One Two Do

Developer Guide

Table of Contents

Introduction

This developer guide is a comprehensive document for software engineers who are contributing to our to-do list application "OneTwoDo". We welcome all of you to contribute this awesome application.

OneTwoDo is a simple yet elegant and powerful to-do list application. It is easy to use, extremely flexible and keyboard-friendly. So get ready to know more about our application!

1. Setting Up

Here are the settings you need to configure to get started with your first contribution.

1.1 Prerequisites

Make sure your computer has the following software installed to develop OneTwoDo.

1. JDK 1.8.0_60 or later. Having any Java 8 version is not enough. This app will not work with earlier versions of Java 8.
2. Eclipse IDE
3. e(fx)clipse plugin for Eclipse . You may refer to the tutorial by doing the steps 2 onwards given here: http://www.eclipse.org/efxclipse/install.html#for-the-ambitious
4. Buildship Gradle Integration plugin from the Eclipse Marketplace
5. Checkstyle Plug-in plugin from the Eclipse Marketplace

1.2 Importing the project into Eclipse

Carry out the following steps to import OneTwoDo into Eclipse Workspace.

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
3. in the prerequisites above)
4. Click File > Import
5. Click Gradle > Gradle Project > Next > Next
6. Click Browse, then locate the project's directory
7. Click Finish

Notes:

- 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.

1.3 Configuring Checkstyle

To adhere to the common coding style adopted for this project, you may want to import Checksyle into WorkSpace as well.

1. Click Project -> Properties -> Checkstyle -> Local Check Configurations -> New...
2. Choose External Configuration File under Type
3. Enter an arbitrary configuration name e.g. onetwodo
4. Import checkstyle configuration file found at config/checkstyle/checkstyle.xml
5. Click OK once, go to the Main tab, use the newly imported check configuration.
6. Tick and select files from packages, click Change..., and select the resources package
7. Click OK twice. Rebuild project if prompted

Notes

- To click on the files from packages text after ticking in order to enable the Change... button.

1.4 Troubleshooting project setup

Here we provide solutions to some common problems you may encounter in the process of setting up.

Problem: Eclipse reports compile errors after new commits are pulled from Git

- Reason: Eclipse fails to recognize new files that appeared due to the Git pull.
- Solution: Refresh the project in Eclipse:
  Right click on the project (in Eclipse package explorer), choose Gradle -> Refresh Gradle Project.

Problem: Eclipse reports some required libraries missing

- Reason: Required libraries may not have been downloaded during the project import.
- Solution: Run tests using Gradle once (to refresh the libraries).

2. Design

2.1 Architecture

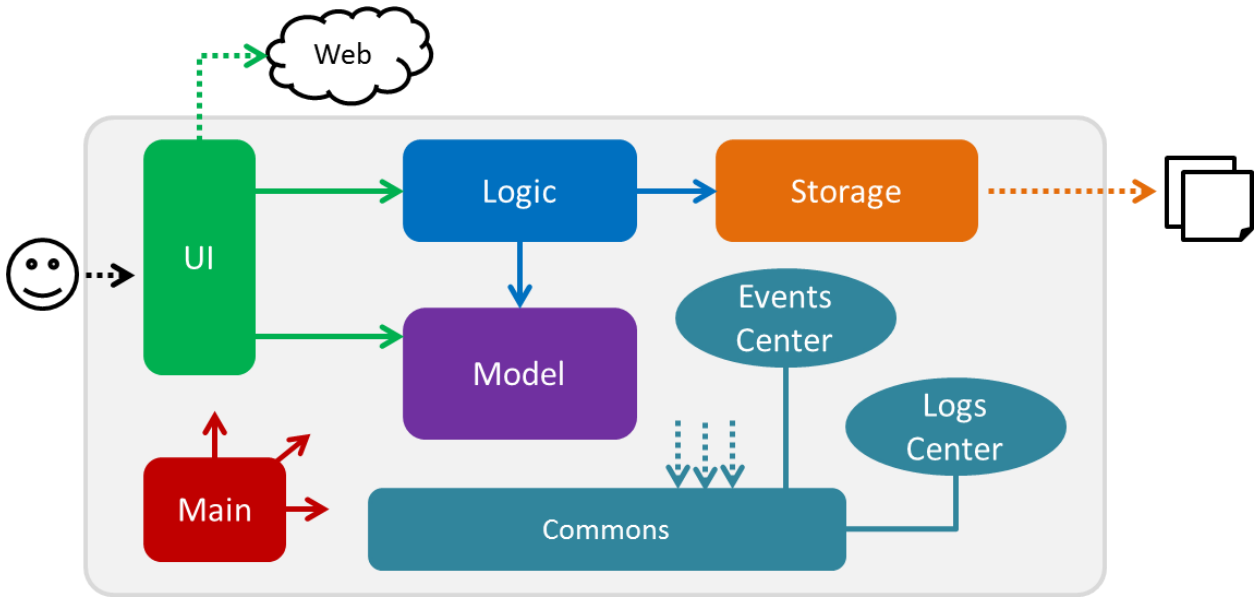

Figure 2.1.1 : Architecture Diagram

The **Architecture Diagram** given above explains the high-level design of OneTwoDo.

