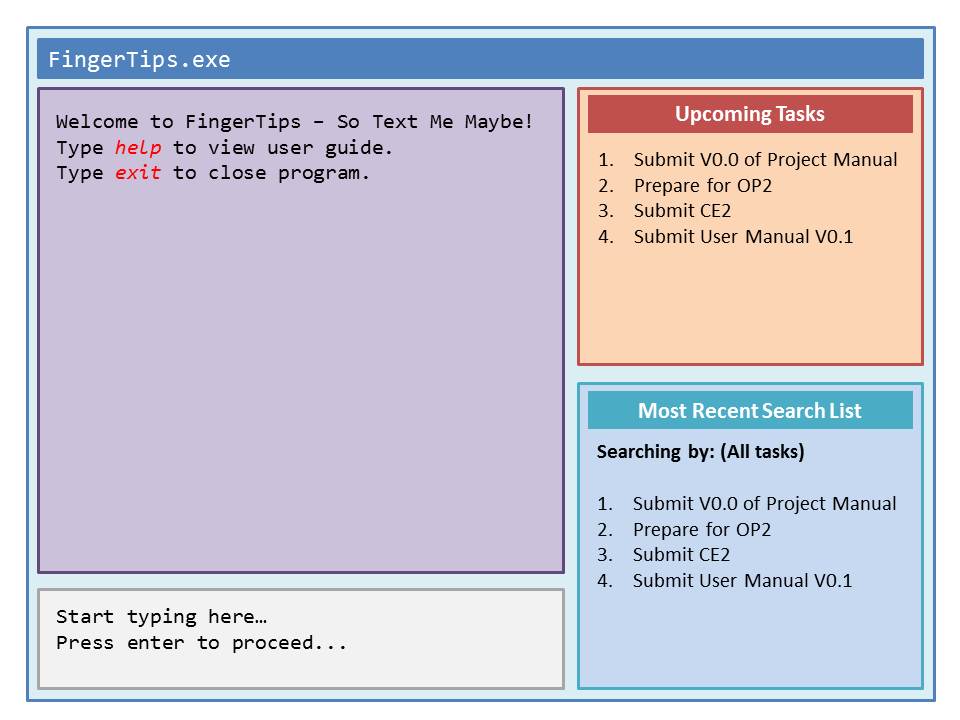
**FingerTips**

**So Text Me Maybe!**



|  |  |  |  |
| --- | --- | --- | --- |
| Raymond Cheng  Team lead, testing, coder | Farhan  Coder | Kathy Phua  Designer, dateline watcher, product tester, documentation | Koh Zhen Zhen  Designer, product tester, documentation |

# Contents

[Contents 2](#_Toc338807138)

[1 Overview 2](#_Toc338807139)

[2 User Guide 3](#_Toc338807140)

[2.1 Basic Features 3](#_Toc338807141)

[2.2 Basic Command Formats 3](#_Toc338807142)

[2.2.1 Add Event 4](#_Toc338807143)

[2.2.2 Remove Event 4](#_Toc338807144)

[2.2.3 Edit 4](#_Toc338807145)

[2.2.4 Undo Last Action 5](#_Toc338807146)

[2.2.5 Display 5](#_Toc338807147)

[2.2.6 Done 5](#_Toc338807148)

[2.2.7 Clear 6](#_Toc338807149)

[3 Developer Guide 6](#_Toc338807150)

[3.1 Introduction 6](#_Toc338807151)

[3.2 Architecture Overview 6](#_Toc338807152)

[3.3 Working With FingerTips 7](#_Toc338807153)

[3.3.1 Determine Command Type Function 7](#_Toc338807154)

[3.3.2 Add Function 7](#_Toc338807155)

[3.3.3 Display Function 8](#_Toc338807156)

[3.3.4 Remove Function 8](#_Toc338807157)

[3.3.5 Edit Function 9](#_Toc338807158)

[3.3.6 Undo Last Action Function 10](#_Toc338807159)

[3.3.7 Done Function 10](#_Toc338807160)

[3.3.8 Clear Function 10](#_Toc338807161)

[3.4 Important APIs 10](#_Toc338807162)

[3.5 Code Samples 11](#_Toc338807163)

[3.6 Testing 11](#_Toc338807164)

[3.6.1 Unit Testing 12](#_Toc338807165)

[3.6.2 Automated Testing 12](#_Toc338807166)

[4 Change Log 12](#_Toc338807167)

# Overview

*What* is **FingerTips**?

* **FingerTips** is a to-do task companion that accepts natural and simple but structured language commands via the keyboard.

