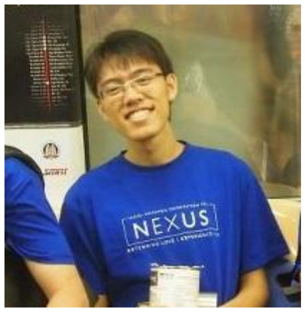
YourDay

 Ignatius Damai Team Leader Deadline Watcher	 Huang Da Lead Coder	 Wu Pei Lead Tester	 Nguyen Ngoc Nhu Thao Lead UI

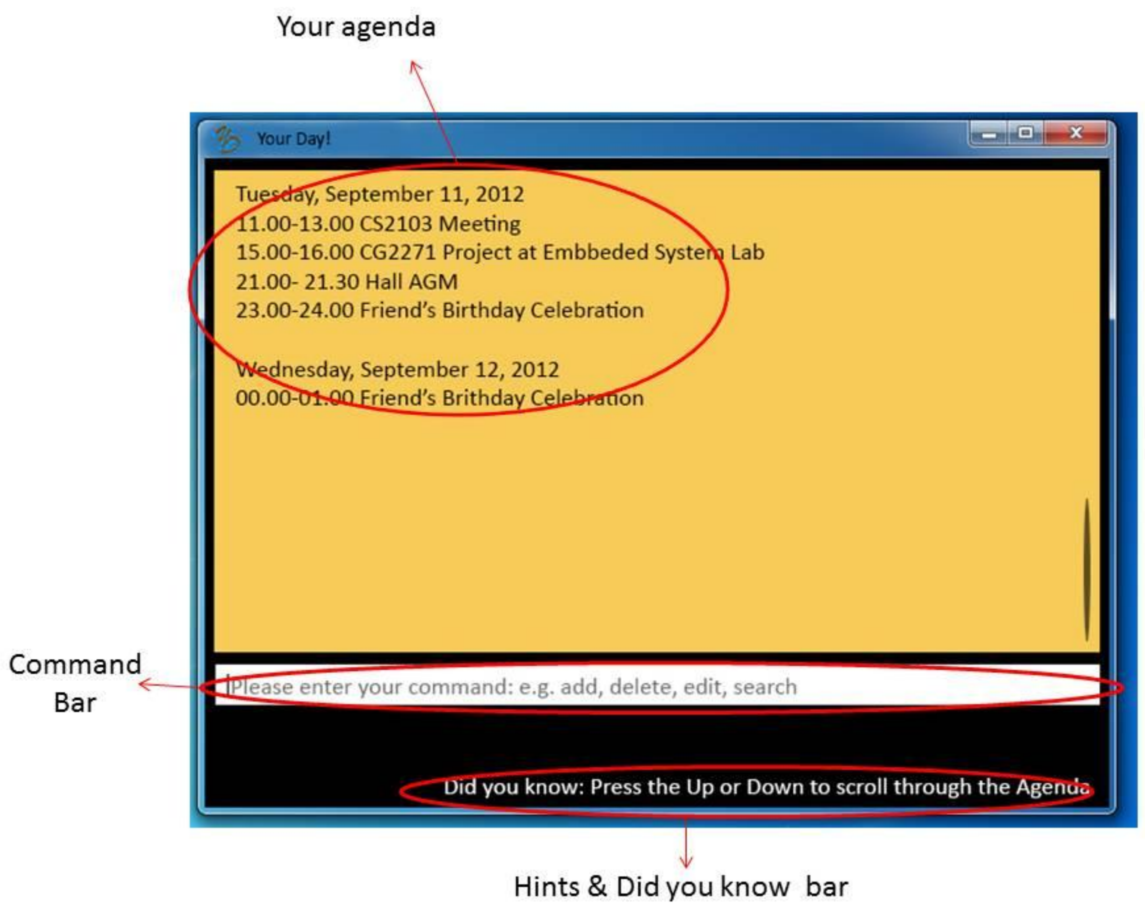
Quick Introduction

Tired of missing your appointments or having difficulties in managing your agenda? YourDay will help you to keep track of your appointments and tasks, as well as to keep them organized. YourDay is a powerful software which accepts semi-natural language commands via keyboard, provides easy-to-use functionalities and utilizes simple user interface.

User Guide

1. The Main Screen

On the home screen, you can give commands such as add, edit, delete, or search agenda entries. The following sections will guide you how to use each of these functions.



Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12, 2012
00.00-01.00 Friend's Brithday Celebration

Did you know: Press the Up or Down to scroll through the Agenda

2. Adding an Entry

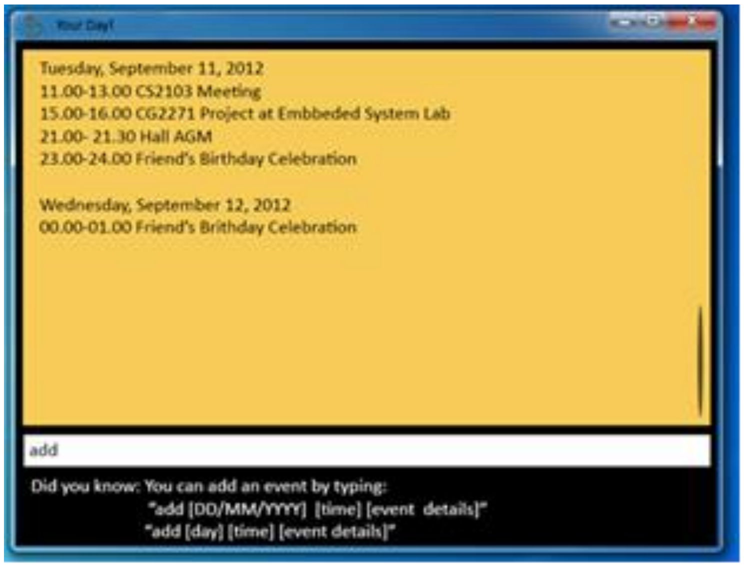
Our product lets you add new agenda entries easily. There are several flexible formats for task addition:

Command:	**[add command] [DD/MM/YYYY] [Time] [Event Details]**
Possible add commands:	**add, create, +**

You can add more tasks details such as priority or location easily. To add location details of an entry, add "**at**" followed by the location after your event details. To add priority, simply add "**priority**" after your event details. The accepted priorities are:

- 0 : Highest Priority
- 1 : High priority

- 2 : Normal priority. This is the default priority if the user does not give a specific priority
- 3 : Low Priority.



