SyncSharp: Plug & Sync

File Synchronization Software

Developer Guide V0.9

3/18/2010

Azhar Bin Mohamed Yasin
Loh Jianxiong Christopher
Hong Lei
Guo Jiayuan
Tian Shuang
Tan Yewkang

TABLE OF CONTENTS

Chapter 1 Introduction

1.1 What is SyncSharp 03
1.2 SyncSharp Features 04
1.3 System Requirements 05
1.4 Support and Feedback 05

Chapter 2 Developers Guide

Part 1: Design Methodology
2.1.1 Top-Down Incremental Compilation Flow 06

Part 2: Design Flow
2.2.1 System Architecture 07
2.2.2 Domain Model Analysis 09
2.2.3 Sequence Diagram 10
2.2.4 Class Diagram 11
2.2.5 Use cases 12
2.2.6 Activity Diagram 18
2.2.6.1 Detector Activity Diagram and Algorithm 18
2.2.6.2 Reconciler Activity Diagram and Algorithm 22
2.2.6.2.1 Handling File Name Collision Conflicts 23

Chapter 3 Glossary 27

Chapter 1 Introduction

1.1 What is SyncSharp

~~This documentation is written in response to develop a file (and folder) synchronization tool for CS3215 Software engineering Project module.~~

File synchronization tools are used to synchronize files and folders across multiple computers. Users are able to modify and update files in two or more locations through certain rules. Most synchronization tools provide users with one-way sync, where files and folders are copied in one direction only, while some provide two-way sync, where files and folders are replicated in both locations.

However, most of the sync tools that are available in the market required installation which may considered as a hassle to some users. Not all computers are pre-installed with file synchronization software, and users may not be granted with administrative rights to install software, and this poses problems for users who need to perform file synchronization.

In order to solve the abovementioned problems, our team has developed a file synchronization tool called SyncSharp which provides users with a streamline file synchronization operation and installation free application.

1.2 SyncSharp Features

SyncSharp allows users to sync files and folders between multiple computers through a USB device. Our product also allows user to back up files to another directory. Moreover, our product provides two-way synchronization which can automatically detect which files and folders have been modified and perform synchronization actions between source and destination directory based on pre-determined user preferences. In addition, SyncSharp is easy to use and supports all the basic features of a typical synchronization tool. SyncSharp offers users with automated synchronization with minimal user interaction.

A summary of SyncSharp features is as follows:

- Create, edit and delete synchronization profiles
- Import/export synchronization profiles
- Ability to use environment variables in folder paths
- Configure user settings
- Perform 2-way synchronization between the file/folder pairs
- Set inclusion/exclusion filters
- Backup files in source to target folder
- Generate log file after each synchronization operation

1.3 System Requirements

- **Operating System**

Windows 2000, Windows 2003, Windows XP, Windows Vista, Windows 7[1]

- **PC Configuration Requirement:**

128 RAM or more, X MB of hard disk space

1.4 Support and Feedback

- **Technical support**

For technical support, please contact us by email at CS3215-13@gmail.com

- **Feedback**

If you have any comments or suggestions for the next release, please direct them to cs3215-13@gmail.com. Your feedback is highly important for us. In order to get idea of how to make SyncSharp a better product for you, the current release is highly influenced by comments from users.

[1] For current release, PlugSync feature is not fully functioning in Windows 7 due to system restrictions.

Chapter 2 Developers Guide

Part 1: *Design Methodology*

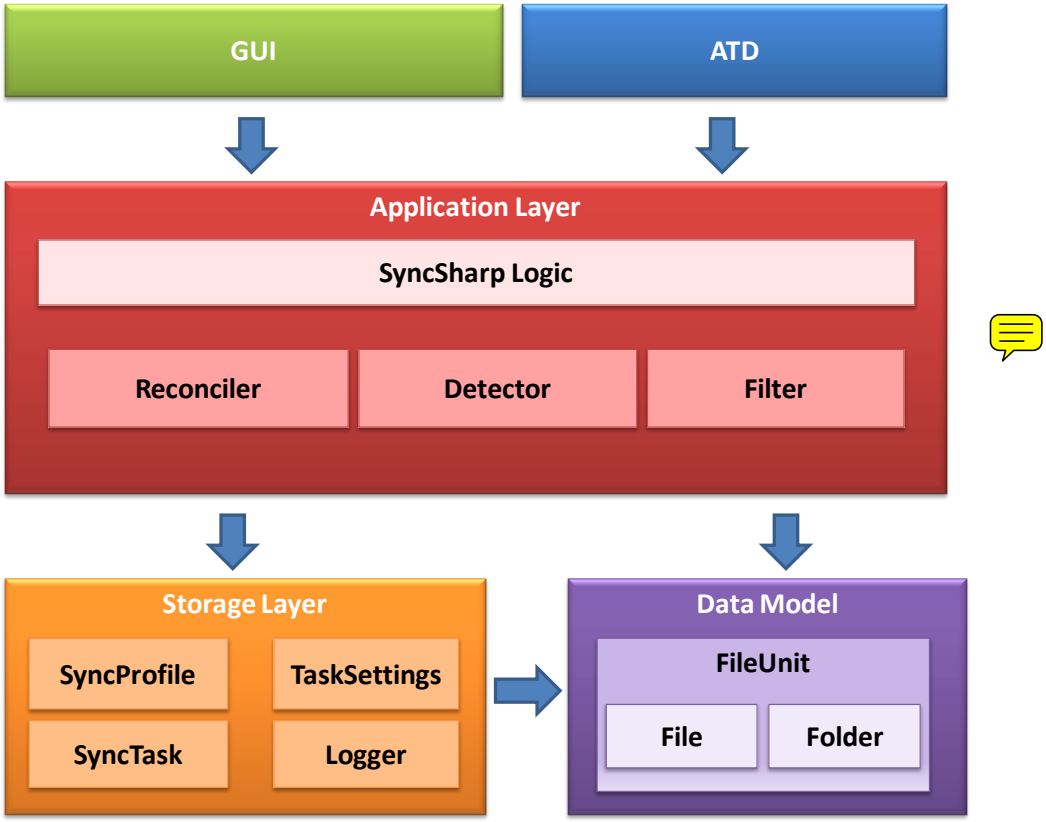
2.1.1 Top-Down Incremental Compilation Flow

