CleanSync

Version 2.0

Developer's Guide

Presented by Team 0110

Gu Yang

Li Yi

Li Zichen

Lu WenHao

Tso Shuk Yi

Yu Qiqi

TABLE OF CONTENTS

1. CLEANSYNC

1.1 INTRODUCTION

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 through a removable device, while keeping disk space usage on the removable device, to a minimum. With Clean Synchronization, users can synchronize between workstations using a removable device without keeping track of the two separate synchronization jobs of the external drive on each of the computers separately.

1.2 FUNCTIONALITY FEATURES

Below are some of the main features that are implemented by CleanSync:

Clean Synchronization

Clean Synchronization is the method through which CleanSync does synchronization. It is especially designed to synchronize folders in 2 different computers through an external drive. There are 2 distinct forms of Clean Synchronization: normal clean synchronization and re-synchronization.

Safe Synchronization

If a synchronization process is interrupted, CleanSync will attempt to restore the folders and files to their original state. CleanSync will also backup deleted and overwritten files and folders on folders.

Preview feature

CleanSync can preview a list to changes made to each computer before synchronization so that users can see what will be changed before allowing synchronization to take place.

Conflict Handling

CleanSync can handle conflicts and allow users to choose from 3 options: Using the one on the removable device, using the one on the computer or choosing to update both, in which case files will be copied in both directions, and will be renamed.

External drive Plug-in Plug-out detection

CleanSync will detect whenever an external drive is plugged in or plugged out and load or unload any synchronization job found on the external drive respectively.

Automation

CleanSync allows the user to select the option of synchronizing a job on the removable device whenever the device is plugged in.

1.3 GLOSSARY OF TERMS

Removable Device

A removable device refers to an external hard drive through which folders in 2 computers can be synchronized.

FileMeta / FolderMeta

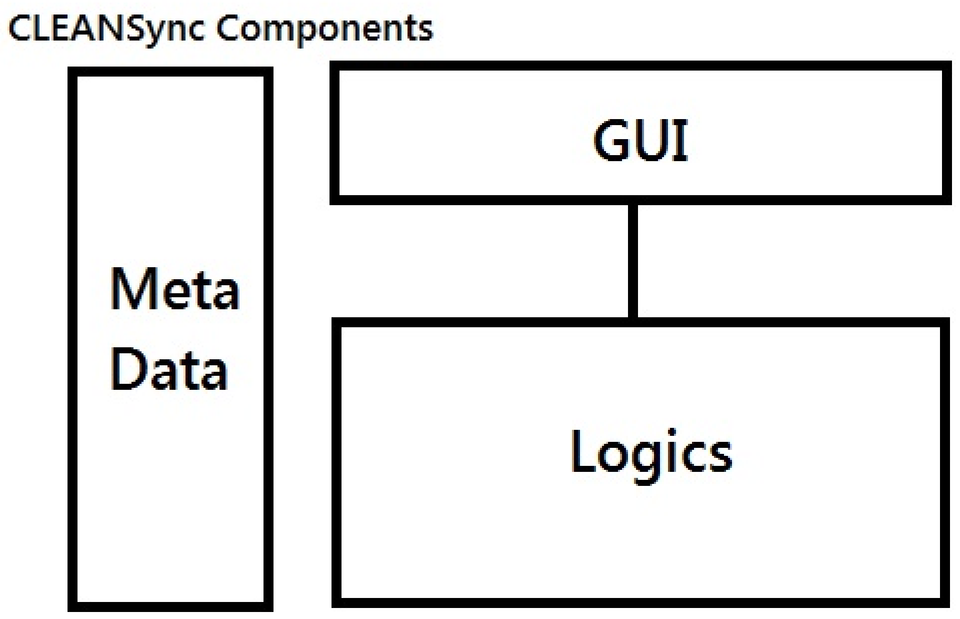
FileMeta and FolderMeta are serializable classes used to store information about files and folders respectively. Using FileMeta and FolderMeta, a directory structure can be represented.

Job

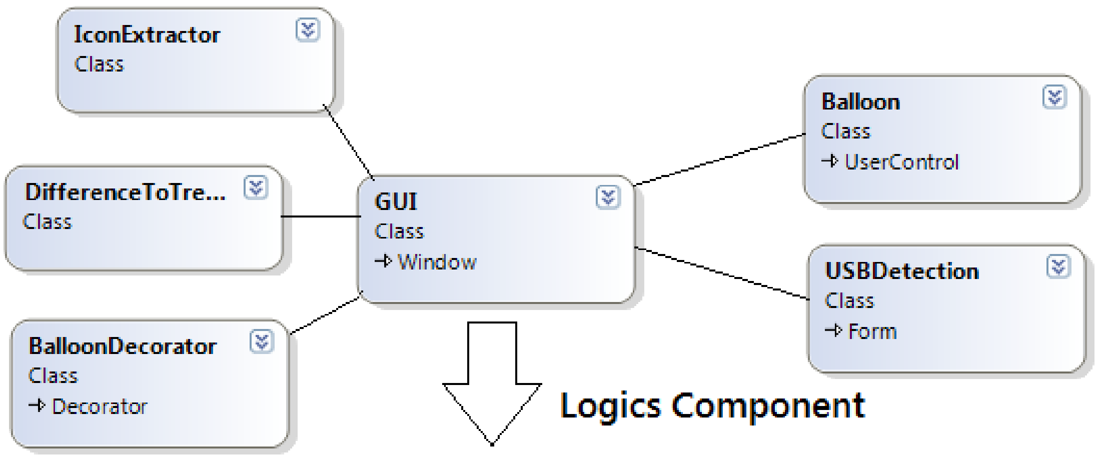
A job stores the information of the directory on both computers and the removable device. When the user wants to synchronize folders in 2 different computers, users create a job. In implementation, a job consists of 2 serializable PCJob classes which is stored in the computers and a serializable USBJob which is stored in the removable device.