Given below is a quick overview of each component.

To update a diagram, modify the diagram in the pptx file, select the objects of the diagram, and choose **Save as picture**.

Main has only one class called **MainApp**. It is responsible for,

- At app launch: Initializes the components in the correct sequence, and connects them up with each other.
- At shut down: Shuts down the components and invokes 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.

- EventsCenter : 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 of four components.

- **UI** : The UI of the App.
- **Logic :** The command executor.
- **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

- Defines its **API** in an interface with the same name as the Component.
- Exposes 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.

Events-Driven nature of the design

The **Sequence Diagram** below shows how the components interact for the scenario where the user issues the

command `delete 1`

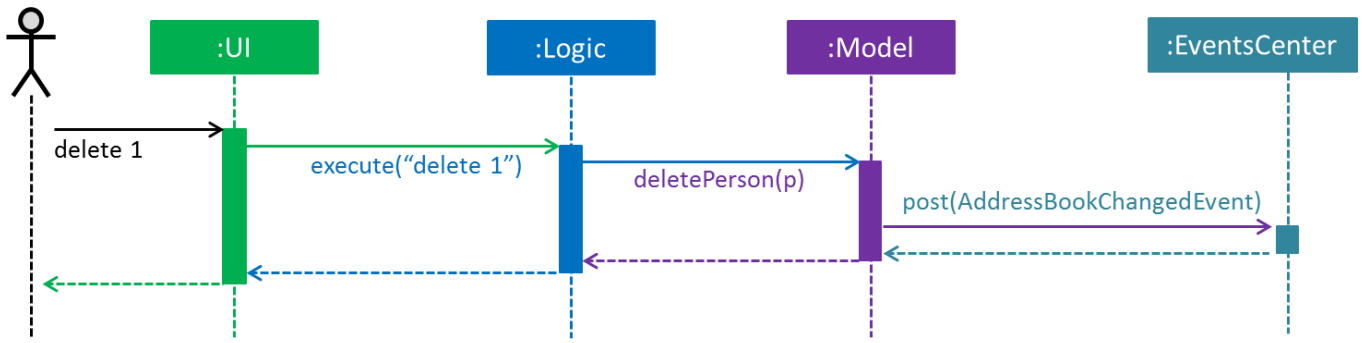

Figure 2.1.3a : Component interactions for delete 1 command (part 1)

Note how the **Model** simply raises a **OneTwoDoChangedEvent** when the OneTwoDo data are changed, instead of asking the Storage to save the updates to the hard disk.

The diagram below shows 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.

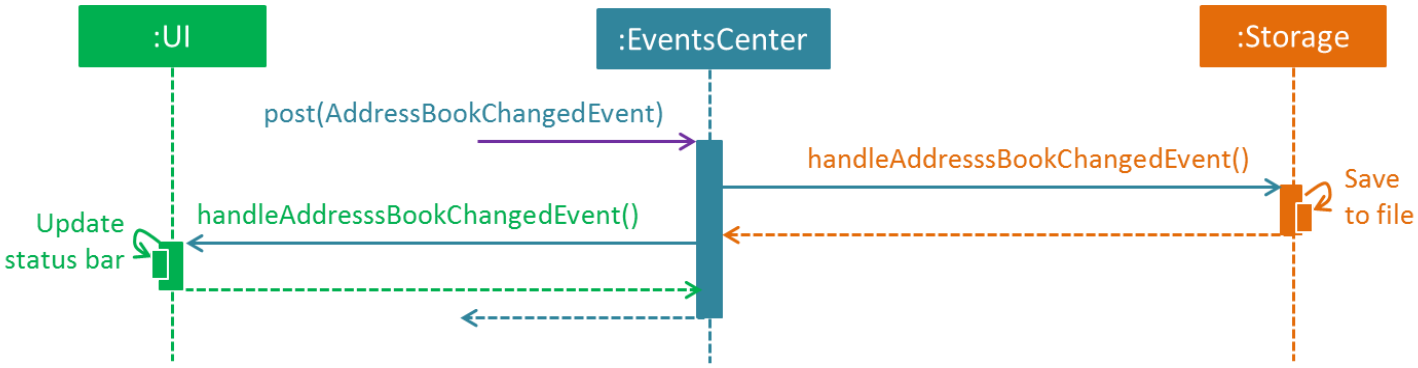

Figure 2.1.3b : Component interactions for delete 1 command (part 2)

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.