Your Day!

Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12, 2012
00.00-01.00 Friend's Brithday Celebration

add

Did you know: You can add an event by typing:
"add [DD/MM/YYYY] [time] [event details]"
"add [day] [time] [event details]"

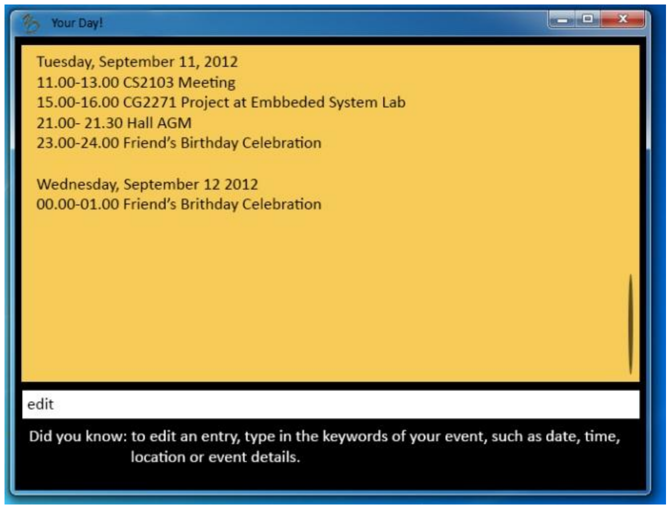
Date format: **DD/MM/YYY**

Time format: **HH:MM-HH:MM**

3. Editing an Entry

Editing an entry has never been easier. To update an entry simply type:

Command:	**[edit commands] [keyword(s)]**
Possible edit commands:	**edit, update, change, modify, !**



Your Day!

Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

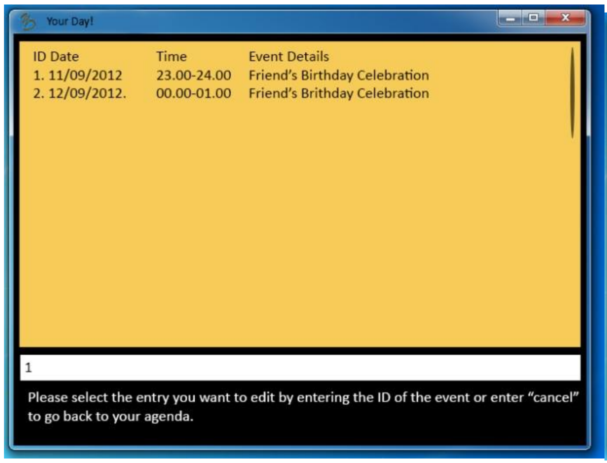
Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration

edit

Did you know: to edit an entry, type in the keywords of your event, such as date, time, location or event details.

The keywords can be a word or a part of the entry details, date, or time of the entry. After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to edit by entering the event ID shown ur search result.

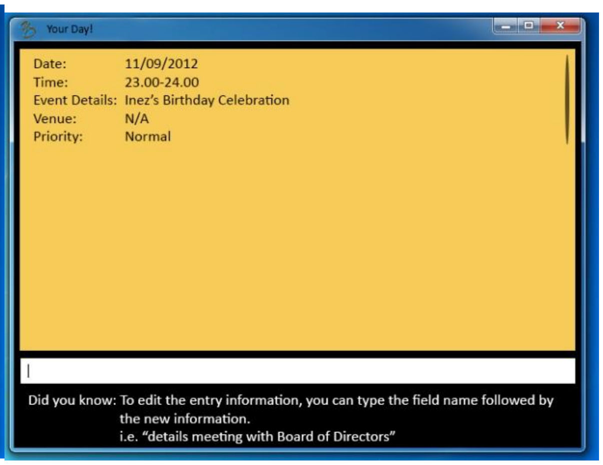
You will then be shown the entry's details. To edit the entry, type the field name you want to edit (details, time, date, venue, or pri) and type the new value for the respective field.


Your Day!

```
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration
```

1

Please select the entry you want to edit by entering the ID of the event or enter "cancel" to go back to your agenda.


Your Day!

```
Date:           11/09/2012
Time:           23.00-24.00
Event Details:  Inez's Birthday Celebration
Venue:          N/A
Priority:       Normal
```

Did you know: To edit the entry information, you can type the field name followed by the new information.
i.e. "details meeting with Board of Directors"

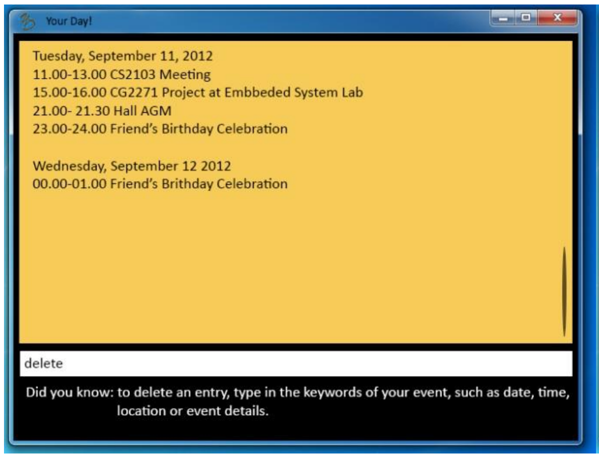
4. **Deleting an Entry**

In order to delete an existing entry, you need to enter a command in the following format:

Command:	**[delete command] [keyword(s)]**
Possible delete commands:	**delete, del, remove, cancel, clear, kill, -**

