdated, Conflicts fileConflict, int USBorPCIndex) |

# 2.1.7 Differences

## Handle basically five different type of differences:

### 1. Folder created.

### 2. Folder deleted.

### 3. File created.

### 4. File deleted.

### 5. File modified.

## Attributes:

## private List<FolderMeta> deletedFolderDifference =new List<FolderMeta>();

Contains differences of type folder deleted.

**private List<FolderMeta> newFolderDifference = new List<FolderMeta>();**

Contains differences of type folder created.

**private List<FileMeta> deletedFileDifference = new List<FileMeta>();**

Contains differences of type file deleted.

**private List<FileMeta> newFileDifference = new List<FileMeta>();**

Contains differences of type file created.

**private List<FileMeta> modifiedFileDifference = new List<FileMeta>();**

Contains differences of type file modified.

## Methods:

## public void AddDeletedFolderDifference(FolderMeta deletedFolder)

## public void AddNewFolderDifference(FolderMeta newFolder)

## public void AddNewFileDifference(FileMeta newFile)

## public void AddDeletedFileDifference(FileMeta newFile)

## public void AddModifiedFileDifference(FileMeta newFile)

## public List<FolderMeta> getNewFolderList()

## public List<FolderMeta> getDeletedFolderList()

## public List<FileMeta> getDeletedFileList()

## public List<FileMeta> getNewFileList()

## public List<FileMeta> getModifiedFileList()

## public void removeFolderFromDeletedFolderList(FolderMeta folder)

## public void removeFolderFromNewFolderList(FolderMeta folder)

## public void removeFileFromNewFileList(FileMeta file)

## public void removeFileFromModifiedFileList(FileMeta file)

## public void removeFileFromDeletedFileList(FileMeta file)

## public override string ToString()

# 2.1.8 DataInputOutput

## Provides a file read/write.

## Methods

## public static T LoadFromBinary(string path)

Performs a read file operation.

**public static void SaveToBinary(string path, T dataObject)**

Performs a write file operation.

|  |
| --- |
| **2.1.9 JobDefinition**  **JobDefinition is a serializable abstract class used to describe the meta data for jobs. USBJob and PCJob inherit from JobDefinition** Attributes:Public  * **enum JobStatus { Complete, Incomplete, NotReady}**   + Complete indicates that both PCs synchronized in this Job have already been identified.   + Incomplete indicates that   + NotReady * **string JobName** * **JobStatus JobState** * **string RelativeUSBPath**   **Methods:** Public:****void ToggleStatus(JobStatus**** ****state)**** Toggles the state of the job, based on a given state. PCJob Class Description **PCJob stores the metadata of the job information that is stored on the PC.** Attributes:Public****string PCPath**** **FolderMeta FolderInfo**  Stores the last known metadata of the root folder in the PC.  **string PCID** Private **USBJob usbJob**  Methods: Public: **USBJob GetUsbJob()**  **void SetUsbJob(USBJob usb)** USBJob Class Description **USBJob stores the metadata of the job information that is stored on the USB.** Attributes:Public **string PCOnePath**  **string PCTwoPath**  **Differences diff**  **string PCOneID**  **string PCTwoID**  **string MostRecentPCID**  The ID of the most recent PC which did a synchronization.  **bool PCOneDeleted**  Checks if PCOne has deleted this job.  **bool PCTwoDeleted**  Checks if PCtwo has deleted this job. |

**2.1.10 JobLogic**

JobLogic handles all requests for job-related work. It manages the jobs in the system and delegates the work further to CompareLogic and SyncLogic.

**Attributes:**

## Private

**List<string> StoredPCJobInfoPaths**

**List<string> StoredUSBJobInfoPaths**

**SyncLogic sync**

**CompareLogic compareLogic**

## Internal

**List<PCJob> PCJobs**

**List<USBJob> USBJobs**

**Constructors:**

**JobLogic()**

**Methods:**

## Public

**void InitializePCJobInfo()**

Loads all pcJob found on the PC

**void InitializeUSBJobInfo(string usbRoot,string pcID)**

Searches and Loads all usbJob found on the USB drives and mount them to the respective pcJobs.

