CLEANSync

Developer's Guide

CS3215 Team9: 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 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. Components

2.1 Logic Components

2.1.1 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.

2.1.2 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.

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)**

2.1.3 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.

Methods

CompareLogic currently handles three different jobs: Checking for differences between folders, checking for conflicts between 2 differences and converting the differences from list form to a tree form.

Checking for differences
Public

- **Differences ConvertFolderMetaToDifferences(FolderMeta folderMeta)**
- **Differences CompareDirectories(FolderMeta newTree, FolderMeta oldTree)**

Private

- **void CompareDirectories(FolderMeta newTree, FolderMeta oldTree, Differences differences)**
- **void CompareFiles(FolderMeta newFolder, FolderMeta oldFolder, Differences differences)**
- **void CompareFolders(FolderMeta newFolder, FolderMeta oldFolder, Differences differences)**

Checking for conflicts
Public

- **List<Conflicts> ComparePCwithUSB(Differences USBFoldersAndFiles, Differences PCFoldersAndFiles)**

Private

- **void DetectDeletedFolderConflict(List<Conflicts> conflicts, List<FolderMeta> folderList, Differences differences, int PCorUSBIndex)**
- **void DetectNewAndModifiedFileConflict(List<Conflicts> conflicts,Differences PCDifferences, Differences USBDifferences)**
- **void DetectDeletedFileConflict(List<Conflicts> conflicts,Differences PCDifferences, Differences USBDifferences)**
- **void DetectNewAndModifiedFileConflict(List<Conflicts> conflicts, List<FileMeta> fileList, Differences USBDifferences, Conflicts.ConflictType type)**
- **void DetectNewFolderConflictWithDeletedFolderList(List<Conflicts> conflicts, List<FolderMeta> newFolderList, List<FolderMeta> deletedFolderList, int PCorUSBIndex)**
- **void DetectFileConflictWithDeletedFolderList(List<Conflicts> conflicts,Conflicts.ConflictType type, List<FileMeta> fileList, List<FolderMeta> deletedFolderList, int PCorUSBIndex)**
- **void DetectNewFolderConflictWithNewFolderList(List<Conflicts> conflicts, Differences PCDifferences, Differences USBDifferences)**

Converting list to tree

These methods are used for converting differences from list to tree form. As the differences consists of lists of ComponentMeta, these methods will use these ComponentMeta to construct a tree of folder and component meta which contains these differences only.

Public

- **FolderMeta ConvertDifferencesToTreeStructure(FolderMeta root, Differences differences)**
- **void AppendFileListToFileList(List<FileMeta> fileList, List<FileMeta> baseList)**
- **void AppendFolderListToFolderList(List<FolderMeta> folderList, List<FolderMeta> baseList)**

Private

- **void RemoveFile(FileMeta fileToBeRemoved, List<FileMeta> fileList)**
- **void RemoveFolder(FolderMeta folderToBeRemoved, List<FolderMeta> folderList)**
- **FileMeta CheckFileInList(FileMeta fileToBeChecked, List<FileMeta> fileList)**
- **FolderMeta CheckFolderInList(FolderMeta folderToBeChecked, List<FolderMeta> folderList)**
- **void ConvertSubFolders(FolderMeta root, Differences differences, List<FolderMeta> subFolders)**
- **void AddDeletedFilesToRootFileList(FolderMeta root, List<FileMeta> deletedFileList)**
- **void RemoveUntouchedFilesFromRootFileList(FolderMeta root, Differences differences, List<FileMeta> subFiles)**
- **static void ClearRootUnTouchedFolders(FolderMeta root)**
- **void ClearFileList(FolderMeta root, List<FileMeta> fileList, ComponentMeta.Type type)**

- **void ClearFolderList(FolderMeta root, List<FolderMeta> folderList, ComponentMeta.Type type)**
- **bool checkFileExistence(FileMeta file, Differences differences)**
- **bool checkFolderExistence(FolderMeta folder, Differences differences)**

2.1.4 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.