We used top-down incremental design methodology in this project. We think this is the best methodology for developing SyncSharp. It is because the incremental design methodology allows us to preserve what we have done and save compilation time by taking previous compilation results so that only the portion that has just been modified are compiled each time.

In the incremental compilation flow, we wrote the codes and compiled them incrementally. It is much easier to fix bugs when we face them; and it saved a lot of time by allowing us to modify the critical portions while processing the other portions.

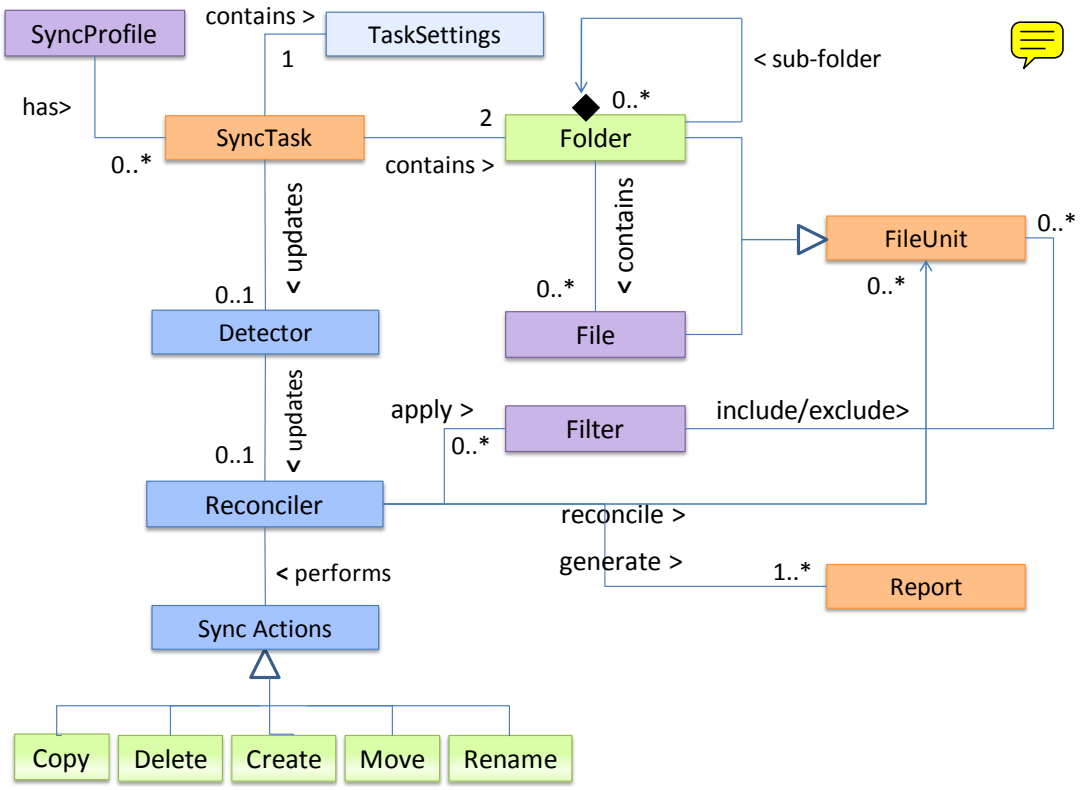
Part 2 Design Flow

2.2.1 System Architecture

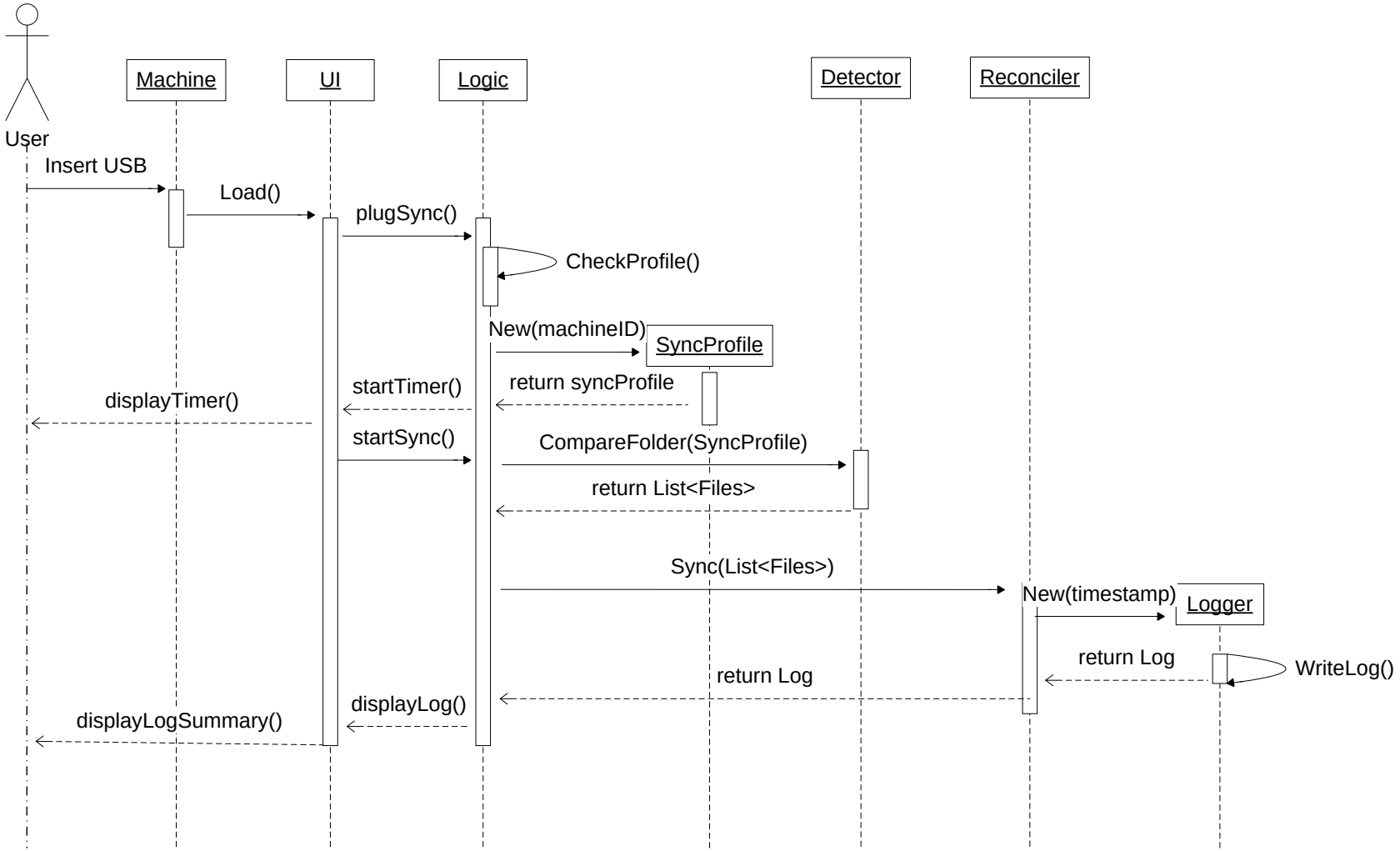


Component	Description
GUI	Provides the interface between users and application.
ATD	Provides automated testing of the application functionalities during development.
SyncSharp Logic	Receives input from GUI component and initiates a response by making function calls to various sub-components.
Detector	Evaluates the changes on the designated folders / files based on the last synchronization operation which stores a small amount of information (called metadata). Metadata captures a snapshot of every file and folders' state. Detector then passes a list of files to the reconciler to perform synchronization.
Reconciler	Performs file synchronization on the list of files obtained from the Detector. The file synchronization operation is based on the analyzed results. In the rise of conflicting updates, pre-determined users' preferences will be used to resolve the updating conflicts. The summary of the updates will be passed to the Logger. Reconciler then updates the metadata of the replica.
Filter	Provides a list of filter rules that will be used by Detector for files retrieval.
Sync Profile	Stores the machine identity and contains a list of SyncTask associated with the profile.
SyncTask	Defines the pair of folders to be used for synchronization
TaskSettings	Stores all the configuration settings for each SyncTask.
Logger	Generates the summary of the file synchronization tasks.
FileUnit	Abstract representation of a file or folder.