*Who* is **FingerTips** designed for?

* **FingerTips** is designed specifically for users who prefer typing over using touch-motion devices like the mouse and the trackpad.
* **FingerTips** is ideal for people who are always on the go, as it works as a stand-alone application without any internet connection required.
* **FingerTips** is as easy to use as ABC with simple and short commands that even a computer dummy can pick up easily.
* Best of all, for people who are on a tight budget, it's free.

*What* does **FingerTips** do?

* Similar to a normal to-do list, the program helps the user organise and manage present and future events and tasks, stores relevant details for easy reference, through simple keyboard commands.

*How* do I install **FingerTips**?

* **FingerTips** does not need to be installed. As the program is designed to be portable, you just have to start the executable and it will be fully functional.

*How* do I start **FingerTips**?

* You can open **FingerTips** either by double clicking the program icon.

# User Guide

## 2.1 Basic Features

|  |  |
| --- | --- |
| **Function** | **Description** |
| **Add** | Create an event and add it to the database. |
| **Edit** | Add/remove information related to your current tasks/events. |
| **Remove** | Removes an event/task from the database. |
| **Display** | Shows the details of an event/task. |
| **Undo** | Undo the previous action. |
| **Done** | Updates a task’s status to completed |
| **Clear** | Deletes all entries within FingerTips |
| **Help** | Displays a list of functions, and how to access them |
| **Quit** | Exits **FingerTips** |

Bonus Features:

* **Tagging**: User can add keywords to events/tasks to make them more searchable.
* **Sorting**: User can generate a list and define how the events/tasks are sorted.
* **Priority ranking**: High priority events/tasks are displayed first to alert user to these events.

## 2.2 Basic Command Formats

### 2.2.1 Add Event

|  |
| --- |
| **Sample structure:** **a** “Meeting” 21/09/2012 6pm @U-Town  **General format:** **a**<space>”description” or **add**<space>”description”  start time am/pm<space>end time am/pm<space>dd/mm/yyyy  <space>@venue<space>#hash tag<space>priority tag  Priority – **HIGH** / **MED** / **LOW**  Note: There should be no space in each field (except for the description field). Example: @MarinaBaySands |

The **add** function fits into the natural flow of how most people would save the data in their minds. The key fields for this function to create the event would be the description.

### 2.2.2 Remove Event

|  |
| --- |
| **Sample structure:** **r** 2  **General format:** **r**<space>number or **rmv** or **remove** |

The **remove** function would be similar to how people would update a task or event as completed on a pen and paper list. This would however, remove the task from **FingerTips**, permanently. Fortunately, this action is reversible (see [**2.2.4**](#_2.2.4_Undo_Last)), but only the last action can be restored.

### 2.2.3 Edit

|  |
| --- |
| **Sample structure:** **e** 2  **General format:** **e**<space>number or **edit** |

The **edit** function should be preceded by a display function (see [**2.2.5**](#_2.2.5_Display)). The list derived from the **display** function will be used when **edit** is called. Each item on the list will be numbered. The number entered by the user will call the corresponding item on the list to be edited.

If used without doing a ‘display’ command, user will edit the item number based on the list containing all the events.

### 2.2.4 Undo Last Action

|  |
| --- |
| **Sample structure:** **undo**  **General format:** **u** or **undo** |

This function reverses the last action taken. Enter **y** to confirm the change when prompted. However, you can only undo the last action committed.

### 2.2.5 Display

|  |
| --- |
| **Sample structure:** **d** due Today  **General format:** **d**<space>due day/date<space>keyword/hashtag |

The **display** function lets you view the event details. You can either call up events with a specified keyword, or hashtag.

### 2.2.6 Done

|  |
| --- |
| **Sample structure:** **done** 3  **General format:** **done**<space>number or **fin** or **finish** |

The **done** function lets you mark a task as completed. The task will then be moved over to the archive list.

### 2.2.7 Clear

|  |
| --- |
| **Sample structure:** **clear**  **General format:** **clear** or **clr** |

The **clear** function erases ALL entries that is currently stored in FingerTips. Use this with extreme caution!