The delete function interface is similar to that of the edit function. The keywords can be a word or a part of the entry details, date, or time of the entry. After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to delete by entering the entry ID shown on your search result. You can delete multiple entries by entering:

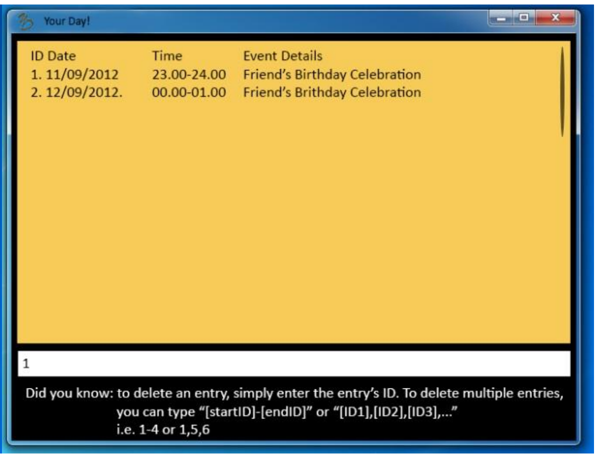
Command:	**[start ID]-[endID]**
	[ID1],[ID2],[ID3],…
	i.e. 1-6 or 1,2,4


Your Day!

```
Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration
```

delete

Did you know: to delete an entry, type in the keywords of your event, such as date, time, location or event details.


Your Day!

```
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration
```

1

Did you know: to delete an entry, simply enter the entry's ID. To delete multiple entries, you can type "[startID]-[endID]" or "[ID1],[ID2],[ID3],…"
i.e. 1-4 or 1,5,6

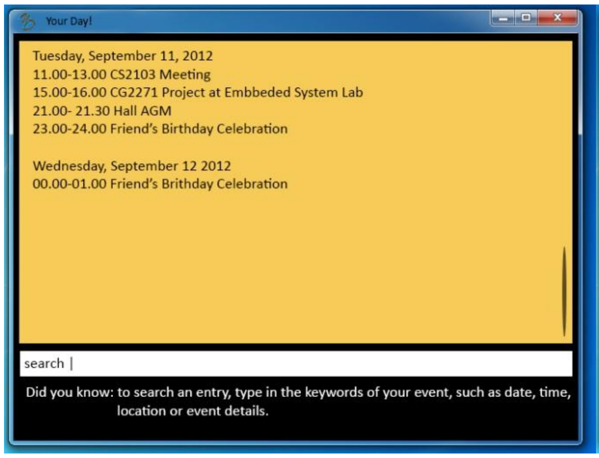
5. **Search Function (Power Search)**

In order to search a specific event, you need to enter a command in the following format:

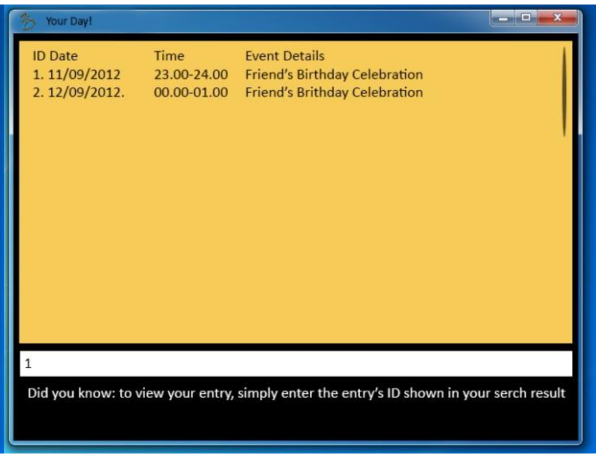
Command:	**[search command] [keyword(s)]**

Possible search commands: **search, find, lookup, ?**

After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to view by entering the entry ID shown on your search result. The keyword(s) can be anything from the event's details, location priority, time, and date. You would not even need to enter the exact word character by character. Our software will detect what you want to search.



```
Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration

search |
Did you know: to search an entry, type in the keywords of your event, such as date, time,
location or event details.
```



```
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration

1
Did you know: to view your entry, simply enter the entry's ID shown in your serch result
```

6. **Undo Function**

In order to undo an input, you need to enter a command in the following format:

Command: **undo**

Why YourDay?

YourDay offers so much more than your ordinary agenda. YourDay works very efficiently with our special feature, PowerSearch. Our PowerSearch features includes QuickSearch, Suggestions, MitsakeMeNot and

Glossary

PowerSearch: a powerful search function that provides suggestion and auto correction to user's search input.

QuickSearch : a powerful search feature that enables the program to output the entry found from the user's input without the need to push the return button or other special character to trigger the search engine

Suggestions : a powerful search feature that enables the user to input few characters and suggest a word that is included inside the entry database.

