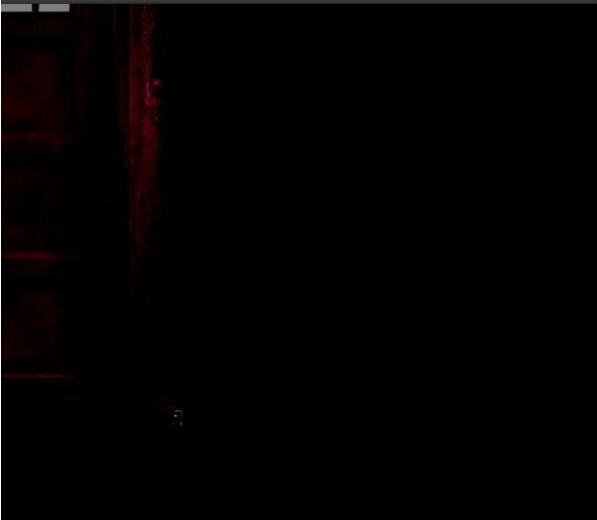


SmartyDo

Menu

Meet Galahad to Buy Shoes
Date: 6 Dec 2016
Time: 18:12
Description: Oxfords not brogues
Location: Savile Row
Tags: [Kingsmen]



File View Help

select 3

1. Collect laundry on the way home ☐





2. **Presentation** 26-Jul-2016 9:00AM ☐

3. **Meet Galahad to Buy Shoes** 06-Dec-2016 6:12PM [Kingsmen] ☐

Selected Task #3

Last Updated: Fri Nov 04 16:42:19 ...

Supervisor: Prof Damith

			
Kenneth Tan Xin You	Matthew Chan	Filbert Cheong	Moon Byunghun

Acknowledgements

About Us

Kenneth Tan Xin You



- Components in charge of: [Model](#)
 - Aspects/tools in charge of: Team Lead, Integration, Git, Issue Tracking
 - Features implemented:
 - [Edit Command](#)
 - [Minimise](#)
 - [Confirmation Prompt](#)
 - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
 - Other major contributions:
 - Set up Travis and Coveralls
 - Implemented the logical ordering of tasks [#37](#)
 - Advised on implementation of several features and changes to UI
 - Did the initial copy-editing of docs for CS2101[\[link to commit\]](#)
 - Assisted Filbert in Testing and Bug Fixing
 - Assisted Matthew in improvements to UI [#80](#)
-

Filbert Cheong



- Components in charge of: [Logic](#)
 - Aspects/tools in charge of: Testing, Parsing
 - Features implemented:
 - [Adding Tasks - Floating, Untimed etc.](#)
 - [Some Flexibility in commands\(Date Parsing\)](#)
 - [Undo and Redo](#)
 - [Mark Completed Tasks](#)
 - Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
 - Other major contributions:
 - Contributed to initial copy-editing of docs for CS2101[\[link to commit\]](#)
-

Matthew Chan



- Components in charge of: [UI](#)
- Aspects/tools in charge of: UI, UX, Documentation
- Features implemented:
 - [Find By Tag](#)
 - [View Command](#)
 - [Locate Command](#)
- Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
- Other major contributions:
 - Overhauled UI [\[#56\]](#)[\[#84\]](#)
 - Revise Documentation for Consistency [\[#65\]](#)

Moon Byunghun



- Components in charge of: [Storage](#)
- Aspects/tools in charge of: Documentation, Markdown
- Features implemented:
 - [Save file](#)
 - [Load file](#)
- Code written: [\[functional code\]](#)[\[test code\]](#)[\[docs\]](#)
- Other major contributions:
 - Did refactoring from Person to Task [\[#20\]](#)
 - Copy edited UserGuide for CS2101 [\[#38\]](#)
 - Enhancement in task card UI [\[#50\]](#)[\[#78\]](#)

User Guide

Table of Contents

1. [Introduction](#)
2. [Quick Start](#)
3. [Getting Started](#)
 1. [Requesting Help](#)
 2. [Choosing Your Save Location](#)
 3. [Loading Save Files](#)
 4. [Adding Tasks](#)
 5. [Editing Task Details](#)
 6. [Deleting Tasks](#)
 7. [Marking Completed Tasks](#)
 8. [Undoing and Redoing](#)
 9. [Selecting Specific Tasks](#)
 10. [Finding Specific Tasks](#)
 11. [Filtering the Task List](#)
 12. [Locating a Destination](#)
 13. [Clearing Saved Data](#)

- 14. [Exiting SmartyDo](#)
- 4. [Smart Features](#)
 - 1. [FlexiCommand](#)
 - 2. [Saving The Data](#)
- 5. [Summary](#)
 - 1. [Command Summary](#)
 - 2. [Keyboard Shortcuts](#)

1. Introduction

SmartyDo is a **to-do-list** application. With SmartyDo, forgetting upcoming deadlines and sleepless nights over incomplete tasks are a thing of the past. SmartyDo **increases your efficiency** by showing the lists of tasks that can be completed simultaneously. Treat SmartyDo like your personal assistant and just focus on **completing your tasks!**

2. Quick Start

Launch SmartyDo: Simply double-click on the `SmartyDo.jar` file to start SmartyDo. You will be greeted with a simple interface that has three components: a **Visual Box**, a **Message Box** and a **Command Bar**.



Figure 1. Welcome Screen

Command Bar is where you enter short commands to tell SmartyDo what to do.

Visual Box is where you can see a comprehensive list of your tasks.

Message Box shows the result of your command.

3. Getting Started

In this section, you will be introduced to the various commands that you will need when using SmartyDo. These commands will be described to you in the format described below.

Command Format

Words in lower case are the command.

Words in upper case are the parameters.

Items in square brackets are optional.

The order of parameters is flexible.

3.1. Requesting Help From SmartyDo

You can use the help command to gain access to this user guide should you need any help with the commands and their format. Should you enter an invalid command (e.g. abcd), information will be shown, when possible, to help correct your mistakes. You may also access this function through a keyboard shortcut.

Format: `help` Keyboard Shortcut: `Ctrl+F1`

Example:

If you wish to get help on using SmartyDo, you may enter `help` into the Command Bar.

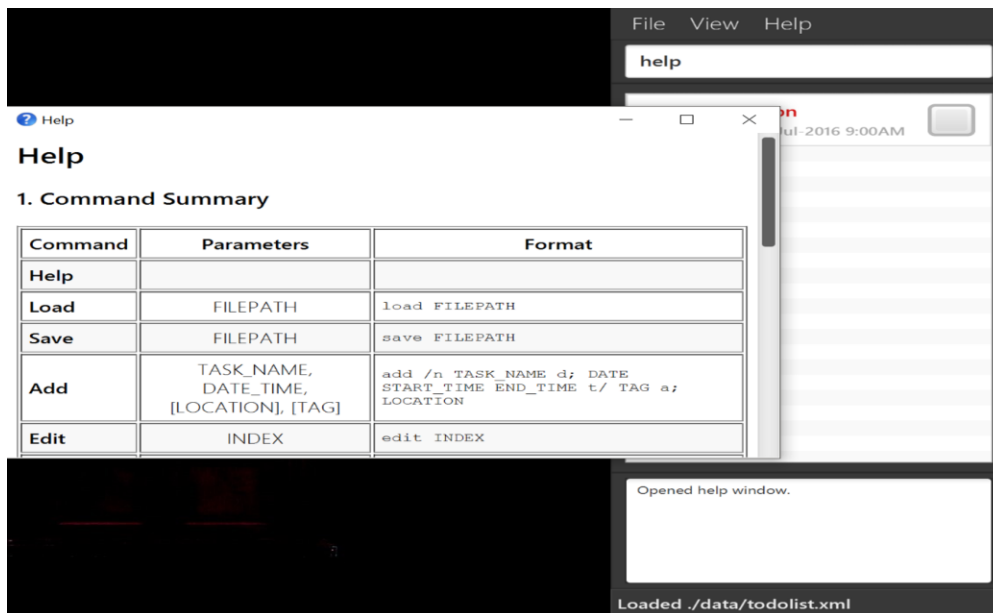


Figure 2. SmartyDo's Help Command

After entering the command, a new window will appear showing you a summary of all commands and keyboard shortcuts.

3.2. Choosing Your Save Location

You can choose where to save your data on your computer by using the `save` command. The save location will be referenced from the directory in which SmartyDo is stored. From this point, all data will be saved to the file you specified.

Format: `save FILEPATH.xml`

Example:

If you wish to save your files to the filepath `data/todolist.xml`, you may enter `save data/todolist.xml` into the Command Bar.



Figure 3. SmartyDo's Save Command

After entering the command, MessageBox will show you if your new save file has been successfully created.

3.3. Loading Save Files

You can load different save files from your computer into SmartyDo by using the `load` command. The location from which your save file is retrieved will be referenced from the directory in which SmartyDo is stored. From this point, all data will be saved to the file you specified.

Format: `load FILEPATH.xml`

Example:

If you wish to load a previously saved file stored in `data/my_todo_list.xml`, you may enter `load data/my_todo_list.xml` into the Command Bar.



Figure 4. SmartyDo's Load Command

After entering the command, MessageBox will show you if the save file has been successfully loaded to SmartyDo.

3.4. Adding Tasks Into SmartyDo

You can add a task into SmartyDo by using the add command. There are number of parameters that you can use to add more details to the task. Below is a summary of the various parameters and their usage:

Here is the summary of the parameters and their usage:

Parameter	Flag	Format Requirements	Optional
TASK_NAME	<i>n</i> ; required if TASK_NAME is not the first parameter		No

Parameter	Flag	Format Requirements	Optional
DATE_TIME	t;	[Date] [Start_Time] [End_Time] , delimited by spaces	Yes
TAG	t/	alphanumeric	Yes
LOCATION	a;	alphanumeric	Yes
DESCRIPTION	d;	alphanumeric	Yes

Table 3.2. Add Command Parameters

- **TASK_NAME** is the name of the task and this parameter is compulsory.
- **Date** is the date of the task supports date format of dd/mmm/yyyy eg:20-Jan-2017 and dd/mm/yyyy with / interchangeable with . and – eg: 20-01-2017.
- **START_TIME** and **END_TIME** is the starting time and ending time of the task respectively. You may consider to use these parameters when starting time and/or deadline is known. You may omit the details of 'DATE_TIME' which will result in a task that has no time frame.
- **TAG** is the characteristic you can add to the task. Such tags can be "Urgent", "HighPriority" and etc.
- **LOCATION** is the place of task being done. You can use this parameter to remind you where to go to complete the task.
- '**DESCRIPTION**' is where you can include some extra information about your task.

