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 2 options: Using the one on the removable device or using the one on the computer.

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

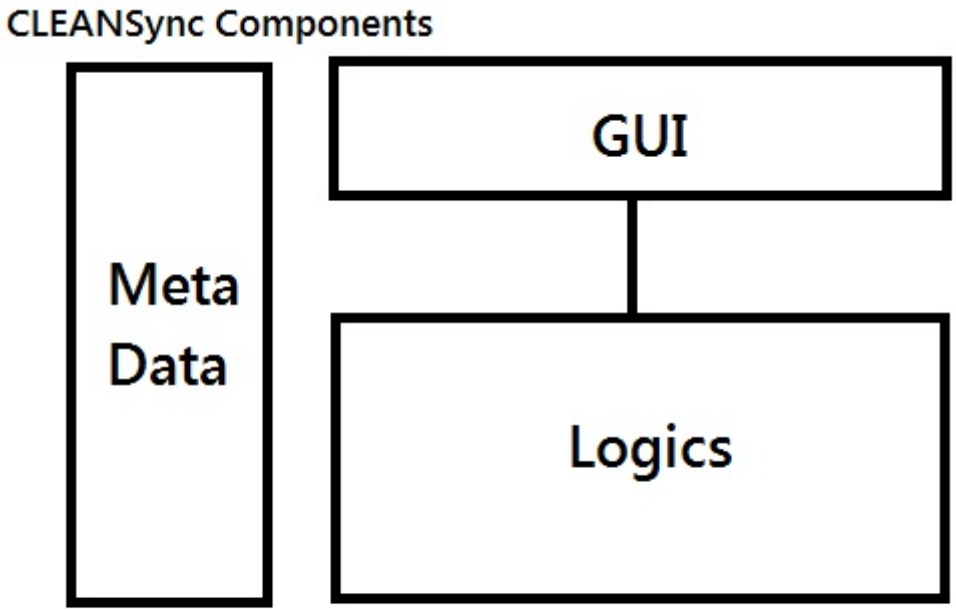
FileMeta and FolderMeta are serializable classes used to store information about files and folders respectively. Using FileMeta and FolderMeta, a directory structure can be represented.

Job

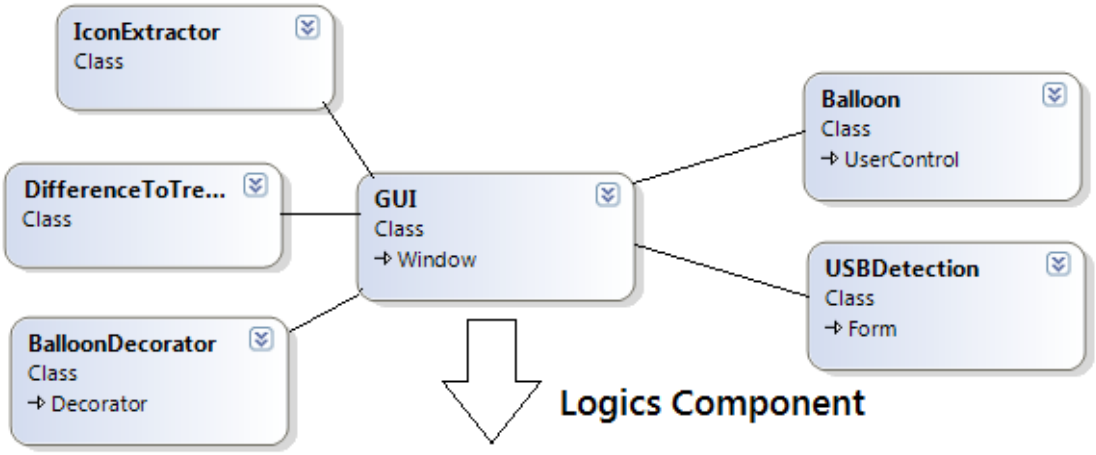
A job stores the information of the directory on both computers and the removable device. When the user wants to synchronize folders in 2 different computers, users create a job. In implementation, a job consists of 2 serializable PCJob classes which is stored in the computers and a serializable USBJob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

2.1.4 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.

2.1.5 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.

