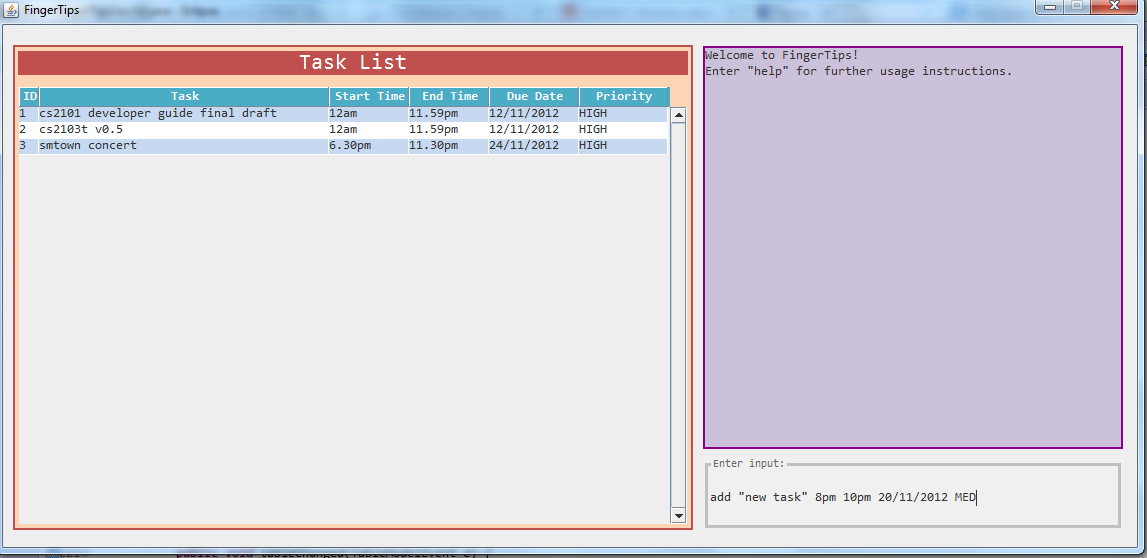
**FingerTips**

**So Text Me Maybe!**



|  |  |  |  |
| --- | --- | --- | --- |
| Raymond Cheng  Team lead,  Coder,  Product Tester | Farhan  Coder,  Documentation | Kathy Phua  Designer,  Product Tester,  Documentation | Koh Zhen Zhen  Designer,  Product Tester,  Documentation |

# Contents

[Contents 2](#_Toc340146582)

[1 Overview 3](#_Toc340146583)

[2 User Guide 3](#_Toc340146584)

[2.1 Basic Features 3](#_Toc340146585)

[2.2 Basic Command Formats 4](#_Toc340146586)

[2.2.1 Add Event 4](#_Toc340146587)

[2.2.2 Remove Event 4](#_Toc340146588)

[2.2.3 Edit 5](#_Toc340146589)

[2.2.4 Undo Last Action 5](#_Toc340146590)

[2.2.5 Redo Last Action 5](#_Toc340146591)

[2.2.6 Display 6](#_Toc340146592)

[2.2.7 Mark Done 6](#_Toc340146593)

[2.2.8 Clear 6](#_Toc340146594)

[2.2.9 Help 6](#_Toc340146595)

[2.2.10 Quit 7](#_Toc340146596)

[2.3 Bonus Features Format 7](#_Toc340146597)

[2.3.1 Hash Tag Tagging 7](#_Toc340146598)

[2.3.2 Priority Tagging 7](#_Toc340146599)

[3 Developer Guide 8](#_Toc340146600)

[3.1 Introduction 8](#_Toc340146601)

[3.2 Architecture Overview 8](#_Toc340146602)

[3.3 Working With FingerTips 8](#_Toc340146603)

[3.3.1 Determine Command Type Function 9](#_Toc340146604)

[3.3.2 Add Function 9](#_Toc340146605)

[3.3.3 Display Function 10](#_Toc340146606)

[3.3.4 Remove Function 10](#_Toc340146607)

[3.3.5 Edit Function 11](#_Toc340146608)

[3.3.6 Undo Last Action Function 12](#_Toc340146609)

[3.3.7 Redo Last Action Function 12](#_Toc340146610)

[3.3.8 Mark Done Function 12](#_Toc340146611)

[3.3.9 Clear Function 12](#_Toc340146612)

[3.4 Important APIs 12](#_Toc340146613)

[3.5 Code Samples 13](#_Toc340146614)

[3.6 Testing 14](#_Toc340146615)

[3.6.1 Unit Testing 14](#_Toc340146616)

[3.6.2 Automated Testing 15](#_Toc340146617)

[3.6.3 User Acceptance Test 15](#_Toc340146618)

[4 Change Log 15](#_Toc340146619)

# 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 application and it will be fully functional.

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

* You can open **FingerTips** 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. |
| **Redo** | Undoes the undo command. |
| **Mark 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** |

Table 1: List of functions for FingerTips

Bonus Features:

* **Tagging**: User can add keywords to events/tasks to make them more searchable.
* **Priority ranking**: High priority events/tasks are displayed first to alert user to these events.