Format : add TASK_NAME [t; DATE START_TIME] [a;LOCATION] [t/TAG] [d;

You don't have to enter the optional parameters when you don't need them. The order of the parameters are not fixed. You can enter the parameters in any order. For example, add t/[TAG] t; DATE START_TIME is also correct format.

Example:

Let's say you want to add task named "Presentation" which is scheduled for 18 July 2016, 9:00AM. All you need to do is enter the following as shown below.

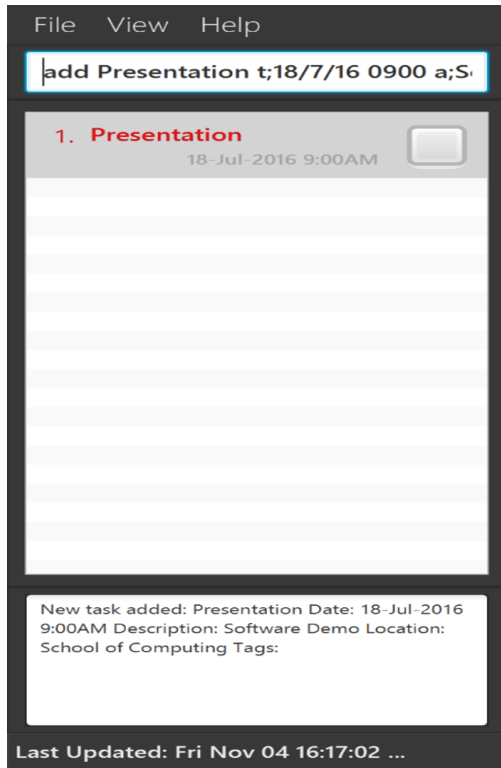


Figure 5. Example of add command

After entering the command, MessageBox will show you task is successfully added into SmartyDo and you will see the updated list of task in the Visual Box.

3.5. Editing Task Details

You might want to edit details of a task for several reasons. For example, when deadline was extended you will need to update the SmartyDo for this change. Using `edit` command will solve this problem.

Format: `edit INDEX PARAMETER_TYPE NEW_VALUE`

`PARAMETER_TYPE` the type of the parameter we want to change and `NEW_VALUE` is the new value for that parameter.

`edit` command edits the task at the specified `INDEX`. You can easily identify the `INDEX` of the task by looking at the Visual Box.

If the task you want to edit is not shown in the Visual Box, you can use `view` or `find` command to find the required task.

Example:

Let's say you want to add deadline time for task named "Presentation". Then, you must first check the `INDEX` of that task. In this case, the `INDEX` of the task is 1. So to add deadline for

this task, enter `edit 1 t; DEADLINE`. This will update the deadline of the task. A demonstration of this functionality shown below.

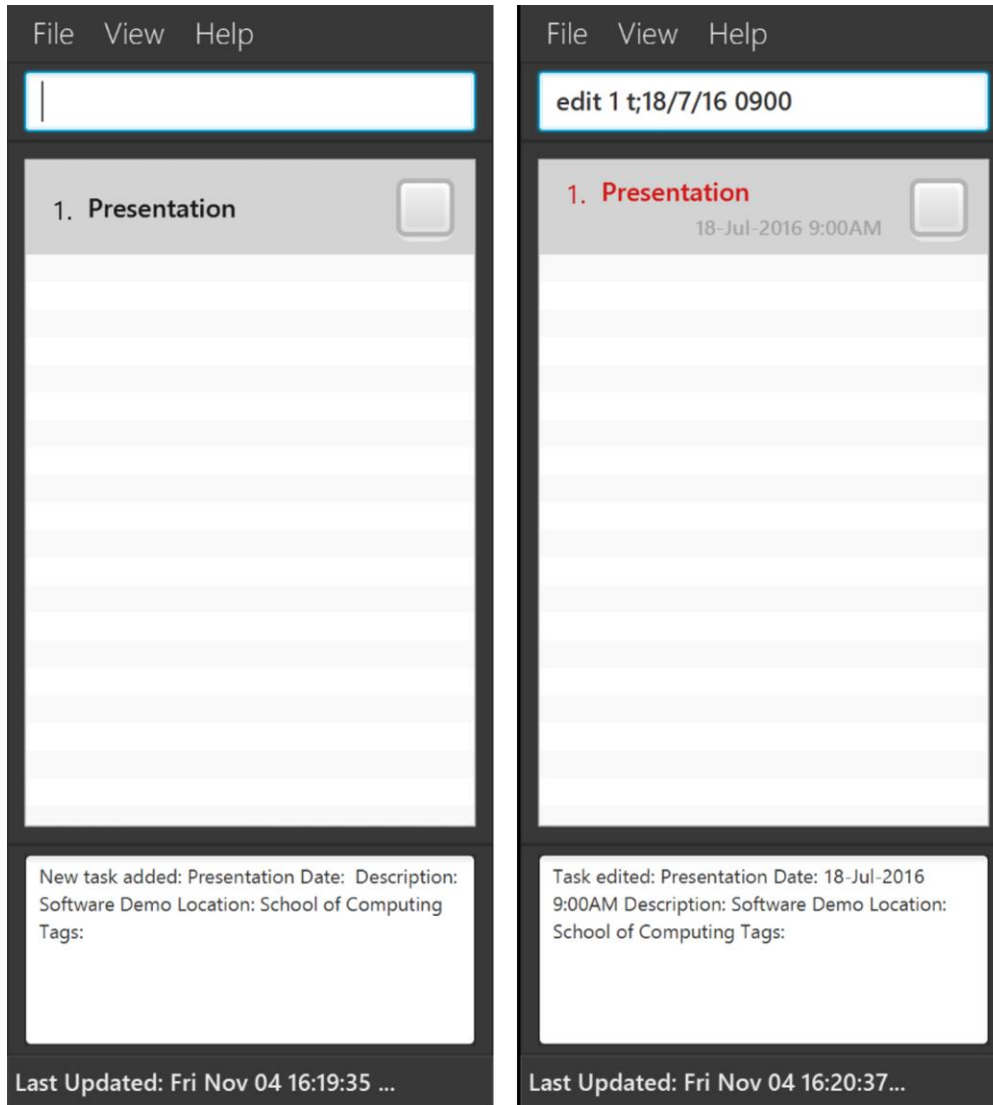


Figure 6. Before(left) and after(right) of an edit command

3.6. Deleting Tasks

Sometimes, you may also want to delete tasks due to unexpected circumstances. To help you to handle such problem, `delete` command can be used. `delete` command is simply deleting task from SmartyDo.

Format: `delete INDEX`

Similar to `edit` command, `delete` command also uses `INDEX`. `INDEX` can be found in Visual Box by using `view` command and `find` command.

Example:

If you want to delete specific task, find the INDEX of that task. Let's say the INDEX is 1. Then, enter `delete 1` in the command bar.

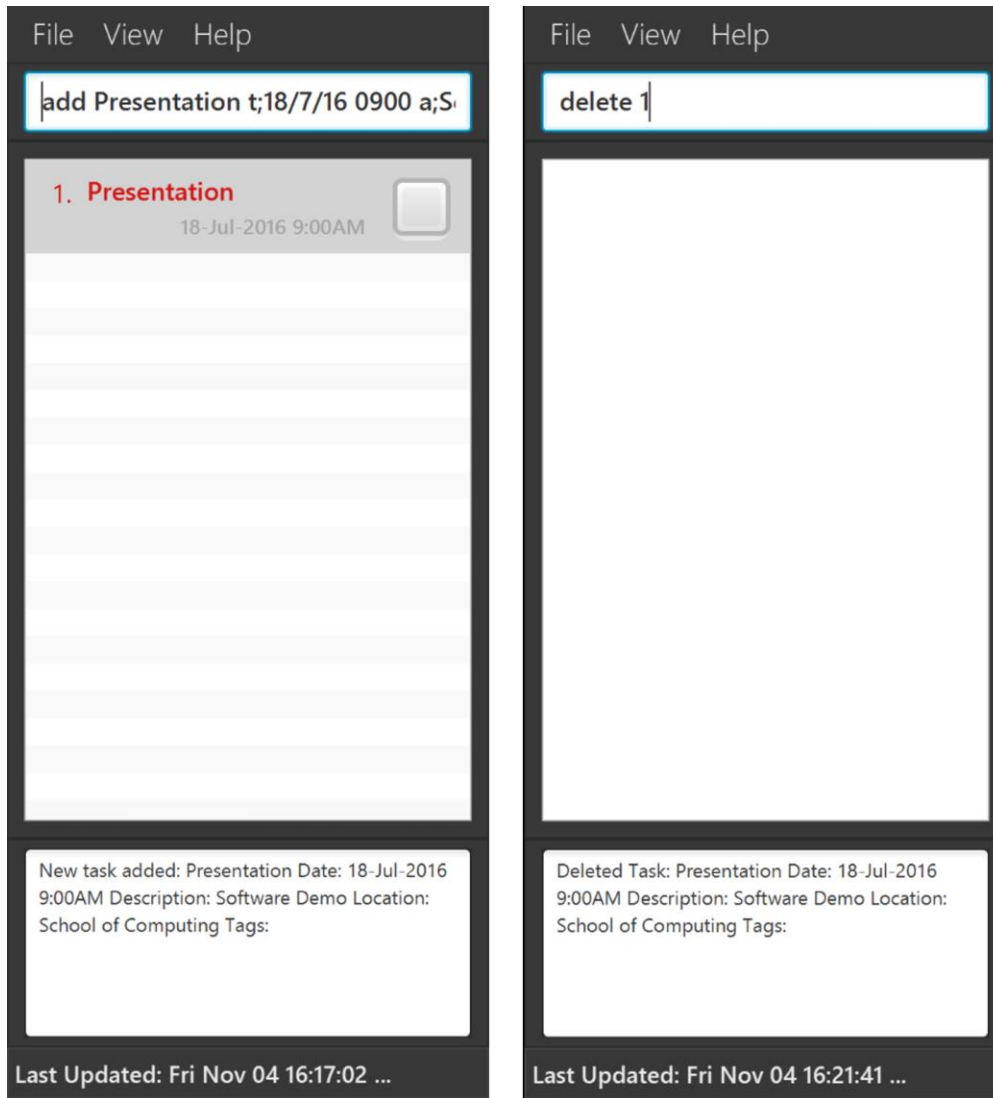


Figure 7. Example of delete command

After entering `delete` command, SmartyDo will delete the task specified by the INDEX and will show the updated list in the Visual Box. In the screenshot above, you can see that the "Presentation" task has been deleted from SmartyDo.

3.7. Marking Completed Tasks

Instead of deleting the task, you may want to mark the task as complete and store the details of the task in the SmartyDo. In this case, you can use `done` command. By using `done` command, you can easily identify the completed tasks from the list.