# 3 Developer Guide

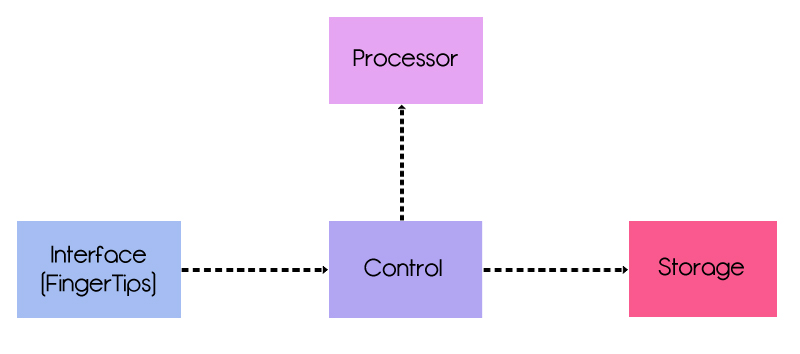
## 3.1 Introduction

This developer guide helps the reader to familiarise the various aspects of **FingerTips** by walking through the development process of **FingerTips**. In addition to providing background information on the capabilities and core functions of **FingerTips**, this guide provides examples for interacting with the application by implementing specific features.

This guide is intended for programmers who want to develop **FingerTips**. *Java* is the main language used for **FingerTips** and this guide also assumes that the reader knows how to program in *Java*.

## 3.2 Architecture Overview

The following *dependency diagram* captures the interactions between multiple components of **FingerTips**.



**FingerTips** makes use of the *n-tier* architectural style. Beginning with the implementation of a command-line Interface *(CLI),* text commands are typed into the CLI. Upon entry of a command, the entry will be parsed into the Control, which will be routed into the Processor. The Processor will tokenise the command into two parts: the command string, and the arguments string. It will then invoke the relevant command handler (e.g. *add* function), based on the command string.

Upon invoking the command handler, it will return a **COMMAND\_TYPE** object to Control. If the command is a valid operation, the command and argument strings will be passed to the Storage component, where the necessary edits will be made. This will be returned back to the user Interface as a success/failure message, with the relevant data printed, if any.

## 3.3 Working With FingerTips

**FingerTips** is a hassle-free to-do list that helps the user to manage and store tasks for easy referencing in the future, through the use of simple keyboard commands. The User Interaction with **FingerTips** will be simplified in the sections below. All the Command Type Objects are called from the Processor component.

### 3.3.1 Determine Command Type Function

**determineCommandType** will return a **COMMAND\_TYPE** object in the Processor that has a commandString attached to it. The current list of **COMMAND\_TYPES** is as follows:

|  |  |
| --- | --- |
| Task/Event Related:   * Add * Remove * Edit * Display | User Interaction:   * Help * Undo * Done * Quit |

### 3.3.2 Add Function

|  |  |
| --- | --- |
| sd add |  |
|  | |

The add function takes in the input from the user and the Processor helps to sort the input into different fields before passing it to Storage via Control. When the input is stored, Storage will then return a signal to Control that the data is successfully added. The Control will then output the success message.

### 3.3.3 Display Function

|  |  |
| --- | --- |
| sd display |  |
|  | |

The display function calls up the relevant data from the Storage component. The Processor first determines which data to be displayed, either based on **(1)** day, **(2)** date, or **(3)** hashtag. The data will then be passed back to Control, and printed out on the Interface. However, only the first 10 entries of the display results will be printed. To print the next 10 entries, user will have to input ‘next’.

The display function must be called prior to running the remove or the edit function. The display function outputs the items in a numbered list. The numbers, or indexes, are required as part of the input for the remove and edit function.

### 3.3.4 Remove Function

|  |  |
| --- | --- |
| sd remove |  |
|  | |

The remove function takes in the index of the task to be removed, in which the indexes of the tasks are shown when the display function is implemented. The input is passed to Storage which will update the file and stores a copy of the removed entry in case the undo command is called. The Storage then sends a signal back to Control that the task is successfully removed and the Control outputs the success message to the user.

### 3.3.5 Edit Function

|  |  |
| --- | --- |
| sd edit |  |
|  | |

The edit function takes in the index of the item to be edited, in which the indexes of the items are shown when the display function is implemented. The input (index/id of the task) is passed to the Storage and it will return the task and all the entered fields.

The task is displayed and the user is prompted to enter which field to edit. The id of the field to edit will be passed to the Processor, which updates the edited field in Storage and also keeps a copy in case the undo function is called.

Storage then sends a signal to inform that the data is edited successfully and a success message is returned.

### 3.3.6 Undo Last Action Function

|  |  |
| --- | --- |
| sd undo |  |
|  | |

The undo function can take place in three scenarios: **(1)** adding a task **(2)** editing a task details **(3)** removing a task. In each case, the following end results will appear respectively: **(1)** task is removed **(2)** task details are back to its prior state before editing **(3)** task is re-added.

The Control then returns a message to the user via Interface, on whether the undo function has been successful, and what data has been restored (or removed).