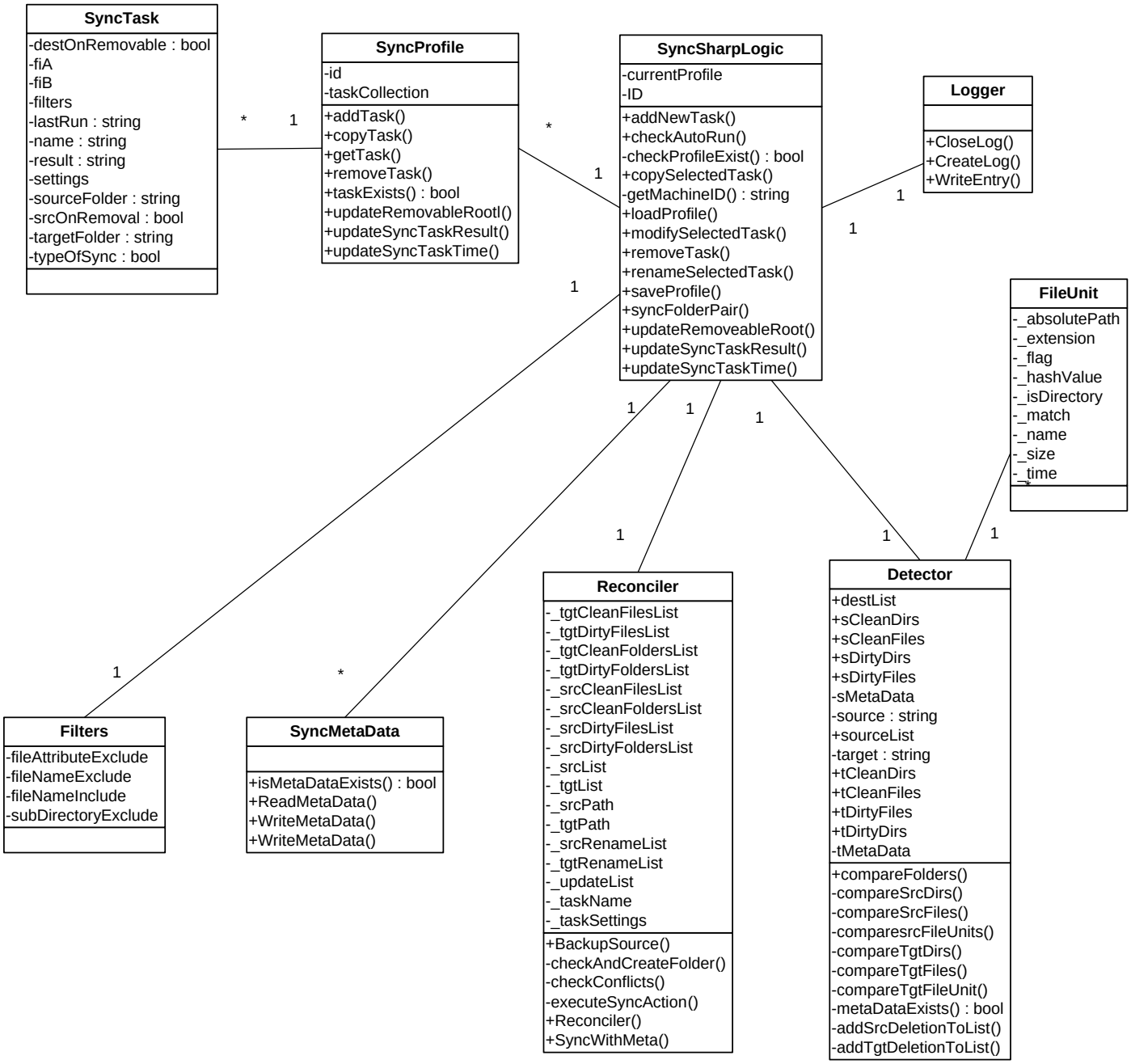
2.2.2 Domain Model Analysis



2.2.3 Sequence Diagram



2.2.4 Class Diagram



2.2.5 Use Cases

The following table is a summary of all use cases

No	User Case
1	Create Synchronization Tasks
2	Edit Synchronization Tasks
3	Delete Synchronization Tasks
4	Run PlugSync
5	Compare Source and Target Directories
6	Perform 2-way Synchronization Between Source and Target Directories
7	Backup Files
8	Restore Files
9	View Source/Target Folders
10	Export Synchronization Profiles
11	Import Synchronization Profiles
12	View Log Files
13	View Help Files

Use Case Number: 1 Use Case Name: Create Synchronization Tasks
Pre-Conditions: SyncSharp is running and at the main window
Post-Conditions: System creates new Synchronization task and is displayed on the main window
Actors: User, System
Main Success Scenario: 1. User clicks on “New” 2. System displays new SyncTask setup wizard 3. User enters name for SyncTask 4. System requests for SyncTask type: ‘Synchronize’ or ‘Backup’ 5. User selects SyncTask type 6. System requests path for Source and Target folder 7. User enters path for Source and Target folder 8. System creates new SyncTask and updates the main window Extensions(s): 3a. User enters a name that already exists for a SyncTask 3a1. System displays name already exist error Use case resumes from step 2.

5a. User did not select a SyncTask type before attempting to proceed
5a1. System prompts user to select a SyncTask type
Use case resumes from step 4.
7a. User enters non-existing/empty path for Source/Target folders
7a1. System prompts user to enter a valid path name
Use case resumes from step 6.
7b. User selects same path for Source/Target folders
7b1. System displays error that Source/Target folders cannot be the same
Use case resumes from step 6.