MitsakeMeNot : Error correction for user's input inside the search function

Developer Guide

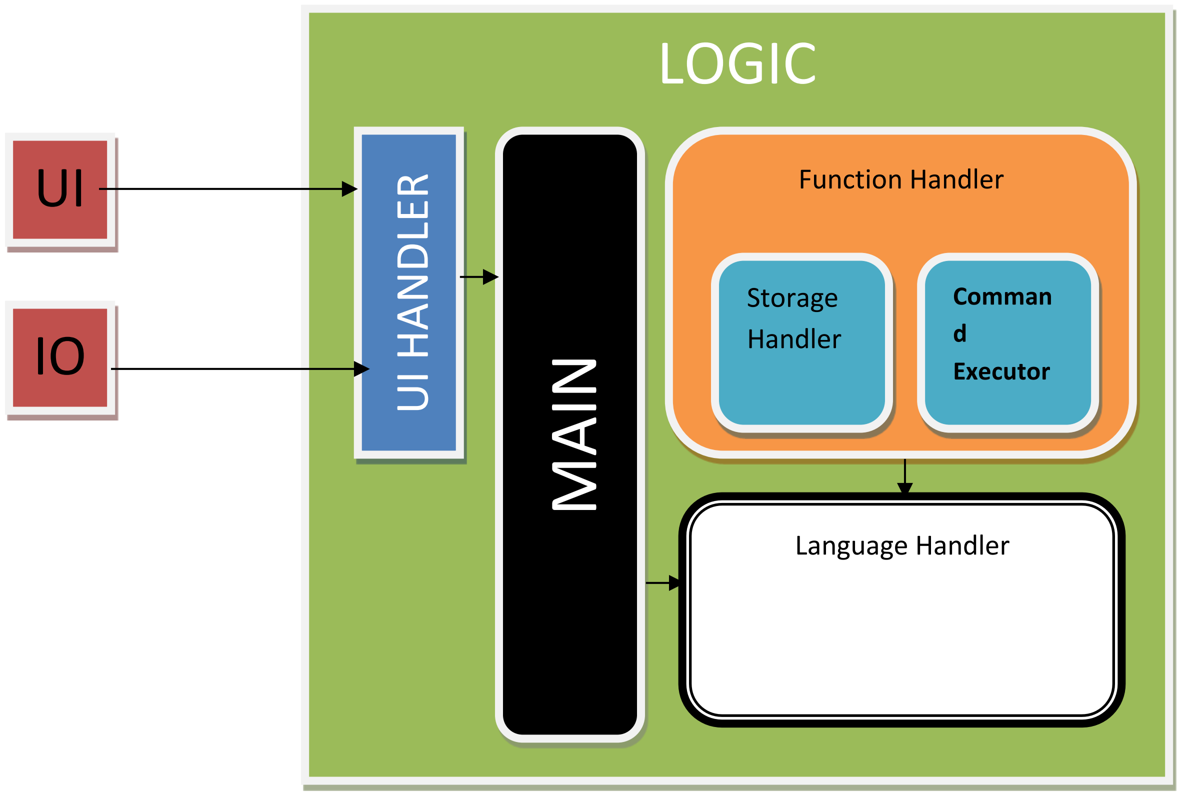
Purpose

The developer's guide is intended for future developers that intend to further enhance the program. The guide will include how the program is structured, how the different classes work together and the main API's that runs the backbone of the program. This guide serves to introduce the program classes and methods, not the actual implementation of the program. With the information from this guide, the reader will be able to understand how each of our functions works and the architecture of our program.

