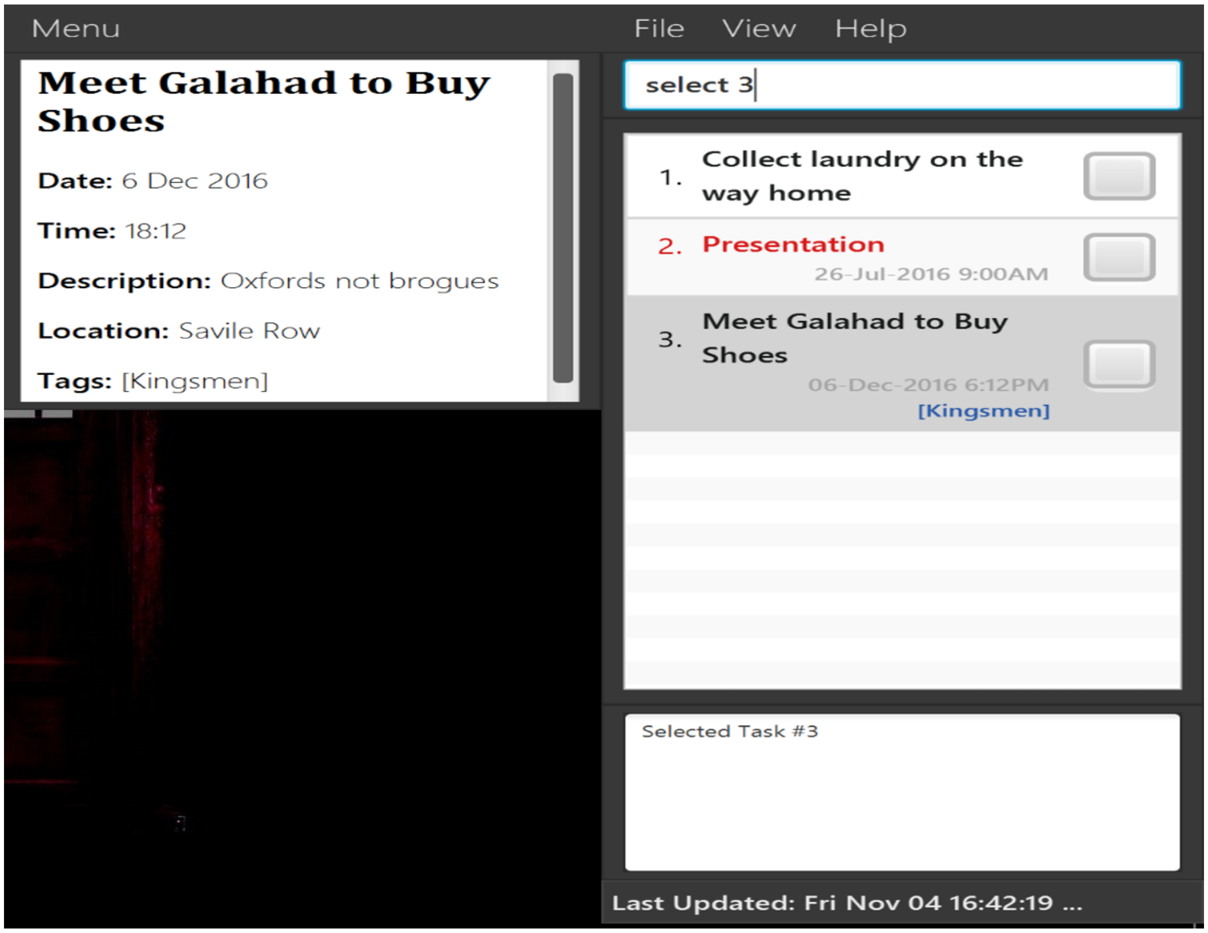
SmartyDo



Meet Galahad to Buy Shoes

Date: 6 Dec 2016

Time: 18:12

Description: Oxfords not brogues

Location: Savile Row

Tags: [Kingsmen]

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]

SmartyDo User Guide

Table of Contents

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**.

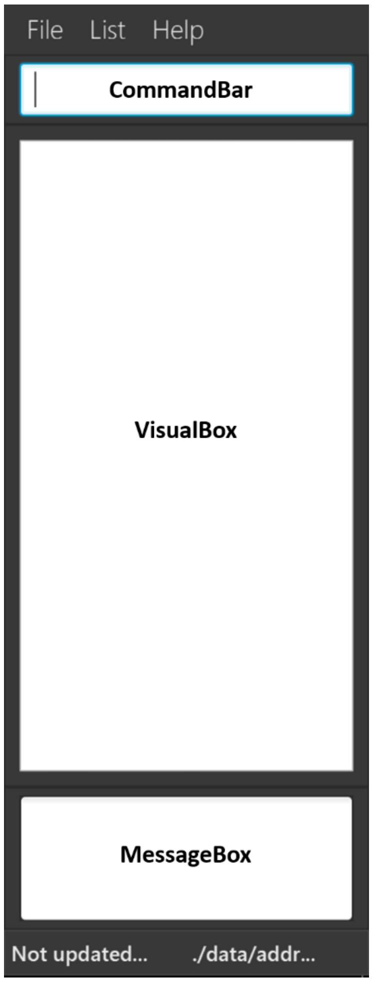


Figure 1: SmartyDo's Main Screen

Command Bar is where you enter short commands to tell SmartyDo what to do.
Visual Box is the box where the list of tasks is shown.
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:

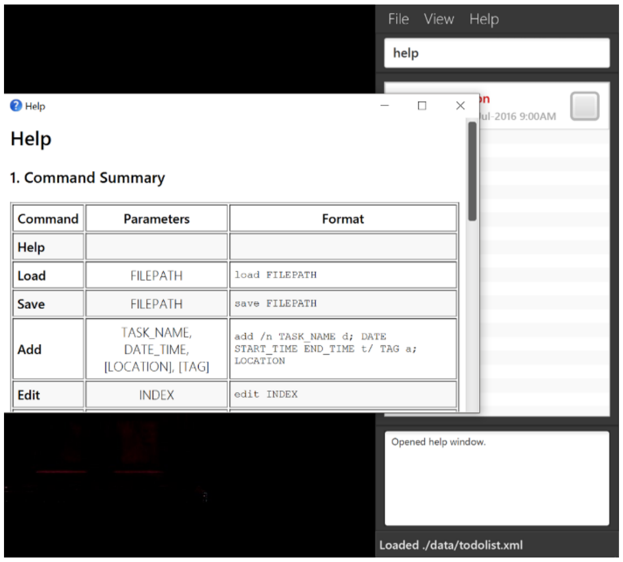
- Words in `lower_case` represent the command.
- Words in `UPPER_CASE` represent the parameters.
- Words in `[SQUARE_BRACKETS]` represent optional parameters.
- The order of parameters is flexible.

