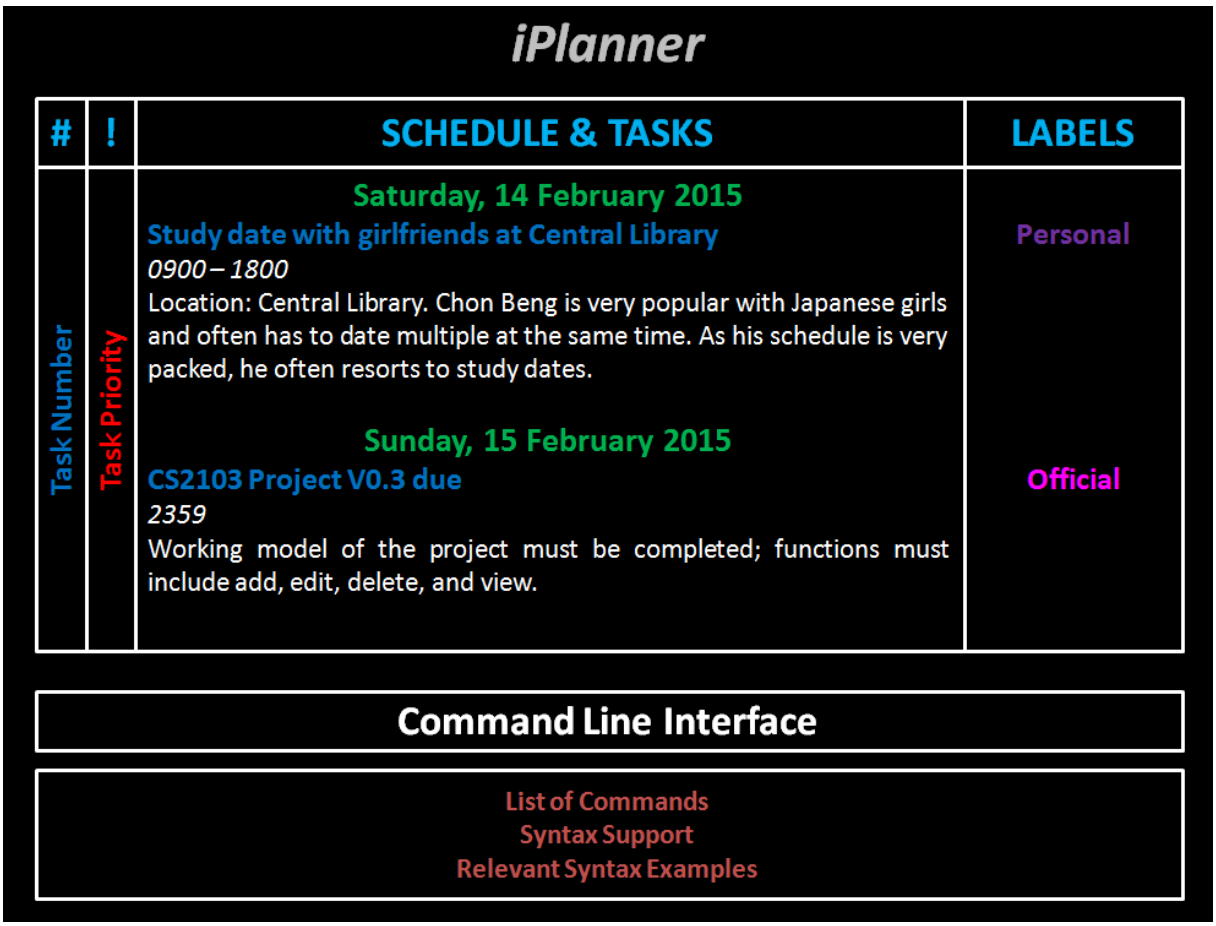
iPlanner



Supervisor: Huang Da

Extra Feature: Recurring Task

Andy Soh Wei Zhi

Documentation
Code Quality

Lee Joon Fai

Integration
Documentation

Yu Youngbin

Team Lead
Scheduling*
Testing

Ng Chon Beng

Testing
Integration

Shri Kishen Rajendran

Code Quality
*Scheduling**

* Role includes scheduling, tracking, and ensuring adherence of deliverables to deadlines.

User Guide

Welcome to iPlanner – planning and scheduling at your fingertips!

iPlanner is a light and user-friendly task managing software, targeted at the busy individual who requires an efficient way of managing tasks on a desktop or a laptop PC, even when offline. This quick-start guide will bring you through the features provided by iPlanner, everything at your fingertips!

How Commands Work

iPlanner supports the entering of multiple relevant commands in the same command line. For example, entering:

::add Boss's birthday party ::start 15/03 1300 ::end 15/03 1430 ::desc 24 Kent Ridge Park, buy present ::! H

adds a new event, titled "Boss's birthday party" from 15 March, 1pm to 15 March, 2:30pm, with additional information specified by the user – including location (Kent Ridge Park) and a task to do (buy a present). Additionally, the event is marked as high priority.