Format: `done INDEX`

Similar to `delete` command and `edit` command, `INDEX` is used in `done` command.

Example:

You have now completed the task named "Presentation" and now you want to mark this task as complete. To do this, you will need to check the `INDEX` of this task. In this case, `INDEX` of this task is 1. So, entering `done 1` will mark "User Guide" task as complete.

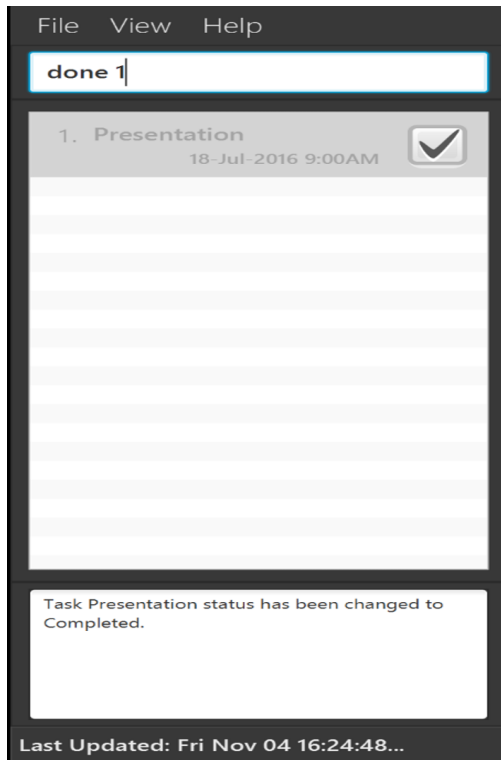


Figure 8. Example of `done` command

After entering the `done` command, you are now able to identify the completed task easily from the list.

3.8. Undoing and Redoing

With `undo`, you are allowed to reverse your previous changes sequentially while `redo` allows you to reverse the change done by `undo`.

- `undo` command requires the application to have executed atleast one undoable command after launching.
- `redo` command requires the application to have executed atleast one succussful `undo` command after launching.

Undoable Commands: add, delete, edit, done

SmartyDo **does not store** history of actions in your computer. Your history of actions resets when SmartyDo is launched. Also, if you enter any undoable command after entering `redo` or `undo`, your history of actions would be *removed*.

Format: `undo`, `redo`

Example:

Let's say you have added a task and your friend told you that your tutor has changed the date. You would like to undo it. You can undo it as long as you just added it, as shown below.

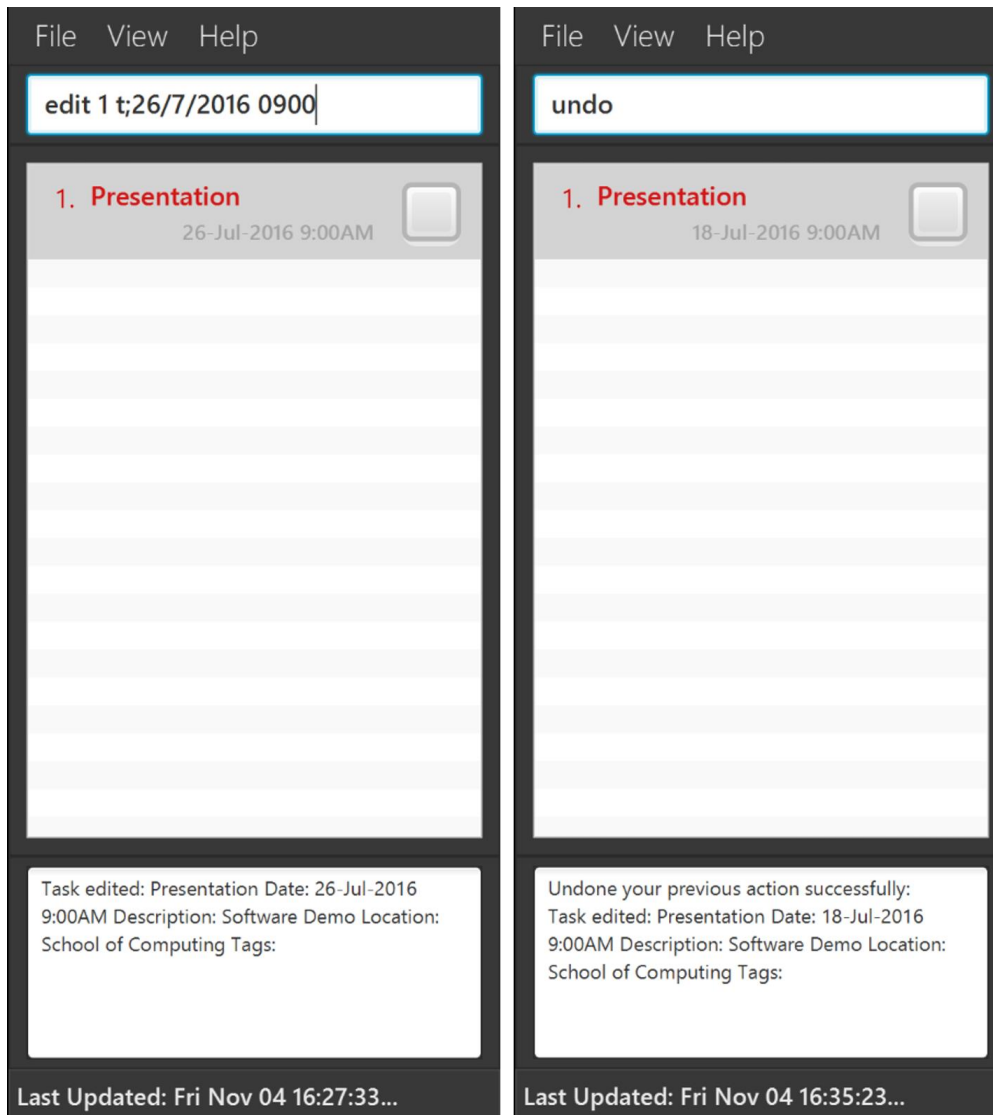


Figure 9.1. Before(left) and after(right) of an undo command

By entering `undo` command, SmartyDo updates your list of tasks to how it was before you executed an undoable action. From the screenshot above, you can see that the date of the task named "Presentation" had changed.

However, you realized that your friend was wrong and you want to change the date back again. In this case, you do not need to use edit command again. Instead you can simply use `redo` command, as shown below.

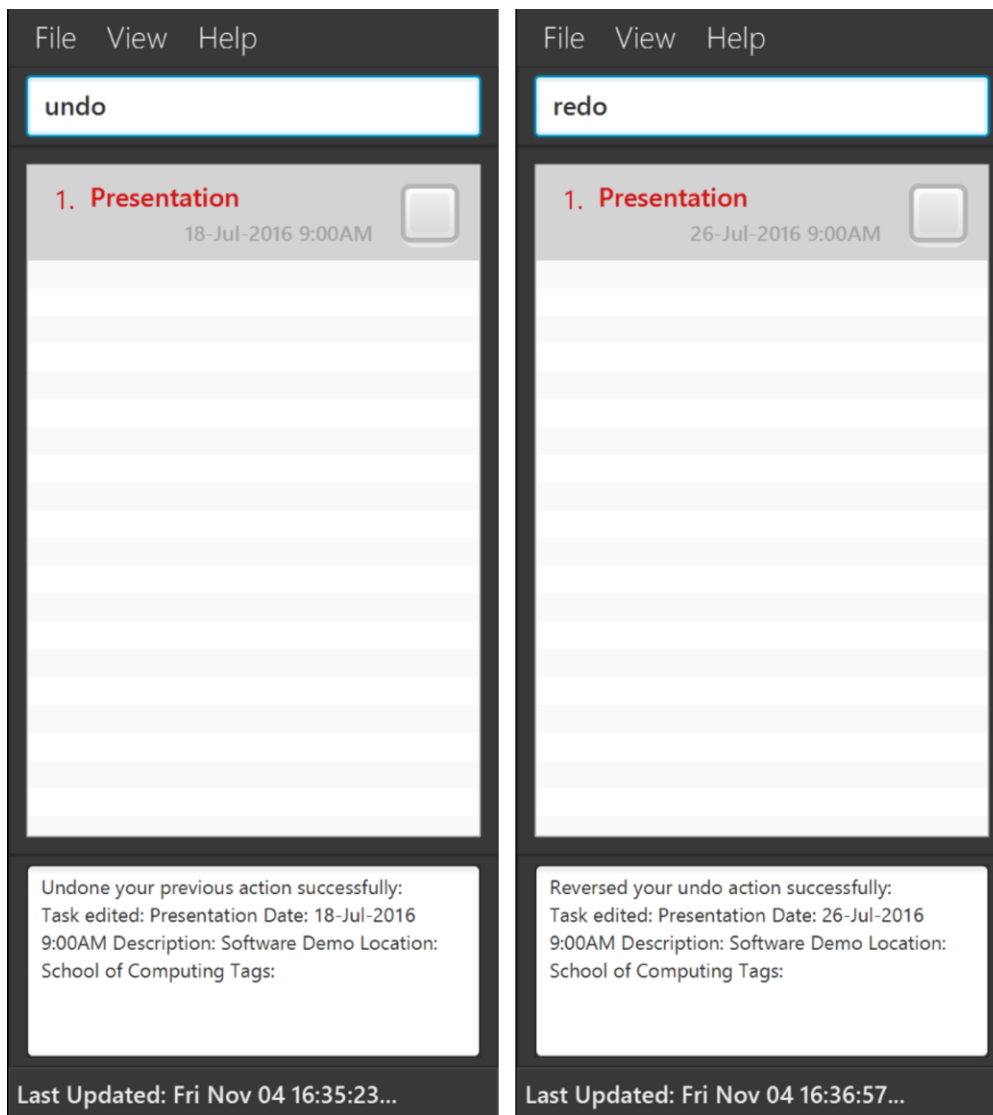


Figure 9.2. Before(left) and after(right) of an undo command

By using `redo` command, SmartyDo updates your list of tasks to how it was before you executed `undo` command. From the screenshot above, you can see that the "Presentation" task has been restored to its previous state.

3.9. Selecting Specific Tasks

Select the task identified by the parameter. A full detailed description will appear in a pop up window.

Format: `select PARAM`

Example:

Let's say you want to know detailed information about the third task in the Visual Box. All you need to do is enter `select 3` into command bar, just as shown below.