3.1. Requesting Help

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:



Help

1. Command Summary

Command	Parameters	Format
Help		
Load	FILEPATH	`load FILEPATH`
Save	FILEPATH	`save FILEPATH`
Add	TASK_NAME, DATE_TIME, [LOCATION], [TAG]	`add /n TASK_NAME d; DATE START_TIME END_TIME t/ TAG a; LOCATION`
Edit	INDEX	`edit INDEX`

```
Opened help window.
```

Loaded ./data/todolist.xml

If you wish to get help on using SmartyDo,

You may enter,

```
help
```

into the Command Bar.

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

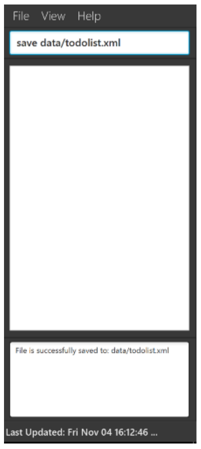
Figure 2: SmartyDo's Help Command

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:



```
save data/todolist.xml
```

```
File is successfully saved to: data/todolist.xml
```

```
Last Updated: Fri Nov 04 16:12:46 ...
```

Figure 3: SmartyDo's Save Command

If you wish to save your files to the filepath **data/todolist.xml**,

You may enter,

```
save data/todolist.xml
```

into the Command Bar.

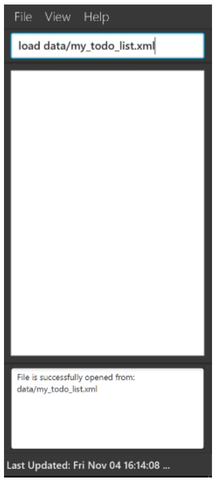
After entering the command, Message Box will show you if your new save file has been successfully created.

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:



```
load data/my_todo_list.xml
```

```
File is successfully opened from:
data/my_todo_list.xml
```

Last Updated: Fri Nov 04 16:14:08 ...

Figure 4: SmartyDo's Load Command

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.

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

3.4. Adding Tasks

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

Parameter Type	Usage	Details	Restrictions
`TASK_NAME`	[n;]	Name of the task	Alphanumeric
`[TIME]`	t;	Time specifications of the task	
`[DESCRIPTION]`	d;	Detailed description of the task	Alphanumeric
`[LOCATION]`	a;	Location relevant to the task	Alphanumeric
`[TAG]`	t/	One word description of the task	Alphanumeric

Table 1: Add Command Parameters

Format: **`add TASK_NAME d; TIME d; DESCRIPTION a; LOCATION t/TAG [t/ADDIONAL_TAGS]`**...

You do not have to enter the optional parameters if you do not need them. You may also add multiple tags to a particular task by prepending each tag with the `t/` parameter.

For example,

```
add TASK_NAME a;LOCATION t/TAG1 t;TIME t/TAG2
```

is also an accepted format for the add command.

If the `TASK_NAME` *is not the first parameter being entered, an additional delimiter* **`n;`** *is required for SmartyDo to read your command correctly.*

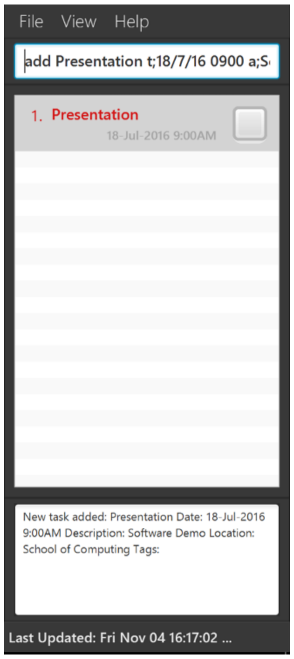
For example, the equivalent of

```
add TASK_NAME a;LOCATION t/TAG1 t;TIME t/TAG2
```

is,

```
add a;LOCATION t/TAG1 t;TIME t/TAG2 n;TASK_NAME
```

Example:



add Presentation t;18/7/16 0900 a;S

1. Presentation
18-Jul-2016 9:00AM

New task added: Presentation Date: 18-Jul-2016 9:00AM Description: Software Demo Location: School of Computing Tags:

Last Updated: Fri Nov 04 16:17:02 ...

Figure 5: SmartyDo's Add Command

If you wish to add a task named *Presentation* which occurs on 18 July 2016 at 9 am at the School of Computing,

You may enter,

```
add Presentation t;18/7/2016 0900 a;
        School of Computing
          d; Software Demo
```

into the Command Bar.

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

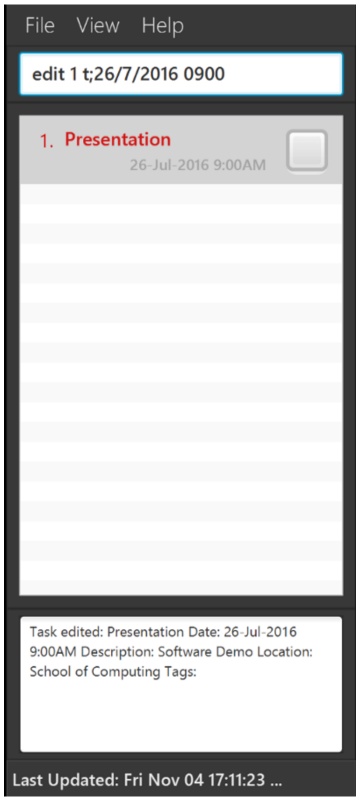
3.5. Editing Task Details

You can edit tasks you have previously entered by using the `edit` command. You can also add and remove details to a task using this function. Each task can be referred to by the index displayed in front of its title.

Format: **`edit INDEX PARAMETER_TYPE NEW_VALUE`**

> ***`PARAMETER_TYPE`*** *refers to the type of parameter (see Section 3.2) we wish to edit and* ***`NEW_VALUE`*** *is the new value for the specified parameter.*

Example:



```
edit 1 t;26/7/2016 0900
```

1. Presentation
26-Jul-2016 9:00AM

```
Task edited: Presentation Date: 26-Jul-2016
9:00AM Description: Software Demo Location:
School of Computing Tags:
```

Last Updated: Fri Nov 04 17:11:23 ...