2. COMPONENTS

CleanSync consists of 3 main components: GUI, Logics and MetaData. Below are their description, the classes that form the component and description of some of the more notable methods found in the classes.



2.1 GUI



GUI is implemented using WPF. It consists of the main class GUI, which is the main window for CleanSync, and several GUI helper classes.

2.1.1 GUI

The GUI class is the main window for the user interface.

2.1.2 USBDETECTION

This class is in charge of detecting whenever a removable device has been plugged in or plugged out.

2.1.3 BALLOON / BALLOON DECORATOR

The balloon class is used to create information balloons displayed on the system tray. The balloon decorator is used to define the balloon.

2.1.3 ICONEXTRACTOR

The IconExtractor is used to load icons from the system registry. It is mainly used to load the new, modified and deleted icons at the analysis results screen.

2.1.4 DIFFERENCETOTREECONVERTOR

DifferenceToTreeConvertor converts a difference into tree form for display purposes. Note that difference to tree convertor is also called by SyncLogic and CompareLogic.

Methods

Public

FolderMeta ConvertDifferencesToTreeStructure(Differences difference)

Private

bool ConvertFolderListToTree(FolderMeta root, bool haveDifference, List<FolderMeta> folders)

- Adds a list of FolderMeta into a given root. Parent folders are created if it does not exist yet.

bool ConvertFileListToTree(FolderMeta root, bool haveDifference, List<FileMeta> files)

- Adds a list of FileMeta into a given root. Parent folders are created if it does not exist yet.

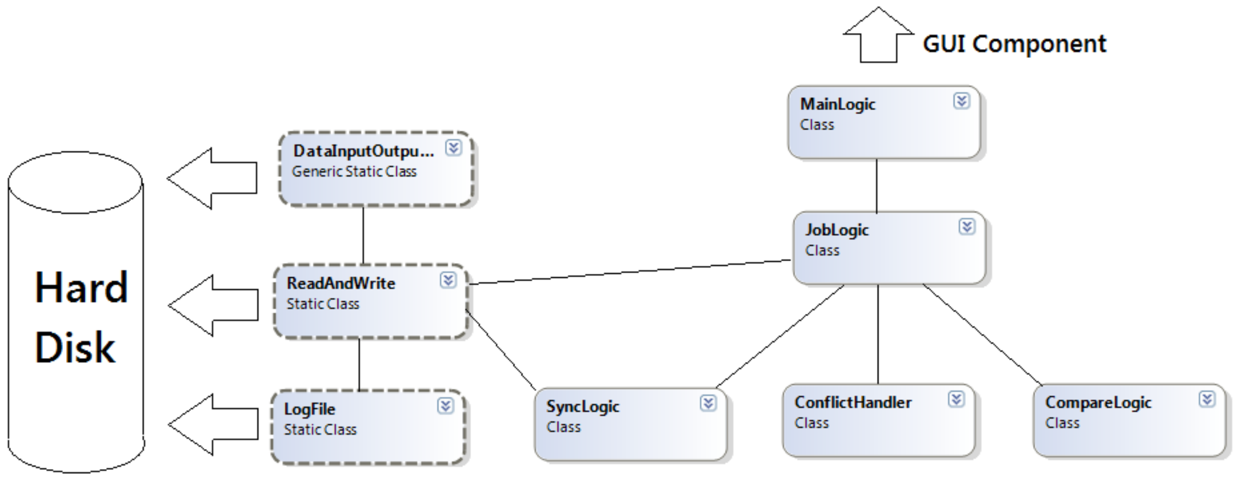
2.1.5 ENUMDISPLAYNAMEATTRIBUTE

This class is used to store a string representation of an enumeration.

2.1.6 ENUMTYPECONVERTER

This class is called to represent enumerations for job configurations in string form.

2.2 LOGIC COMPONENTS



2.2.1 MAINLOGIC

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

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)

string GetPCID()

Get the identification string of the current computer. The identification string is created at the first time when CleanSync runs on a computer using the System.Environment.TickCount, the string is then stored inside the data folder and following GetPCID() will just read the string from this file.

2.2.2 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.

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.

2.2.3 COMPARELOGIC

CompareLogic 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.

CompareLogic currently handles two different jobs: Checking for differences between folders and checking for conflicts between 2 differences.

2.2.4 SYNCLOGIC

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.

In SyncLogic there are two methods called by external classes.

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.

public void InitializationSynchronize(Differences PCToUSB, PCJob pcJob, System.ComponentModel.BackgroundWorker worker,System.ComponentModel.DoWorkEventArgs e)

This method is called when synchronizing for the first time.