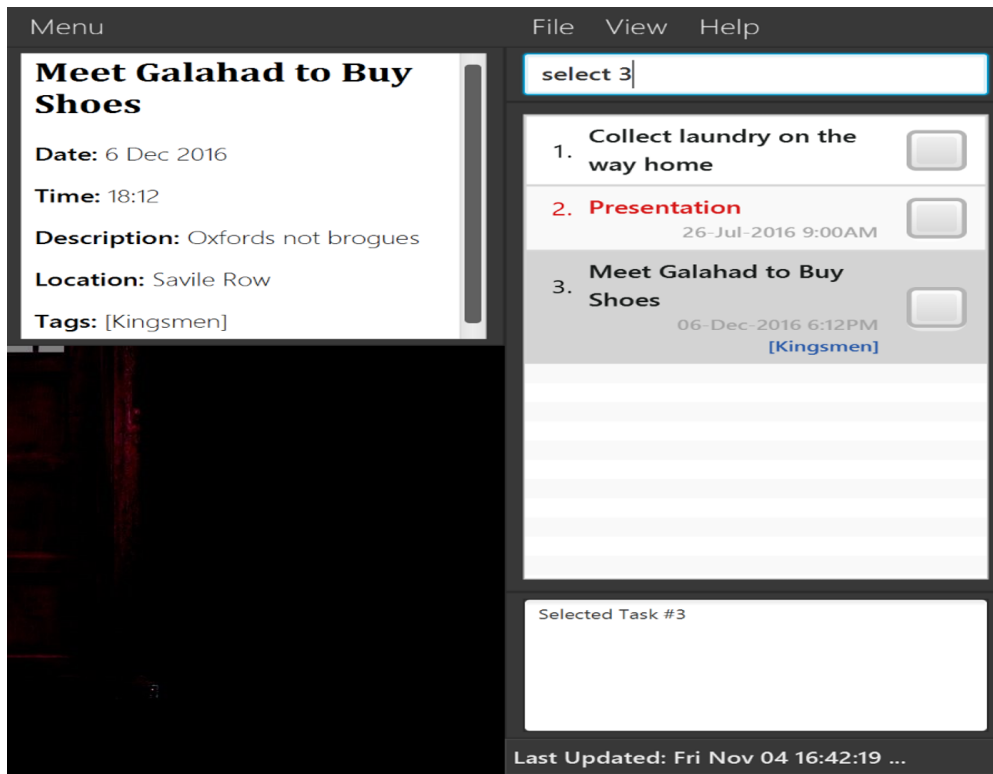


Figure 10. Example of select command

After entering the command, Browser Panel will show a detailed description about task 3.

3.10. Finding Specific Tasks

If you want to find tasks that contain specific keyword in their name, you can use `find` command. `find` command is a command that will list all the tasks matching atleast one keyword. You can enter more than one keyword for `find` command.

Format: `find KEYWORD [MORE_KEYWORDS]`

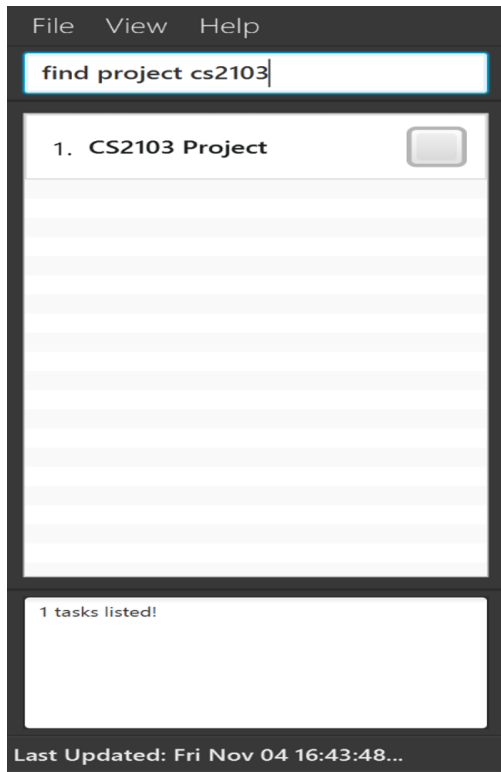


Figure 11. Example of find command

Beware that keywords are case sensitive and only the task's name is searched. However, the order of the keywords does not matter. e.g. `find cs2103 project` is same as `find project cs2103`

3.11. Filtering the Task List

You can filter the list of tasks that you are viewing on the Visual Box.

Format: `view KEYWORD` where **KEYWORD** in this case are any of the following:
`ALL/OVERDUE/UPCOMING/COMPLETED/INCOMPLETE`

For example, after finding specific tasks, to return the Visual Box back to where it lists all the tasks, simply input `view ALL` just as shown below.

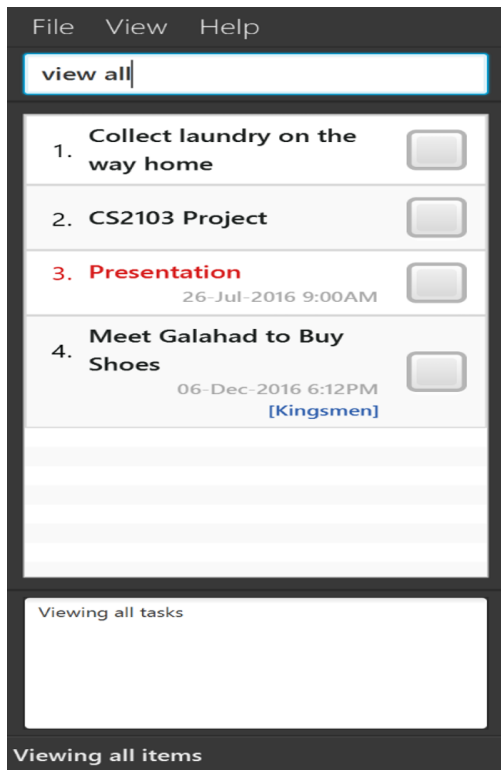


Figure 12. Example of view command

3.12. Locating a Destination

You may search for destinations listed in the LOCATION parameter of your task by using the `locate` command. A separate window will appear showing the details of the location mentioned (if any) in your task. Each task can be referred to by the index displayed in front of its title.

Format: `locate INDEX`

Example:

If you wish to search for the location of the task named Presentation which has the index of 3, you may enter `locate 3` into the Command Bar.

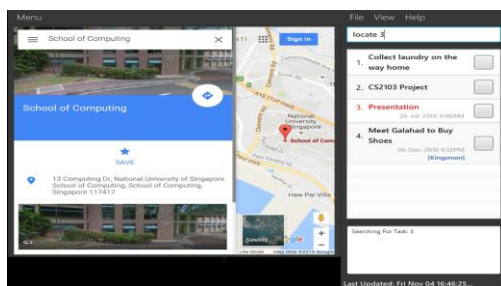


Figure 13. Example of locate command

After entering the command, a new window will appear showing you the details of the task you requested.

3.13. Clearing Saved Data

You may clear all data stored in SmartyDo by using the `clear` command. SmartyDo will prompt you to confirm this action. Enter `yes` to complete the command. Entering a different command will cancel the `clear` command.

Format: `clear`

Example

If you wish to clear all data in SmartyDo, you may enter `clear` into the CommandBox.

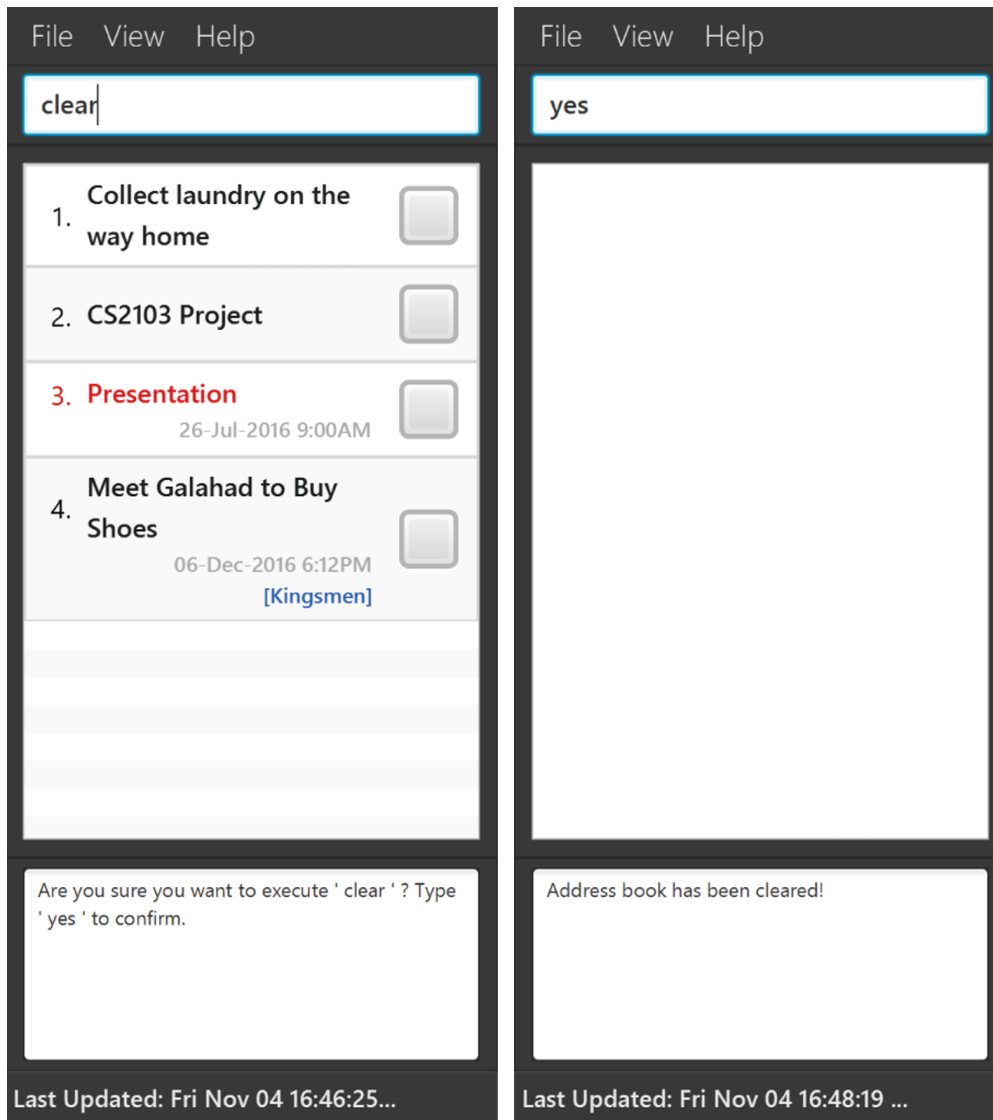


Figure 14. Example of locate command

After entering the command, a prompt will appear in the MessageBox asking you to confirm this action. Enter `yes` to proceed.

3.14. Exiting SmartyDo

After using SmartyDo, you can exit the program by using `exit` command.

Format: `exit`

By entering `exit` command in the command box, SmartyDo will quit and save the data.