**PCJob CreateJob(string jobName, string PCPath, string AbsoluteUSBPath, string PCID)**

**PCJob CreateJob(USBJob jobUSB, string PCPath, string PCID)**

**bool CheckNameConflict(string JobName)**

**void InsertJob(USBJob usbJob)**

**void InsertJob(PCJob pcJob)**

**bool setupJob(PCJob pcJob, System.ComponentModel.BackgroundWorker worker)**

**string GetFullFolderPath(PCJob pcJob, FolderMeta folderInfo)**

**void WriteIncompleteFileInfoOnUSB(USBJob usbJob)**

**void CleanSync(ComparisonResult comparisonResult, PCJob pcJob, System.ComponentModel.BackgroundWorker worker)**

**ComparisonResult Compare(PCJob pcJob)**

**void USBPlugIn(string drive, string pcID)**

**void USBRemoved()**

**ComparisonResult handleConflicts(ComparisonResult comparisonResult, int userChoice)**

**PCJob ConnectUSBJobwithPCJob(USBJob usbJob)**

**void DeleteJob(PCJob pcJob,string pcID)**

## Private

**int CheckPCTwoDelete(int i, USBJob usbJob)**

**int CheckPCOneDelete(int i, USBJob usbJob)**

**int GetPCNumber(USBJob usbJob,string pcID)**

**void WriteIncompleteFileInfoOnUSB(USBJob usbJob)**

**void RemovePCJob(PCJob pcJob)**

**void RemoveUSBJob(USBJob usbJob)**

**2.1.11 LogFile** 

**LogFile is in charge of logging. The path to the file is specified and every subsequent log is written to that file.**

**Attributes:**

## Private

**static string LogFilePath = Directory.GetCurrentDirectory()**[+@"\CleanSyncLog.txt](mailto:+@%22\CleanSyncLog.txt)**"**

**Methods:**

## Public

**static void FileDeletion(string path)**

**static void FolderDeletion(string path)**

**static void FileCreation(string path)**

**static void FolderCreation(string path)**

**static void FileCopy(string source, string destination)**

**static void FolderCopy(string source, string destination)**

**static void NewLine()**

**static void ExportToPC(string path)**

**static void ExportToUSB(string path)**

**static void ReportSolvedConflicts(string type, string PCpath, string AbsoluteUSBPath, string userChoicePath)**

**2.1.12 MainLogic**

**MainLogic handles request from the GUI and distributes the work to the various classes in the logic component.**

**Attributes:**

## Public

**List<string> GetCurrentDrives**

## Private

**JobLogic jobLogic**

**string thisPCID**

**List<string> CurrentDrives**

**List<USBJob> IncompleteList**

**Constructors:**

**MainLogic()**

**Methods:**

## Public

**void InitializePCJobInfo()**

**PCJob CreateJob(string JobName,string PCPath, string pathName )**

**bool FirstTimeSync(PCJob pcJob, System.ComponentModel.BackgroundWorker worker)**

**List<USBJob> AcceptJob(List<string> drives)**

This method searches all USB drives connected to the computer for incomplete jobs. If there is an incomplete job, check if this computer is the computer that created the job. If it is not, it will return it as an incomplete USB job available to be accepted by this computer.

**PCJob CreateJob(USBJob jobUSB, string PCPath)**

**ComparisonResult Compare(PCJob pcJob)**

**void CleanSync(ComparisonResult comparisonResult, PCJob pcJob, System.ComponentModel.BackgroundWorker worker)**

**void setUSBJobList(List<USBJob> usbJobs)**

**void USBPlugIn(List<string> drives)**

**void USBRemoved(string drives)**

**List<PCJob> GetPCJobs()**

**List<USBJob> GetUSBJobs()**

**ComparisonResult handleConflicts(ComparisonResult comparisonResult, int userChoice)**

**void DeleteJob(PCJob pcJob)**

**string GetPCID()**

Uses the MAC address of this computer as the PCID.

**2.1.13 ReadAndWrite**

**Other than for logging, ReadAndWrite performs all copy and delete operations on files and folders. It is also called to fetch data from the hard disk.**

**Methods:**

## Public

**static FolderMeta BuildTree(string rootDir)**

**static void DeleteFile(string path)**

**static void DeleteFolder(string path)**

This method will delete the specified folder and all its contents.

**static void DeleteFolderContent(string path)**

This method will not delete the specified folder, only all its contents.

**static void EmptyFolder(string path)**

this method will only delete the subfiles within the specified folder.