The command line has to contain only one main command (i.e. ::add / ::del / ::edit / ::search /::sort / ::display / ::exit / ::undo). The order of the commands does not have to be exactly as stated above whereby the user may choose to input ::start and ::end before ::add. Comments which are not the main command (i.e. ::start / ::end / ::desc / ::label / ::priority) may be omitted from the command line.

In the case where the user accidentally types in more than one whitespace between commands and texts, the command line interface will still be able to recognise the input. For example, entering:

::add Boss's birthday party ::desc 24 Kent Ridge Park, buy present

Add New Item

Create a new item by entering *::add [name of item]* into the command line interface.

Edit Item

Edit an existing item by entering *::edit <item number>* into the command line interface.

Delete Item

Delete an existing task by entering *::delete <item number>* into the command line interface.

Assign Timings

In iPlanner, items are categorized into Events (with specific start and end dates/times), Deadlines (with specific due dates/times) and Tasks (without specific dates/times attached). When adding a new item or editing an existing one,

1. Specify/modify start time by entering *::start [dd/mm/yy hhmm]*;
2. Specify/modify end time by entering *::end [dd/mm/yy hhmm]*; and
3. Specify/modify deadline by entering *::due [dd/mm/yy hhmm]*.

Remove the assigned start time, end time, and deadline by entering *::start*, *::end*, or *::due* respectively into the command line interface.

Set Recurring Events

Set an event to recur on each day of the week by entering *::recur [day]* into the command line interface, where *[day]* will take on *mon*, *tue*, *wed*, *thu*, *fri*, *sat*, or *sun*.

Set an event to recur every certain number of days by entering *::recur [number of days]* into the command line interface.

End the series of recurring events by entering *::recur.end [dd/mm/yy]* to specify the end date or *::recur.end [number of recursions]* to specify the total number of recurring events.

Adding Details/Descriptions

When adding a new item or editing an existing one, add additional details or descriptions of the items by entering *::desc [description]* into the command line interface. This may include the location of the event, *e.g. Singapore Flyer*, and other relevant details, *e.g. bring a rose, wear a bowtie*. Note that *[description]* will take on the user's input until the end of the command line or the characters *::* is found.

Remove the details entered by entering *::desc* into the command line interface.

Assign Priority

When adding a new item or editing an existing one, assign priority to an item by entering *::! [priority]* into the command line interface. Note that priority can only take on the values of low, medium, or high, and *[priority]* in the command line interface should only take on *L*, *M*, or *H*.

Remove the assigned priority by entering *::!* followed by a *<space>* into the command line interface.

Labels

Items can be labelled according to their nature when adding a new item or editing an existing one, as follows:

1. Label an item as personal, *e.g. family picnic, watching a football game, doing the laundry,* by entering *::# P* into the command line interface. Note that labelling an item as personal would automatically remove the 'official' label, if any.
2. Label an item as official, *e.g. product pitch to a customer, product design meeting with colleagues,* by entering *::# O* into the command line interface. Note that labelling an item as official would automatically remove the 'personal' label, if any.
3. Label an item as a milestone, *e.g. promotion, anniversary, birthday*, by entering *::# M* into the command line interface.

Removed the labels by entering *::#* into the command line interface.

Mark Done/Undone

User can mark an item as completed by entering *::done <item number>*. User can mark an item as incomplete by entering *::undone <item number>*.

Undo/Redo Previous Action

Undo the previous action by entering *::undo* into the command line interface. Should you decide to redo the undone item, enter *::redo*.

Search

Search for an existing item by entering *::search [keywords]* into the command line interface. Note that *[keywords]* will take on the user's input until the end of the command line or the characters *::* is found., and only results that match the entire string will be returned.

Sort Items

The default arrangement of events/deadlines is chronologically, with the most recent ones first, i.e. an event on 4 Jan 2015 will appear before an event on 14 Feb 2015. Tasks (without timestamps) will appear at the end. Revert to this arrangement by entering *::sort* or *::sort date* into the command line interface.

Sort items into the sequence that you have keyed them in by entering *::sort key* into the command line interface.

Sort items in order of priority by entering *::sort !* into the command line interface. High-priority items will appear first, followed by medium-priority ones, followed by low-priority ones. Items without priority indicators will appear last.

View Items

View only items on a certain date by entering *::view date [dd/mm/yyyy]* into the command line interface. Events starting, ending, or spanning that time period and deadlines on that day will be returned.

View only high-priority, medium-priority, or low-priority items by entering *::view !H*, *::view !M*, or *::view !L* respectively into the command line interface.

View only items labelled personal, official, or as a milestone by entering *::view P*, *::view O*, or *::view M* respectively into the command line interface.

View only items that have been done or that have yet to be done by entering *::view done* or *::view undone* respectively into the command line interface.

Storage

When the program is executed for the first time, the user will be prompted for a directory to store the schedule, following which the program will create the required text file and access it in the subsequent runs.