In the sections below we will provide more details of each component. As OneTwoDo is written in Object-Oriented Paradigm, you may want to pay attention to how these components work and interact with each other.

2.2. UI component

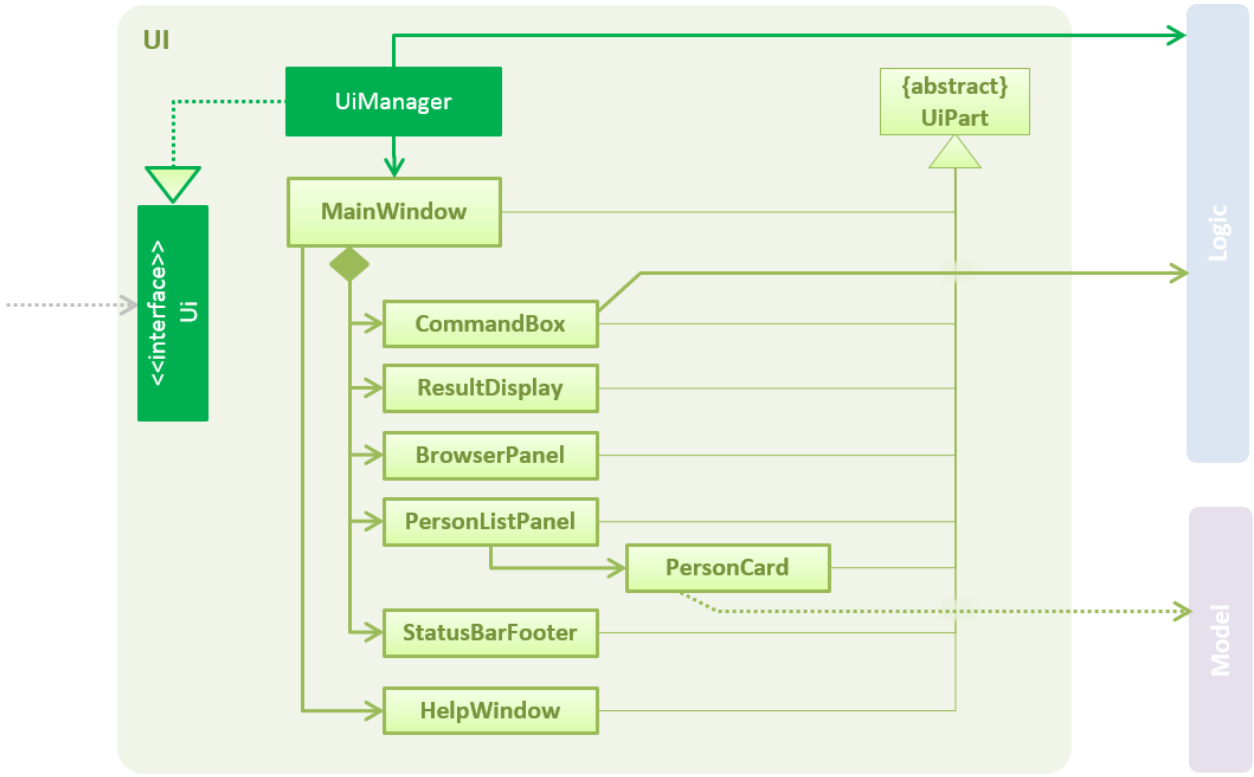

Figure 2.2.1 : Structure of the UI Component

You can find the API for this component in **Ui.java.**

The UI consists of a **MainWindow** that is made up of parts e.g.CommandBox, ResultDisplay, PersonListPanel, StatusBarFooter, BrowserPanel etc. All these, including the **MainWindow**, inherit from the abstract **UiPart** class.

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,

- Executes user commands using the Logic component.
- Binds itself to some data in the Model so that the UI can auto-update when data in the Model change.
- Responds to events raised from various parts of the App and updates the UI accordingly.