Private methods

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, based on the tree structure of the differences.

void NormalCleanSync(ComparisonResult comparisonResult, PCJob pcJob)

void NormalCleanSyncFolderPcToUsb(FolderMeta pcToUsb, string originDirectoryRoot, string destinationDirectoryRoot, Differences pcToUsbDone)

void NormalCleanSyncFolderUsbToPC(FolderMeta usbToPC, string originDirectoryRoot, string destinationDirectoryRoot, Differences usbToPCDone)

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 ReSynchronizeFolders(FolderMeta oldDifferencesRoot, FolderMeta newDifferencesRoot, string sourceDirectory, string destinationDirectory, Differences pcToUSBDone)

void ReSynchronizeFiles(FolderMeta oldDifferencesRoot, FolderMeta newDifferencesRoot, string sourceRoot, string destinationRoot)

The previous two methods are called to resync a modified folder to a previously modified folder.

void CompareNewDeletedWithOldModifiedFolders(FolderMeta folderNew, FolderMeta folderOld)

void CompareNewDeletedWithOldModifiedFiles(FolderMeta folderNew, FolderMeta folderOld)

These two methods are called to resynchronize a modified folder with a deleted folder.

void ReSynchronizeToNewFolder(FolderMeta oldDifferencesRoot, FolderMeta newDifferencesRoot, string sourceRoot, string destinationRoot)

void ReSynchronizeToNewFolderFiles(FolderMeta folderOld, FolderMeta folderNew, string sourceRoot, string destinationRoot)

These two methods are called to resynchronize a previously new folder with a modified folder

Restoration

void RestoreIncompletePCChanges(FolderMeta changes, string pcPath)

This method is called during normal synchronization if the job if an exception is thrown, to restore the folder to its previous state.

2.2.5 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.

static FolderMeta BuildTree(string rootDir)

This is the main method invoked to construct a directory's metadata. It calls a private method of the same name which is recursive, and returns the root folder's foldermeta.

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 MoveFolderContents(string source, string target)

This method will move the contents inside the source to the target. Files and folders will not be overwritten.

static void MoveFolderContentWithReplace(string source, string target)

This method will move the contents inside the source to the target. Files and folders will explicitly be overwritten.

2.2.6 LOGFILE

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

2.2.7 CONFLICTHANDLER

This method handles any conflicts found. Based on the job setting, different conflicts will be handled differently.

2.2.8 JOBSRESTORELOGIC

static void RestoreInterruptedPCJobPCChanges(PCJob pcJob)

This method is invoked to restore a PCJob from a previous synchronization crash due to some PC related failure (e.g. unexpected power off).

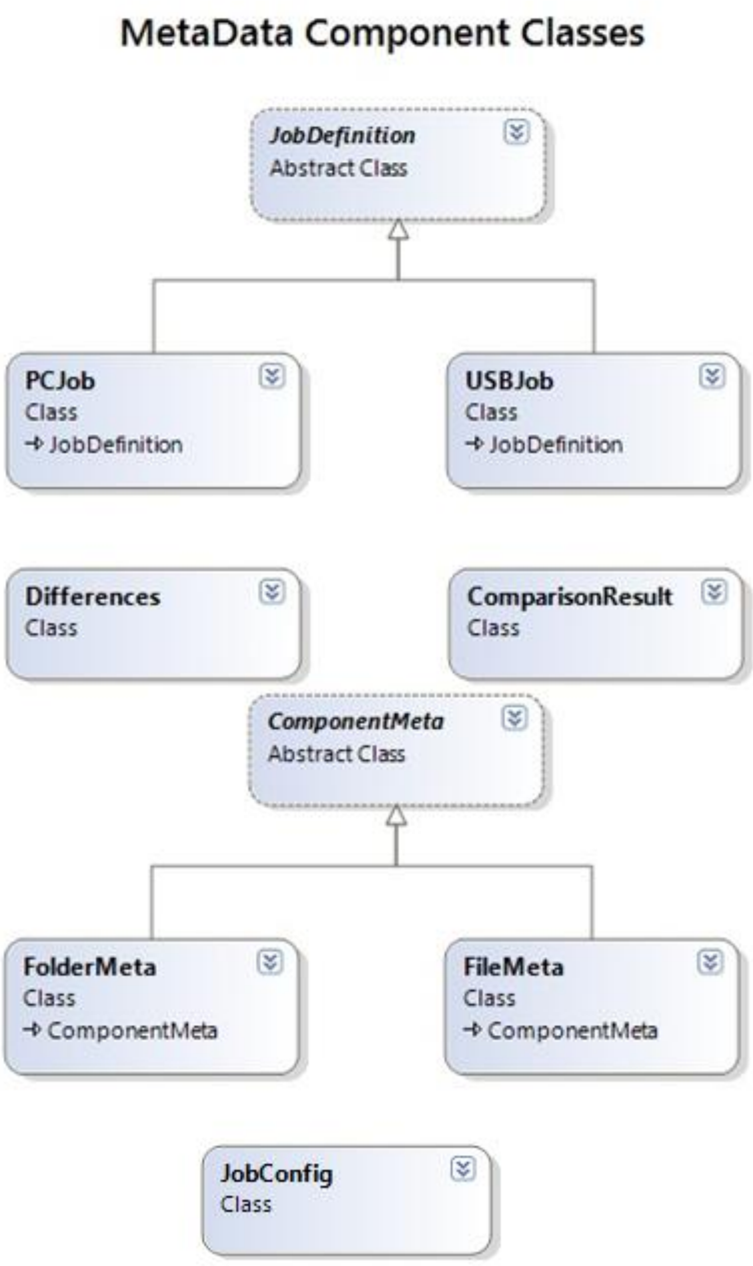
static void RestoreInterruptedUSB(PCJob pcJob)

This method is invoked to restore a USBJob from a previous normal synchronization crash due to some removable device related failure (e.g. plug-out removable device during synchronization).

static void RestoreReSyncUSB(PCJob pcJob)

This method is invoked to restore a USBJob from a previous re-synchronization crash due to some removable device related failure (e.g. plug-out removable device during synchronization).