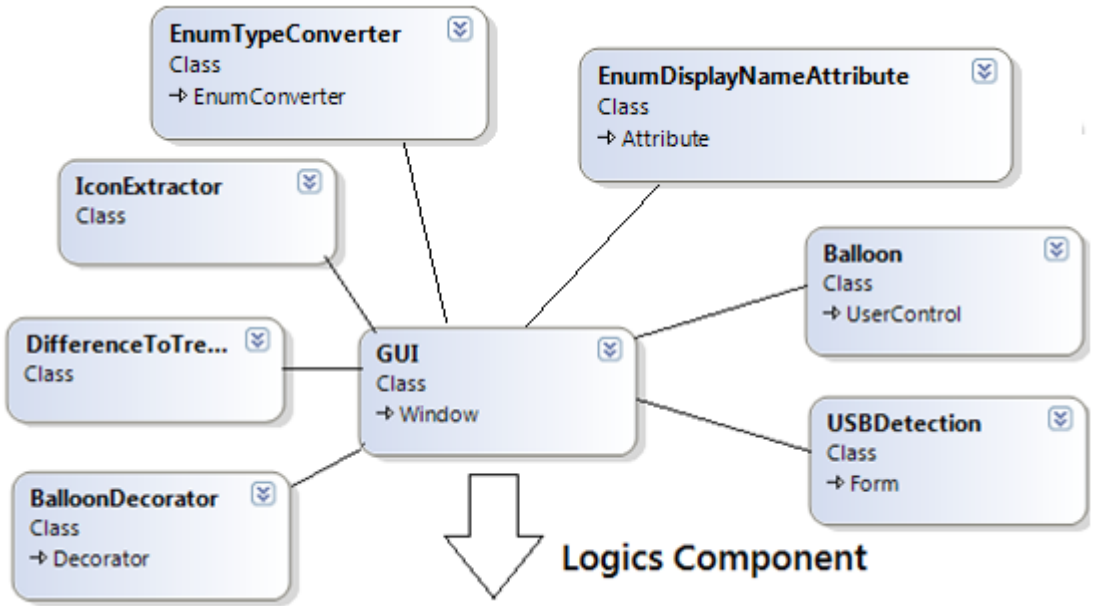
2.1.6 ENUMDISPLAYNAMEATTRIBUTE

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

2.1.7 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 CONFLICTHANDLER

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

2.2.7 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).

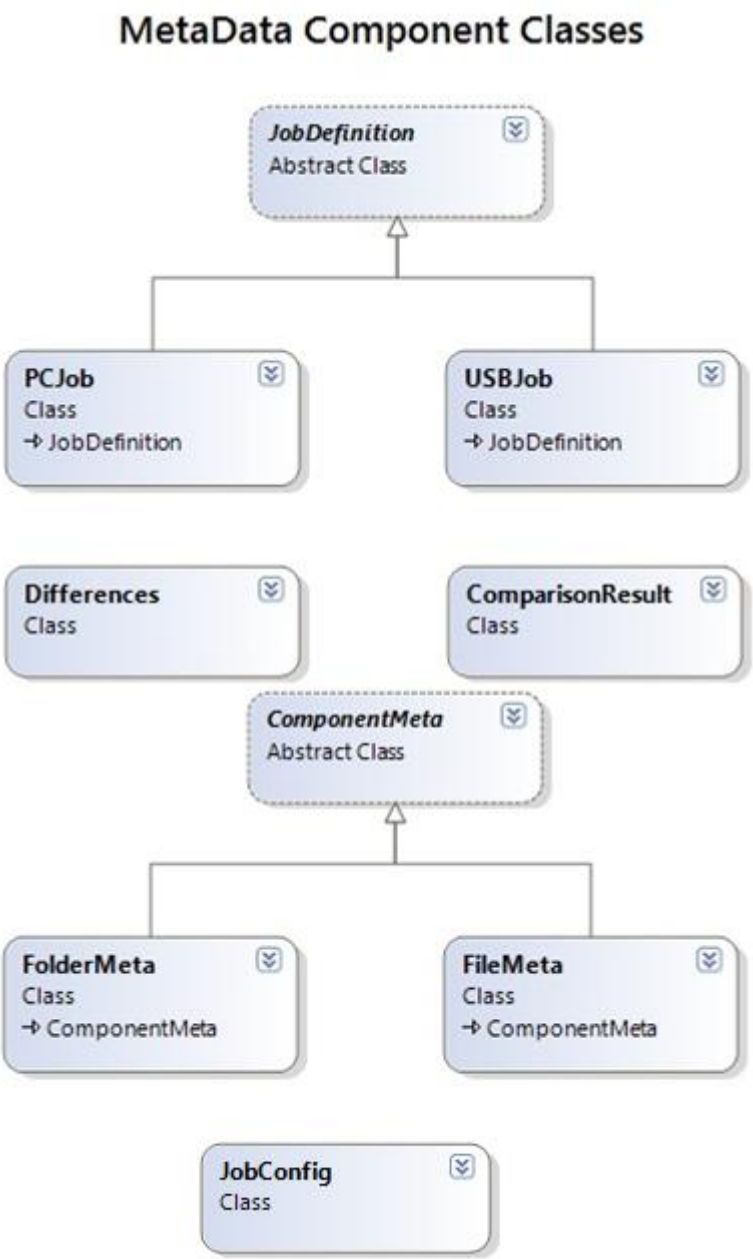
static void RestoreInterruptedUSB(PCJob pcJob)