Constructor

- **SyncLogic()**

Methods
Public

In SyncLogic there is only one method called by external classes, which is to do 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> 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> 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.
- **void RemoveNullComponentsFolders(PCJob pcJob, List<FolderMeta> folders)**
  - This method does not change the name of any folder on the hard disk. Only the null components on the list are removed.

2.1.5 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.

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)**

2.1.6 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"**

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)**

2.1.7 ConflictHandler

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

Methods
Public

- **public ComparisonResult HandleConflicts(ComparisonResult comparisonResult)**

Private

- **void HandleFileConflict(Differences PCDifferences,Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void HandleFolderConflict(Differences PCDifferences,Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void HandleFolderVSSubFolderConflict(Differences PCDifferences, Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void HandleSubFolderVSFolderConflict(Differences PCDifferences, Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void HandleFileVSFolderConflict(Differences PCDifferences, Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void HandleFolderVSFileConflict(Differences PCDifferences, Differences USBDifferences, Conflicts conflict, Conflicts.UserChoice userChoice)**
- **void DeleteFolderFromRootFolder(FolderMeta folderToBeDeleted, FolderMeta baseFolder)**
- **void DeleteFileFromRootFolder(FileMeta fileToBeDeleted, FolderMeta baseFolder)**
- **void DeleteDifferencesFileEntry(Conflicts.ConflictType conflictType,FileMeta FileToBeUpdated, Differences differencesToBeUpdated)**
- **void DeleteDifferencesFolderEntry(Conflicts.ConflictType conflictType,FolderMeta FolderToBeUpdated, Differences differencesToBeUpdated)**

2.1.7 DifferenceToTreeConvertor

DifferenceToTreeConvertor converts a difference into tree form for display purposes.

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.

2.2 Metadata

2.2.1 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

Constructor

- **ComponentMeta(string path, string rootDir)**

Methods

Public

- **static bool operator >(ComponentMeta first, ComponentMeta second)**
    - Compare if the first component metadata is larger than second.
- **static bool operator <(ComponentMeta first, ComponentMeta second)**
    - Compare if the first component metadata is smaller than second.

2.2.2 FileMeta

An instance extends from ComponentMeta which specifies the file informations.

Attributes

Public

- **Type FileType**
    - Type of the file: using the Type in the 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

- **FileMeta(string path, string rootDir) : base(path, rootDir)**
    - Inheritated from the ComponentMata.
- **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.
- **string getString()**
    - Provide a string representation of the file metadata.

2.2.3 FolderMeta

An instance extends from ComponentMeta which specifies the folder informations.

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.

Constructor

- **FolderMeta(string path, string rootDir)**
    - Inheritated from the ComponentMata.
- **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.
- **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.

2.2.4 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.

2.2.5 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**

Methods
Public

- **USBJob GetUsbJob()**
- **void SetUsbJob(USBJob usb)**

2.2.6 USBJob

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.

2.2.7 Differences

CLEANSync handles five different types of differences:
1. Folder created.
2. Folder deleted.
3. File created.
4. File deleted.
5. File modified.

This class stores the differences between 2 folders.

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.

Methods
Public

- **void AddDeletedFolderDifference(FolderMeta deletedFolder)**
- **void AddNewFolderDifference(FolderMeta newFolder)**
- **void AddNewFileDifference(FileMeta newFile)**
- **void AddDeletedFileDifference(FileMeta newFile)**
- **void AddModifiedFileDifference(FileMeta newFile)**
- **List<FolderMeta> getNewFolderList()**
- **List<FolderMeta> getDeletedFolderList()**
- **List<FileMeta> getDeletedFileList()**
- **List<FileMeta> getNewFileList()**
- **List<FileMeta> getModifiedFileList()**
- **void removeFolderFromDeletedFolderList(FolderMeta folder)**
- **void removeFolderFromNewFolderList(FolderMeta folder)**
- **void removeFileFromNewFileList(FileMeta file)**
- **void removeFileFromModifiedFileList(FileMeta file)**
- **void removeFileFromDeletedFileList(FileMeta file)**
- **override string ToString()**

Private

- **void DuplicateFolderList(List<FolderMeta> newList, List<FolderMeta> baseList)**
- **void DuplicateFileList(List<FileMeta> newList, List<FileMeta> baseList)**

2.2.8 Conflict

Defines the basic object of conflicts between files.