Figure 6: SmartyDo's Edit Command

If you wish to change the date of the task named *Presentation* from 18 July to 26 July,

You may enter,

```
edit 1 t; 26/7/2016 0900
```

into the Command Bar.

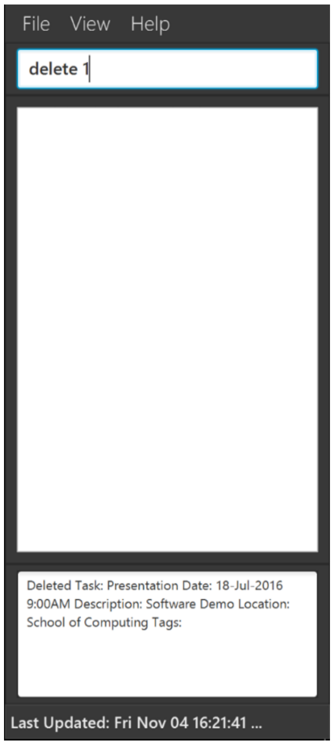
After entering the command, Message Box will show you if the task is successfully edited and you will see the updated list of tasks in the Visual Box.

3.6. Deleting Tasks

You may remove a task from the SmartyDo by using the `delete` command. Each task can be referred to by the index displayed in front of its title.

Format: **`delete INDEX`**

Example:



Deleted Task: Presentation Date: 18-Jul-2016 9:00AM Description: Software Demo Location: School of Computing Tags:

Last Updated: Fri Nov 04 16:21:41 ...

Figure 7: SmartyDo's Delete Command

If you wish to delete the task named *Presentation,*

You may enter,

```
delete 1
```

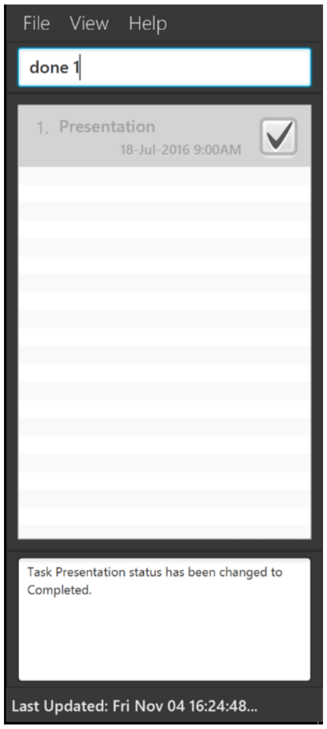
into the Command Bar.

After entering the command, Message Box will show you if the task has been successfully deleted and you will see the updated list of tasks in the Visual Box.

3.7. Marking Completed Tasks

You can mark a task as complete by using the done command. A check mark will appear next to the task, allowing you to identify the task as completed.

Format: `done INDEX`



Task Presentation status has been changed to Completed.

Last Updated: Fri Nov 04 16:24:48...

Figure 8: SmartyDo's Done Command

If you wish to mark the task named *Presentation* as complete,

You may enter,

```
done 1
```

into the Command Bar.

After entering the command, the task will be updated to reflect its new status.

3.8. Undoing and Redoing

You can reverse a change you have made to a task by using the `undo` command. Subsequently, you may also use the `redo` command to restore the changes made by `undo`. Below is a list of commands which can be undone using the `undo` command:

Undoable Commands
`add`
`edit`
`delete`
`done`

Table 2: List of Undoable Commands

Format: `undo`, `redo`

undo *requires the application to have executed at least one undoable command after launching.*

Similarly, ***redo*** *requires the application to have executed at least one successful* ***undo*** *command after launching.*

SmartyDo ***does not store*** *its history on your computer. The history of your actions resets when SmartyDo is closed. Also, if you enter any undoable command after entering redo or undo, the history of your actions will be* ***reset****.*

Example:

If you have previously made a change to a task, for example,

```
edit 1 t; 26/7/2016
```

and you wish to reverse this edit,

You may enter,

```
undo
```

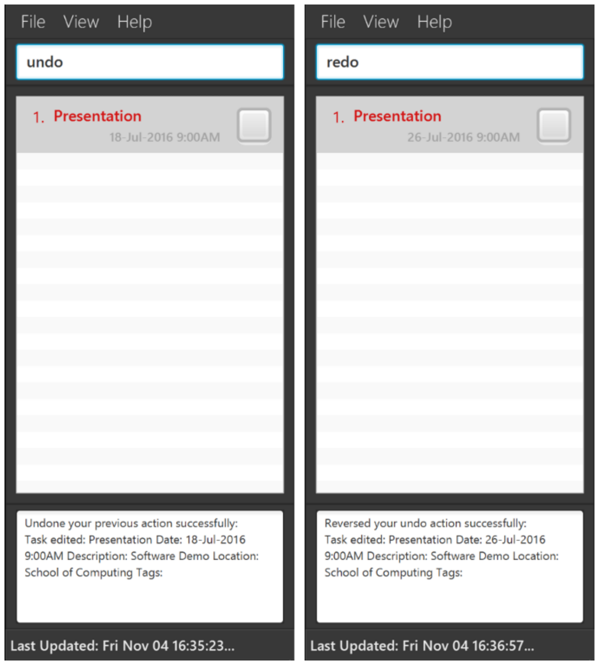
into the Command Bar.

After entering the command, Message Box will show you if the task is successfully undone and you will see the updated list of tasks in the Visual Box.

You may then reinstate the date change by entering,

```
redo
```

Into the Command Bar.



```
Undone your previous action successfully:
Task edited: Presentation Date: 18-Jul-2016
9:00AM Description: Software Demo Location:
School of Computing Tags:
```

Last Updated: Fri Nov 04 16:35:23...

```
Reversed your undo action successfully:
Task edited: Presentation Date: 26-Jul-2016
9:00AM Description: Software Demo Location:
School of Computing Tags:
```

Last Updated: Fri Nov 04 16:36:57...