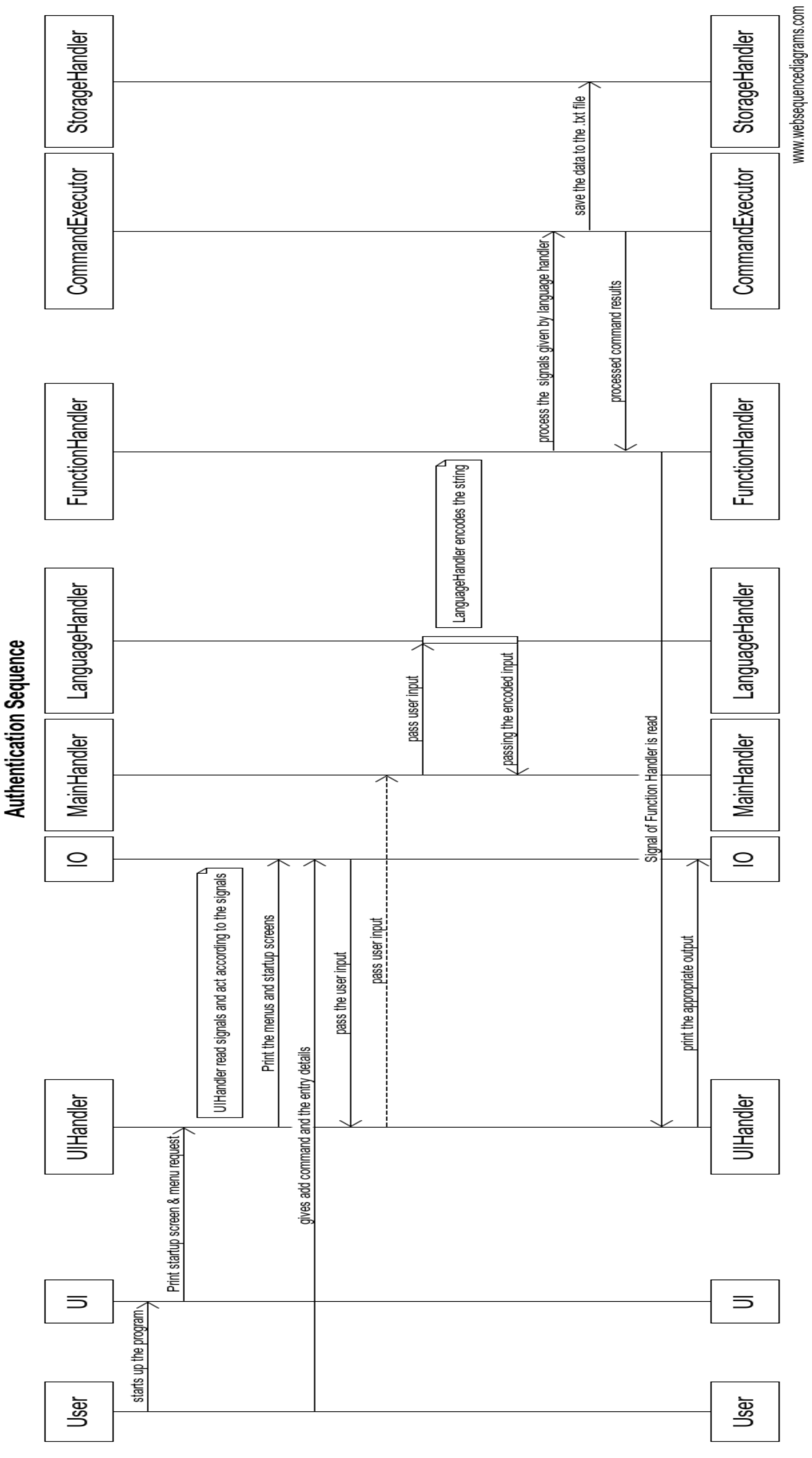
Intended Audience

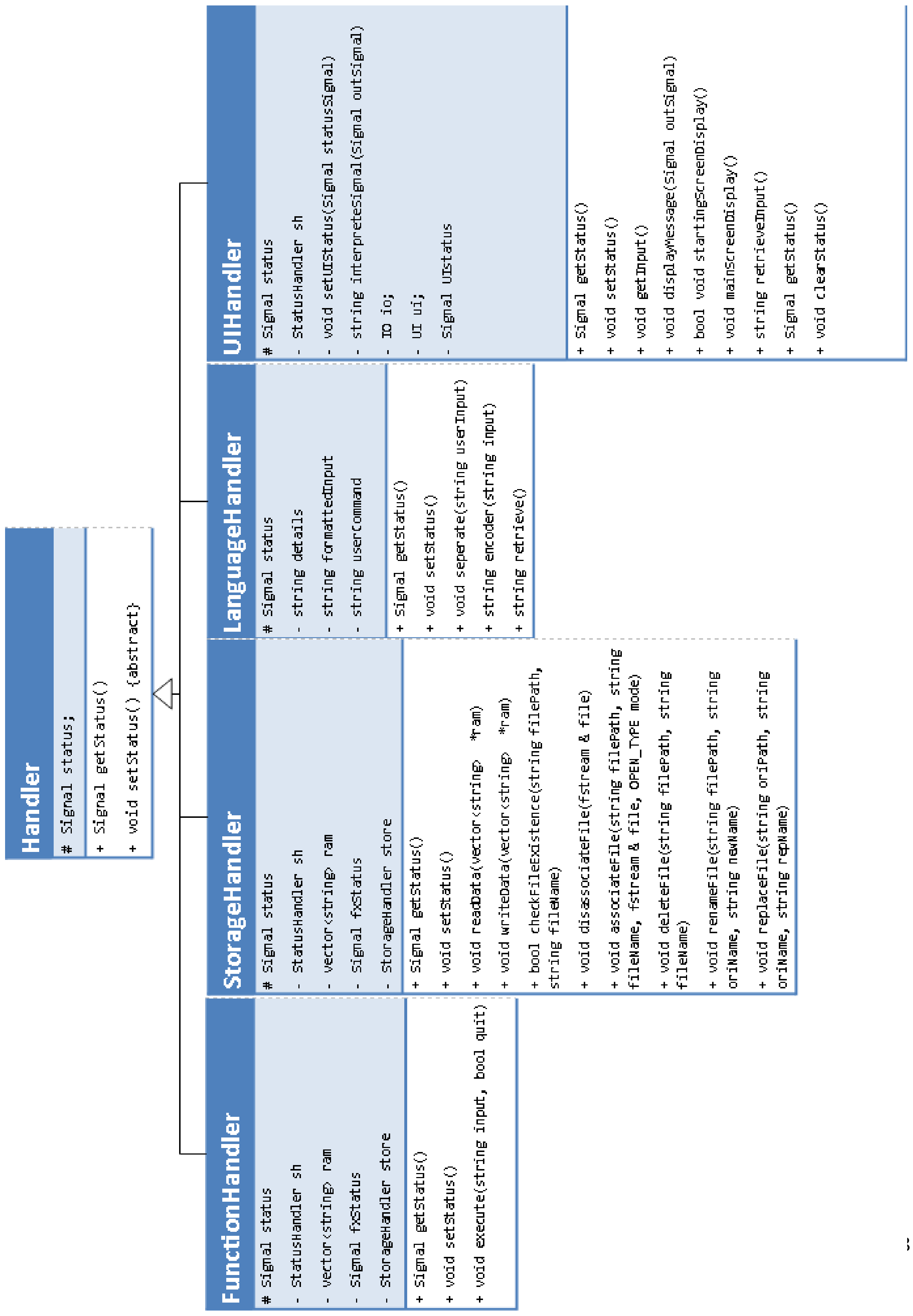
The information in this guide assumes that the reader has a decent level of C++ programming experience such as working with STL Vectors and Classes(Object Oriented Programming).