2.3 METADATA


MetaData Component Classes
JobDefinition
Abstract Class
PCJob
Class
JobDefinition
USBJob
Class
JobDefinition
Differences
Class
ComparisonResult
Class
ComponentMeta
Abstract Class
FolderMeta
Class
ComponentMeta
FileMeta
Class
ComponentMeta
JobConfig
Class

2.3.1 COMPONENTMETA

This is the basic file and folder metadata definition extends form the basic metadata definition "**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 created
- Modified: being modified by other side
- Deleted: Deleted by other side
- NotTouched: No change on both sides

string Name

Name of the file/folder metadata

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".

string AbsolutePath

Stores the absolutePath of the file/folder metadata

string rootDir

Give the root folder selected by user

Methods

Private

static bool operator >(ComponentMeta first, ComponentMeta second)

static bool operator <(ComponentMeta first, ComponentMeta second)

The above 2 methods compare 2 ComponentMetas using their names.

.

2.3.2 FILEMETA

An instance extending from ComponentMeta describes a given file.

Methods

Public

static int ConvertToKiloByte(FileInfo fileInfo)

Return file size in Kbytes.

string getString()

Provide a string representation of the file metadata.

2.3.3 FOLDERMETA

An instance extending from ComponentMeta which describes a given folder.

Attributes

Public

Type FolderType

Type of the folder: using the Type in the ComponentMeta enum.

List<**FolderMeta**> **folders**

List of folders inside one folder.

List<**FileMeta**> **files**

List of files inside one folder.

long Size

Size of the folder.

Methods

Public

void AddFile(FileMeta file)

Add one file to the file list.

void AddFolder(FolderMeta folder)

Add one folder to the folder list.

IEnumerator<FolderMeta> GetFolders()

Get all the folders in the folder metadata.

IEnumerator<FileMeta> GetFiles()

Get all the files in the folder metadata.

String getString()

Return a string representation of the folder metadata.

2.3.4 JobDefinition

JobDefinition is a serializable abstract class used to describe the meta data for jobs. PCJob and USBJob 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.

2.3.5 PCJOB

USBJob 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

- Whenever a removable device is plugged in, if the USBJob corresponding to a PCJob is found it will set this attribute to the USBJob and consider this job 'mounted'.

bool Synchronizing

- True when synchronizing. Thie attribute is used to check if a synchronization process was interrupted.

2.3.6 USBJOB

USBJob stores the metadata of the job information that is stored on the USB.

Attributes

Public

bool Synchronizing

- True when synchronizing. Thie attribute is used to check if a synchronization process was interrupted.

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.

2.3.7 DIFFERENCES

CleanSync handles five different types of differences:

Attributes

Private

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

- Contains differences of type folder deleted.

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

- Contains differences of type folder created.

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

- Contains differences of type file deleted.

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

- Contains differences of type file created.

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

- Contains differences of type file modified.

2.2.8 Conflict

Each conflict object represents one conflict found during synchronization analysis.

Attributes

Public

enum FolderFileType

- FileConflict
- FolderVSFileConflict

- FileVSFolderConflict
- FolderVSSubFolderConflict
- SubFolderVSFolderConflict
  - 3 different kinds of conflicts are identified by CleanSync. The first one is when two files are modified in both the PC and the USB. The second one is when a folder is modified on one side while a sub-file in that folder is modified in the other side. The third kind of conflict happens when a folder is modified in one side and a sub-folder is modified on the other side.

enum UserChoice

- KeepPCUpdates
- KeepUSBUpdates
- Untouched
  - Users are given 3 choices to handle conflicts: Keep the update on the PC, keep the update on the USB and keep both updates and do not synchronize the conflict.

enum ConflictType

- New
- Modified
- Deleted

FolderMeta CurrentPCFolder

FolderMeta USBFolder

FileMeta CurrentPCFile

FileMeta USBFile

FolderFileType FolderOrFileConflictType

ConflictType PCFolderFileType

ConflictType USBFolderFileType

string Name

bool USBSelected

bool PCSelected

- The above 2 booleans represents the user's choice of resolving this conflict.

2.2.9 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

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.

2.2.10 JOBCONFIG

JobConfig is stored in every PCJob, and stores the configurations of the job on this PC. 2 configurations are currently handled: automation and auto-conflict resolving. Automation enables the job to be synchronized automatically whenever it's removable device is plugged in or whenever CleanSync starts up with the removable device plugged in. Conflict resolving allows user to specify the job to always copy either the computer or the removable device's files and folders to the other.