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



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.

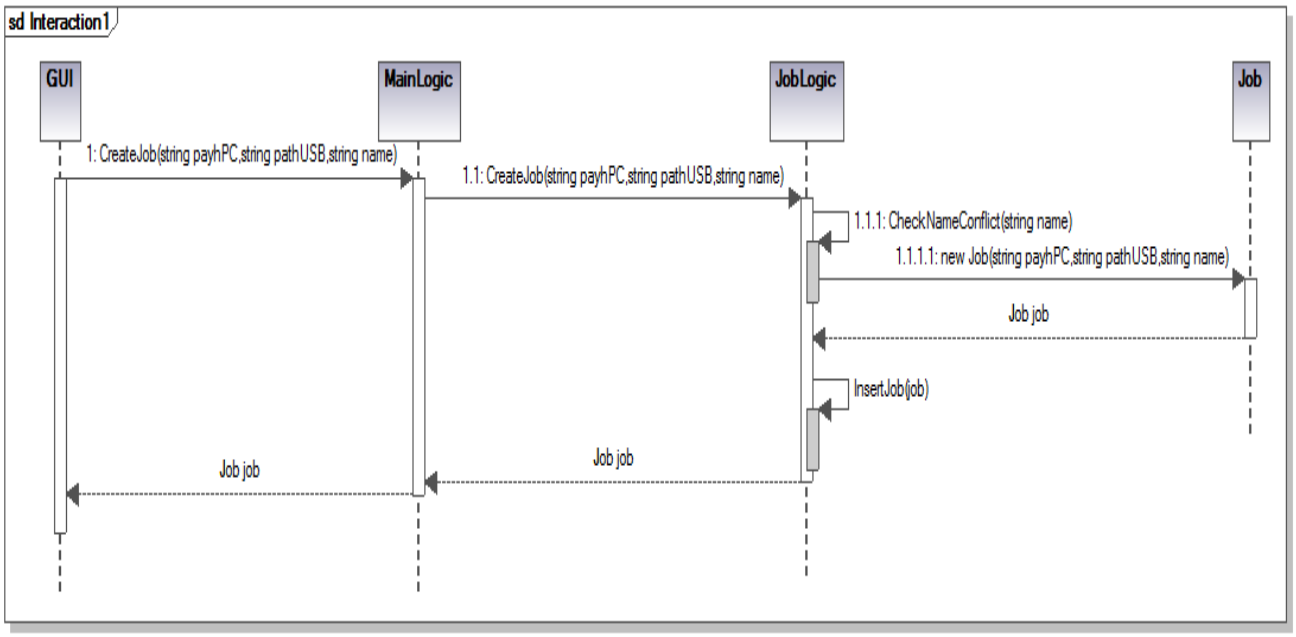
2.3 EXTERNAL COMPONENTS