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**
- **ConflictComplexity Complexity**
- **string Name**
- **bool USBSelected**
- **bool PCSelected**

Constructors

- **Conflicts(FolderFileType type,FolderMeta currentPCFolder, FolderMeta USBFolder, ConflictType PCFolderFileType, ConflictType USBFolderFileType, ConflictComplexity complexity)**
- **Conflicts(FolderFileType type,FolderMeta currentPCFolder, FileMeta USBFile, ConflictType PCFolderFileType, ConflictType USBFolderFileType, ConflictComplexity complexity)**
- **Conflicts(FolderFileType type, FileMeta currentPCFile, FolderMeta USBFolder, ConflictType PCFolderFileType, ConflictType USBFolderFileType, ConflictComplexity complexity)**
- **Conflicts(FolderFileType type,FileMeta currentPCFile, FileMeta USBFile, ConflictType PCFolderFileType, ConflictType USBFolderFileType, ConflictComplexity complexity)**

Methods
Public

- **override string ToString()**
- **Conflicts.UserChoice getUserChoice()**

Private

- **string GetName()**

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.

Constructors

- **ComparisonResult(Differences USBDifferences, Differences PCDifferences, List<Conflicts> conflictList)**

Methods
Public

- **List<string> ConvertComparisonResultToListOfString(ComparisonResult comparisonResult)**