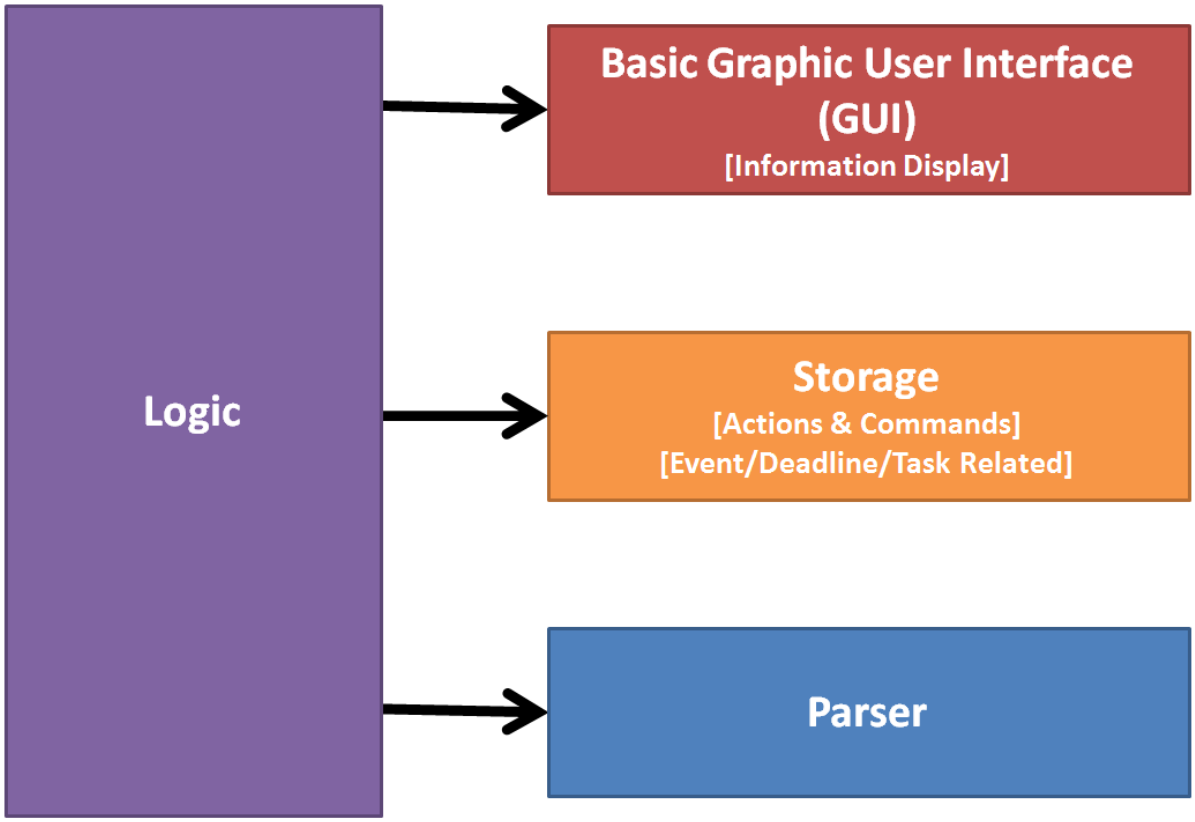
Help Menu

View the list of commands supported, their specifications, and syntax by entering *::help* into the command line interface.

Developer Guide

At this point in time, we have implemented the most basic functions (Create, Read, Update, Delete), and have set up the preliminary skeletons and code for other more advanced ones (Sort, Search, Filter, etc.) although the latter are not fully implementable at this period in time. Additionally, plans have been made to modify some of the current implementation and APIs in order to conform more closely to the Object Oriented paradigm. In the following section of the guide, an overview of the project is provided.