4. Smart Features

4.1. FlexiCommand

It is okay if you cannot remember the syntax entirely! As long as you remember the keyword some reshuffling of the parameters entered is fine. Our program will ask you for confirmation if we are unsure what you want.

4.2. Saving the Data

SmartyDo will automatically save your data in the hard disk after any command that changes the data. There is no need to save manually.

5. Summary

5.1. Command Summary

Command	Parameters	Format
Help		<code>help</code>
Save	FILEPATH.xml	<code>save FILEPATH.xml</code>
Load	FILEPATH.xml	<code>load FILEPATH.xml</code>
Add	TASK_NAME(n; required if TASK_NAME is not the first parameter), DATE_TIME,[LOCATION], [TAG]	<code>add n; TASK_NAME d; DATE START_TIME END_TIME t/ TAG a; LOCATION</code>
Edit	INDEX	<code>edit INDEX</code>
Delete	INDEX	<code>delete INDEX</code>

Command	Parameters	Format
Done	INDEX	done INDEX
Undo		undo
Redo		redo
Select	INDEX	select INDEX
Find	KEYWORD, [MORE_KEYWORD]	find KEYWORD [MORE_KEYWORD]
View	PARAM	view PARAM
Locate	INDEX	locate INDEX
Clear		clear
Exit		exit

Table 5. Command Summary

5.2. Keyboard Shortcuts

Command	Shortcut
help	Ctrl+F1
list all	Ctrl+1
list overdue	Ctrl+2
list upcoming	Ctrl+3
list completed	Ctrl+4
list incomplete	Ctrl+5

Developer Guide

Table of Content

1. [Introduction](#)
2. [Setting Up](#)
 1. [Prerequisites](#)
 2. [Importing the project into Eclipse](#)
3. [Design](#)
 1. [Architecture](#)
 2. [UI component](#)
 3. [Logic component](#)
 4. [Model component](#)
 5. [Storage component](#)
 6. [Common classes](#)
4. [Implementation](#)
 1. [Logging](#)
 2. [Configuration](#)
5. [Testing](#)
6. [Dev Ops](#)
 1. [Build Automation](#)
 2. [Continuous Integration](#)
 3. [Making a Release](#)
 4. [Managing Dependencies](#)
7. [Appendix](#)
 1. [Appendix A: User Stories](#)
 2. [Appendix B: Use Cases](#)
 3. [Appendix C: Non Functional Requirements](#)
 4. [Appendix D: Glossary](#)
 5. [Appendix E : Product Survey](#)

1. Introduction

Welcome to the developer guide for SmartyDo. This guide is meant to enable budding developers like yourself to better understand the implementation of our program. Through this guide, we hope that you will be able to learn not only about how SmartyDo is implemented, but about different parts of the application that you are able to improve yourself.

2. Setting up

2.1 Prerequisites

To ensure that you are able to run SmartyDo smoothly, do ensure that you have met the following prerequisites:

1. Installed **JDK 1.8.0_60** or later

This app may not work as intended with earlier versions of Java 8.

This app will not work with earlier versions of Java.

2. Installed **Eclipse IDE**
3. Installed **e(fx)clipse** plugin for Eclipse (Follow the instructions given on [this page](#))
4. Installed **Buildship Gradle Integration** plugin from the Eclipse Marketplace

2.2 Importing the project into Eclipse

To import the latest version of this project into Eclipse, follow the instructions as given below:

1. Fork this repo, and clone the fork to your computer
2. Open Eclipse (Note: Ensure you have installed the **e(fx)clipse** and **buildship** plugins as given in the prerequisites above)
3. Click `File > Import`
4. Click `Gradle > Gradle Project > Next > Next`
5. Click `Browse`, then locate the project's directory
6. Click `Finish`
 - If you are asked whether to 'keep' or 'overwrite' config files, choose to 'keep'.
 - Depending on your connection speed and server load, it can even take up to 30 minutes for the set up to finish (This is because Gradle downloads library files from servers during the project set up process)
 - If Eclipse auto-changed any settings files during the import process, you can discard those changes.

3. Design

3.1 Architecture

The *Architecture Diagram* given above will explain to you the high-level design of the App. Below, we will give you a quick overview of each component.

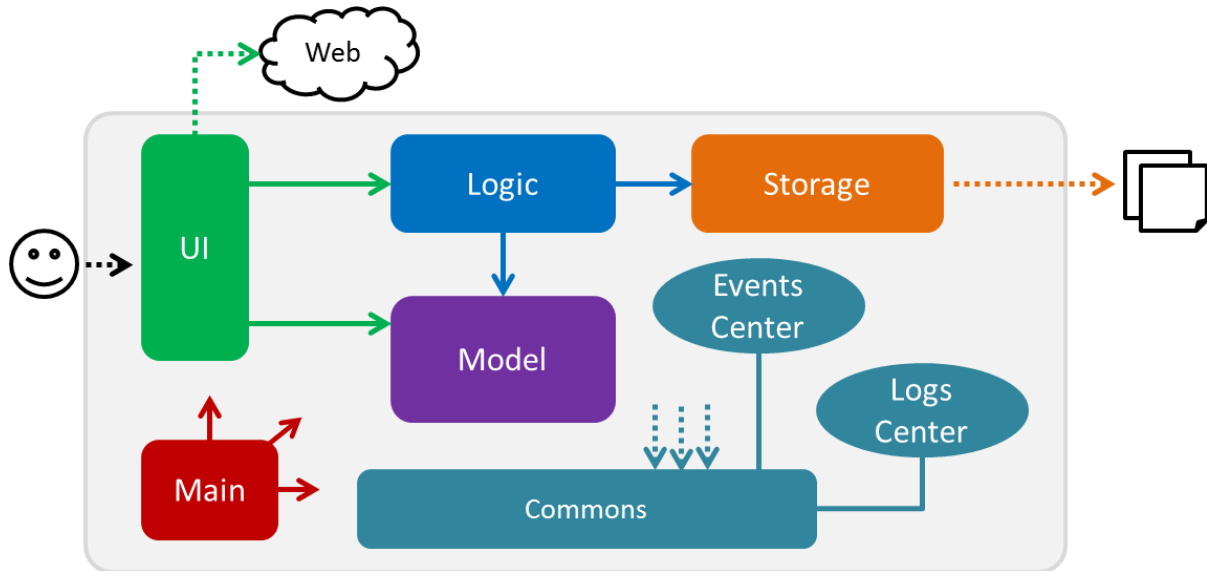


Figure 1. Overview of Main

Main has only one class called [MainApp](#). It is responsible for,

- At app launch: Main will initialize the components in the correct sequence, and connect them up with each other.
- At shut down: Main will shut down the components and invoke cleanup method where necessary.

[Commons](#) represents a collection of classes used by multiple other components. Two of those classes play important roles at the architecture level.

- [EventsCentre](#): This class (written using [Google's Event Bus library](#)) is used by components to communicate with other components using events (i.e. a form of *Event Driven* design)
- [LogsCenter](#): Used by many classes to write log messages to the App's log file.

The rest of the App consists four components.

- [UI](#): The UI of the App.
- [Logic](#): Executes commands given by the user.
- [Model](#): Holds the data of the App in-memory.
- [Storage](#): Reads data from, and writes data to the hard disk.

Each of the four components will

- Define its *API* in an `interface` with the same name as the Component.
- Expose its functionality using a `{Component Name}Manager` class.

For example, the `Logic` component (see the class diagram given below) defines it's API in the `Logic.java` interface and exposes its functionality using the `LogicManager.java` class.

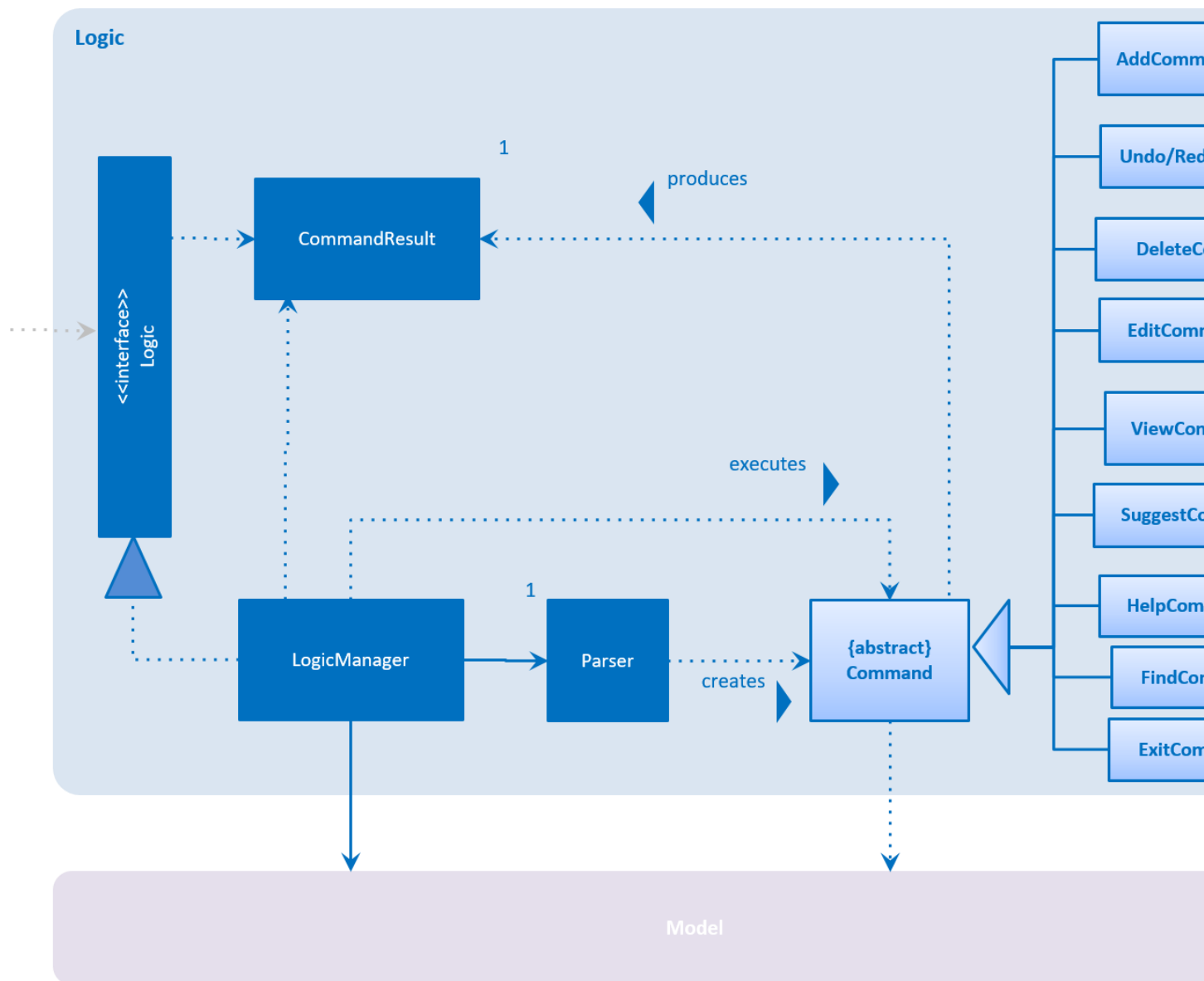


Figure 2. Overview of Logic

The *Sequence Diagram* below will show you how the components interact for the scenario where the user issues the command `delete 3`.