Private

- **string ConflictlistToString(List<Conflicts> conflictList)**

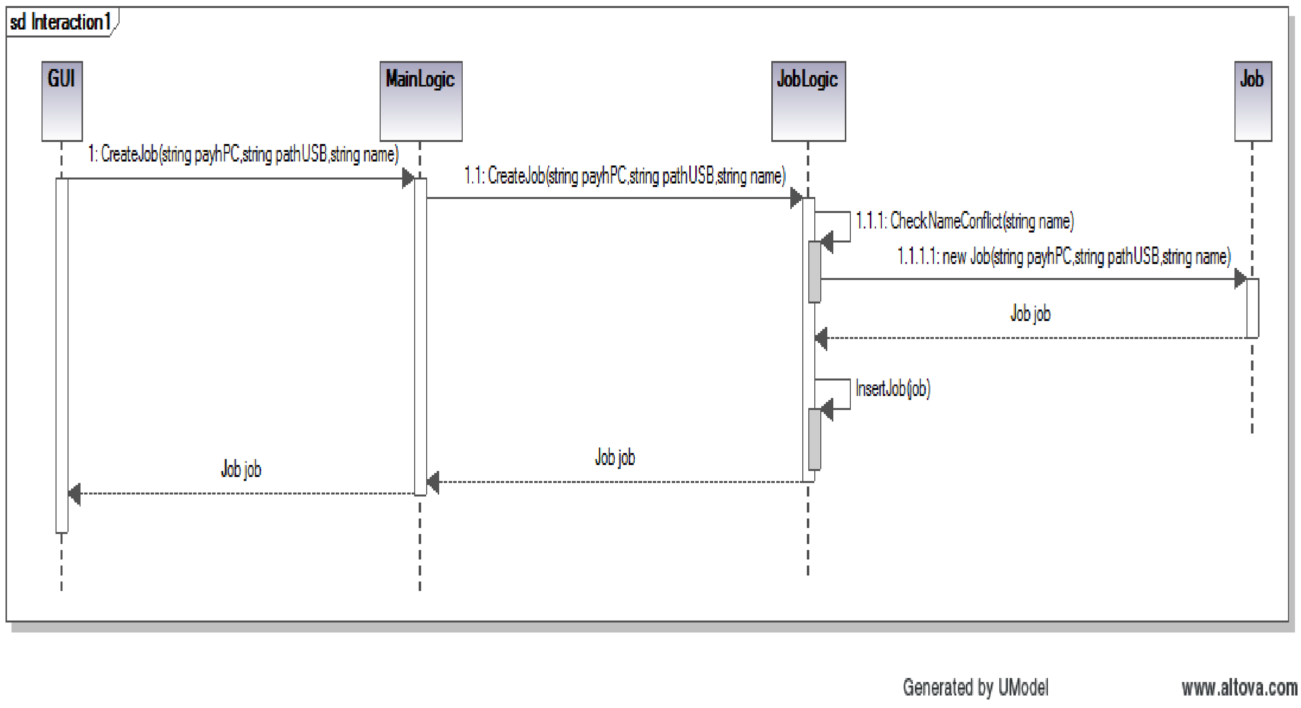
2.3 GUI

GUI is implemented using WPF.