User Case Number: 2 User Case Name: Edit Synchronization Tasks
Pre-Conditions: At least 1 SyncTask has already been created
Post-Conditions: SyncTask settings are updated and main window is updated to reflect any changes
Actors: User, System
Main Success Scenario: 1. User selects a SyncTask from the main window and clicks on “Modify” 2. System displays the task setup window 3. User modifies the SyncTask settings as desired 4. System updates the SyncTask settings and updates the main window to reflect any changes Extensions(s): 3a. User provides some invalid settings 3a1. System prompts user to correct any errors Use case resumes from step 2.

User Case Number: 3 User Case Name: Delete Synchronization Tasks
Pre-Conditions: At least 1 SyncTask has already been created
Post-Conditions: Select SyncTask is deleted and removed from the main window
Actors: User, System
Main Success Scenario: 1. User selects a SyncTask from the main window and clicks on “Delete” 2. System confirms with user to delete selected SyncTask 3. User selects ‘OK’ 4. System deletes selected SyncTask and removes it from the main window Extension(s): 3a. User selects ‘Cancel’

Use case ends.

User Case Number: 4 User Case Name: Run PlugSync
Pre-Conditions: At least 1 SyncTask has been created and PlugSync is enabled for this SyncTask, SyncSharp is run from removable USB device, Computer's AutoPlay is enabled
Post-Conditions: Source/Target folder contents are synchronized
Actors: User, System
Main Success Scenario: 1. User inserts removable USB device 2. System automatically initiates 3. System retrieves a list of SyncTasks from current profile that has PlugSync enabled 4. System displays countdown that PlugSync is about to start 5. User waits for countdown period to end 6. System performs synchronization 7. System returns back to main window. Normal usage continues Extension(s) 5a. User cancels PlugSync by clicking on "Back to Main" Use case resumes from step 7

Use Case Number: 5 Use Case Name: Compare Source and Target Directories
Pre-Conditions: At least 1 SyncTask has been created
Post-Conditions: A window is displayed to the user that shows all the differences and SyncActions that would be performed by synchronization
Actors: User, System
Main Success Scenario: 1. User selects SyncTask and clicks on 'Analyze' 2. System compares Source/Target folders and displays results to user Extension(s): 2a. System determines that Source/Target folders are already synchronized, and displays message to user User case ends

Use Case Number: 6 Use Case Name: Perform 2-way Synchronization Between Source and Target Directories
Pre-Conditions: At least 1 SyncTask has been created, with 'Synchronization' type
Post-Conditions: Source/Target folder contents are synchronized
Actors: User, System
Main Success Scenario: 1. User selects SyncTask and clicks on 'Synchronize' 2. System proceeds to synchronize the Source/Target folders 3. System updates "Successful", and last run time in the main window for the selected SyncTask Extension(s): 2a. System encounters error during synchronization 2a1. System updates "Unsuccessful", and last run time in the main window for the selected SyncTask Use case ends

Use Case Number: 7 Use Case Name: Backup Files
Pre-Conditions: At least 1 SyncTask has been created with 'Backup' type
Post-Conditions: Any changes made to files/folders on Source directory will be updated on Target directory
Actors: User, System
Main Success Scenario: 1. User selects SyncTask and clicks on 'Backup' 2. System proceeds to backup the Source folder to the Target folder 3. System updates "Successful", and last run time in the main window for the selected SyncTask Extension(s): 2a. System encounters error during backup 2a1. System updates "Unsuccessful", and last run time in the main window for the selected SyncTask Use case ends

Use Case Number: 8 Use Case Name: Restore Files
Pre-Conditions: At least 1 SyncTask has been created with 'Backup' type
Post-Conditions: Files/folders on the Target directory will be copied to the Source Directory
Actors: User, System

Main Success Scenario:

1. User selects SyncTask and clicks on 'Restore'
2. System proceeds to restore the Target folder to the Source folder
3. System updates "Successful", and last run time in the main window for the selected SyncTask

Extension(s):

2a. System encounters error during restore

2a1. System updates "Unsuccessful", and last run time in the main window for the selected SyncTask

Use case ends

Use Case Number: 9 Use Case Name: View Source/Target Folders
Pre-Conditions: At least 1 SyncTask has been created
Post-Conditions: Source/Target folders are opened and displayed to the user using windows explorer
Actors: User, System
Main Success Scenario: 1. User selects SyncTask and clicks on 'Action-> Open Source/Target Folder' 2. System opens and displays Source/Target folders in windows explorer Extension(s): 2a. Source/Target folder does not exist. 2a1. System displays error that Source/Target path cannot be found Use case ends

Use Case Number: 10 Use Case Name: Export Synchronization Profiles
Pre-Conditions: At least 1 SyncTask has been created
Post-Conditions: All SyncTasks for profile are exported to a *.profile file
Actors: User, System
Main Success Scenario: 1. User selects 'Export Task' from main menu 2. System requests from user location and filename for exported file 3. User selects location and enters filename for exported file 4. System exports all SyncTasks for current profile into location with filename selected by user

Use Case Number: 11 Use Case Name: Import Synchronization Profiles
Pre-Conditions: A SyncProfile has been previously exported
Post-Conditions: SyncTasks from exported profile will be imported and added into current profile
Actors: User, System
Main Success Scenario: 1. User selects 'Import Task' from main menu 2. System request from user location of file to import 3. User selects file to import 4. System imports all SyncTasks from the export file into the current profile Extension(s): 4a. System determines that user selected file to import is not valid 4a1. System displays error message to user Use case resumes from step 2