Architecture

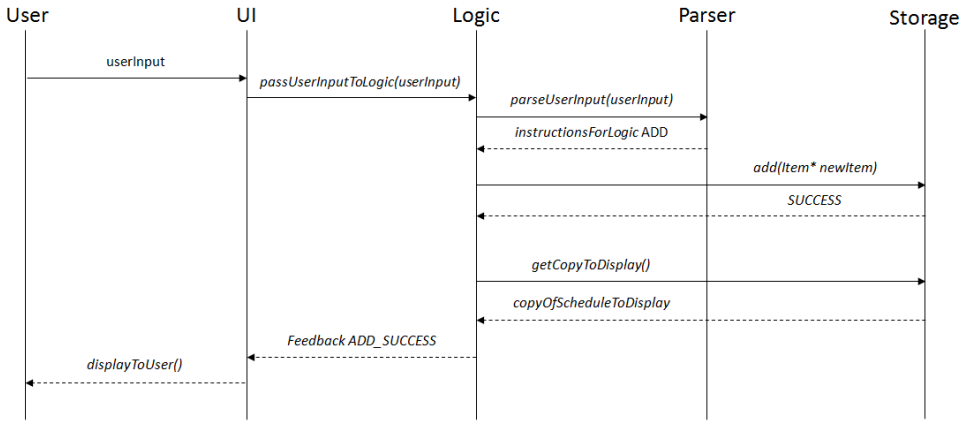


Sequence Diagrams

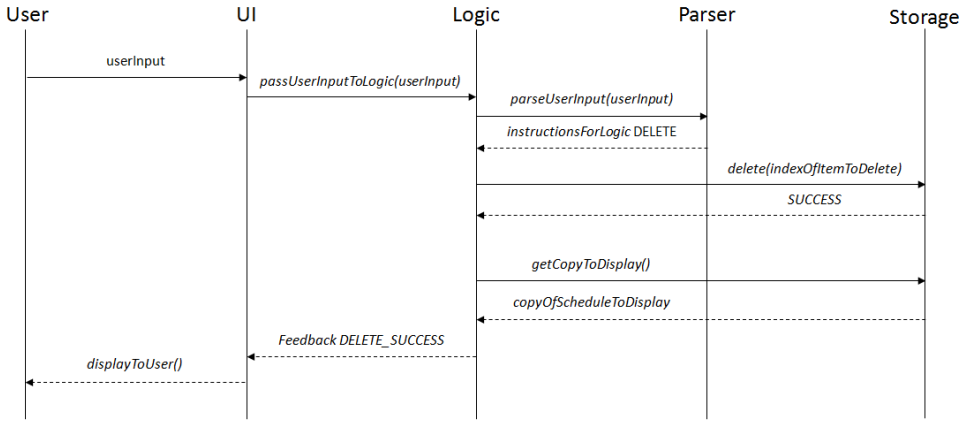
1. Assuming that no exceptions occur and all cases are valid, after the user enters input in the User Interface, the input string is passed to the Logic component, which goes on to pass it to the Parser component.
2. Parser decomposes and interprets the input string, returning the parsed command back to Logic in the form of an Instruction object.
3. According to the instructions received from Parser, Logic will either
    a. Add: Create a new Item and pass its address to Storage,
    b. Delete: Instruct Storage to delete and existing Item,
    c. Edit: Retrieve a copy of an existing Item, make necessary changes to it, and pass the address of an edited Item to Storage to replace the old Item.
4. Once Storage has made its necessary changes, it returns a Success message back to Logic.
5. Upon receiving Success confirmation from Storage, Logic calls Storage again to retrieve a copy of vector of Items that will be displayed to the user.
6. Logic passes this vector to UI, along with the Success message, in a Feedback object.