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

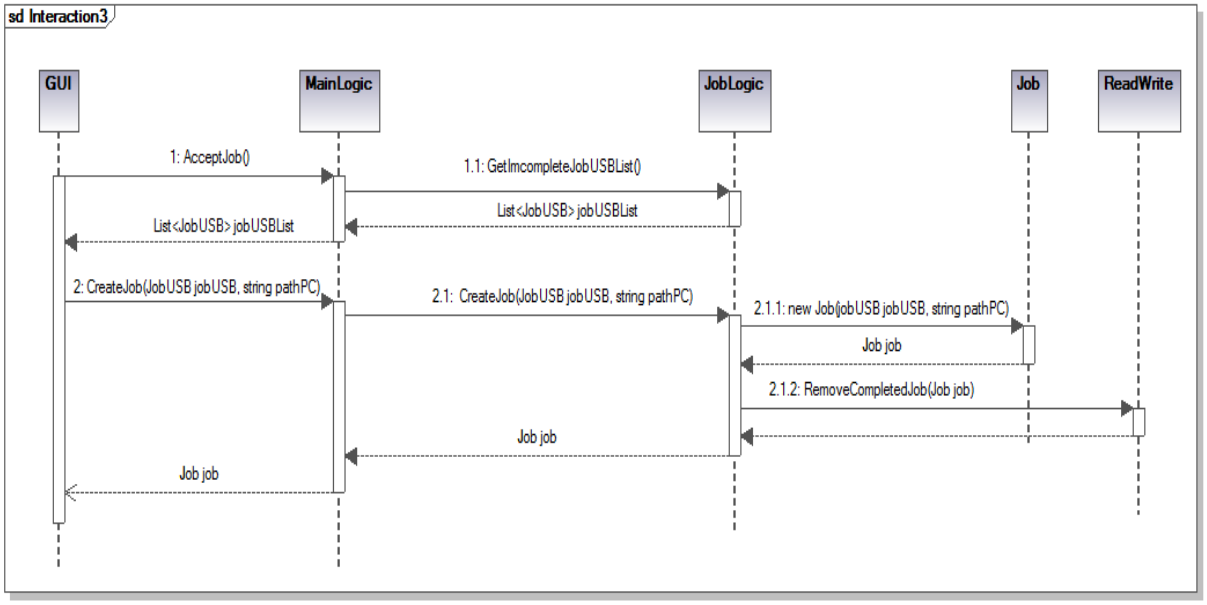
3 TECHNICAL DETAILS

3.1 SEQUENCE DIAGRAMS

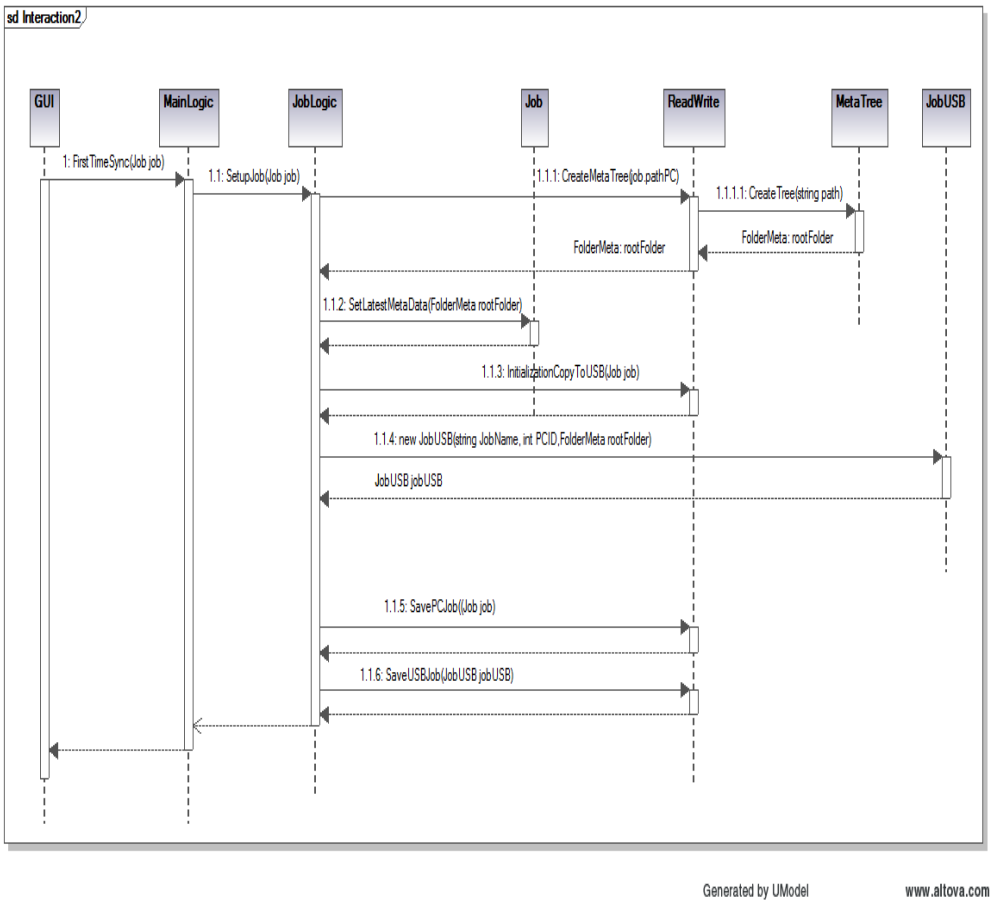