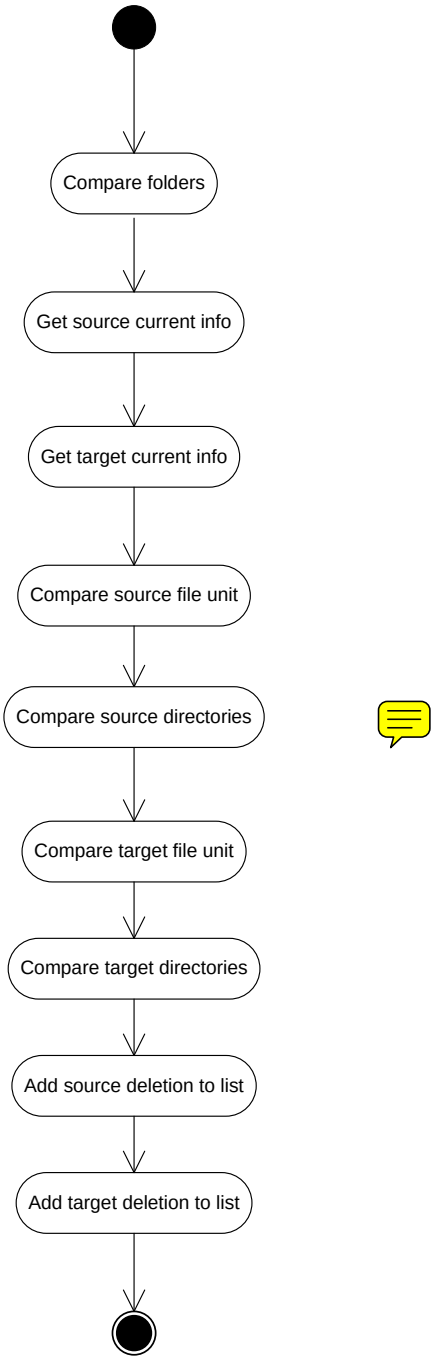
Use Case Number:12 Use Case Name: View Log File
Pre-Conditions: At least 1 SyncTask has been created
Post-Conditions: System displays log file to the user
Actors: User, System
Main Success Scenario: 1. User selects SyncTask and clicks on 'Action -> View Log' 2. System displays log file to user Extension(s): 2a. System cannot find log file associated with selected SyncTask 2a1. System informs user that no log file exists for select SyncTask Use case ends

Use Case Number: 13 Use Case Name: View Help File
Pre-Conditions: -
Post-Conditions: Help file is displayed to user
Actors: User, System
Main Success Scenario: 1. User clicks on 'Help' on the main menu 2. System displays help file to user End of User case

2.2.6 Activity Diagrams

2.2.6.1 Detector ~~Activity Diagram and~~ Algorithm

The following ~~diagram~~ is the ~~activity diagram~~ for Detector:



Activity 1: Compare folders

```
private void CompareFolderPair()
```

Description:
Detect file/folder changes on both source and target folders based on folders' current states and its metadata.

Activity 2: Get current source file info

```
private static void getCurrentSrcInfo(List<FileUnit> srcFiles, Stack<string> stack)
```

Description:
Get file information (name, size, hash code, last modified date, etc) for all files in the source directory and store them in a list.

Activity 3: Get current target file info

```
private static void getCurrentTgtInfo(List<FileUnit> destFiles, Stack<string> stack)
```

Description:
Get file information (name, size, hash code, last modified date, etc) for all files in the target directory and store them in a list.

Activity 4: Compare source file unit

```
private void compareSrcFileUnits(int sRevPathLen, List<FileUnit> srcFiles)
```

Description:
For each file in the source directory, compare it with corresponding file unit in source meta data. If the file was modified or newly created, add it to sDirFiles (source dirty files list) and flag it as 'M-' or 'C-' . If the file has not been modified, add it to sCleanFiles (source clean files list). Lastly, delete this file unit from source meta data.

Activity 5: Compare source directories

```
private void compareSrcDirs(FileUnit u, String folderRelativePath)
```

Description:
For each sub directory in source directory, compare it with source meta data. If the folder info was found in the metadata, add it to sCleanDirs (source clean directories). Else if there is no relevant metadata, add it to sDirtyDirs (source dirty directories). Lastly, delete this folder info from source meta data.

Activity 6: Compare target file unit

```
private void compareTgtFileUnits(int tRevPathLen, List<FileUnit> destFiles)
```

Description:
For each file in the target directory, compare it with corresponding file unit in target meta data. If the file was modified or newly created, add it to tDirFiles (target dirty files list) and flag it as 'M-' or 'C-' . If the file has not been modified, add it to tCleanFiles (target clean files list). Lastly, delete this file unit from target meta data.

Activity 7: Compare target directories

```
private void compareTgtDirs(FileUnit u, String folderRelativePath)
```

Description:
For each sub directory in target directory, compare it with target meta data. If the folder info was found in the metadata, add it to tCleanDirs (target clean directories). Else if there is no relevant metadata, add it to tDirtyDirs (target dirty directories). Lastly, delete this folder info from target meta data.