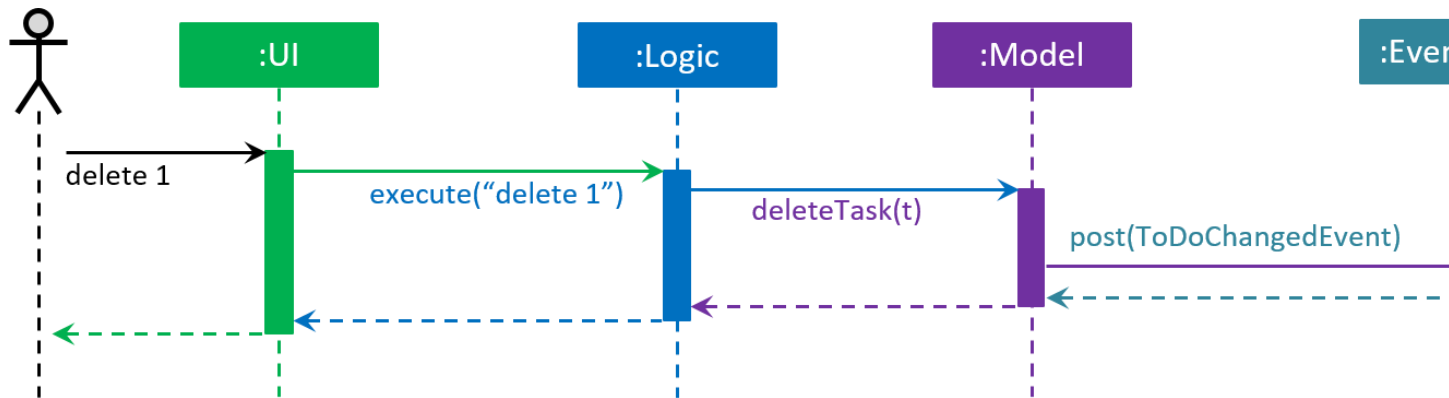


Figure 3. Sequence Diagram: Delete 1

Note how the `Model` simply raises a `TodoChangedEvent` when the To-Do data are changed, instead of asking the `Storage` to save the updates to the hard disk.

The diagram below will show you how the `EventsCenter` reacts to that event, which eventually results in the updates being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.

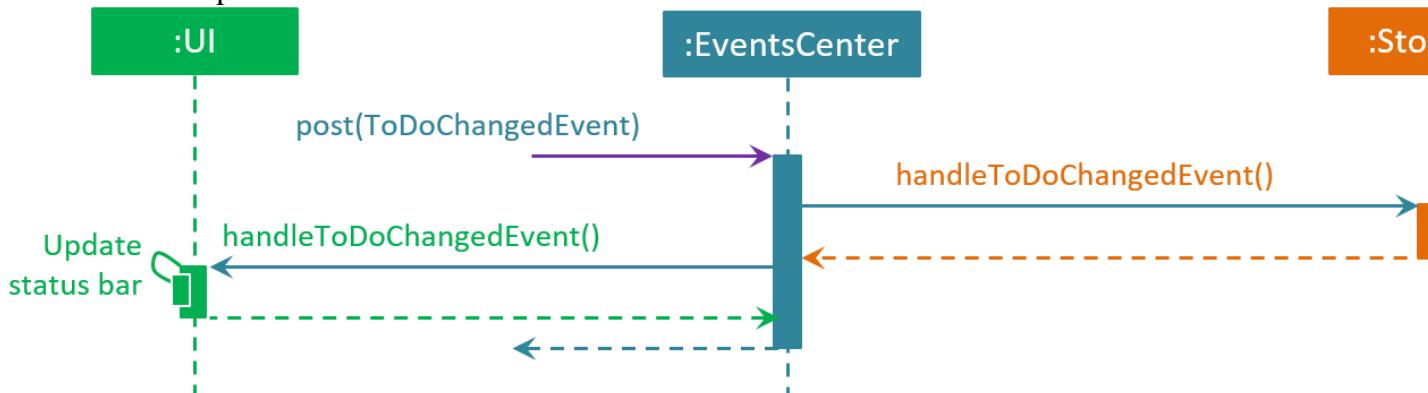


Figure 4. Sequence Diagram: TodoEventChange

Note how the event is propagated through the `EventsCenter` to the `Storage` and `UI` without `Model` having to be coupled to either of them. This is an example of how this Event Driven approach helps us reduce direct coupling between components.

The following sections will give you more details about each component.

3.2 UI component

The UI consists of a `MainWindow` that is made up of parts e.g. `CommandBox`, `ResultDisplay`, `TaskListPanel`, `StatusBarFooter`, `BrowserPanel` etc. All these, including the `MainWindow`, inherit from the abstract `UiPart` class and they can be loaded using the `UiPartLoader`.

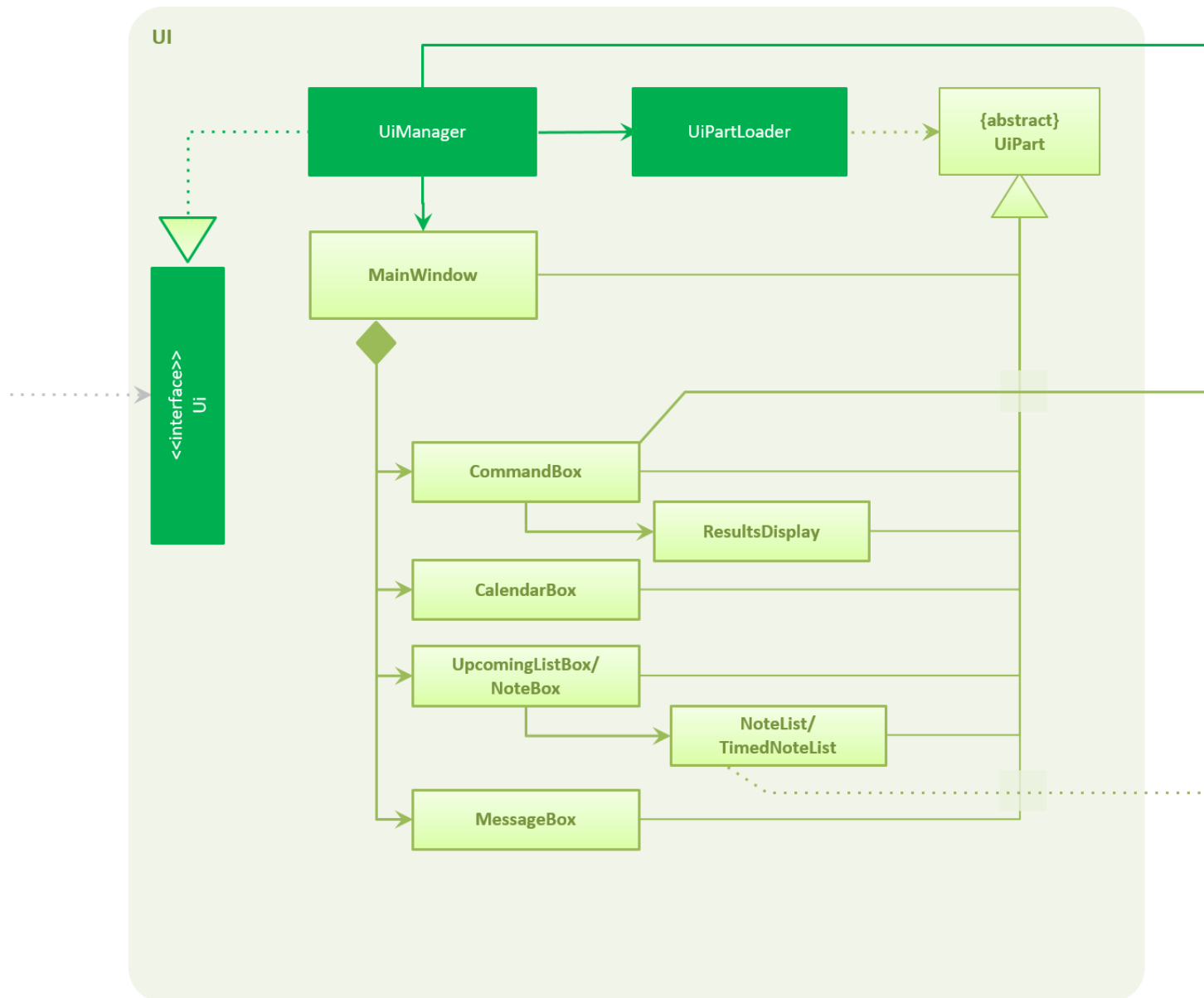


Figure 5. Overview of UI

API : [Ui.java](#)

The `UI` component uses JavaFx UI framework. The layout of these UI parts are defined in matching `.fxml` files that are in the `src/main/resources/view` folder.

For example, the layout of the [MainWindow](#) is specified in [MainWindow.fxml](#)

The `UI` component will

- Execute user commands using the `Logic` component.

- Bind itself to some data in the `Model` so that the UI can auto-update when data in the `Model` change.
- Respond to events raised from various parts of the App and updates the UI accordingly.

3.3 Logic component

Logic is in charge of reading user input and executing the correct commands. It is also in charge of give the user feedback on their input.