3.1.1 CREATE A JOB



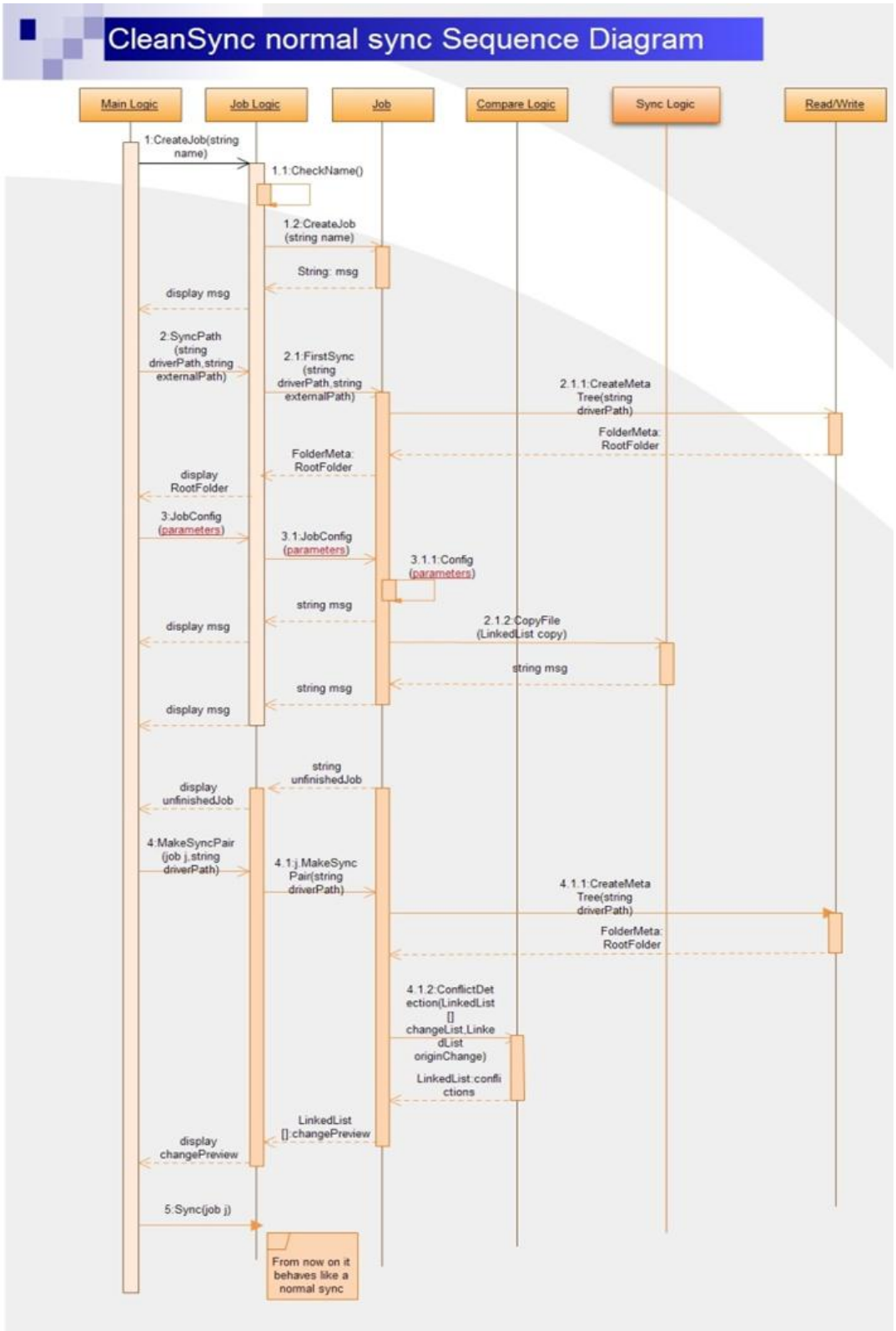
3.1.2 ACCEPT A JOB



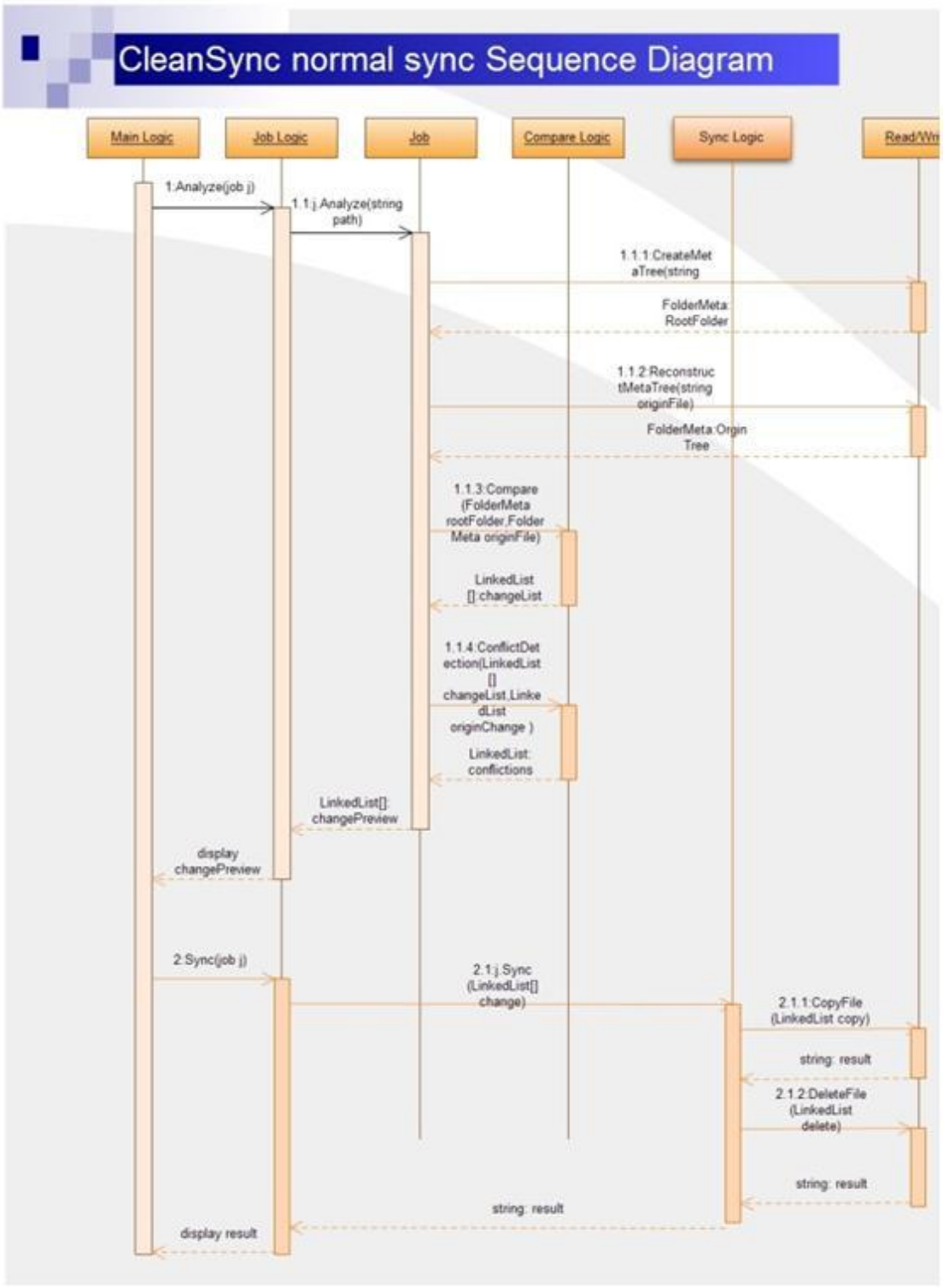
3.1.3 ACCEPT & SYNC



3.1.4 FIRST SETUP



3.1.5 NORMAL SYNC



3.2 IMPLEMENTATION DETAILS

3.2.1 CONFLICT HANDLING

There exist 8 kinds of possible conflicts listed below:

1: New File VS New File Conflict.