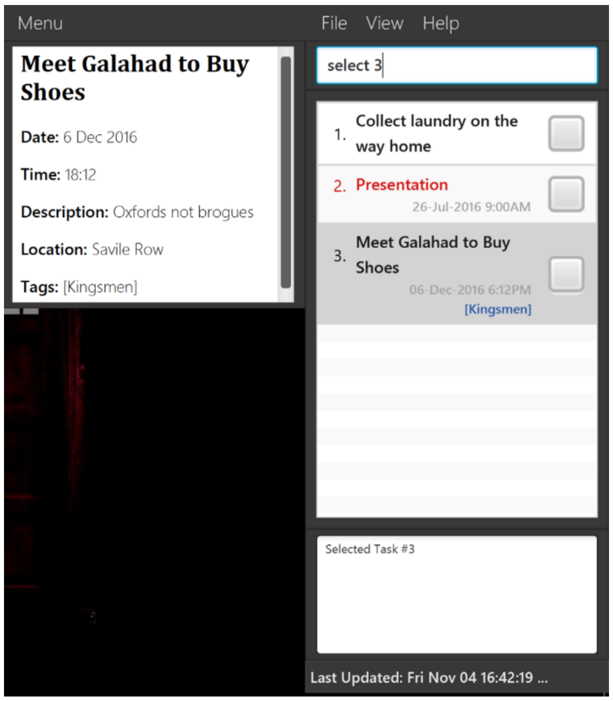
Figure 9: SmartyDo's Undo & Redo Commands

3.9. Selecting Specific Tasks

You can see details a specific task in the listed in the Visual Box by using the `select` command. A separate window will appear showing the details of the selected task. Each task can be referred to by the index displayed in front of its title.

Format: `select INDEX`

Example:



select 3

Meet Galahad to Buy Shoes

Date: 6 Dec 2016

Time: 18:12

Description: Oxfords not brogues

Location: Savile Row

Tags: [Kingsmen]

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 ...

Figure 10: SmartyDo's Select Command

If you wish to view details of the task named *Meet Galahad to Buy Shoes* which has the index of 3,

You may enter,

```
select 3
```

into the Command Bar.

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

3.10. Finding Specific Tasks

You can search for specific tasks stored in SmartyDo by using the `find` command. You may use one or more keywords as your search parameters, and tasks that match at least one keyword will be returned.

Format: **`find KEYWORD [MORE_KEYWORDS]`**

`find` is not case-sensitive and will search both the TASK_NAME and TAG for tasks that match the keywords given. In addition, the order of the keywords is not important.

For example,
`find CS2103 Project` *is equivalent to* ***`find project cs2103`***

3.11. Filtering the Task List

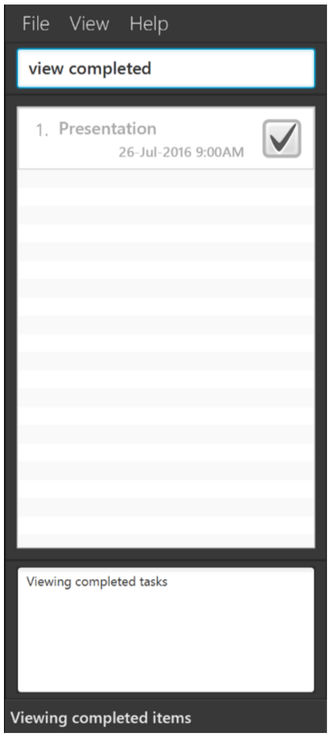
You may filter which tasks are shown in the Visual Box by using the `view` command. There are five options for filtering tasks as described below:

List Parameters	Keyboard Shortcut	Details
`all`	Ctrl + 1	Shows all tasks stored in SmartyDo
`overdue`	Ctrl + 2	Shows all incomplete tasks that are due before the current date
`upcoming`	Ctrl + 3	Shows all incomplete tasks that are due starting from the current date
`completed`	Ctrl + 4	Shows all completed tasks
`incomplete`	Ctrl + 5	Shows all incomplete tasks

Table 3: List Command Parameters

Format: `view LIST_PARAMETER`

Example:



view completed

1. Presentation 26-Jul-2016 9:00AM ✓

Viewing completed tasks

Viewing completed items

Figure 11: SmartyDo's View Command

If you wish to view a list of all completed tasks,

You may enter,

`view completed`

into the Command Bar.

After entering the command, you will see the updated list of tasks in the Visual Box.

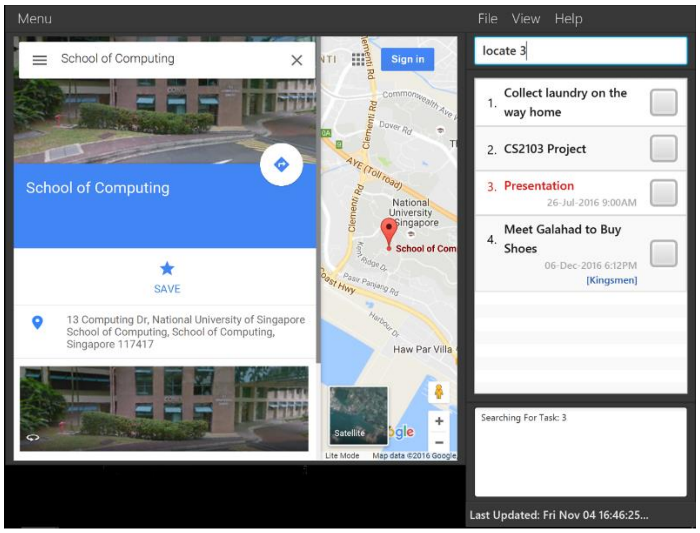
3.12. Locating a Destination

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`**

Do note that `locate` requires a working internet connection to function correctly.

Example:



locate 3

1. Collect laundry on the way home
2. CS2103 Project
3. Presentation 26-Jul-2016 9:00AM
4. Meet Galahad to Buy Shoes 06-Dec-2016 6:12PM [Kingsmen]

Searching For Task: 3

Last Updated: Fri Nov 04 16:46:25...

13 Computing Dr, National University of Singapore School of Computing, School of Computing, Singapore 117417

Figure 12: SmartyDo's Locate Command

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.