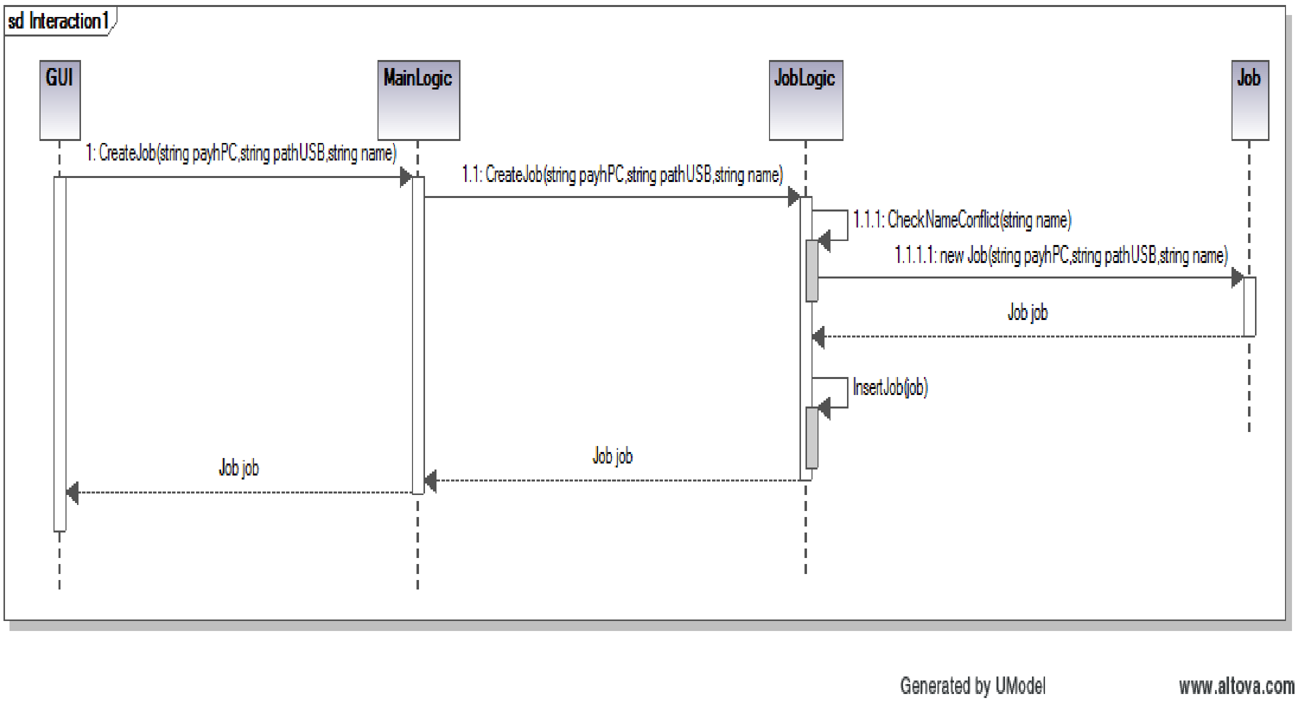
2.3 EXTERNAL COMPONENTS

CleanSync uses http://www.hardcodet.net/projects/wpf-notifyicon for balloon notification.