2.3. Logic component

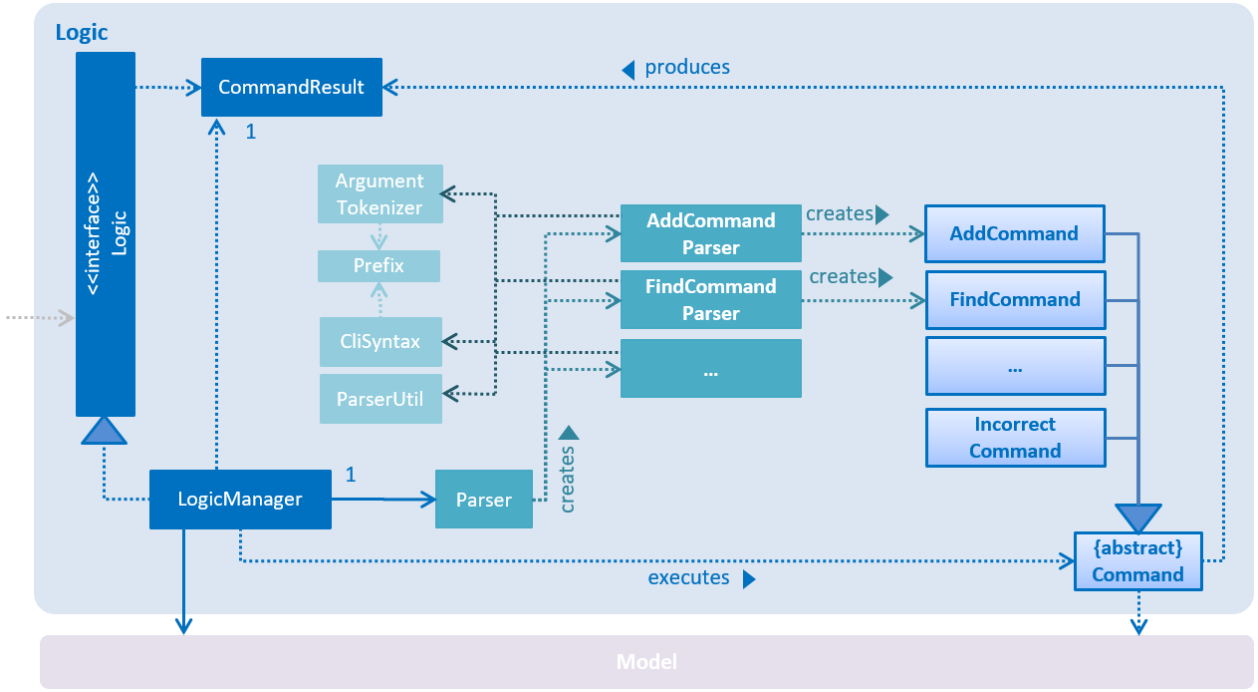

Figure 2.3.1 : Structure of the Logic Component

You can find the API for this component in **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 person) and/or raise events.
4. The result of the command execution is encapsulated as a **CommandResult** object which is passed back to the Ui.

Given below is the Sequence Diagram for interactions within the Logic component for the execute("`delete 1`") API call.