**static void CopyFile(string source,string destination)**

**static void CopyFile(string source, string destination, System.ComponentModel.BackgroundWorker worker, double onePercentSize)**

**static void CopyFolder(string source, string destination, System.ComponentModel.BackgroundWorker worker, double onePercentSize)**

**static void CopyFolder(string source,string destination)**

**static void RenameFile(string source, string destination)**

**static void RenameFolder(string source, string destination)**

**static List<string> ImportJobList()**

**static USBJob ImportIncompleteJobFromUSB(string incompleteUSBJobPath)**

**static void ExportPCJobPathsList(List<string> pcJobPathsList)**

**static void ExportPCJob(PCJob pcJob)**

**static void ExportIncompleteJobToUSB(USBJob incompleteUSBJob)**

**static string GetStoredPathOnPC(PCJob pcJob)**

**static string GetPCJobListPath()**

**static string GetUSBJobListPathLocation(string usbPath)**

**static string GetIncompleteUSBFilePath(USBJob usbJob)**

**static string GetIncompleteUSBFolderPath(string AbsoluteUSBPath)**

**static string GetIncompleteUSBFolderLocation(string usbRoot)**

**static List<USBJob> GetIncompleteUSBJobList(string USBRoot)**

**static List<PCJob> GetPCJobList(string rootFolder)**

**static List<USBJob> GetUSBJobList(string rootFolder)**

**static void RemoveIncompleteUSBJob(USBJob jobUSB)**

**static List<PCJob> GetPCJobs(List<string> jobPaths)**

**static string GetStoredPathOnUSB(USBJob usbJob)**

**static string GetStoredFolderOnUSB(string usbPath)**

**static void ExportUSBJobPathsList(List<string> StoredUSBJobInfoPaths,USBJob usbJob)**

**static void ExportUSBJob(USBJob usbJob)**

**static string GetUSBRootPath(USBJob usbJob)**

**static string GetUSBRootPath(string usbPath)**

**static List<string> GetUSBJobListPath(string AbsoluteUSBPath)**

**static List<USBJob> GetUSBJobs(List<string> usbJobListPath)**

## Private

**static FolderMeta BuildTree(string sourceDir, string rootDir)**

**static string GetPCRootPath()**

**static void CreateDataFolder(string rootPath)**

**static void ReSetPCFolderAccess()**

**static void SetPCFolderAccess()**

**static void ReSetUSBFolderAccess(string path)**

**static void SetUSBFolderAccess(string path)**

**static void GrantAccess(string folderPath)**

# 2.1.14Sync Logic Description

**Sync Logic handles synchronization between two folders. After comparing differences of the component metas, the results are brought here to execute synchronization and the copying of files and folders between the PC and the USB.**

**Files and Folders stored in the USB are renamed and their renaming is also handled by Sync Logic.**

**Constructor:**

**SyncLogic()**

**Methods:**

## Public

In SyncLogic there is only one method called by external classes, which is to do a synchronization.

**void CleanSync(ComparisonResult comparisonResult, PCJob pcJob, System.ComponentModel.BackgroundWorker worker)**

This method checks whether this PC is the last PC to do a synchronization with the USB by matching the PCID with the usbJob's MostRecentPCID. If it is not the last PC to synchronization with the USB, a normal synchronization is performed. Else, it will do a re-synchronization. The worker object is used to update the progress of synchronization.

## Private

Private methods are divided into three groups.

### Calculate Size

These methods are called to calculate the total size of data needed to be copied for this job.

**void initializeTotalSize(ComparisonResult comparisonResult)**

**void updateFolderSize(List<FolderMeta> folders)**

**void updateFileSize(List<FileMeta> files)**

### Normal Synchronization

These methods are called to do a normal synchronization. In the methods that copies files and folders from the USB to the PC, the individual files and folders in the USB will be deleted immediately after being copied to the PC. The algorithm is basic, just going through the differences in the comparison result and copying and renaming the files and folders accordingly.