## 2.2 Basic Command Formats

### 2.2.1 Add Entry

|  |
| --- |
| **Sample structure:** **a** “Meeting” 21/09/2012 6pm @U-Town #school HIGH  **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 between words 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 field for this function to create a task/event would be the description.

### 2.2.2 Remove Entry

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

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)).

### 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 [**2.2.6**](#_2.2.6_Display)). The list derived from the **display** function will be used when **edit** is called. Each entry on the list will be numbered. The number entered by the user will call the corresponding entry 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 in the main display panel.

### 2.2.4 Undo Last Action

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

This function reverses the last action done.

### 2.2.5 Redo Last Action

|  |
| --- |
| **Sample structure:** **redo**  **General format:** **rd** or **redo** |

This function reverses the last undo action done.

### 2.2.6 Display

|  |
| --- |
| **Sample structure:** **d** meeting  **General format:** **d**<space >keyword/hashtag |

The **display** function lets you view the event details, by searching for specific keywords, or hashtag. Calling the display function without any keyword or hashtag would show all the entries.

### 2.2.7 Mark 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.

### 2.2.8 Clear

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

The **clear** function erases ALL data that is currently stored in FingerTips. Use this with extreme caution! This action is reversible by using the undo command (see [**2.2.4**](#_2.2.4_Undo_Last)).

### 2.2.9 Help

|  |
| --- |
| **Sample structure:** **help**  **General format:** **help** or **h** |

Calling up the **help** function displays all the possible command syntaxes/formats within **FingerTips**, (as shown above), as well as the syntax/format for the edit mode for a task.

### 2.2.10 Quit

|  |
| --- |
| **Sample structure:** **quit**  **General format:** **quit** or **q** or **exit** |

To close FingerTips, simple type the quit command. All data and any changes made will be saved.

## 2.3 Bonus Features Format

### 2.3.1 Hash Tag Tagging

|  |
| --- |
| **Sample structure:** **a** “Meeting” 21/09/2012 6pm @U-Town **#school** HIGH  **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 |

The hash tag tagging can be done in **add** or **edit** functions. Similar to what is done in Twitter, users can group similar tasks together, i.e. #school or #homework. Users can add only one hash tag per entry, so choose carefully!

### 2.3.2 Priority Tagging

|  |
| --- |
| **Sample structure:** **a** “Meeting” 21/09/2012 6pm @U-Town #school **HIGH**  **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** |

Tasks can be grouped according to their level of priority – High (Very Important), Medium (Important) and Low (Not So Important). In this way, you can keep your tasks organised, and better manage your time by prioritising the tasks that have a relatively higher priority.

# 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**:

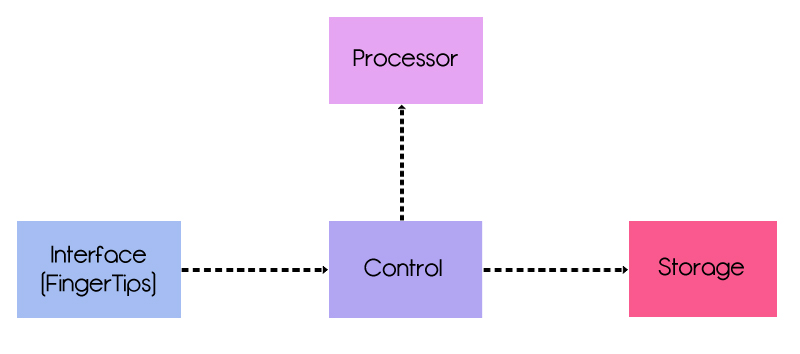


Figure 1: Dependency, Architectural Diagram for **FingerTips**

**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 portion.

Upon invoking the respective command handler (e.g. add function), 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 via **Control**, 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 explained in the sub-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 * Clear | User Interaction:   * Help * Undo * Redo * Mark Done * Quit |

Table 2: Breakdown of User Commands available

### 3.3.2 Add Function

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

Figure 2: Sequence Diagram for add function

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 |  |
|  | |

Figure 3: Sequence Diagram for display function

The display function calls up the relevant data from the Storage component. The Processor first determines the 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 |  |
|  | |

Figure 4: Sequence Diagram for remove function

The pre-requirement for this function would be to have the user enter the display function first. The edit function then takes in the index of the task to be edited, based on the index of tasks shown after the display function. **FingerTips** will then retrieve the data from Storage, returning the task details.