Activity 8: Add deleted file unit on source to list

```
private void addSrcDeletionToList()
```

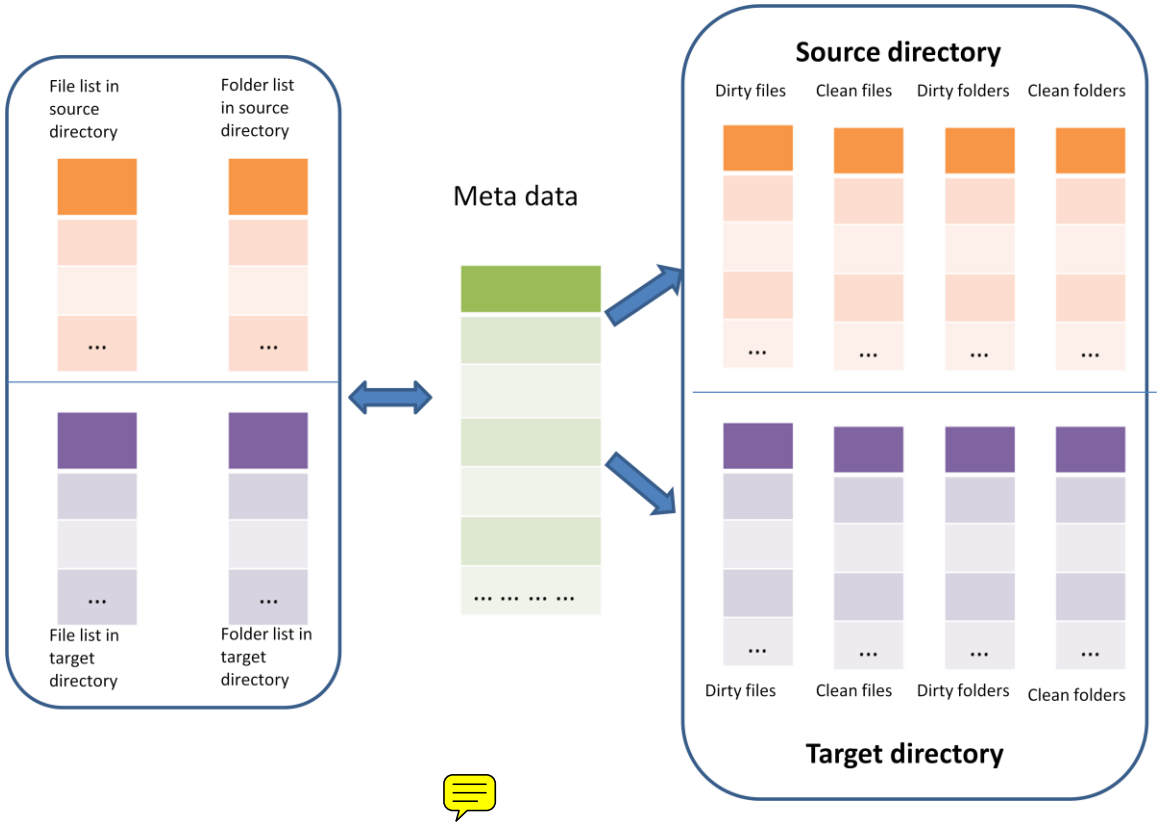
Description:
For each file unit left in source meta data, if the file unit is a directory, add it into sDirtyDir (source dirty directories). If the file unit is a file, add it into sFileDir (source dirty files). Flag the file unit as 'D-'.

Activity 9: Add deleted file unit on target to list

```
private void addTgtDeletionToList()
```

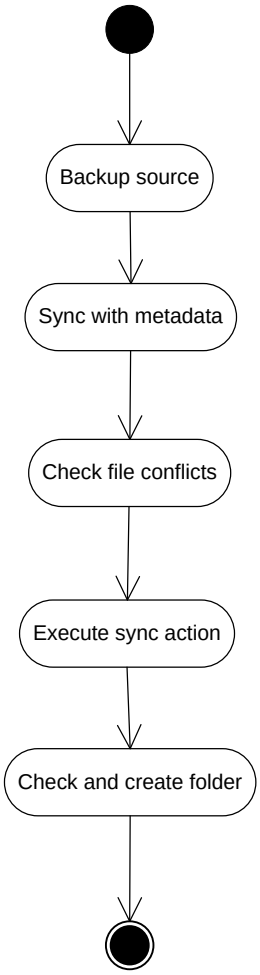
Description:
For each file unit left in target meta data, if the file unit is a directory, add it into tDirtyDir (target dirty directories). If the file unit is a file, add it into tFileDir (target dirty files). Flag the file unit as 'D-'.

Algorithm Description



Compare the current file/folder status with its corresponding metadata. If the current file/folder has been modified, deleted or created, mark it as “M-”, “D-”, and “C-” respectively and then put it into the dirty source file/folder list. Otherwise, we put it into clean source file/folder list. We do the same comparison on destination directory.