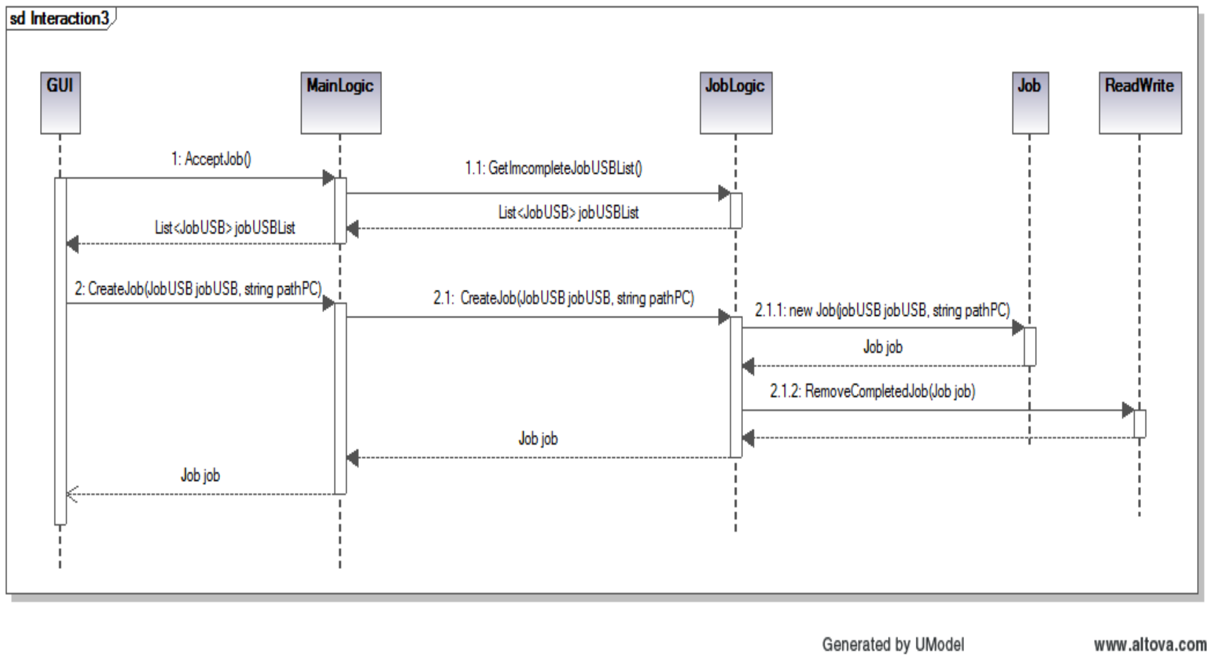
3 TECHNICAL DETAILS

3.1 SEQUENCE DIAGRAMS

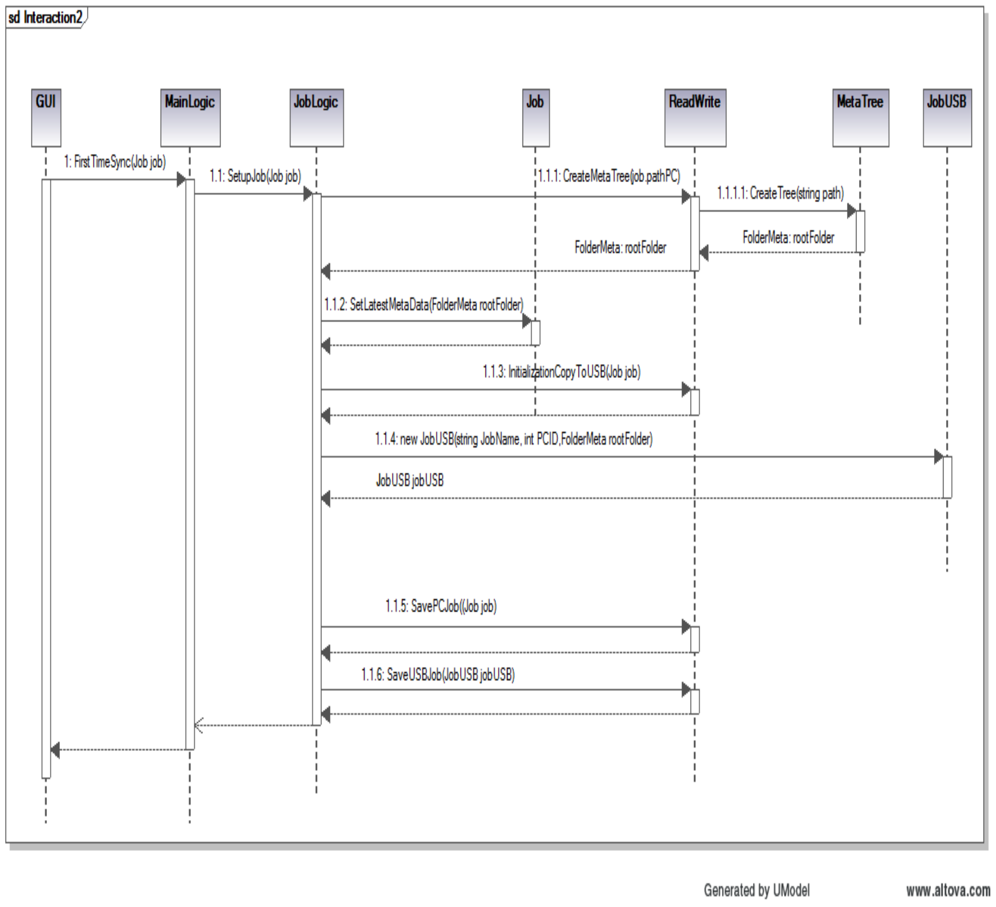
3.1.1 CREATE A JOB


sd Interaction1
GUI
MainLogic
JobLogic
Job
1: CreateJob(string payhPC,string pathUSB,string name)
1.1: CreateJob(string payhPC,string pathUSB,string name)
1.1.1: CheckNameConflict(string name)
1.1.1.1: new Job(string payhPC,string pathUSB,string name)
Job job
InsertJob(job)
Job job
Job job
Generated by UModel
www.altova.com