### 3.3.7 Done Function

The done function takes in the index of the item that is marked as done by the user. The index is then passed on to Storage where the data of the item marked as done gets moved to the Archive list.

### 3.3.8 Clear Function

The clear function generates a new file for the active list and archive list. A copy of the file is kept for the undo command, thereafter is permanently deleted if user’s next command input is not undo.

## 3.4 Important APIs

CMD.java

|  |
| --- |
| public CMD(Processor.COMMAND\_TYPE command, Object data) |

Receives a command from COMMAND\_TYPE list from Processor and returns a Data String to be output.

|  |
| --- |
| public Processor.COMMAND\_TYPE getCommandType() |

Receives a cmd String input from Interface and returns a commandType Object from Processor.

|  |
| --- |
| public void setCommandType(Processor.COMMAND\_TYPE commandType) |

Receives commandType Object from Processor.

Control.java

|  |
| --- |
| public CMD performAction(String userInput) |

Receives a cmd String input from Interface and returns a CMD Object from Processor.

|  |
| --- |
| public getPrintEntry(ArrayList<String> toPrint, Entry entry) |

Receives an Entry Object from Interface and returns an ArrayList of toPrint.

|  |
| --- |
| public String[] processEditMode(String userInput) |

Receives a cmd String input from Interface and returns a Data String from Processor.

Processor.java

|  |
| --- |
| public CMD translateToCMD(String userInput) |

Receives userInput String from Control and returns userCMD Object and String output.

Storage.java

|  |
| --- |
| public void loadFromStorage(File source, ArrayList<Entry> entries) |

Receives source File and returns an ArrayList of entries.

|  |
| --- |
| public ArrayList<Entry> getActiveEntries() |

Returns an ArrayList of ActiveEntries.

|  |
| --- |
| public ArrayList<Entry> displayAll() |

Clears the ArrayList used to store Entries to be displayed (DisplayEntries) and add on ActiveEntries to ArrayList of DisplayEntries.

## 3.5 Code Samples

This will be updated in V0.3.

## 3.6 Testing

In developing or expanding the current functions of **FingerTips**, the developer is advised to update the test cases as well. This is to prevent regressions, and ensure that most, if not all, the functions/classes are working well.

We have made use of the in-built *JUnit* in *Eclipse* to carry out unit testing for **FingerTips**. The test cases are as written in the test folder. Currently, we have built testing for the five main functions (*add, remove, edit, display, undo*), as well as handled error exception during file reading and user command input.

**Summary of Use-Test Cases**

| Use Case | Test Case No. | Test Case Description | Expected Results |
| --- | --- | --- | --- |
| add | a01 | Add normally? | New entry added |
| add | a02 | Add with invalid date | Returns invalid date error |
| add | a03 | Add with no description | Prompts user to re-enter command string again |
| remove | r01 | Remove normally | Existing entry removed |
| remove | r02 | Remove out-of-range entry | Entry to be removed not found; prompts user to re-enter command string again |
| remove | r03 | Remove a specific hashtag | All entries containing that hashtag are removed |
| remove | r04 | Remove invalid hashtag | Entries containing hashtag not found; prompts user to re-enter command string again |
| edit | e01 | Edit normally | Existing entry edited |
| edit | e02 | Edit invalid numbering input | Entry to be edited not found |
| display | d01 | Display normally | All entries displayed |
| display | d02 | Display hashtag | All entries with that hashtag displayed |
| display | d03 | Display invalid hashtag | Hashtag not found, nothing to display |
| undo | u01 | Undo previous action | Last action undone |

### 3.6.1 Unit Testing

Items tested under this section would cover:

* Five Main Use Cases/Features of **FingerTips** – Add, Remove, Edit, Undo, Display
* Valid output messages are displayed
* Valid changes made to Storage component are made
* Validation undertaken by methods i.e.*FileNotFoundException* Error

Unit Test Case Code Samples will be added in V0.3.

### 3.6.2 Automated Testing

This will be updated in V0.3.

# 4 Change Log

V0.0 (10 Sep 2012):

* Added in User Guide
  + Listed Main Functions (Add, Remove, Edit, Search, Undo)

V0.1 (16 Oct 2012):

* Updated User Guide with the three additional functions listed below
* Added in Developer Guide
  + Updated Core Functions
    - Search function updated to Display Function
    - Added in new functions: Done, Quit, Help
* First Working iteration of **FingerTips**
* Added in basic Unit Testing

V0.2 (24 Oct 2012):

* Fixed bugs and problematic functions
* Aligned text layout for display
* Added new function: Clear