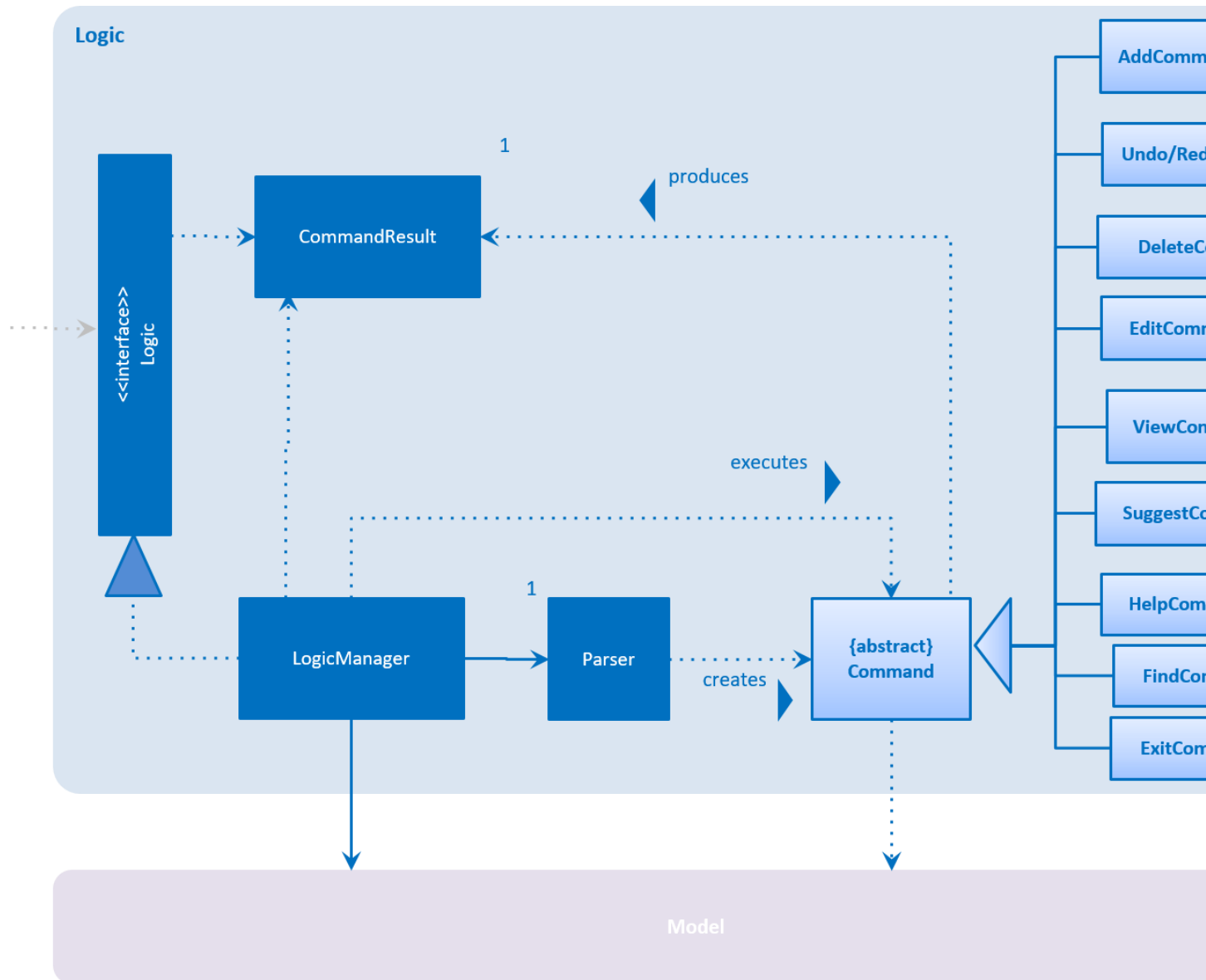


Figure 6. Overview of Logic

API : [Logic.java](#)

1. `Logic` uses the `Parser` class to parse the user command.
2. This results in a `Command` object which is executed by the `LogicManager`.
3. The command execution can affect the `Model` (e.g. adding a task) and/or raise events.
4. The result of the command execution is encapsulated as a `CommandResult` object which is passed back to the `Ui`.
5. `Logic` loads the undo/redo `Manager` which is initially an empty stack. If the command that is recently executed successfully belongs to an undoable command, the undo/redo manager will record it.

Below, you will find the Sequence Diagram for interactions within the `Logic` component for the `execute("delete 1")` API call.

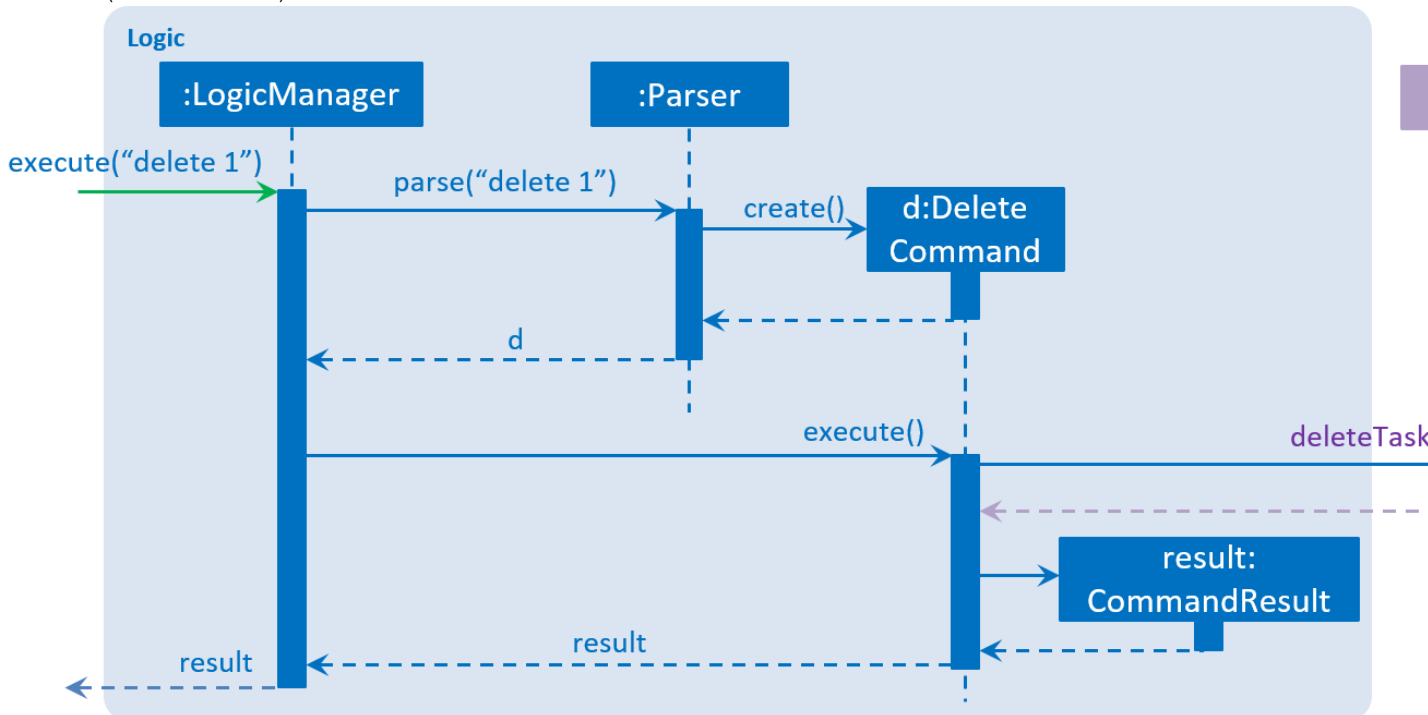


Figure 7. Sequence Diagram: Delete in Logic

3.4 Model component

Model is in charge of the structure of the to-do list, and serves as the manager of the abstraction layer between Logic and the actual list of tasks.

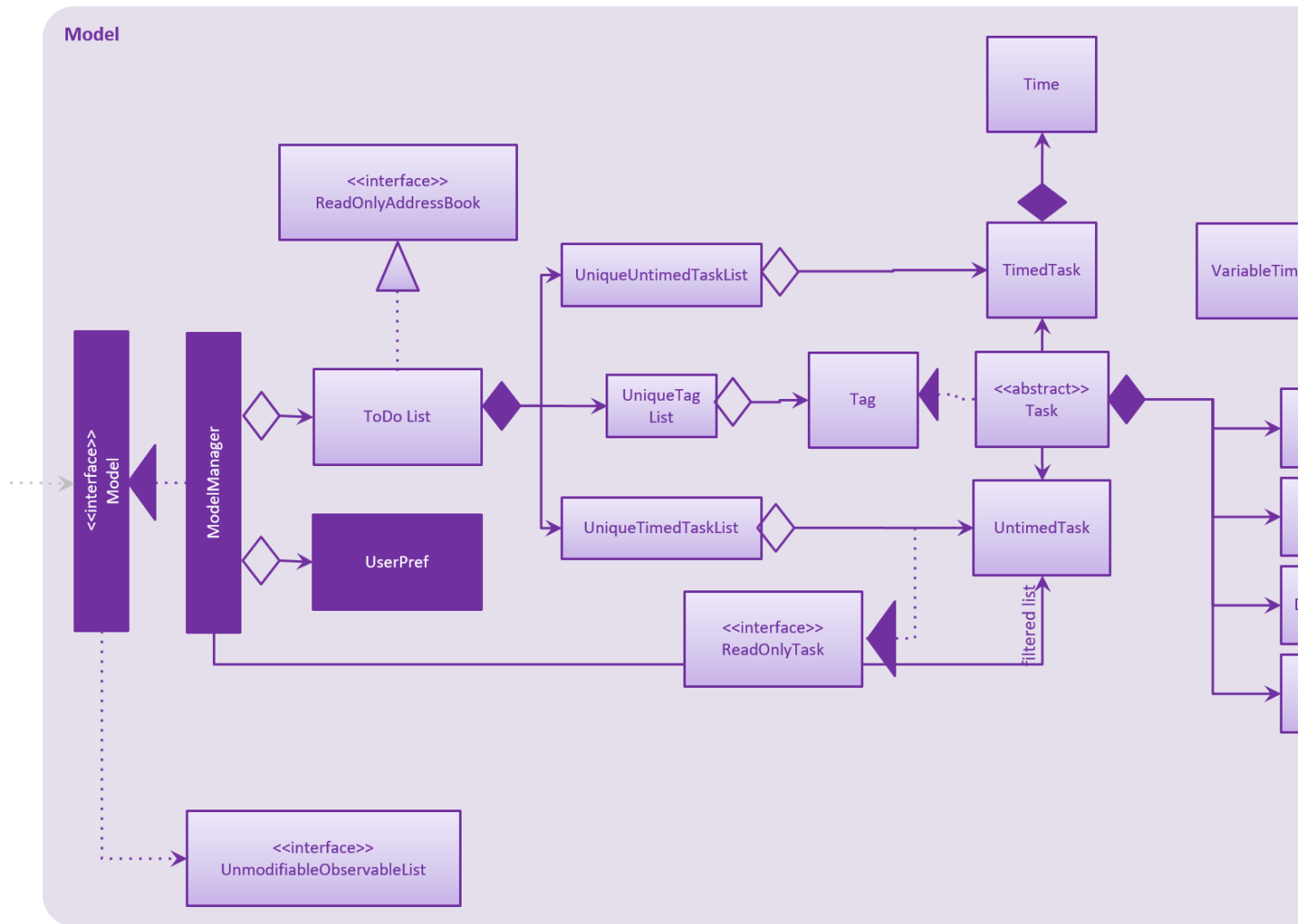


Figure 8. Overview of Model

API : [Model.java](#)

The Model,

- stores a `UserPref` object that represents the user's preferences.
- stores the To-Do data.
- exposes a `UnmodifiableObservableList<ReadOnlyTask>` that can be 'observed' e.g. the UI can be bound to this list so that the UI automatically updates when the data in the list change.
- does not depend on any of the other three components.