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

3.13 Clearing Saved Data

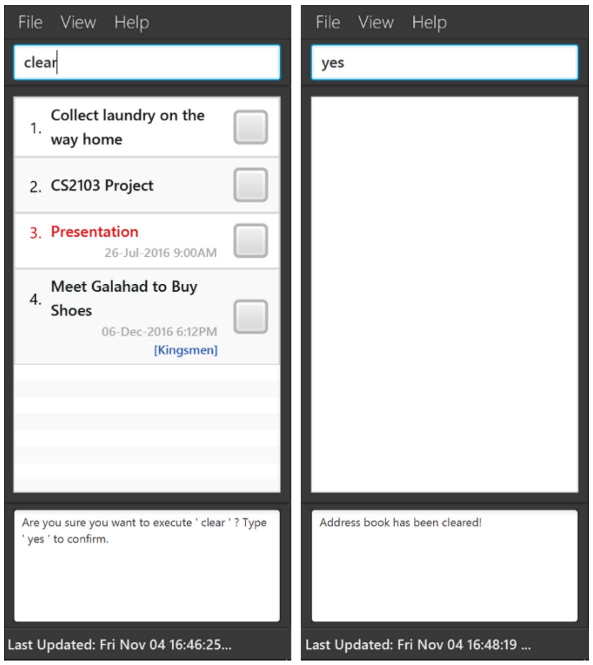
You may clear all data stored in SmartyDo by using the `clear` command. SmartyDo will prompt you to confirm this action. Enter `yes` to complete the command. Entering a different command will cancel the `clear` command.

Format: **`clear`**

*Beware! Data removed using `clear` **cannot** be recovered using `undo`. Please ensure that you have taken precautions to back-up your data if you wish to retrieve any information after using this command.*

Example:

If you wish to clear all data in SmartyDo,



Are you sure you want to execute ' clear ' ? Type ' yes ' to confirm.

Last Updated: Fri Nov 04 16:46:25...

Address book has been cleared!

Last Updated: Fri Nov 04 16:48:19 ...

Figure 13: SmartyDo's Clear Command

You may enter,

`clear`

into the Command Bar.

After entering the command, a prompt will appear in the Message Box asking you to confirm this action.

To proceed, enter

`Yes`

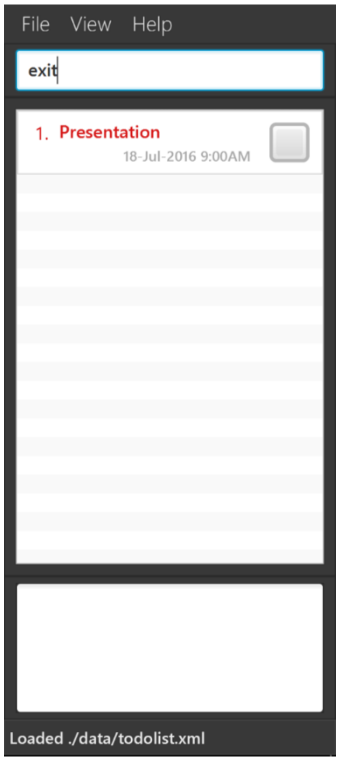
into the Command Bar, and SmartyDo will clear all saved data.

3.14. Exiting SmartyDo

You may close SmartyDo by using the `exit` command.

Format: **`exit`**

Example:



```
exit
1. Presentation
18-Jul-2016 9:00AM
Loaded ./data/todolist.xml
```

Figure 14: SmartyDo's Exit Command

If you wish to exit SmartyDo,

You may enter,

`exit`

into the Command Bar.

SmartyDo will save all your data and terminate.

4. Smart Features

4.1. FlexiCommand

It is okay if you cannot remember the syntax entirely! As long as you remember the keyword, the ordering of subsequent 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		**help**
save	FILEPATH	**save FILEPATH.xml**
load	FILEPATH	**load FIlEPATH.xml**
add	TASK_NAME, [TIME], [DESCRIPTION], [LOCATION], [TAG]	**add TASK_NAME t; TIME d;DESCRIPTION a;LOCATION t/TAG [t/ADDIONTAL_TAGS]**…
edit	INDEX, PARAM_TYPE, NEW_VALUE	**edit INDEX PARAM_TYPE NEW_VALUE**
delete	INDEX	**delete INDEX**
done	INDEX	**done INDEX**
undo		**undo**
redo		**redo**
select	INDEX	**select INDEX**
find	KEYWORD, [MORE_KEYWORDS]	**find KEYWORD [MORE_KEYWORDS]**
view	LIST_PARAM	**view LIST_PARAMETER**
locate	INDEX	**locate INDEX**
clear		**clear**
exit		**exit**

Table 4: Command Summary

5.2. Keyboard Shortcuts

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

Table 5: Keyboard Shortcuts

SmartyDo Developer Guide

Table of Contents

1. Introduction

Welcome to the developer guide for SmartyDo. SmartyDo is a to-do-list application. With SmartyDo, forgetting upcoming deadlines and sleepless nights over incomplete tasks are a thing of the past. 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.
    - Detailed instructions can be found at http://www.eclipse.org/efxclipse/install.html#for-the-ambitious
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 below will explain to you the high-level design of the App. Below, we will give you a quick overview of each component.

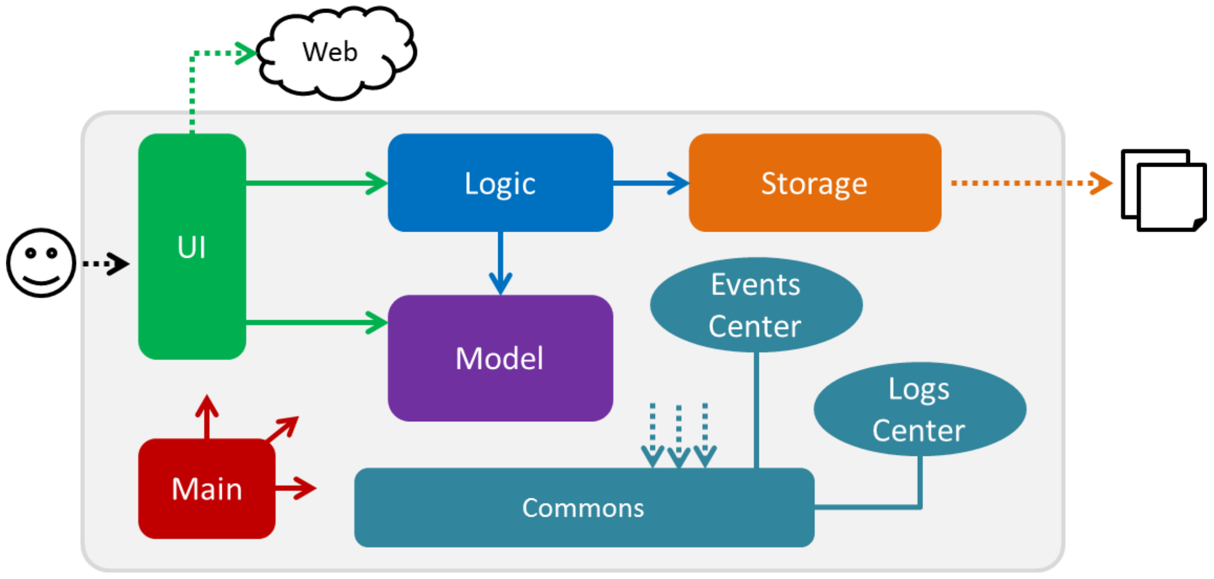


Figure 1. Overview of Main

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 its API in the Logic.java interface and exposes its functionality using the LogicManager.java class.