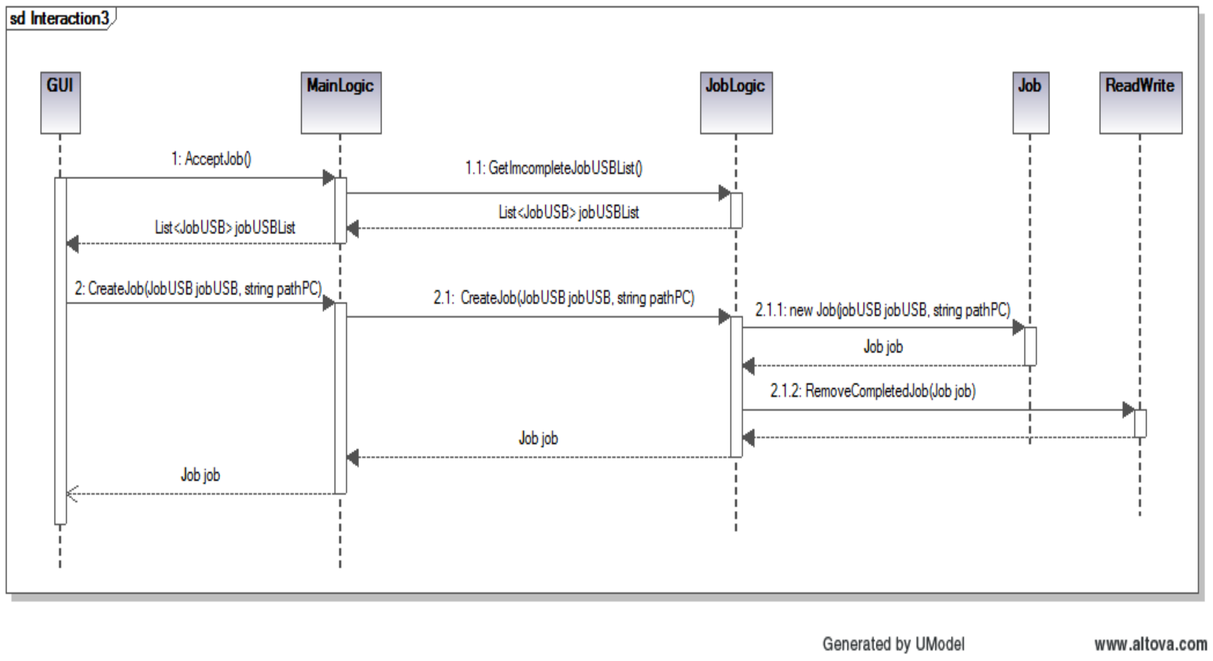
3 Technical Details

3.1 Sequence Diagrams

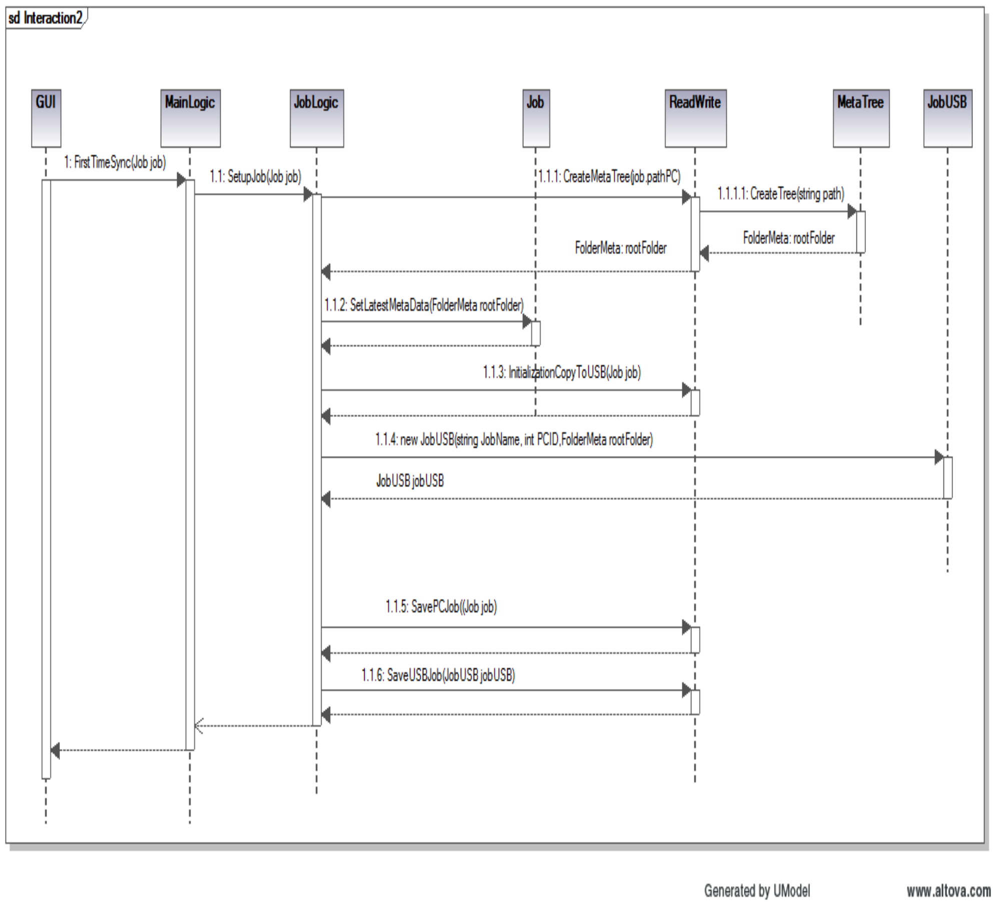
3.1.1 Create a job


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

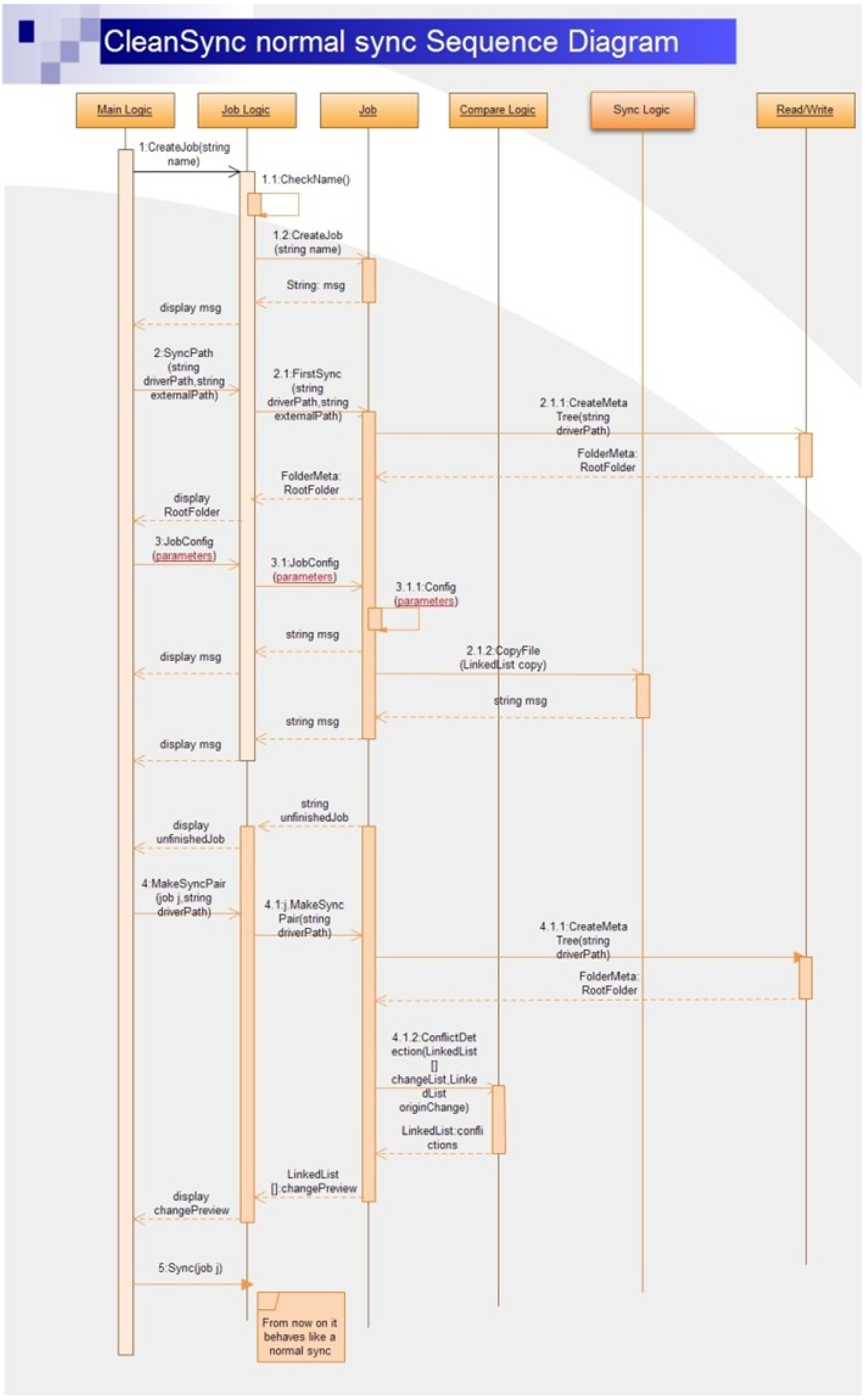
3.1.2 Accept a job


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

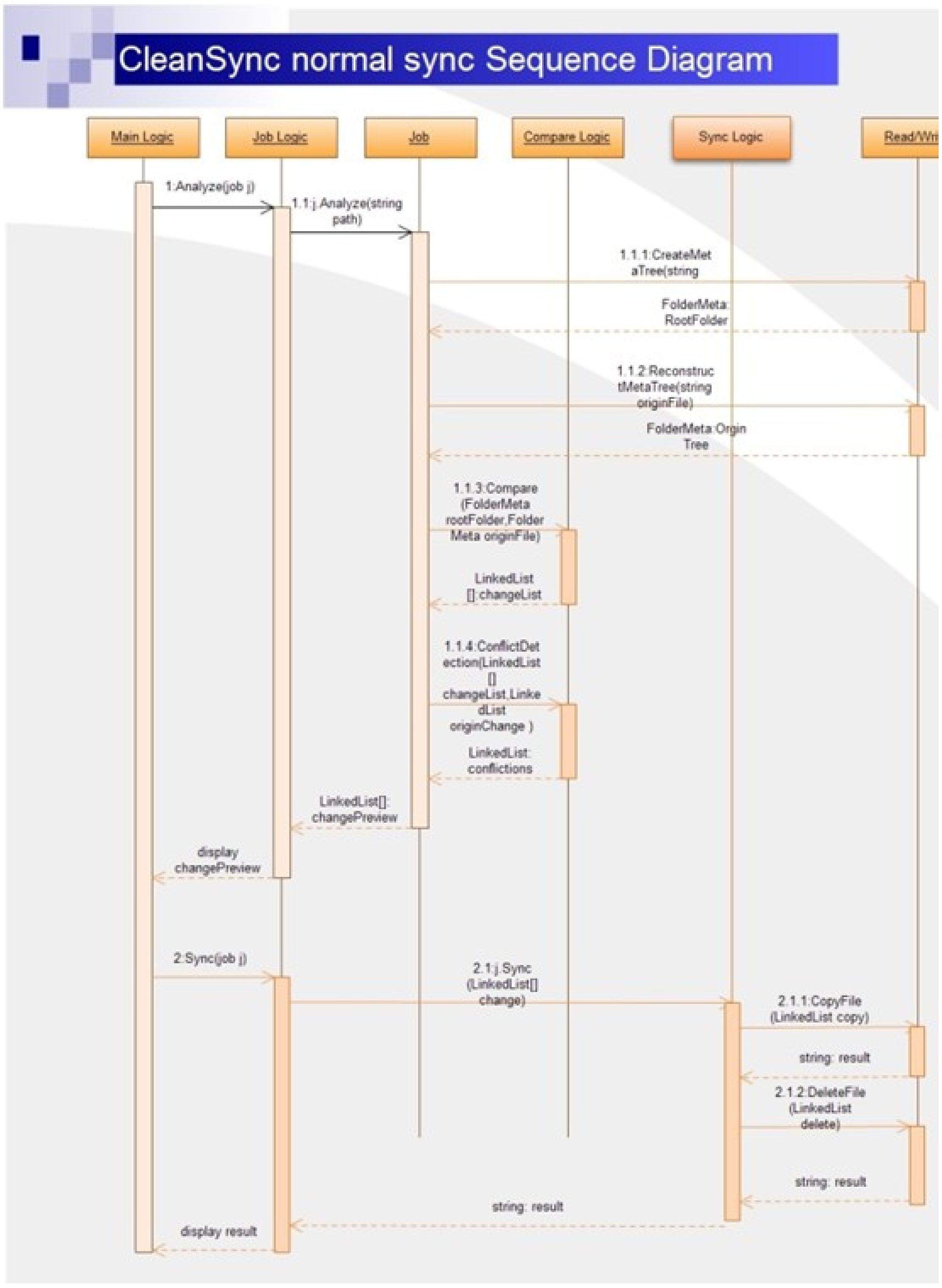
3.1.3 Accept & Sync


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 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.

3.2.2 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.

4 Known Issues

CLEANSync v0.9 currently has the following known issues:

- Users can currently set a folder from the PC's drive as the USB folder. When this happens, the job can never be completed since the computer will always detect it as incomplete. Also, the next time it loads the USB's job will not be detected as CLEANSync only scans external drives for USB job. This leads to a USB job lost on the hard disk.

- An incomplete USBJob can be corrupted if it is plugged in after CLEANSync has started, and a new job is created with the same name as the USB job. Doing the first synchronization causes the USB job to be written over.
- If the PC is the most recent PC to synchronize with the USB, analysis of comparison results does not detect and the preview window will display the wrong preview, as if it is a normal synchronization.
- Re-synchronization with the computer after first synchronization before the other computer accepts the job is not possible.
- Folder and file authorization is detected but is not handled. If a file or folder which is protected but is changed and CLEANSync attempts to modify or copy it, the program will crash.
- Ａ rare bug may occur during conflicts resolving, between a deleted folder and a new subfolder in the computer. If the user chooses to delete the parent folder except for the new subfolder, the new subfolder will still be deleted in the other computer.