**void NormalCleanSync(ComparisonResult comparisonResult, PCJob pcJob)**

**void SyncPCToUSB(Differences PCToUSB, PCJob pcJob)**

**void SyncPCToUSBModifiedFile(PCJob pcJob, List<FileMeta> modifiedFileList)**

**void SyncPCToUSBNewFile(PCJob pcJob, List<FileMeta> newFileList)**

**void SyncPCToUSBNewFolder(PCJob pcJob, List<FolderMeta> newFolderList)**

**void SyncUSBToPC(Differences USBToPC, PCJob pcJob)**

**void SyncUSBToPCDeleteFile(PCJob pcJob, List<FileMeta> deletedFileList)**

**void SyncUSBToPCModifiedFile(PCJob pcJob, List<FileMeta> modifiedFileList)**

**void SyncUSbToPCNewFile(PCJob pcJob, List<FileMeta> newFileList)**

**void SyncUSBtoPCDeleteFolder(PCJob pcJob, List<FolderMeta> deleteFolderList)**

**void SyncUSBToPCNewFolder(PCJob pcJob, List<FolderMeta> newFolderList)**

### Resynchronization

These methods are called to do a resynchronization. Updates of files and folders in the PC will be updated on the USB. The renaming of these files and folders are also handled here. The pair of differences in the comparisonResult, instead of being treated as in a normal synchronization, are now viewed as old and new differences. Old differences are the differences in the USB that is to be synchronized with the other PC, new differences are the differences in the PC that are now to be copied to the USB.

**void CleanSyncReSync(ComparisonResult comparisonResult, PCJob pcJob)**

**void ReSyncFiles(Differences oldDifferences, Differences newDifferences, PCJob pcJob)**

**void ReSyncFolders(Differences oldDifferences, Differences newDifferences, PCJob pcJob)**

**void ReSyncNewFiles(PCJob pcJob, Differences oldDifferences, List<FileMeta> newFilesNew)**

**void ReSyncModifiedFiles(Differences oldDifferences, PCJob pcJob, List<FileMeta> modifiedFilesOld, List<FileMeta**[**?**](http://code.google.com/p/cleansync/w/edit/FileMeta)**>modifiedFilesNew)**

3 possible cases:

The modified file was previously a new file in the USB. In this case, the modified file will be copied over and treated as a new version of the new file yet to be copied to the other PC.

The modified file was previously a modified file in the USB. Thus, the modified file will also be copied over and treated as a new version of the modified file.

Else, the modified file's information will be added to the old difference, and the file will be copied over to the USB.

**void ReSyncDeletedFiles(Differences oldDifferences, PCJob pcJob, List<FileMeta> newFilesOld, List<FileMeta**[**?**](http://code.google.com/p/cleansync/w/edit/FileMeta)**> deletedFilesNew, List<FileMeta> modifiedFilesOld)**

3 possible cases:

The deleted file was previously a new file in the USB. In this case, the file will be deleted and its information removed from the old difference.

The deleted file was previously a modified file in the USB. In this case, the file will be deleted, and its information moved from the modified list to the deleted list in the old difference.

Else, the deleted file's information will be added to the old difference's deleted list.

**void ReSyncNewFolders(Differences oldDifferences, PCJob pcJob, List<FolderMeta> newFoldersNew)**

**void ReSyncDeletedFolders(Differences oldDifferences, PCJob pcJob, List<FolderMeta> newFoldersOld, List<FolderMeta>deletedFoldersNew)**

2 possible cases:

The deleted folder was previously a new folder in the USb. In this case, the folder will be deleted from the USB and its information removed from the old difference.

Else, the deleted folder's information will be added to the old difference.

**void RemoveNullComponentsFiles(PCJob pcJob, List<FileMeta> files, string listType)**

This method deletes the null entries in a given FileMeta List. Files are re-organized accordingly and their temporary names are also changed.

**void RemoveNullComponentsFolders(PCJob pcJob, List<FolderMeta> folders, string listType)**

This method is the same as the previous, but works on a given FolderMeta list instead.

## Renaming of Files and Folders in the USB

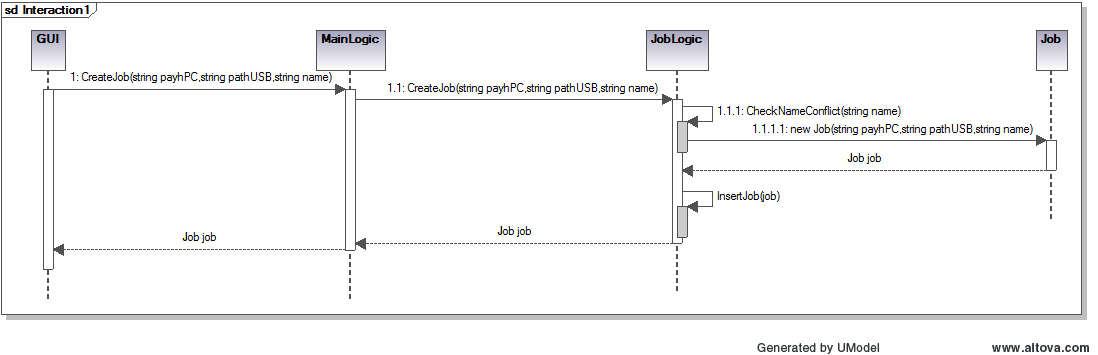
Files and folders copied to the USB are renamed to avoid name conflicts between files and folders of the same name but in different folders in the PC. They are named as such: pcJob.JobName + difference type modifier+ index in the list. Files are renamed similarly, but with an extension .temp.