7. UI displays a message to inform the user about the completion of the action and displays the contents of the vector.

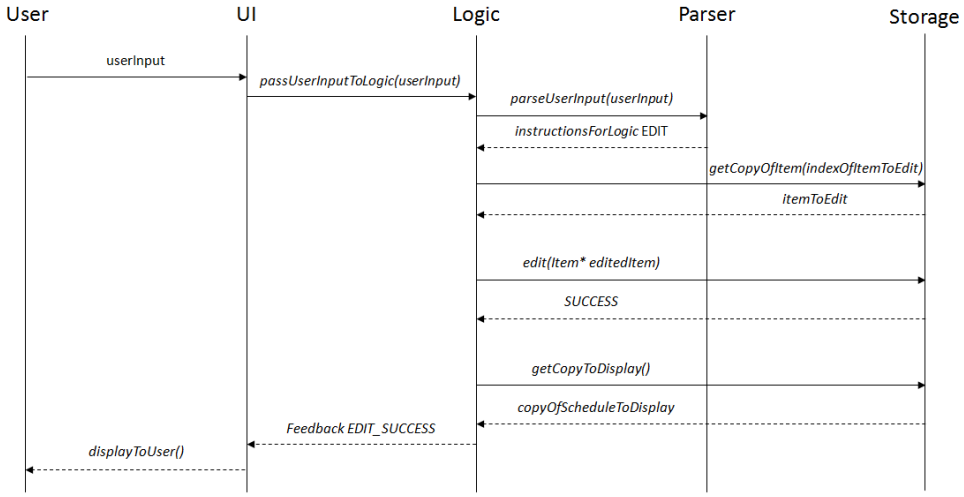
Addition of Valid Item



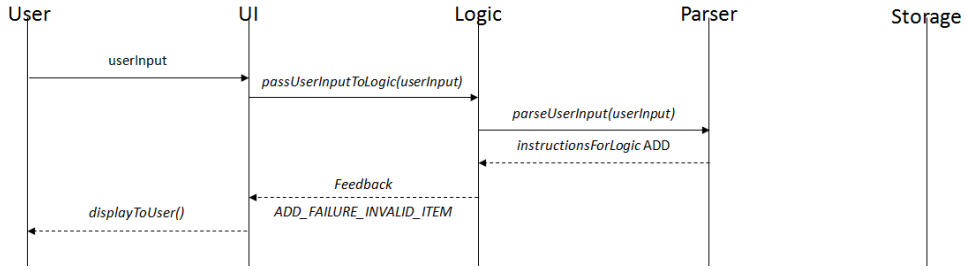
Deletion of Valid Item



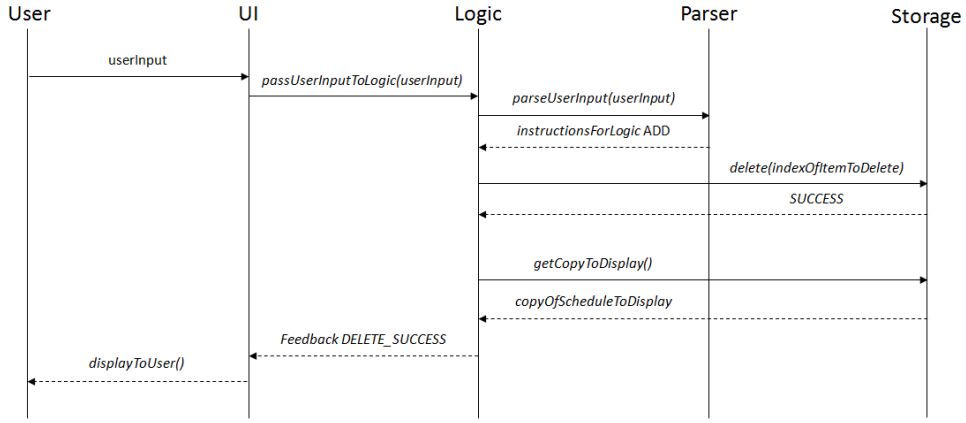
Editing of Valid Item



Addition of Invalid Item



Deletion of Invalid Item



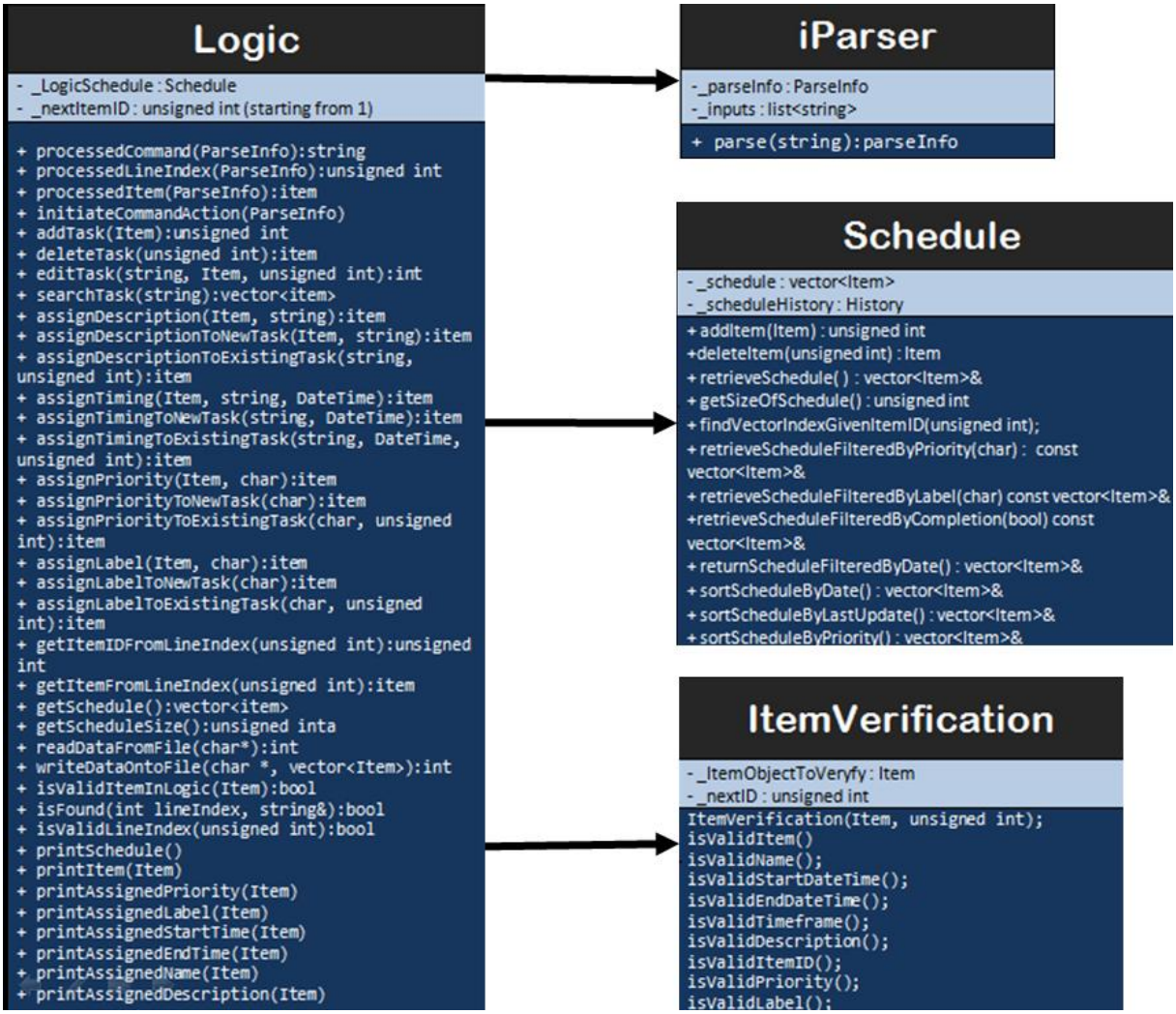
User Interface

User interface is currently Command Prompt, and a basic graphic user interface will be implemented in subsequent versions.