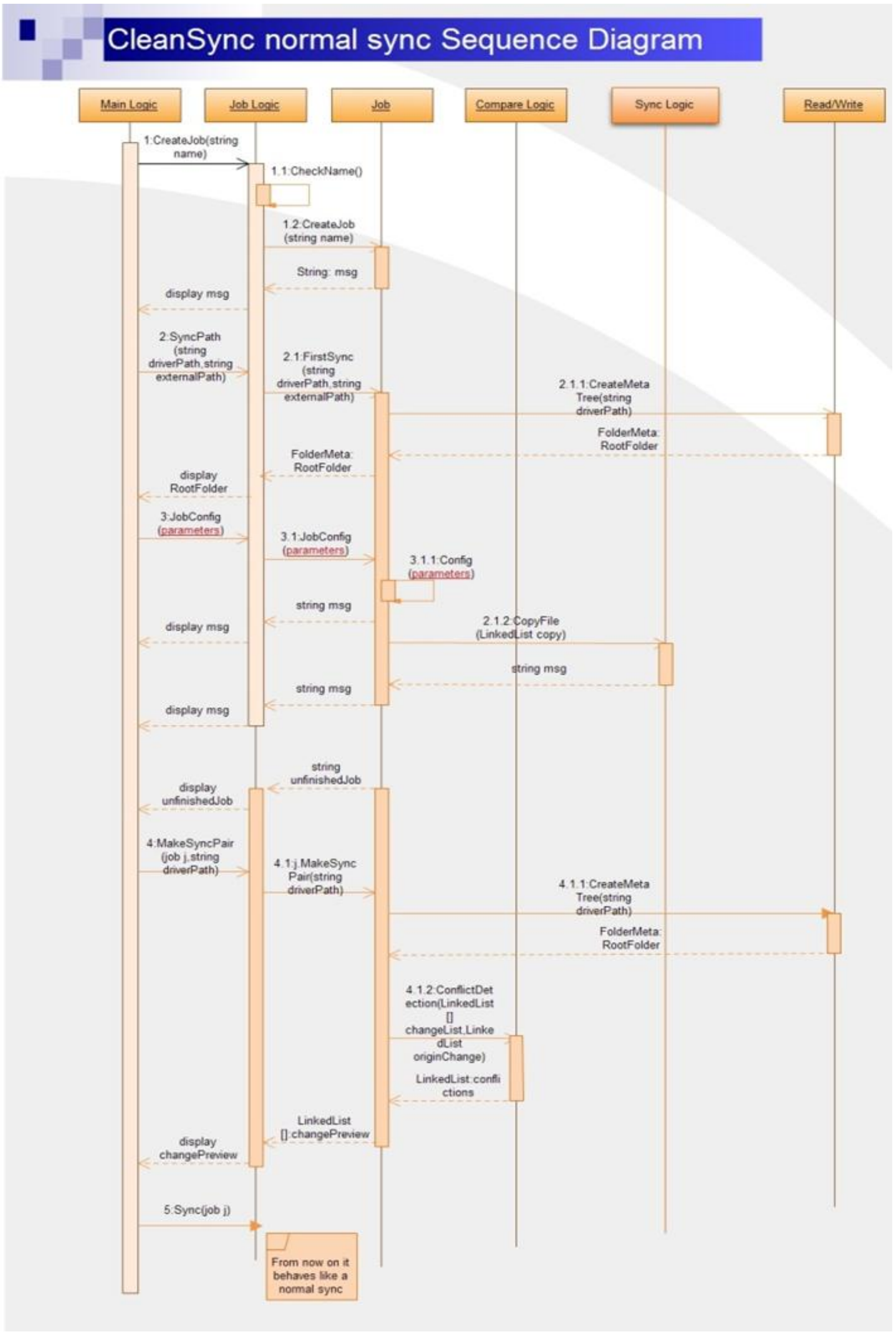
3.1.2 ACCEPT A JOB


sd Interaction3
GUI
MainLogic
JobLogic
Job
ReadWrite
1: AcceptJob()
1.1: GetImcompleteJobUSBList()
List<JobUSB> jobUSBList
List<JobUSB> jobUSBList
2: CreateJob(JobUSB jobUSB, string pathPC)
2.1: CreateJob(JobUSB jobUSB, string pathPC)
2.1.1: new Job(jobUSB jobUSB, string pathPC)
Job job
2.1.2: RemoveCompletedJob(Job job)
Job job
Job job
Generated by UModel
www.altova.com

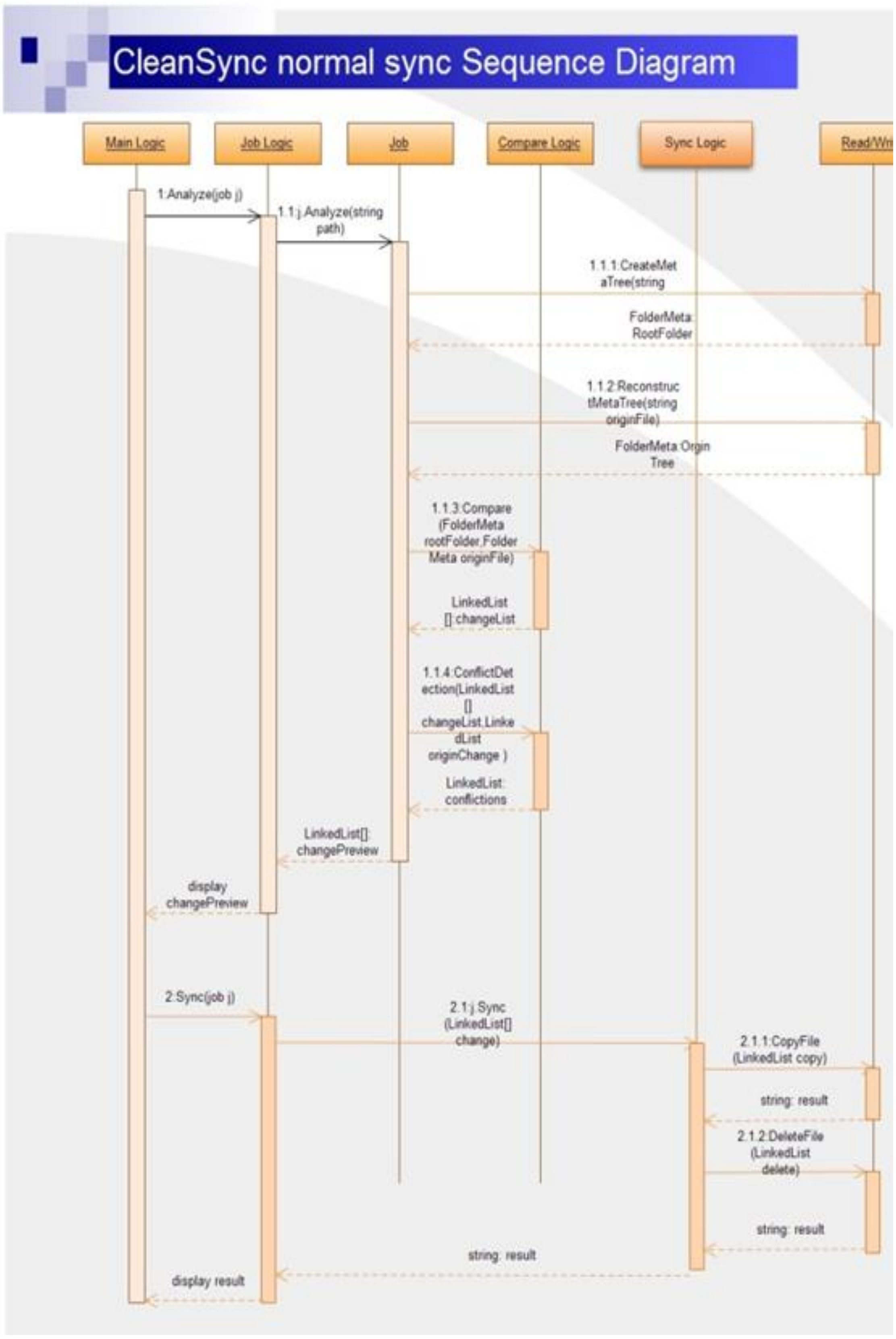
3.1.3 ACCEPT & SYNC


sd Interaction2
GUI
MainLogic
JobLogic
Job
ReadWrite
MetaTree
JobUSB
1: FirstTimeSync(Job job)
1.1: SetupJob(Job job)
1.1.1: CreateMetaTree(job.pathPC)
1.1.1.1: CreateTree(string path)
FolderMeta: rootFolder
FolderMeta: rootFolder
1.1.2: SetLatestMetaData(FolderMeta rootFolder)
1.1.3: InitializationCopyToUSB(Job job)
1.1.4: new JobUSB(string JobName, int PCID,FolderMeta rootFolder)
JobUSB jobUSB
1.1.5: SavePCJob((Job job)
1.1.6: SaveUSBJob(JobUSB jobUSB)
Generated by UModel
www.altova.com