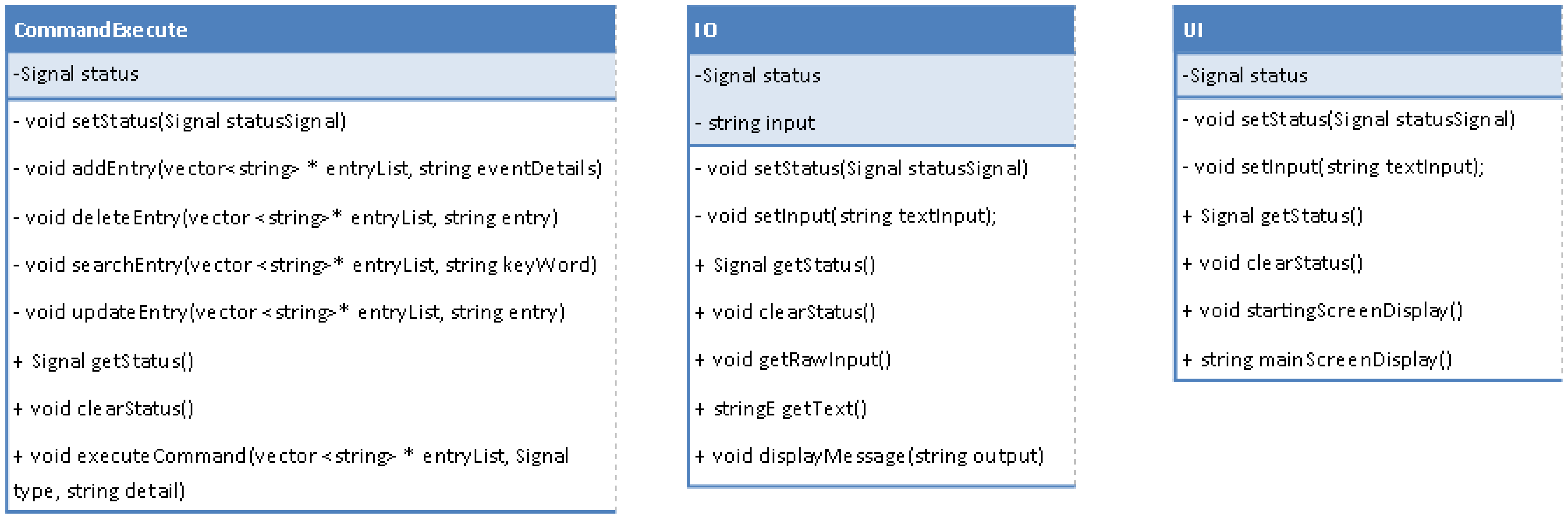
Transaction Architecture



The next 3 pages containing the UML Sequence Diagram, classes' attributes and properties, and inheritance will be presented in landscape.


Authentication Sequence
User
UI
UIHandler
IO
MainHandler
LanguageHandler
FunctionHandler
CommandExecutor
StorageHandler
starts up the program
Print startup screen & menu request
UIHandler read signals and act according to the signals
Print the menus and startup screens
gives add command and the entry details
pass the user input
pass user input
pass user input
LanguageHandler encodes the string
passing the encoded input
process the signals given by language handler
save the data to the .txt file
processed command results
Signal of Function Handler is read
print the appropriate output
www.websequencediagrams.com


Handler
# Signal status;
+ Signal getStatus()
+ void setStatus() {abstract}
FunctionHandler
# Signal status
- StatusHandler sh
- vector<string> ram
- Signal fxStatus
- StorageHandler store
+ Signal getStatus()
+ void setStatus()
+ void execute(string input, bool quit)
StorageHandler
# Signal status
- StatusHandler sh
- vector<string> ram
- Signal fxStatus
- StorageHandler store
+ Signal getStatus()
+ void setStatus()
+ void readData(vector<string> *ram)
+ void writeData(vector<string> *ram)
+ bool checkFileExistence(string filePath, string fileName)
+ void disassociateFile(fstream & file)
+ void associateFile(string filePath, string fileName, fstream & file, OPEN_TYPE mode)
+ void deleteFile(string filePath, string fileName)
+ void renameFile(string filePath, string oriName, string newName)
+ void replaceFile(string oriPath, string oriName, string repName)
LanguageHandler
# Signal status
- string details
- string formattedInput
- string userCommand
+ Signal getStatus()
+ void setStatus()
+ void seperate(string userInput)
+ string encoder(string input)
+ string retrieve()
UIHandler
# Signal status
- StatusHandler sh
- void setUIStatus(Signal statusSignal)
- string interpreteSignal(Signal outSignal)
- IO io;
- UI ui;
- Signal UIstatus
+ Signal getStatus()
+ void setStatus()
+ void getInput()
+ void displayMessage(Signal outSignal)
+ bool void startingScreenDisplay()
+ void mainScreenDisplay()
+ string retrieveInput()
+ Signal getStatus()
+ void clearStatus()



CommandExecute

-Signal status

- void setStatus(Signal statusSignal)
- void addEntry(vector<string> * entryList, string eventDetails)
- void deleteEntry(vector <string>* entryList, string entry)
- void searchEntry(vector <string>* entryList, string keyWord)
- void updateEntry(vector <string>* entryList, string entry)
+ Signal getStatus()
+ void clearStatus()
+ void executeCommand(vector <string> * entryList, Signal type, string detail)

IO

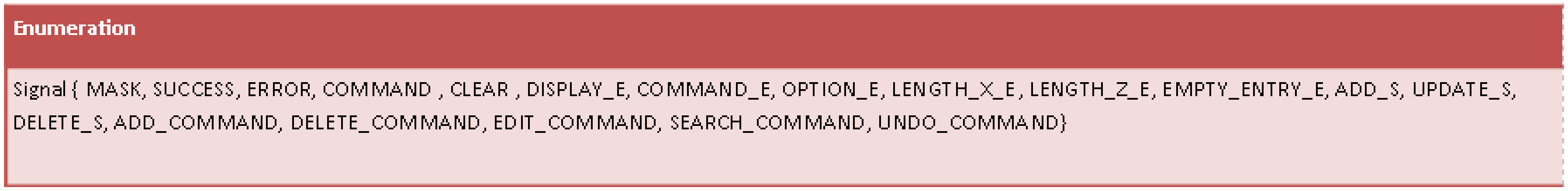
-Signal status
- string input

- void setStatus(Signal statusSignal)
- void setInput(string textInput);
+ Signal getStatus()
+ void clearStatus()
+ void getRawInput()
+ stringE getText()
+ void displayMessage(string output)

UI

-Signal status

- void setStatus(Signal statusSignal)
- void setInput(string textInput);
+ Signal getStatus()
+ void clearStatus()
+ void startingScreenDisplay()
+ string mainScreenDisplay()



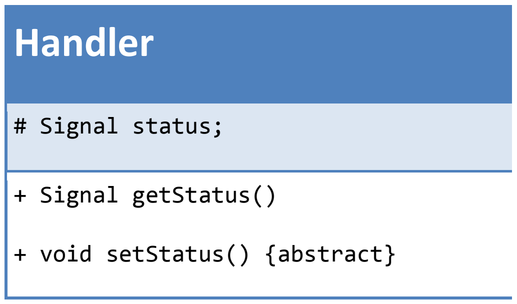
Enumeration

Signal { MASK, SUCCESS, ERROR, COMMAND , CLEAR , DISPLAY_E, COMMAND_E, OPTION_E, LENGTH_X_E, LENGTH_Z_E, EMPTY_ENTRY_E, ADD_S, UPDATE_S, DELETE_S, ADD_COMMAND, DELETE_COMMAND, EDIT_COMMAND, SEARCH_COMMAND, UNDO_COMMAND}