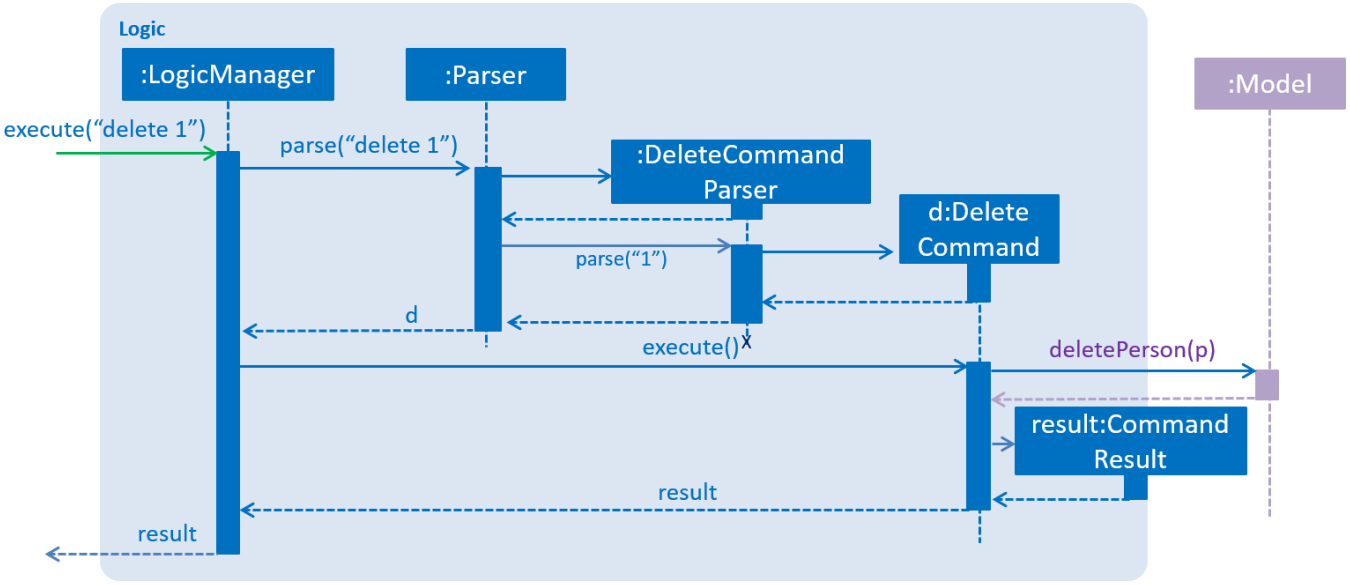

Figure 2.3.1 : Interactions Inside the Logic Component for the delete 1 Command

2.4. Model component

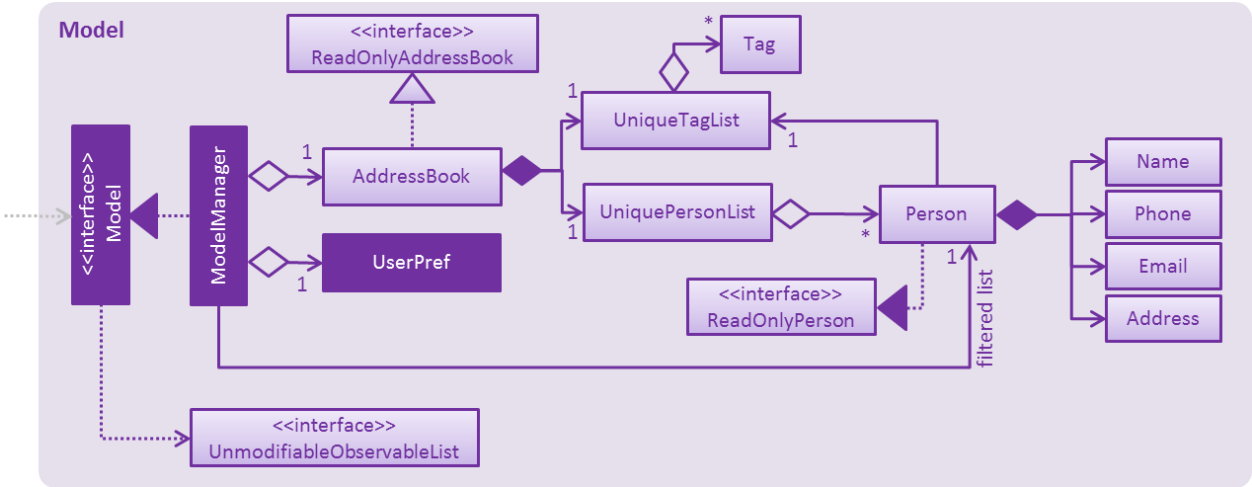

Figure 2.4.1 : Structure of the Model Component

You can find the API for this component in **Model.java**.

The Model,

- stores a **UserPref** object that represents the user's preferences.
- stores the OneTwoDo data.
- exposes a **UnmodifiableObservableList<ReadOnlyPerson>** 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.

2.5. Storage component

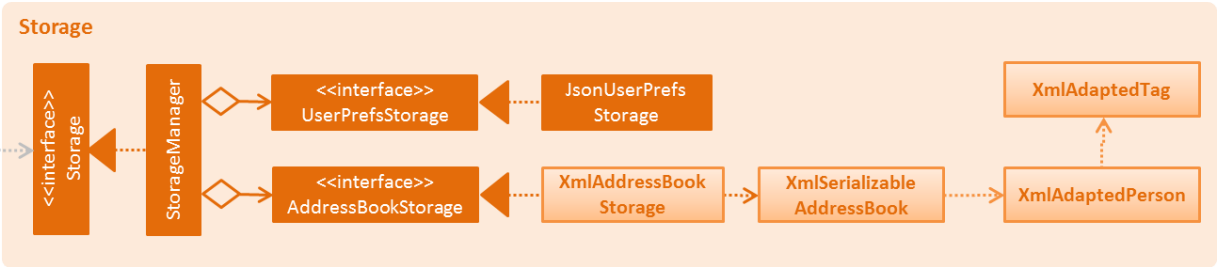

Figure 2.5.1 : Structure of the Storage Component

You can find the API for this component in **Storage.java**.