The input is passed to Storage which will update the file and store 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 displays the success message to the user.

### 3.3.5 Edit Function

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

Figure 5: Sequence Diagram for edit function

The pre-requirement for this function would be to have the user enter the display function first. The edit function then takes in the index of the task to be edited, based on the index of tasks shown after the display function. **FingerTips** will then retrieve the data from Storage, returning the task details.

The task is displayed and the user is prompted to enter the field to be edited. The id of the field to be edited 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 displayed.

### 3.3.6 Undo Last Action Function

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

Figure 6: Sequence Diagram for undo function

The undo function can take place in the following four scenarios: **(1)** adding a task **(2)** editing a task details **(3)** removing a task and **(4)** using the clear function (see [**3.3.9**](#_3.3.8_Clear_Function)). 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 **(4)** all entries are restored in the active and archive list.

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

### 3.3.7 Redo Last Action Function

The redo function reverses the last undo action committed (for more details on the undo function, see [**3.3.6**](#_3.3.6_Undo_Last)).

### 3.3.8 Mark 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 to Storage where the data of the item marked as done is moved to the Archive list.

### 3.3.9 Clear Function

The clear function recreates new files for the active list and archive list.

## 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 displayed as 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 ActiveEntries to ArrayList of DisplayEntries.

## 3.5 Code Samples

|  |
| --- |
| **public** **static** Control getInstance(){  **if** (*control* == **null**) {  *control* = **new** Control();  }  **return** *control*;  } |

In FingerTips, we have adhered to the singleton pattern principle. This involves restricting to a single instance of Control, Processor, and Storage, through checking on start to see if any past copies have been made.

We have also followed the principle of single responsibility principle. This involves containing all ensuring that each class has a single responsibility, and that responsibility should be encapsulated within that class. Coupling is also reduced by limiting the interaction between variables and objects of different classes. This prevents major changes when only a small code of the section is changed. In this way, the Law of Demeter is upheld.

More code examples will be update in V0.5.

## 3.6 Testing

In developing or expanding the current functions of **FingerTips** (hereby referred to in this section as *Software Under Test*, or *SUT*), 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 strive to have at least 80% code coverage testing in our Unit Testing and Automated Test Cases, and they are explained in the below two sub-sections.

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.

| 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 is removed |
| Remove | r02 | Remove out-of-range entry | Entry to be removed is not found; Prompts user to re-enter command string again |
| Remove | r03 | Remove a specific hashtag | All entries containing that specified hashtag(s) are removed |
| Remove | r04 | Remove invalid hashtag | Entries containing hashtag is not found; prompts user to re-enter command string again |
| Edit | e01 | Edit normally | Existing entry to be edited is displayed |
| Edit | e02 | Edit invalid numbering input | Entry to be edited is not found |
| Display | d01 | Display normally | All entries are displayed |
| Display | d02 | Display hashtag | All entries with specified hashtag are displayed |
| Display | d03 | Display invalid hashtag | If hashtag is not found, an error message will appear, and no entries will be displayed |
| Undo | u01 | Undo previous action | Last action is undone |
| Clear | c01 | Clear all existing entries | All entries are cleared |

Table 3: Summary of Use-Test Cases

### 3.6.1 Unit Testing

Items tested under this section would cover:

* The Main Use Cases/Features of the SUT – Add, Remove, Edit, Undo, Display, Clear, Mark Done
* Valid output messages are displayed
* Commands are executed correctly by each method in the respective classes, i.e. the method(s) functions as intended
* Valid changes to the Storage component are made
* Validation undertaken by methods i.e.*FileNotFoundException* Error

Unit Testing Code Samples will be added in V0.5.

### 3.6.2 Automated Testing

This involves running FingerTips through a wide variety of data inputs to simulate a user using the SUT. At the moment, only partial Automated Testing is done (i.e. Code Coverage is currently not at 80%).

Automated Testing Code Samples will be updated in V0.5.

### 3.6.3 User Acceptance Test

Since the software is designed with the Users in mind, what better way to test this software then having a sample of users doing the User Acceptance Test (UAT)? This involves conducting a field trial to verify the working integrity of the software through daily and rigorous usage. This allows us to find flaws in the SUT’s design or architecture, which may not be found via the two testing methods above (see [**3.6.1**](#_3.6.1_Unit_Testing), and [**3.6.2**](#_3.6.2_Automated_Testing)).

# 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, Undo

V0.3 (31 Oct 2012):

* Added partial implementation of GUI
  + FingerTips will still run via CLI as this stage since not all functions are working in the GUI
* Added in Partial Unit Testing and Automated Testing
* Streamlined the codes for Processor, Control and Storage
* Applied several patterns and principles that help to improve **FingerTips** workflow

V0.4 (7 Nov 2012):

* Bug fixes for GUI
* Refactored code from v0.3