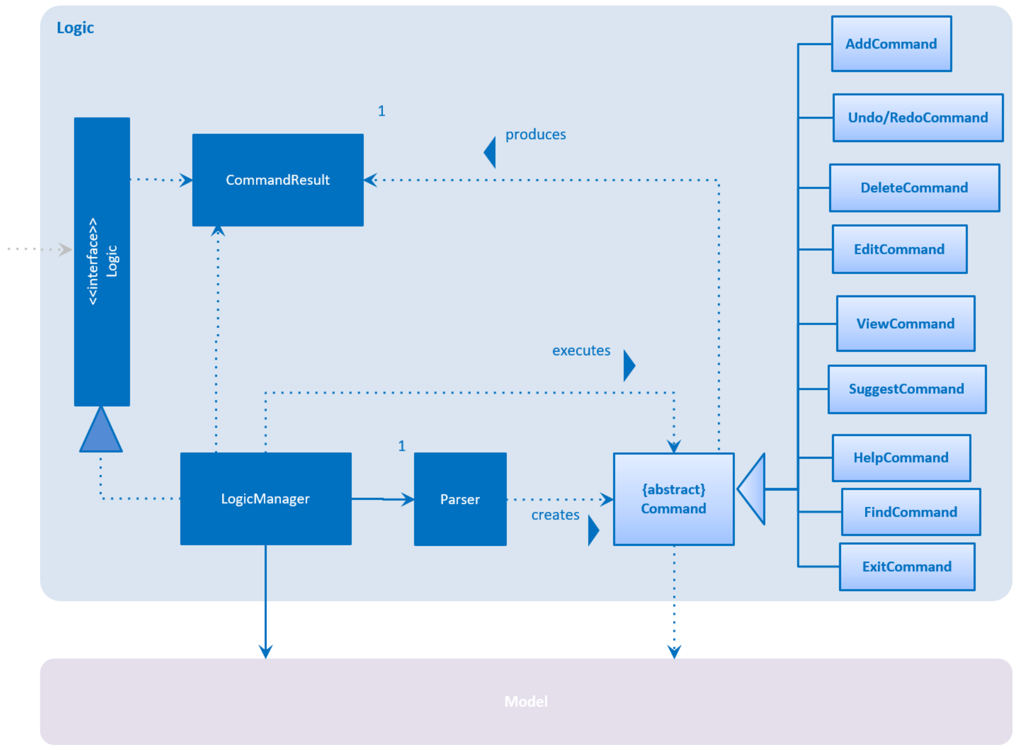


Figure 2. Overview of Logic

The *Sequence Diagram* below will show you how the components interact for the scenario where the user issues the command `delete 1`.

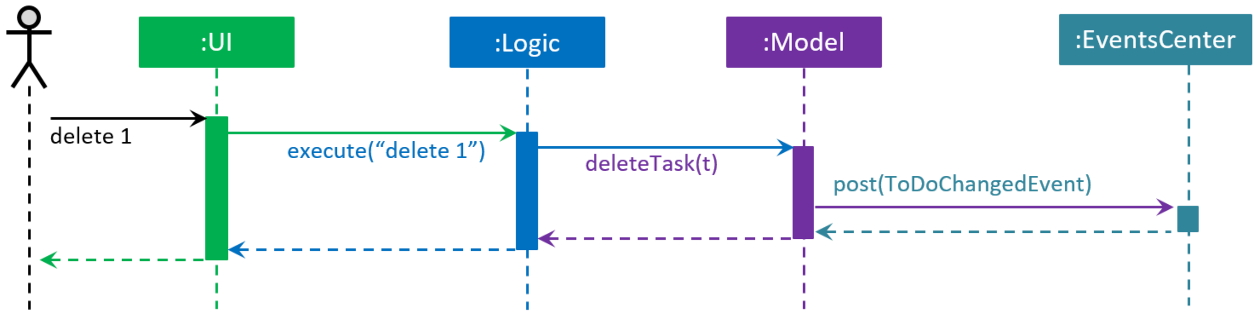


Figure 3. Sequence Diagram: Delete 3

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.

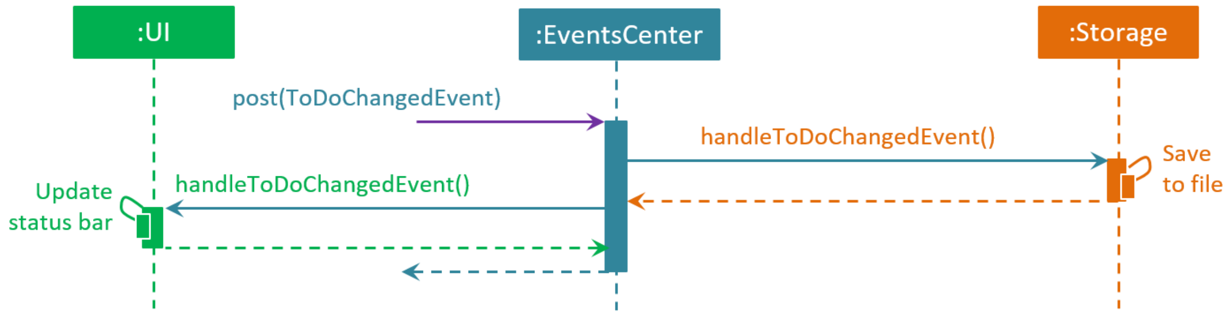


Figure 4. Sequence Diagram: ToDoChangeEvent

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 an 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

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.