The Storage component,

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

2.6. Common classes

Classes used by multiple components are in the **seedu.onetwodo.commons** package.

3. Implementation

3.1. Logging

We are using **java.util.logging** package for logging. The **LogsCenter** class is used to manage the logging levels and logging destinations.

- The logging level can be controlled using the **logLevel** setting in the configuration file (See **Configuration**)
- The Logger for a class can be obtained 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.

There are 4 logging levels. They are:

- **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 is not usually noteworthy but may be useful in debugging
- e.g. print the actual list instead of just its size

3.2. Configuration

Certain properties of the application can be controlled (e.g App name, logging level) through the configuration file (default: **config.json**):

4. Testing

4.1. Running tests

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

Running tests in Eclipse:

- To run all tests, you can right-click on the **src/test/java** folder and choose Run as > **JUnit Test**
- To run a subset of tests, you can right-click on a test package, test class, or a test and choose to run as a **JUnit test**.

Running tests using Gradle:

- You may refer to **UsingGradle.md** for 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,
    - **Unit tests** targeting the lowest level methods/classes, e.g. seedu.onetwodo.commons.UrlUtilTest.
    - **Integration tests** that are checking the integration of multiple code units (those code units are assumed to be working), e.g. seedu.onetwodo.storage.StorageManagerTest.
    - **Hybrids** of unit and integration tests. These test are checking multiple code units as well as how the are connected together, e.g. seedu.onetwodo.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.

4.2. Troubleshooting tests

Here we provide solutions to some common problems you may encounter in the process of testing.

Problem: Tests fail because **NullPointException** when AssertionError is expected

- Reason: Assertions are not enabled for JUnit tests. This can happen if you are not using a recent Eclipse version (i.e. *Neon* or later)
- Solution: Enable assertions in JUnit tests as described here: **stackoverflow.com/questions/2522897/eclipse-junit-ea-vm-option**
  Delete run configurations created when you ran tests earlier.

5. Dev Ops

5.1. Build Automation

Please see **UsingGradle.md** to learn how to use Gradle for build automation.

5.2. Continuous Integration

We use **Travis CI** and **AppVeyor** to perform Continuous Integration on our projects.
See **UsingTravis.md** and **UsingAppVeyor.md** for more details.

5.3. Publishing Documentation

Please see **UsingGithubPages.md** to learn how to use GitHub Pages to publish documentation to the project site.

5.4. 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
4. and upload the JAR file you created.

5.5. Converting Documentation to PDF format

We use Google Chrome for converting documentation to PDF format, as Chrome's PDF engine preserves hyperlinks used in webpages. Here are the steps to convert the project documentation files to PDF format.

1. Make sure you have set up **GitHub Pages** as described in **UsingGithubPages.md**.
2. Using Chrome, go to the **GitHub Pages** version of the documentation file. e.g. For UserGuide.md, the URL will be

https://cs2103jan2017-f14-b1.github.io/main/docs/UserGuide.html.

3. Click on the Print option in Chrome's menu.
4. Set the destination to Save as PDF, then click Save to save a copy of the file in PDF format. For best results, use the settings indicated in the screenshot below.

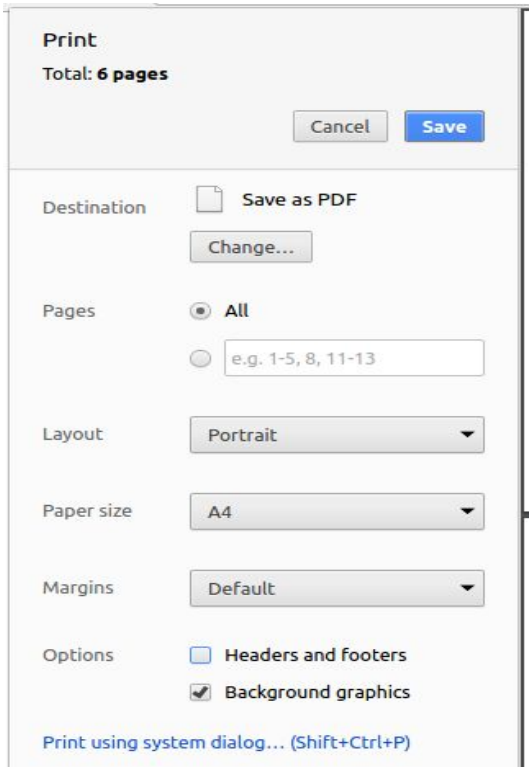

Figure 5.4.1 : Saving documentation as PDF files in Chrome

5.6. Managing Dependencies

A project often depends on third-party libraries. For example, OneTwoDo 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.