Logic

Logic calls for a parser to get command. It executes basic functions like add, delete and edit depending on *parseInfo* given by the parser. Upon receiving *parseInfo*, logic calls *ItemVerification* to check whether the parsed item is valid to execute certain function. And finally, if the item is verified, logic is given access to schedule and modify it appropriately.

For now, logic is granted to access schedule and item and to make changes directly. However, from V0.2 onwards, logic will only be able to pass the address or pointer to *Schedule* and only *Schedule* will make the necessary adjustments.

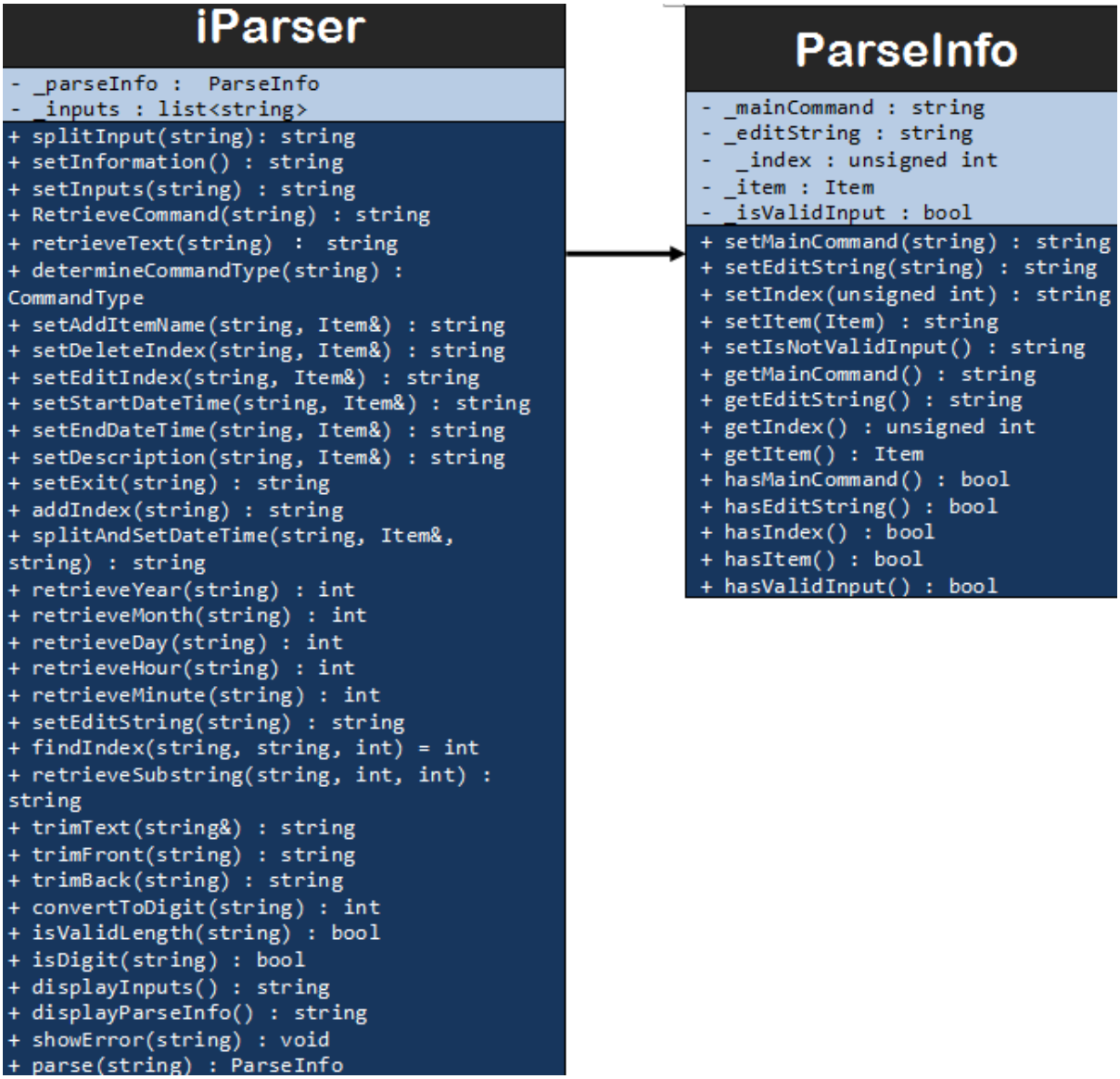


Parser

Parser receives user input passed as a string data type and parser's job is to split the user input into the respective commands and texts. Each user input is identified by the token "::". It is to be noted that the strings split must contain a command but is not necessary accompanied with a text. With each split strings, the parser does a check in the command type and adds the data into *ParseInfo* class attribute.