2: Modified File VS Modified File Conflict.

3: Modified File VS Deleted File Conflict.

4: Deleted File VS Deleted File Conflict.

5: Modified File VS Deleted Root Folder Conflict.

6: Deleted File VS Deleted Root Folder Conflict.

7: Deleted Folder VS Deleted Root Folder Conflict.

8: New Folder VS New Folder Conflict

Two things need to be noticed here:

Firstly, conflicts like 'New File VS Modified File' or 'New File VS Deleted File' are logically incorrect and are not included in the lists. For our implementation, we have avoided these kinds of conflicts.

Secondly, 'New File VS Deleted Root Folder' and 'New Folder VS Deleted Root Folder' are not considered as conflicts. We always keep the files and folders if they are newly created on the remote PC but deleted on current PC for safety issues.

Based on the above understanding, we created two methods:

1: DetectFileConflict(…).

2: DetectFolderConflict(…).

The first method DetectFileConflict will retrieve the NewFileList, ModifiedFileList, DeletedFileList from PCDifferences(current PC changes) and USBDifferences(remote PC changes), and examines conflicts type 1 to type 6 in the above conflicts list.

For type 1 conflicts (New File VS New File): DetectFileConflict compares the two NewFileLists, if there exists files of same names but different size and modification time, a new Conflict object is created and added into the conflicts list. However, if they the two files has the same modification time and file size, we remove them from the filelist entry (Users may copy the files to the remote PC, in this case the new files and identical and should not be reported as conflicts).

For type 2 conflicts (Modified File VS Modified File) and type 3 conflicts(Modified File VS Deleted File) : DetectFileConflict will retrieve the corresponding filelist, check name conflicts and add it to the conflict list when a conflict is detected.

For type 4 conflicts (Deleted File VS Deleted File): when such conflict is detected, the both fileMeta is removed from the fileList entry, since when both PC agree to delete the file, it is not a conflict.

For type 5 conflicts (Modified File VS Deleted Root Folder Conflict): our method will compare the modifiedList with the DeletedFolderList. If a modified file belongs to a deleted folder, it is an conflict.

For type 6 conflicts (Deleted file VS Deleted Root Folder Conflict): deletedFileList and DeletedFolderList are compared. If a deleted file belongs to a deleted folder, we remove both file metadata entry from the deleteFileList and DeleteFolderList, since it is not a real conflict.

The second method DetectFodlerConflict will detect type 7 and type 8 conflict.

For type 7 conflicts (Deleted Folder VS Deleted Root Folder Conflict): it will compare the two deletedFolderLists. If a deleted folder belongs to a deleted root folder, both folder entries are removed from the deletedFolderLists.

For type 8 conflicts (New Folder VS New Folder Conflict): For this particular case, we do more than keep only one new folder and delete the other folder. Instead, we try to merge the two folders.We will check the two new folders' subfiles and detect the newFile VS newFile conflicts. Do a recursive check for the subfolders. So finally, the New Folder VS New Folder Conflict is broken down into multiple New File VS New File Conflicts.

Conflict Handling Algorithm:

After the Conflict Detection logic, we will have a list of conflicts:

1: FileConflicts.

1.1 Modified File VS Modifed File Conflict

1.2 Modified File Vs Deleted File Conflcit.

1.3 New File VS New File Conflict.

2: FileVSFolderConflict.

2.1 Modified File VS Deleted Root Folder.

(note that the rest 'conflicts' are already handled during conflict detection.)

For FileConflicts, we first get the user choice(keep PC changes or keep USB changes). For instance, if user choose to keep PC changes, we will then delete the corresponding file entry in USB changes, so the USB changes will not take effect and PC changes will be kept and propagated to the remote PC.

For FileVSFolderConflict, the resolution is similar, if user chooses to keep the deleted root folder, the file entry is deleted. If the file is chosen to be kept, we will remove the file entry from the root folder(Upon deletion, the program will not delete the chosen file).

3.2.2 CLEAN SYNCHRONIZATION

Clean Synchronization does synchronization in SyncLogic based on trees. Calling methods provide a ComprisonResult object, and SyncLogic takes the 2 differences in the comparison result and converts them to tree form. After that, synchronization is performed based on the trees generated from the differences. SyncLogic does synchronization differently for normal synchronization and re-synchronization.