To make use of Gradle,

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

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 new task	add task to manage all my to-do
* * *	user	delete a task	remove entries that I no longer need
* * *	user with many tasks	find a task by keywords	search for specific task especially when the list is huge
* * *	user	edit a task	edit a task if there are any changes
* * *	user	clear all tasks	quickly start using the tool from scratch
* * *	user	close application	exit after use
* * *	organized user	add tags to a task	organize them by tags
* * *	user	add description to task	describe tasks in more detail
* * *	user	view task description	view the task in details
* * *	user	list tasks	view all uncompleted task
* * *	advance user	sort task by importance	prioritise which task to do first
* * *	last minute user	sort tasks by most upcoming tasks	get myself prepared for more urgent tasks
* * *	advance user	find tasks by tag	find tasks quickly that contains the specific tag

* * *	advance user	set reminder to a task	be notified when tasks are approaching due date
* * *	user	select folder to save data storage	specify a specific folder and file for data storage
* * *	user	undo most recent action	undo action to rectify mistake
* *	user	redo most recent action	redo action to rectify mistake
* *	user	add task to nearest free slot automatically	add task easily without browsing for free time
* *	user with multiple recurring tasks	add recurring tasks	avoid adding similar task one by one
* *	user with multiple recurring tasks	edit all instances of a recurring task	be more efficient when editing recurring tasks
* *	advance user	filter tasks by multiple keywords	search tasks by more specific criteria
* *	user	list all tags	quickly view all the tags in the to-do list
* *	user	keep track of completed tasks	keep them in track for future references
* *	busy user	postpone task	defer uncompleted task to near future where I am free
* *	advance user	select task and pin it at the top of display	quickly mark certain tasks as demanding of priority attention
* *	existing google calendar user	sync google calendar on create, update and delete tasks	exploit google calendar and sync with this task manager
* *	user with heavy email usage	send email notification for upcoming tasks	remind me on tasks that are approaching due date
* *	user who prefer keyboard	open/close application via keyboard shortcut	quickly open or exit program easily without mouse click
* *	advance user	remove all tasks with certain tag	quickly remove all tasks with some similarity

*	advance user	change all tag names in one go	quickly update all tasks with that tag name
*	user with a many tasks	assign priorities to tasks	distinguish and pinpoint importance task immediately just by looking
*	user	list overdue tasks	filter out overdue tasks to decide for further actions
*	user	remove tasks within certain range	easily specify date to remove tasks within that range.
*	user	list near future free slot	choose suitable free slot to add task

Appendix B : Use Cases

Please note that for all use cases below, the System refers to the OneTwoDo task manager and the Actor is the user, unless specified otherwise.

Use case : Add task

MSS

1. User input information to add task
2. System notifies user that task has been successfully added.
3. Use case ends.

Extensions

2a. Task already exists

2a1. System informs user that task already exists and doesn't add repeated task.

Use case ends

2b. Invalid command format

2b1. System informs user that command format is invalid and outputs sample correct format to user.

Use case ends

Use case : Delete task

MSS

1. System display a list of tasks.
2. User input task index to specify which task to delete.
3. System notifies user that task has been successfully deleted.
4. Use case ends

Extensions

1a. List is empty

1a1. System informs user that there is no task to delete.

Use case ends.

2a. Invalid command format

2a1. System informs user that the input is invalid and outputs sample correct format to user.

Use case resume at 1.

2b. User input index is not valid

2b1. System informs user that the input index is not valid.

Use case resume at 1.

Use case : Edit task

MSS

1. User requests list of tasks
2. System displays list of tasks summaries
3. User selects task to edit
4. System shows user task information
5. User edits and submits new task information
6. System displays changes made to task
7. Use case ends

Extensions

2a. The list is empty

2a1. System informs user that there is no task to edit.

Use case ends

3a. User task selection is invalid

3a1. System shows error message

Use case resumes from step 2

3b. System detects an error in the data that user entered

3b1. System selects valid task

3b2. User enters new data

3b3. Repeat steps 3b1 and 3b2 until data has no errors

Use case resumes from step 4.

3c. User requests to tag a task with a category that is not currently in the system

3c1. OneLine creates the category

Use case resumes from step 4.