APIs

Handler Classes

1. HandlerClass



```
Handler
# Signal status;
+ Signal getStatus()
+ void setStatus() {abstract}
```

   a. Signal setStatus()

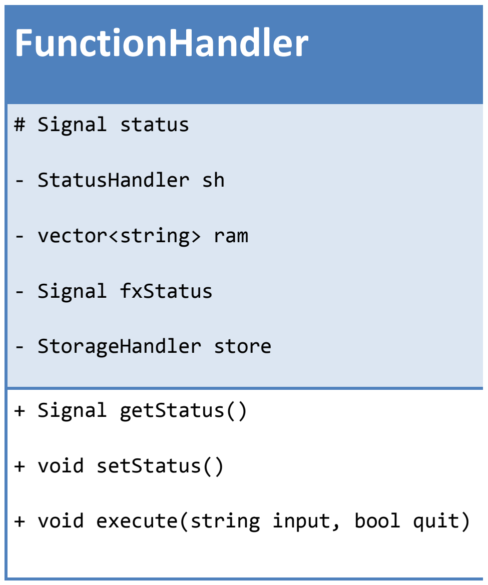
      - sets the status attribute of the class. Signal will be:

CLEAR	: Testing signal
SUCCESS	: Success indicator, all process are successfully handled;
ERROR	: Error indicator, some errors occured when executing.

   b. void getStatus()

      - Retrieves the current status of the class

2. Function Handler Class



```
FunctionHandler
# Signal status
- StatusHandler sh
- vector<string> ram
- Signal fxStatus
- StorageHandler store
+ Signal getStatus()
+ void setStatus()
+ void execute(string input, bool quit)
```

   a. void setStatus()

      - sets the status attribute of the Function Handler class

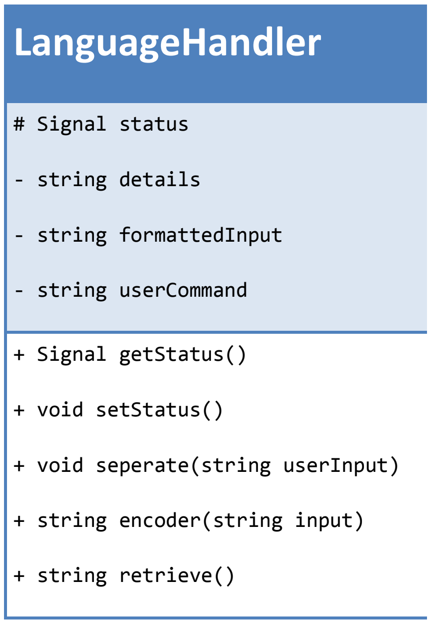
   b. Signal getStatus()

- Retrieves the current status of the class

c. void execute(string input, bool quit)

- The operation is periodically called in main(). It will handle the flow of the logic component.

3. Language Handler Class



LanguageHandler

```
# Signal status
- string details
- string formattedInput
- string userCommand
```

```
+ Signal getStatus()
+ void setStatus()
+ void seperate(string userInput)
+ string encoder(string input)
+ string retrieve()
```

a. void setStatus()

- Seperates user input's string into 2 parts, the input and the string to be processed

b. void separate(string userInput)

- Separates user input's string into 2 parts, the input and the string to be processed

c. string encoder(string input)

- Encodes the raw string into the correct saving format

d. string retrieve()

- Retrieves the processed string pointer after separation method

4. Storage Handler

StorageHandler

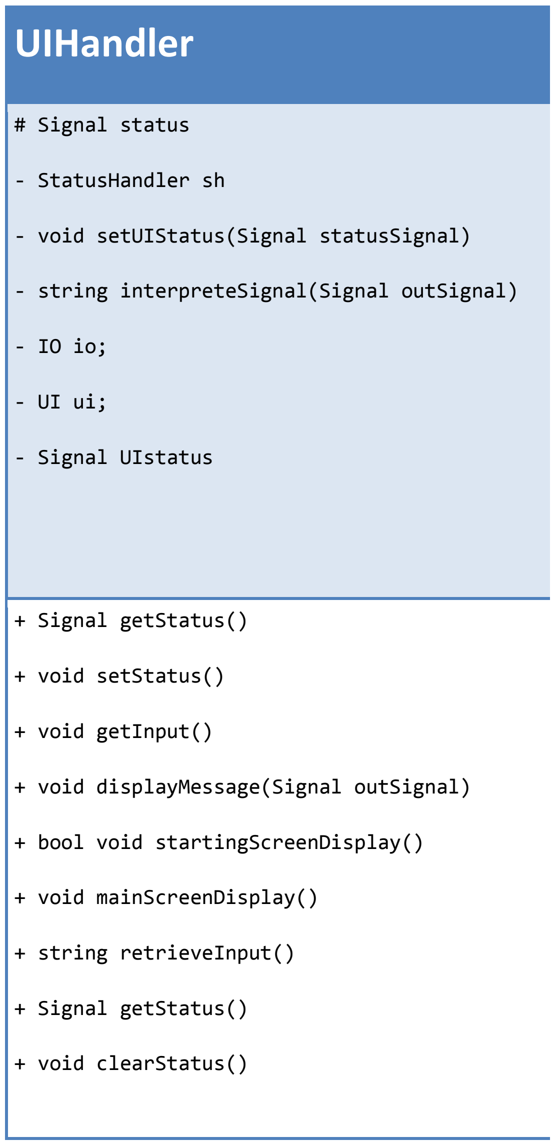
```
# Signal status

- StatusHandler sh

- vector<string> ram

- Signal fxStatus

- StorageHandler store
```

```
+ Signal getStatus()

+ void setStatus()

+ void readData(vector<string>  *ram)

+ void writeData(vector<string>  *ram)

+ bool checkFileExistence(string filePath,
string fileName)

+ void disassociateFile(fstream & file)

+ void associateFile(string filePath, string
fileName, fstream & file, OPEN_TYPE mode)

+ void deleteFile(string filePath, string
fileName)

+ void renameFile(string filePath, string
oriName, string newName)

+ void replaceFile(string oriPath, string
oriName, string repName)
```