3.2.2.1 NORMAL SYNCHRONIZATION

Normal Synchronization is invoked when the previous synchronization happened on the other computer. SyncLogic will then propagate the required updates to both the PC and the removable device. To allow for restoration of previous state and backing up of deleted files and folders, normal synchronization works as follows:

1. Extract changes from the PC to a temporary USB folder.
2. Copy changes from the USB to a temporary PC folder.
3. Propagate the changes from the temporary PC folder to the folder in the PC. During this time, any files and folders deleted from the PC will be moved to the backup folder.
4. Move the old changes from the main USB folder to another temporary USB folder.
5. Move the new changes from the temporary USB folder to the main USB folder.

After the correct changes are in place, the temporary folders' contents will be cleared.

If during the synchronization, the USB is plugged out or the user cancels the process, An exception will be thrown and SyncLogic will move the folders back to the correct locations, depending on where the exception was thrown.

3.2.2.2 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. SyncLogic defines the differences on the USB as the old differences from this computer, and the differences on the USB as the new differences. SyncLogic will compare each file and

folder between the old and new differences and make changes to the old differences accordingly:

Comparing contents between 2 folders which are modified:

If a folder exists in both differences:

1. The old folder is **deleted**, the new folder is **new**
    a. This means the folder is previously deleted, and now a new folder with the same name is created. The folder's difference will then be set to modified, and the sub folders and files will be compared recursively.
2. The old folder is **new**, the new folder is **deleted**
    a. The folder new folder will be removed, and no changes need to be made to the other computer, so the difference will be removed.
3. The old folder is **new**, the new folder is **modified**
    a. In this case, a special method will be called to check accordingly the subfiles and subfolders. If a subfile or subfolder is now deleted, it will be deleted from the differences. Else, any changes will be updated
4. The old folder is **modified**, the new folder is **deleted**
    a. A special method will also be called, to check if the there are any new files that will also be deleted from the differences. Also, previously deleted files and folders will also be added back to the differences.
5. Both folders are **modified**
    a. The folders will be checked recursively again.

If a file exists in both differences:

1. The old file is **deleted**, the new file is **new**
    a. A previously deleted file has been created again, thus, the difference will be set to modified.
2. The old file is **new**, the new file is **deleted**
    a. The file will be removed from the differences.
3. The old file is **new**, the new file is **modified**
    a. The new file will be set as new again.
4. The old file is **modified**, the new file is **modified**
    a. The file will still be modified
5. The old file is **modified**, the new file is **deleted**
    a. The difference will be set to deleted

To allow for restoration of previous state and backing up of deleted files and folders, normal synchronization works as follows:

1. Copy all the changes from the USB to a temporary folder

2. Extract the changes from the PC to a temporary folder on the USB
3. Do the resynchronization to the main USB folder.
4. Delete the temporary USB folder.

If the synchronization is interrupted or cancelled, the temporary folders will be flushed except for the data on the temporary folder containing the original differences which will be set back as the original differences.

3.2.3 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 directory that stores the root folder on the computer is on the application data folder of the user, and on the removable device is the root folder of the drive. On the computer, a folder "CleanSync" will be created and on the removable device,

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.

3.2.4 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 version 2.0 currently has these improvements that we intend to incorporate into future versions:

- Improvement of restoration of interrupted synchronization on the PC's folder.

  If a synchronization process was stopped due do the program being exited (i.e power shortage, etc), CleanSync will attempt to restore the folder the next time users synchronize the folder again. However, while modified and deleted files can be restored to the PC, new files and folders that were already copied to the folder will not be removed from the folder. They will be detected as new files the next time synchronization occurs. Not critical since no files or folders will be lost, but it does not do a full restore. Future versions can include an algorithm that removes new files off the folder.

  During resynchronization, the whole folder on the USB is copied over to a temp folder, taking up a lot of time. In the future, a method will be incorporated which negates the need to copy the whole folder again.

- During previewing after analyzing changes, if there are no changes, it can be stated explicitly so on the GUI window. Now, it only shows an empty tree view.
- SyncLogic has been moved from list-based synchronization to tree-based. Future versions will also migrate conflicts detection, and store differences in trees, negating the need for the Difference class altogether.