|  |  |
| --- | --- |
| Difference Type | Modifier |
| New | n |
| Modified | m |
| Deleted | d |

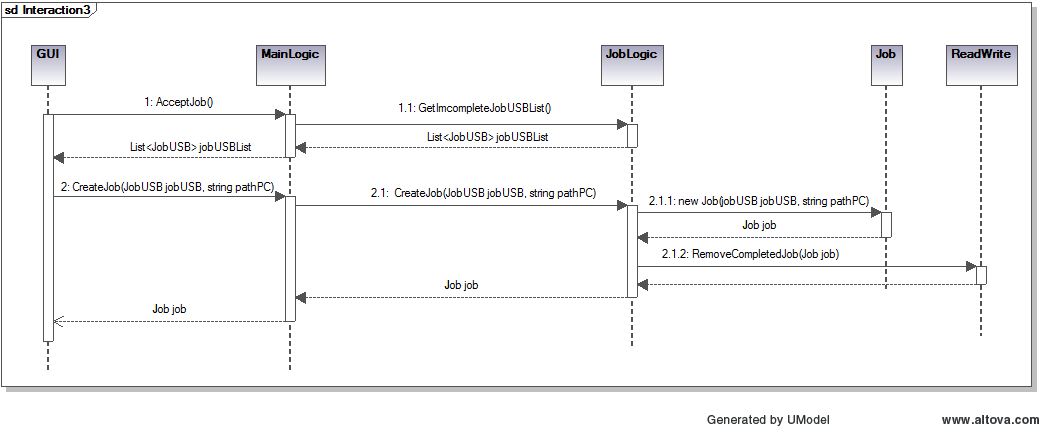
For example, if a file that is modified is to be copied over to the USB, and the file is the 3rd modified file difference in the list. If the job's name is "SyncJob", the file will be renamed as "SyncJobm3.temp". If it is a folder, the folder will be renamed as "SyncJobm3". Note that subfiles and subfolders within renamed folders will not be renamed.