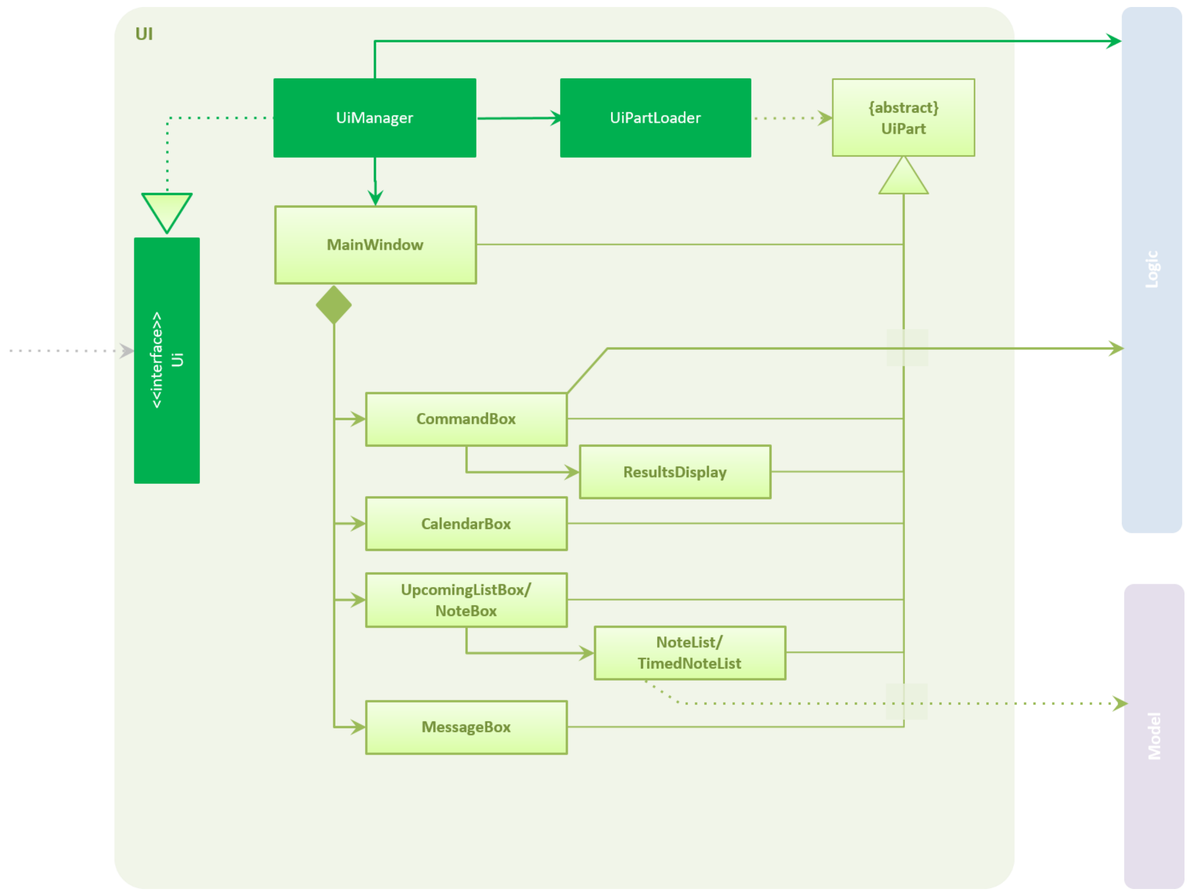


Figure 5. Overview of Ui

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.

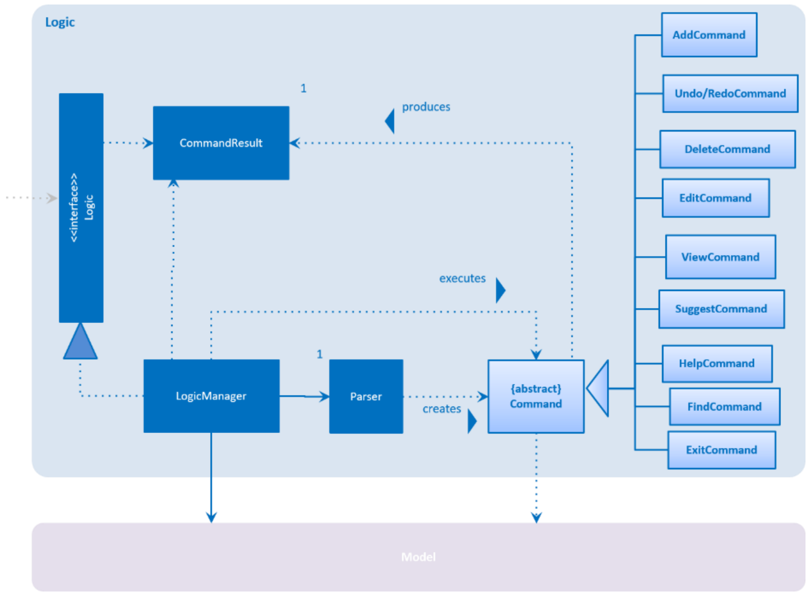


Figure 6. Overview of Logic

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.

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

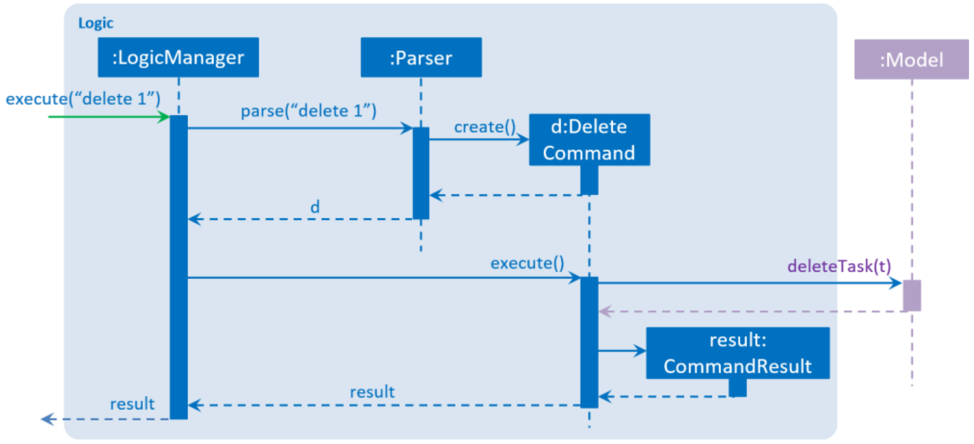


Figure 7. Sequence Diagram: Delete in Logic

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.