3.5 Storage component

Storage is in charge of saving and retrieving data from files stored on the user's device.

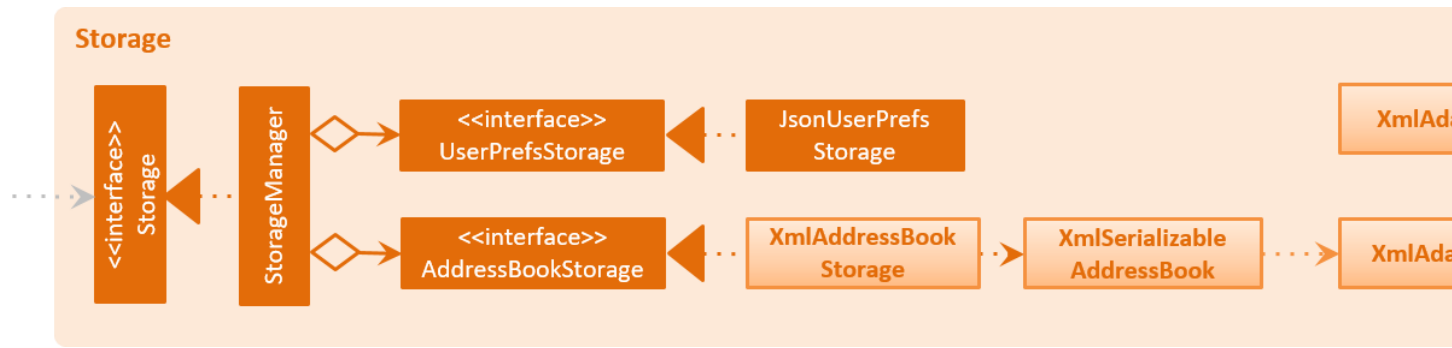


Figure 9. Overview of Storage

API : [Storage.java](#)

The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- can save the SmartyDo data in xml format and read it back.

3.6 Common classes

You may find classes used by multiple components are in the `seedu.addressbook.common` package.

4. Implementation

4.1 Logging

We are using `java.util.logging` package for logging. You can use `LogsCenter` class to manage the logging levels and logging destinations.

- You can control the logging level by using the `logLevel` setting in the configuration file (See [Configuration](#))
- You can obtain the `Logger` for a class by using `LogsCenter.getLogger(Class)` which will log messages according to the specified logging level
- Currently log messages are output through: `Console` and to a `.log` file.

Logging Levels |Level|Details| |--|---| | SEVERE | Critical problem detected which may possibly cause the termination of the application. | WARNING | Can continue, but with caution. | INFO | Information showing the noteworthy actions by the App. | FINE | Details that are not usually noteworthy but may be useful in debugging e.g. printout of the actual list instead of just its size

4.2 Configuration

You can control certain properties of the application (e.g App name, logging level) through the configuration file (default: `config.json`):

5. Testing

You can find tests in the `./src/test/java` folder.

In Eclipse:

If you are not using a recent Eclipse version (i.e. *Neon* or later), you will need to enable assertions in JUnit tests as described [here](#).

- You can run all tests by right-clicking on the `src/test/java` folder and choose `Run as > JUnit Test`
- You can also run a subset of tests by right-clicking on a test package, test class, or a test and choose to run as a JUnit test.

Using Gradle:

- You may refer to [UsingGradle.md](#) to see how to run tests using Gradle.

We have two types of tests:

1. **GUI Tests** - These are *System Tests* that test the entire App by simulating user actions on the GUI. These are in the `guitests` package.
2. **Non-GUI Tests** - These are tests not involving the GUI. They include,
 1. *Unit tests* targeting the lowest level methods/classes.
e.g. `seedu.address.commons.UrlUtilTest`
 2. *Integration tests* that are checking the integration of multiple code units (those code units are assumed to be working).
e.g. `seedu.address.storage.StorageManagerTest`
 3. *Hybrids of unit and integration tests*. These test are checking multiple code units as well as how they are connected together.
e.g. `seedu.address.logic.LogicManagerTest`

Headless GUI Testing : Thanks to the [TestFX](#) library we use, our GUI tests can be run in the *headless* mode. In the headless mode, GUI tests do not show up on the screen. That means the developer can do other things on the Computer while the tests are running. See [UsingGradle.md](#) to learn how to run tests in headless mode.

6. Dev Ops

6.1 Build Automation

You may read [UsingGradle.md](#) to learn how to use Gradle for build automation.

6.2 Continuous Integration

We use [Travis CI](#) to perform *Continuous Integration* on our projects. You may read [UsingTravis.md](#) for more details.

6.3 Making a Release

Here are the steps to create a new release.

1. Generate a JAR file [using Gradle](#).
2. Tag the repo with the version number. e.g. v0.1
3. [Create a new release using GitHub](#) and upload the JAR file your created.

6.4 Managing Dependencies

A project often depends on third-party libraries. For example, SmartyDo depends on the [Jackson library](#) for XML parsing. Managing these *dependencies* can be automated using Gradle. For example, Gradle can download the dependencies automatically, which is better than these alternatives.

- a. Include those libraries in the repo (this bloats the repo size)
- b. Require developers to download those libraries manually (this creates extra work for developers)

7.1 Appendix A : User Stories

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

Priority As a ...	I want to ...	So that I can...
* * * new user	see usage instructions	refer to instructions when I forget how to use the App
* * * user	add a task by specifying a task description only	record tasks that need to be done

Priority As a ...	I want to ...	So that I can...
* * * user	delete a task	remove entries that I no longer need
* * * user	find a task by name	locate details of tasks without having to go through the entire list
* * * user	view list of completed and pending tasks	keep track of what needs to be done
* * user with many tasks at a time	sort my tasks by different criteria	view tasks easily
* * user with large projects/ tasks	add subtasks to main task	break down larger task into smaller tasks
* * user with many unconfirmed events	allocate timeslots for tentative meetings/tasks	avoid having plans that might conflict with unconfirmed plans
* * user	undo 1 previous operation	remove commands executed by accident
* * user	specify a target folder as the data storage location	synchronise file with other applications

7.2 Appendix B : Use Cases

(For all use cases below, the **System** is the `SmartyDo` and the **Actor** is the `user`, unless specified otherwise)

Use case: Add task

MSS

1. User requests to add new task
2. SmartyDo shows list of upcoming tasks with new task added
Use case ends.

Extensions

- 1a. The given index is invalid

Use case ends

Use case: Edit task

MSS

1. User requests to view upcoming tasks
 2. SmartyDo shows a list of upcoming tasks
 3. User requests to edit a specific task in the list
 4. SmartyDo edits the task
- Use case ends.

Extensions

2a. The list is empty

Use case ends

3a. The given index is invalid

3a1. SmartyDo shows an error message

Use case resumes at step 2

Use case: Undo task

MSS

1. User requests to undo the previous command
 2. SmartyDo performs undo and shows updated list of upcoming tasks
- Use case ends.

Extensions

1a. There is no previous command

Use case ends

Use case: Redo task

MSS

1. User requests to redo the command reversed by the undo command
 2. SmartyDo performs redo and shows updated list of upcoming tasks
- Use case ends.

Extensions

1a. There is no previous undo command

Use case ends

Use case: View task

MSS

1. User requests to view upcoming tasks that matches specific `string`
 2. SmartyDo shows a list of upcoming tasks
- Use case ends.

Extensions

1a. The given `string` is invalid

Use case ends

Use case: Mark task

MSS

1. User requests to view upcoming tasks
 2. SmartyDo shows a list of upcoming tasks
 3. User requests to mark a specific task in the list
 4. SmartyDo marks the task
- Use case ends.

Extensions

2a. The list is empty

Use case ends

3a. The given index is invalid

3a1. SmartyDo shows an error message
Use case resumes at step 2

Use case: Delete task

MSS

1. User requests to view upcoming tasks
2. SmartyDo shows a list of upcoming tasks
3. User requests to delete a specific task in the list
4. SmartyDo deletes the task
Use case ends.

Extensions

2a. The list is empty

Use case ends

3a. The given index is invalid

3a1. SmartyDo shows an error message

Use case resumes at step 2

Use case: Locate task

MSS

1. User requests to view upcoming tasks
2. SmartyDo shows a list of upcoming tasks
3. User requests to locate a specific task in the list
4. SmartyDo shows location of the task
Use case ends.

Extensions

2a. The list is empty

Use case ends

3a. The given index is invalid

3a1. SmartyDo shows an error message

Use case resumes at step 2

Use case: Save file

MSS

1. User requests to save file to specific file path
2. SmartyDo saves to file path
Use case ends.

Extensions

1a. The `file path` is invalid

Use case ends

Use case: Load file

MSS

1. User requests to load file from specific `file path`
2. SmartyDo loads from `file path`
Use case ends.

Extensions

1a. The `file path` is invalid

Use case ends

7.3 Appendix C : Non Functional Requirements

1. Should work on any [mainstream OS](#) as long as it has Java 1.8.0_60 or higher installed.
2. Should be able to hold up to 2 years of entries estimated to be 8000 entries.
3. Should come with automated unit tests and open source code.
4. Should favor DOS style commands over Unix-style commands.

7.4 Appendix D : Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

7.5 Appendix E : Product Survey

Existing Product	Pros	Cons
Google Calendar	Allows creation of task and events and reside them in the same view. Free to use. Synchronises with gmail account. Allows conversion of email invites into events	Does not have blockout slots
Sticky	Free on Windows Store. Always open. Easy to bring up. Shows all items, always. Easy addition/editing/removal	No backup mechanism. Unable to change font. Manual sorting. Resets

Existing Product	Pros	Cons
Notes	of tasks. Can store notes/weblinks. Can store handwritten notes. Supports basic text formatting.	to default settings on restart. No “calendar view”. Takes up desktop space. Unable to minimise. Can be quite cluttered and messy
Todo.txt	Does not rely on network access to operate. Able to synchronise with cloud storage. Allows priority scheduling. Breaks difficult objectives into small steps to reach the goal. Records date of completion for tasks. Simple GUI and lightweight Application	No support for recurring tasks. No reminder for upcoming due dates