3.1.4 FIRST SETUP


CleanSync normal sync Sequence Diagram
Main Logic
Job Logic
Job
Compare Logic
Sync Logic
Read/Write
1:CreateJob(string name)
1.1:CheckName()
1.2:CreateJob (string name)
String: msg
display msg
2:SyncPath (string driverPath,string externalPath)
2.1:FirstSync (string driverPath,string externalPath)
2.1.1:CreateMeta Tree(string driverPath)
FolderMeta: RootFolder
FolderMeta: RootFolder
display RootFolder
3:JobConfig (parameters)
3.1:JobConfig (parameters)
3.1.1:Config (parameters)
string msg
display msg
2.1.2:CopyFile (LinkedList copy)
string msg
string msg
display msg
string unfinishedJob
display unfinishedJob
4:MakeSyncPair (job j,string driverPath)
4.1:j.MakeSync Pair(string driverPath)
4.1.1:CreateMeta Tree(string driverPath)
FolderMeta: RootFolder
4.1.2:ConflictDetection(LinkedList [] changeList,LinkedList originChange)
LinkedList:conflictions
LinkedList [] changePreview
display changePreview
5:Sync(job j)
From now on it behaves like a normal sync

3.1.5 NORMAL SYNC


CleanSync normal sync Sequence Diagram
Main Logic
Job Logic
Job
Compare Logic
Sync Logic
Read/Wri
1:Analyze(job j)
1.1:j.Analyze(string path)
1.1.1:CreateMetaTree(string
FolderMeta: RootFolder
1.1.2:ReconstructMetaTree(string originFile)
FolderMeta:Orgin Tree
1.1.3:Compare (FolderMeta rootFolder,FolderMeta originFile)
LinkedList []:changeList
1.1.4:ConflictDetection(LinkedList [] changeList,LinkedList originChange)
LinkedList: conflictions
LinkedList[]: changePreview
display changePreview
2:Sync(job j)
2.1:j.Sync (LinkedList[] change)
2.1.1:CopyFile (LinkedList copy)
string: result
2.1.2:DeleteFile (LinkedList delete)
string: result
string: result
display result

3.2 IMPLEMENTATION DETAILS

3.2.1 NORMAL CLEAN SYNCHRONIZATION

Normal Clean Synchronization is invoked during synchronization when this PC is not the most recent PC to do a synchronization. In this mode, SyncLogic will first synchronize the differences on the removable device, then the differences on the PC. For synchronizing from the removable device to the computer, every temporary file will be deleted immediately upon being copied over to the computer.

3.2.2 RE - SYNCHRONIZATION

Re-Synchronization is invoked when synchronizing the computer to the USB successively. Instead of a normal synchronization where updates on both sides are updated, re-synchronization only checks what updates from the computer are to be copied over to the USB. A few cases of updates exists which is handled by SyncLogic:

- If a file or folder is created in a previously new folder, the file or folder will just be copied over to the new directory.
- If a file or folder is deleted in a previously new folder, the file or folder will just be deleted from the USB.
- If a file which is previously new is modified, it's metadata In the difference list will be updated and the modified file will be regarded as the new file to be copied over to the other computer.
- If a file which is previously new or modified, or a folder which is previously new is deleted, it will delete its data from its previous difference list, and the temporary file or folder will also be deleted.
- In all other cases, the updates found in the computer will be propagated to the removable device.

3.2.3 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.

3.2.4 DETERMINING FOLDERS TO STORE META DATA

There are two types of meta data. One is meta data for folder on computer (PC), we'll refer to this type as PCJob. The other is meta data for folder on removable device, we'll refer to this type as USBJob.

The root folder to store PCJobs is the path the Clean Sync executable resides in. All meta data are stored in a folder named "_cs_job_data" and the property of the folder is set to System Hidden.

The root folder to store USBJobs is the path of the root drive of the USBJob. All meta data are stored in folder named "_cs_job_data" and the property of the folder is set to System Hidden.

3.2.5 LOADING AND UNLOADING META DATA TO AND FROM THE HARD DISK.

We define two different types of job, one stores the job information for computer(we'll refer to this type of job as PCJob), and one stores the job information for removable device(we'll refer to this type of job as USBJob).

PCJob:

A PCJob is created when a new job is successfully created or when a job is successfully accepted. PCJobs are stored in the root_cs_job_data\JobsList where root is the current path the CleanSync.exe resides.

PCJobs are initialized and loaded by the program when the program starts up.

USBJob:

An incomplete USBJob is created when a new job is successfully created. The incomplete USBJob will be stored in the root:_CleanSync_Data__cs_job_data\incompleteJobs where root is the root drive of the removable device which user chooses to store the intermediary file. An incompete USBJob will be used to notify CleanSync that there is not fully connected job on this removable device.

When an incomplete job is accepted, the incomplete USBJob will be removed and a new USBJob will be created in the path root:_CleanSync_Data__cs_job_data\usbJobsList where root is the root drive of the removable device.

The USBDetection Class is used to handle removable device plug in. When a removable device is detected, the program will check whether the device contains "_CleanSync_Data_" folder. If yes, the program will then search for the incomplete USBJobs to display, and for USBJobs, the program will try to load the USBJob to the corresponding PCJob on this computer. For PCJobs that can be connected with a USBJob, they can proceed with Analyse or Sync functions.

4. KNOWN ISSUES

CleanSync v2.0 currently has the following known issues:

- Currently, users can only choose to either keep one of the updates in the event of conflicts. Users can neither choose to execute both updates nor negate both updates.