2.2.6.2 Reconciler Activity Diagram and Algorithm



Activity 1: Backup source

```
public void BackupSource(CustomDictionary<string, string, FileUnit> srcList)
```

Description:
Iterate through the given dirty list and perform file copy for every entry. The copying actions are performed in a single direction—source to target. Files with the same name and modified time are assumed to be the same and no copy action will be performed.

Activity 2: Sync with metadata

```
public void SyncWithMeta()
```

Description:

Perform two-way synchronization action based on the reconciler algorithm. Please refer to the Handling File Name Collision Conflicts below.

Activity 3: Check files conflicts

```
private SyncAction checkConflicts(FileUnit sourceDirtyFile, FileUnit destDirtyFile, String s, String t)
```

Description:
Determine the desired synchronization action for file conflict and the action are based on the pre-determined user settings.

Activity 4: Execute sync action

```
private void executeSyncAction(FileUnit srcFile, FileUnit tgtFile, String srcFlag, String tgtFlag, String srcPath, String tgtPath)
```

Description:
Execute the synchronization action based on the synchronization action returned by the checkConflict() function.

Activity 5: Check and create folders

```
private void checkAndCreateFolder(String fullPath)
```

Description:
Check for directory existent. If directory not exists, perform directory creation.

2.2.6.2.1 Handling File Name Collision Conflicts:

The file synchronization tool always follows certain key principles when it is trying to resolve conflicts. Conflict resolution must be done in a way that makes sure the convergence of data across all replicas. The most important issue is that data loss must be avoided in all possible scenarios. The general policies used by the file synchronization tool are listed below:

Letters"M", "D", "C", "R" represents the files' flags which were received from Detector.

"M" – "Modified" (i.e. file was modified)

"D" – "Deleted" (i.e. file was deleted)

"C" – "Created" (i.e. file was newly created)

"R" – "Rename" (i.e. file was renamed from another file)

We allow users to decide how the sync action will be performed based on four user settings, ***keep both copies***, ***keep source copy***, and ***keep target copy***. Our default action is to ***keep both copies***.

(In this algorithm, we treat ***file move*** the same as ***file rename***. So there will be no move flag or actions exclusively design for move.)

Conflict Description (Source)	Conflict Description (Destination)	Solution
M	M	This is a concurrent update conflict. If both source and target files have been modified, our default action is to keep the latest copy (check the last modified time)
	D	This is a concurrent update-delete conflict. If the source file is modified but the same file on destination is deleted; then we copy the modified file from source to destination
	C	This is a concurrent update-create conflict. If the source file is modified and the destination file is newly created; our default action is to keep both files (i.e. copy both file across each other)
	R	This is an update-rename collision. If the source file is modified and the destination file is renamed; our default action is to keep both files (i.e. copy both files across each other)
D	M	This is a concurrent update-delete conflict. If the destination file is modified but the same file on source is deleted; then we copy the modified file from destination to source
	D	This is a concurrent delete-delete conflict. If both files have been deleted; then no action will be performed.

	C	This is a concurrent child create-parent delete conflict. If the file in source directory is newly created and the file in destination has been deleted; then we copy the newly created file from source to destination
	R	This is a delete-rename conflict. Since the source file has been deleted so we directly copy the destination file to source file
C	M	This is a concurrent update-create conflict. If the destination file is modified and the source file is newly created; our default action is to keep both files (i.e. copy both file across each other)
	D	This is a concurrent child create-parent delete conflict. If the file in source directory is newly created and the file in destination has been deleted; then we copy the newly created file from source to destination
	C	This is a concurrent create-create collision. If both files are newly created; our default action is to keep both files.
	R	This is create-rename conflict. If the file in source directory is newly created and the file in destination directory has been renamed; our default action is to keep both copies (i.e. copy both files across each other)
R	M	This is an update-rename conflict. If the source file is renamed and the destination file has been modified; our default action is to keep both copies (i.e. copy both files across each other)
	D	This is a rename-delete conflict. If the source file is renamed and the destination file has been delete; we

		copy the renamed file from source to destination
	C	This is create-rename conflict. If the file in destination directory is newly created and the file in source directory has been renamed; our default action is to keep both copy (i.e. copy both files across each other)
	R	This is a rename-rename collision. If both files have been renamed; our default action is to keep both copies (i.e. copy both files across each other)

Chapter 3 Glossary

Term	Definition
SyncSharp	Name of our sync tool
SyncTask	Configuration file that contains source and target directory information to be synchronized
SyncProfile	Contains list of SyncTasks for a Particular PC/Laptop
FileUnit	Abstract representation of a file or folder, contains information such as name, size, hash code, last modified date, etc.
PlugSync	A feature in our program. Program will run by itself while the USB is plugged into a computer. Upon execution, the program will count down for 5 seconds waiting for user interruption. If there is no interruption, the program will automatically try to perform all the saved SyncTask.
1 way sync	Update destination directory to have the same content as source directory
2 ways sync	Update source and destination directories to have the same state
Report/Logger	Log file that records the operations perform in the synchronization process
Target	The destination directory to be sync or compared
Detector	The sub-system that detect changes of the source or destination directory
Reconciler	The sub-system that resolves conflicts between the source & destination directories
TaskSettings	Contains configuration settings made for each SyncTask