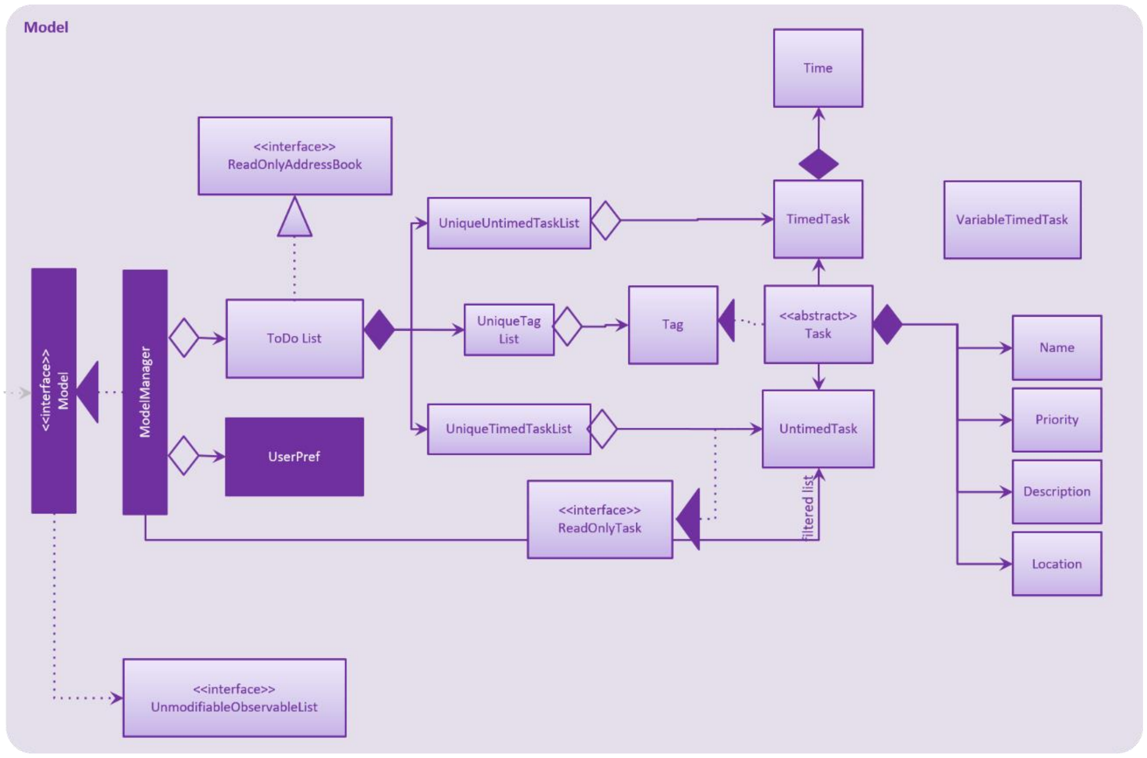


Figure 8. Overview of Model

API : Model.java

The Model,

- stores a UserPref object that represents the user's preferences.
- stores the To-Do data
- exposes an `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.

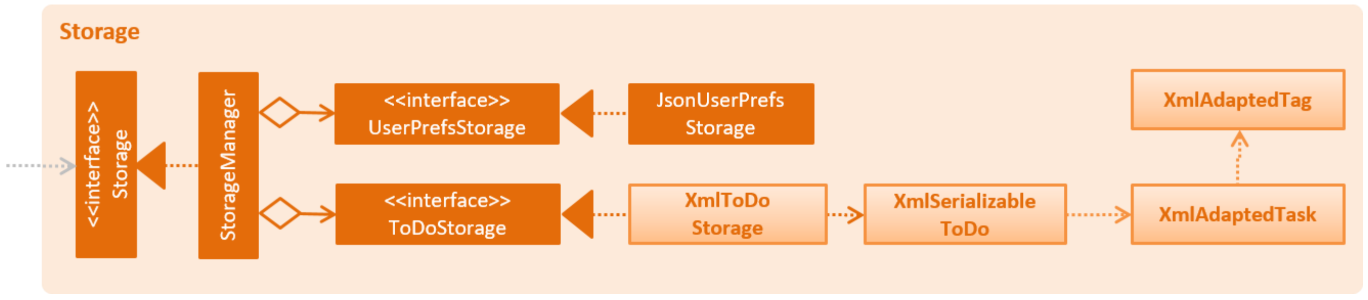


Figure 9. Overview of Storage

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.commons` 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 action 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 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 in this link: (http://stackoverflow.com/questions/2522897/eclipse-junit-ea-vm-option)

- 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,
    a. *Unit tests* targeting the lowest level methods/classes.
       e.g. `seedu.address.commons.UrlUtilTest`
    b. *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`
    c. *Hybrids of unit and integration tests*. These tests 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 you 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 the following alternatives:

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)

7. Appendix

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
* * *	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	sort my tasks by different criteria	view tasks easily
* *	user with large projects/ tasks	add subtasks to the main task	break down a 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	synchronize 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)

7.2.1 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

1.a. The given index is invalid
   Use case ends

7.2.2 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

2.a. The list is empty
   Use case ends

3.a. The given index is invalid
   3.a.1. SmartyDo shows an error message.
      Use case resumes at step 2

7.2.3 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

1.a. There is no previous command.

Use case ends

7.2.4 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

1.a. There is no previous undo command

Use case ends

7.2.5 Use case: View task

MSS

1. User requests to view upcoming tasks that match specific string
2. SmartyDo shows a list of upcoming tasks

   Use case ends

7.2.6 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

2.a. The list is empty

Use case ends

3.a. The given index is invalid

3.a.1. SmartyDo shows an error message

Use case resumes at step 2

7.2.7 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

2.a. The list is empty
    Use case ends

3.a. The given index is invalid
    3.a.1. SmartyDo shows an error message
        Use case resumes at step 2

7.2.7 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

2.a. The list is empty
    Use case ends

3.a. The given index is invalid
    3.a.1. SmartyDo shows an error message
        Use case resumes at step 2

7.2.7 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

1.a. The file path is invalid
   Use case ends

7.2.7 Use case: Delete task

MSS

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

Extensions

1.a. 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 • Synchronizes with Gmail account • Allows conversion of email invites into events	• Does not have blockout slots.
Sticky Notes	• Free to use • Easy to bring up • Shows all items, always • Easy addition/editing/removal of tasks • Can store notes/weblinks • Can store handwritten notes • Supports basic text formatting	• No backup mechanism. • No sorting. • No “calendar view”. • Takes up desktop space.
Todo.txt	• Does not rely on network access to operate • Able to synchronize with cloud storage • Allows priority scheduling • Breaks difficult objectives into small steps to reach the goal. • Records date of completion for tasks. • Simple GUI • Lightweight application	• No support for recurring tasks. • No reminder for upcoming due dates