As the *Parser* class is only required to accept a string data type and return the split information to *Logic*, all functions will be set as private functions with only one function, parse, accessible by *Logic*.

For future development, it will be better for *Parser* to return a list of commands and texts to *Logic*. The list will contain string data types which have been interpreted from the user input. This list of string will allow *Logic* to identify what function to execute.

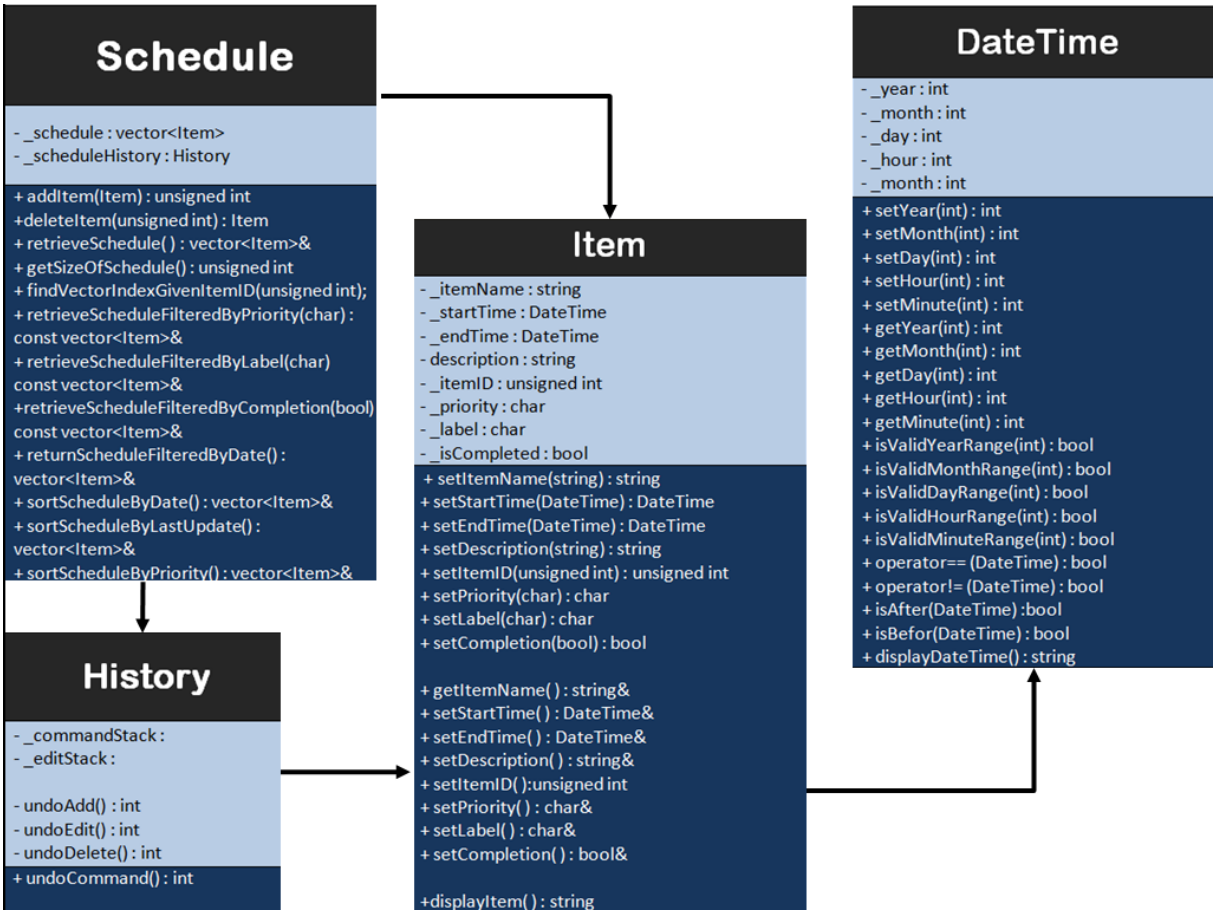


Storage

Storage is currently implemented in the most basic and straightforward method – declaring and calling objects of a certain class. Each of the currently implemented sub-components – *Schedule*, *Item*, and *dateTime*, has the basic setters (used to assign values during *add* and *edit* functions) and getters (used extensively in *display* and *edit* functions). *Schedule* is slightly more functional with certain types of sorting implemented.

However, moving forward, it is important to implement storage as via pointers to objects, and in order to implement OO fully, logic will only pass values/pointers/objects to *Schedule*, and will not directly access the *Item* or *dateTime* objects via the getters (pass by reference). Local copies will be passed via the getter methods.

For the Storage component, test-driven development has been employed in most parts (exception to Schedule). Current tests are fairly comprehensive, although more efficient methods should be written in future to better compare between class objects (overload of Boolean == and !=, for example). This would allow better and less time-consuming usage of *Assert::AreEqual (... , ...)*.