**2.2 Diagrams**

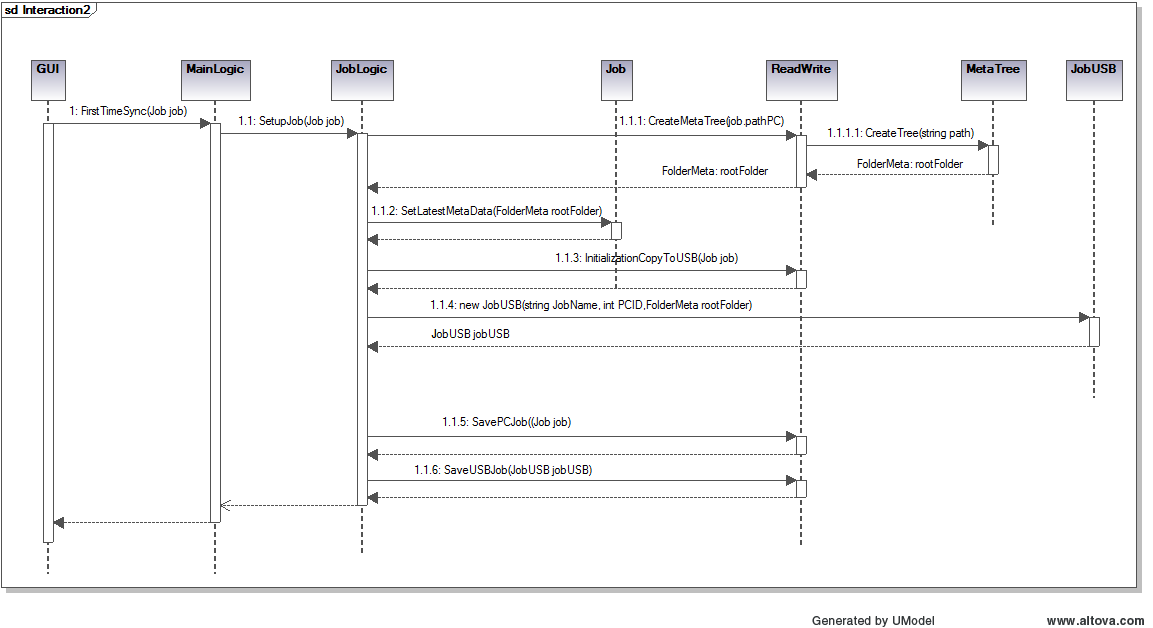
2.2.1. Create Job



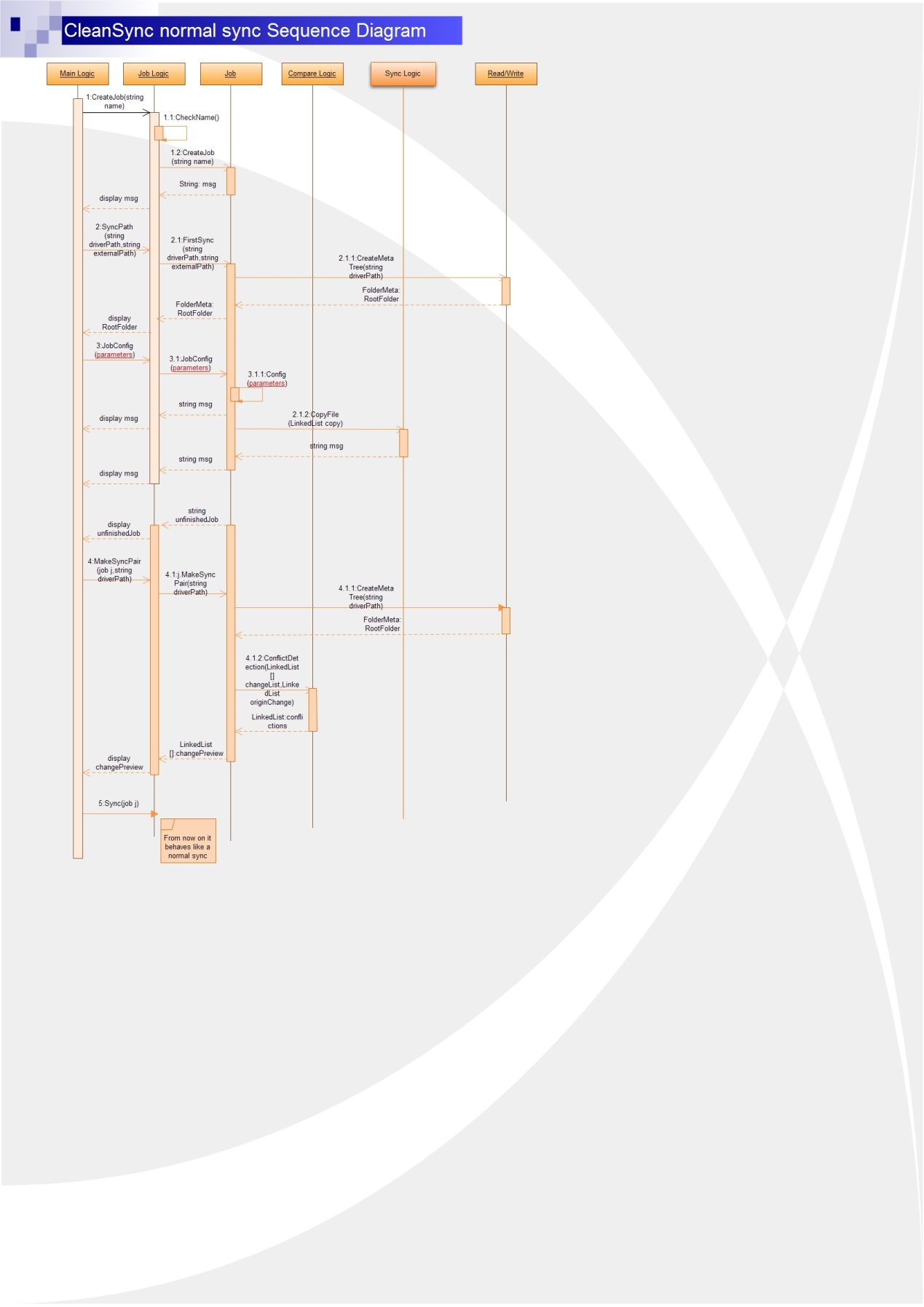
2.2.2 Accept a job



2.2.3 Accept & Sync



2.2.4 First Setup



2.2.5 Normal Sync