Use case : Find task

MSS

1. User searches for task
2. System lists tasks which match the keywords entered by user
3. Use case ends

Extensions

2a. Task does not exist

2a1. System informs user that no tasks found

Use case ends

Use case : List tasks

MSS

1. User requests list of tasks
2. System displays list of tasks.
3. Use case ends

Extensions

1a. User requests to list task in a certain category

1a1. System displays list of tasks in that category

Use case ends

2a. The list is empty

2a1. System displays empty list

Use case ends

Use case : Done task

MSS

1. User requests list of tasks
2. System displays list of tasks summaries
3. User selects task to mark as done

4. System shows user remaining tasks that are uncompleted
5. Use case ends

Extensions

3a. User task selection is invalid

3a1. System shows error message

Use case resumes from step 2

Use case : Undo most recent action

MSS

1. User issues command undo
2. System reverts user's previous command and displays feedback message to user.
3. Use case ends

Extensions

2a. There is no previous command yet.

2a1. System feedbacks to user no previous command found

Use case ends

2b. Previous command is not mutable.

2b1. System feedbacks to user no action required.

Use case ends

Appendix C : Non-Functional Requirements

1. Should work on any mainstream OS.
2. Should be able to store up to 500 tasks.
3. Should work in online or offline mode.
4. Should respond to commands within 2 seconds.
5. Programme should startup within 3 seconds.
6. Programme should be open source code and readily available.
7. Should have an easy to use interface.
8. Should be able to handle exceptions from user input.
9. Should work as a standalone application.
10. Should have an executable file that can be launched by double clicking or keyboard shortcut.
11. Should not use relational databases to store data.
12. Command line interface should be the main mode of input.
13. Should not violate any copyrights.
14. Should not contain obscene or inappropriate content.

Appendix D : Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

Keyword

Words in task names, tag names, and task description

Invalid Commands

Invalid commands includes invalid arguments

Error message

Error message includes suggestion for correct command

START_DATE

START_DATE refers to start date and time entered
Defaults to time of 0000 hrs if no time is indicated

END_DATE

END_DATE refers to end of date and time entered
Defaults to time of 2359 hrs if no time is indicated

Event task

Tasks that have a specific start and end date

Deadline

Tasks that only have a specific end date

To-Do

Tasks that have no start or end date

Category

One of Event, Deadline or To-do.

Appendix E : Product Survey

Review for *Remember the Milk*

Pros:

- Easy to use interface.
- User can add recurring tasks.
- User can set a start date and end date for the task.
- User can add tag to the task.
- User can view a list of all completed tasks.
- User can set priority to task.
- User can postpone a task.

Cons:

- No online support or video tutorials for beginners.
- Pro version is very expensive and most features are only available in pro version.
- No security for data in terms of encryption or HTTPS.

Review for *Google Calendar*

Pros:

- Calendar user interface, click on events on the calendar to view more details.
- Users can edit events by dragging.
- Users are able to import calendars using .ical files.
- Users can edit a single instance of event, or recurring events.
- Users can be notified of approaching events.
- Users have daily, weekly, monthly, and "4-days" types of view.
- Users can sync and access on multiple platforms, like mobile, and desktop.

Cons:

- Have to be online to access calendar.
- No command line interface.
- Updates may take a while to sync due to network issues.

Review for *Wunderlist*

Pros:

- Allows user to note down descriptions and more information for each task.
- Allows user to create subtasks for each task.
- Allows user to set "Reminder" and "Due Date" separately.
- Allows user to highlight(star) a task.
- Separates "Delete" and "Mark as complete".
- Parses the reminder time when user types in the task name.
- Automatically creates "Today" and "Week" folder for user to see what needs to be done for today and this week (more urgent tasks).
- Allows user to categorise tasks using folders, e.g. (work, family).

Cons:

- Just highlighting (starring) is not enough. User may just highlight all tasks and that defeats the purpose of starring.
- Tasks are not sorted in chronological order in the main folder.

Review for *Habitica*

Pros:

- It has simulation game element which keep the work fun and motivating.
- It has colourful and clean user interface that make important tasks stand out.
- It supports multiple platforms such as mobile and desktop.
- It supports recurring task adding.
- It can set multiple and different reminders on any task.
- It can easily be reordered/sorted using name or tags.

Cons:

- Unable to view the calendar when selecting tasks
- Adding tasks require mouse/ touch screen instead of keyboard entering.
- Has the potential of distraction due to the existence of game elements.
- Deadline of the tasks are not shown unless clicked.
- Unable to mark tasks as done without keeping them for future references.