a. void setStatus()
    - Sets the status of the language handler after input processing
b. void readData(vector<string> *ram)
    - Read the data stored in the file, and put it in a vector of string.
c. void writeData(vector<string> *ram)
    - Write the data stored to the file, the data is stored in a vector of string.
d. bool checkFileExistence(string filePath, string fileName)
    - Checks whether the file located in filePath and fileName do exist
e. void disassociateFile(fstream & file)
    - Removes the file association within the program
f. void associateFile(string filePath, string filename, fstream & file, OPEN_TYPE mode)
    - Associate files to the program
g. void deleteFile(string filePath, string fileName)
    - Deletes text file with the same path and name as filePath and fileName

h. void renameFile(string filePath, string oriName, string newName)
   - Renames the file
i. void replaceFile(string oriPath, string oriName, string repName)
   - Replace an old file with the newly updated files

5. UI Handler



UIHandler

```
# Signal status
- StatusHandler sh
- void setUIStatus(Signal statusSignal)
- string interpreteSignal(Signal outSignal)
- IO io;
- UI ui;
- Signal UIstatus
```

```
+ Signal getStatus()
+ void setStatus()
+ void getInput()
+ void displayMessage(Signal outSignal)
+ bool void startingScreenDisplay()
+ void mainScreenDisplay()
+ string retrieveInput()
+ Signal getStatus()
+ void clearStatus()
```

a. void setStatus()
   - Sets the status of the UI handler after printing UI
b. void getInput()
   - This operation is used to get input message from the user. It calls IO to get user input through command line and store it in the private string named input
c. void displayMessage(Signal outSignal)

- This operation is used to display output message to the user. It takes in the feedback signal passed by main(), call private interpretSignal(Signal) to intepret to a string and calls IO to output to the user

d. void startingScreenDisplay()
- This operation is used to display the startup screen to the user at the beginning of the program

e. mainScreenDisplay()
- This operation is used to display the main screen through UI class which interfaces the interaction with user. It will call the UI to display the main screen.

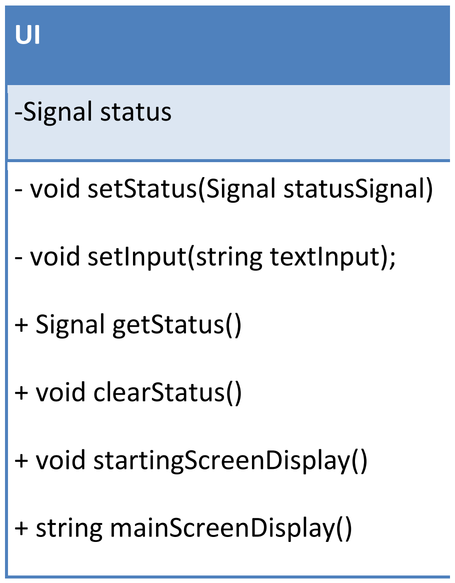
f. string retrieveInput()
- This method retrieves user's input after processed in the IO class

g. Signal getStatus()
- This method returns the status of UI handler

h. void clearStatus();
- This method clears the UI handler's status

Input/Output Classes

1. UI Class



UI

-Signal status

- void setStatus(Signal statusSignal)

- void setInput(string textInput);

+ Signal getStatus()

+ void clearStatus()

+ void startingScreenDisplay()

+ string mainScreenDisplay()

a. void setStatus(Signal statusSignal)
- set the status signal of UI displaying process

b. Signal getStatus()
- get the status signal of UI displaying process

c. void clearStatus()
- clear the status signal of UI process to default CLEAR signal

d. void startingScreenDisplay()
   - show the starting screen to user at the beginning of the program

e. void mainScreenDisplay()
   - show the main screen to interact with the user

2. IO class

IO
-Signal status
- string input
- void setStatus(Signal statusSignal)
- void setInput(string textInput);
+ Signal getStatus()
+ void clearStatus()
+ void getRawInput()
+ stringE getText()
+ void displayMessage(string output)

a. void getRawInput()
   - get user input string through command line

b. void setInput(string textInput)
   - Sets the input string attribute in IO class

c. void setStatus(Signal statusSignal);
   - Sets the status of IO class according to IO process

d. void displayMessage(string output)
   - display feedback message string of the system to user

e. Signal getStatus()
   - get the status signal of IO displaying process

f. void clearStatus()
   - clear the status signal of IO process to default CLEAR signal

g. string retrieveInput();
   - Retrieves the value of input string

Command Processor Classes

1. CommandExecute

CommandExecute
-Signal status
- void setStatus(Signal statusSignal) - void addEntry(vector<string> * entryList, string eventDetails) - void deleteEntry(vector <string>* entryList, string entry) - void searchEntry(vector <string>* entryList, string keyWord) - void updateEntry(vector <string>* entryList, string entry) + Signal getStatus() + void clearStatus() + void executeCommand(vector <string> * entryList, Signal type, string detail)

a. void setStatus(Signal statusSignal)
   - set the status signal of Command Executor displaying process
b. Signal getStatus()
   - get the status signal of Command Executor displaying process
c. void clearStatus()
   - clear the status signal of Command Executor to default CLEAR signal
d. void addEntry(vector<string> * entryList, string eventDetails)
   - Adds an entry to the database
e. void deleteEntry(vector <string>* entryList, string entry)
   - deletes an entry from the database
f. void searchEntry(vector <string>* entryList, string keyWord)
   - search an entry from database by utilizing the keyword
g. void updateEntry(vector <string>* entryList, string entry)
   - updates an entry in the database
h. void executeCommand(vector <string> * entryList, Signal type, string detail)
   - Executes the command given to the program

FUTURE ONGOING DEVELOPMENTS

- Solving issues in coupling of functions
- Implements the UI layer thoroughly

- Implements all of the functionalities