Schedule
- _schedule : vector<Item>
- _scheduleHistory : History
+ addItem(Item) : unsigned int
+deleteItem(unsigned int) : Item
+ retrieveSchedule( ) : vector<Item>&
+ getSizeOfSchedule() : unsigned int
+ findVectorIndexGivenItemID(unsigned int);
+ retrieveScheduleFilteredByPriority(char) :
const vector<Item>&
+ retrieveScheduleFilteredByLabel(char)
const vector<Item>&
+retrieveScheduleFilteredByCompletion(bool)
const vector<Item>&
+ returnScheduleFilteredByDate() :
vector<Item>&
+ sortScheduleByDate() : vector<Item>&
+ sortScheduleByLastUpdate() :
vector<Item>&
+ sortScheduleByPriority() : vector<Item>&
History
- _commandStack :
- _editStack :
- undoAdd() : int
- undoEdit() : int
- undoDelete() : int
+ undoCommand() : int
Item
- _itemName : string
- _startTime : DateTime
- _endTime : DateTime
- description : string
- _itemID : unsigned int
- _priority : char
- _label : char
- _isCompleted : bool
+ setItemName(string) : string
+ setStartTime(DateTime) : DateTime
+ setEndTime(DateTime) : DateTime
+ setDescription(string) : string
+ setItemID(unsigned int) : unsigned int
+ setPriority(char) : char
+ setLabel(char) : char
+ setCompletion(bool) : bool
+ getItemName( ) : string&
+ setStartTime( ) : DateTime&
+ setEndTime( ) : DateTime&
+ setDescription( ) : string&
+ setItemID( ):unsigned int
+ setPriority( ) : char&
+ setLabel( ) : char&
+ setCompletion( ) : bool&
+displayItem( ) : string
DateTime
- _year : int
- _month : int
- _day : int
- _hour : int
- _month : int
+ setYear(int) : int
+ setMonth(int) : int
+ setDay(int) : int
+ setHour(int) : int
+ setMinute(int) : int
+ getYear(int) : int
+ getMonth(int) : int
+ getDay(int) : int
+ getHour(int) : int
+ getMinute(int) : int
+ isValidYearRange(int) : bool
+ isValidMonthRange(int) : bool
+ isValidDayRange(int) : bool
+ isValidHourRange(int) : bool
+ isValidMinuteRange(int) : bool
+ operator== (DateTime) : bool
+ operator!= (DateTime) : bool
+ isAfter(DateTime) :bool
+ isBefor(DateTime) : bool
+ displayDateTime() : string

Appendix A: User Stories. As a user, …

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addTask	add a task by name of task only	can record tasks that I want to do a certain day or some day
editTask	edit a task by changing details of an already created task	make changes to the details of the task should there be any
removeTask	remove a task that had been added earlier	delete a task that is no longer required to be done
rescheduleTask	reschedule a task by changing the deadline or time of an already created task	make changes should there be any
undoLastAction	undo a single action which was previously done (e.g. add/ remove/ reschedule etc)	do not have to delete and remake the whole schedule one more time
specifyDeadline specifyTimedTask	specify a deadline or time for a task should there be any	can keep track of the important deadline or time
clearTasks	clear some or all tasks	do not have to clear them one by one
addToArchive	archive tasks which are done or marked as complete	can view them in the future should there be a need to
markAsComplete	mark or unmark a task as complete	know what is done / not done and can prioritise my tasks
chooseSaveFolder	specify where he wants the data .txt file to be saved	can access it on file-sharing software like Dropbox
viewTasks	view upcoming tasks	can have a clearer understanding of my tasks at hand
searchTask	search for task(s) using keywords	can filter out the task(s) that I am searching for
labelItem	can label tasks into categories	can have an understand which category the tasks belong to
addSubTask	can add a daughter task to a parent task	know what tasks are to be done which belongs to the same task
addDescription	can add description to a task	do not forget what the task is about should the task name be ambiguous
assignPriority	assign priority to a task	know what are the more urgent tasks at hand
recurringTasks [chosen extra feature]	set recurring tasks	do not have to add them over and over again

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
executeHotKey	can make use of hot keys (e.g. ctrl + z)	execute any actions faster
notifyClash	should be informed of clashes in timings of scheduled tasks	can plan what to do with the clashes
changeView	can filter my tasks in views which i want to see (e.g. category, date, priority etc)	can have a clearer understanding of my tasks at hand

setReminder	set reminder to my task	will be informed of the task's at the time of reminder which was set
viewFreeTime	can view my available timeslots on certain day(s)	can make plans on free timings should there be a need
viewHelp	can view help document	have an idea how to use the application

Appendix B: Non-Functional Requirements

Desktop

iPlanner will work on a desktop or laptop without network or Internet connection. It is not a mobile application or cloud-based application.

Windows

iPlanner will work on Windows 7 or Windows Vista OS, and on both 32 bit and 64 bit personal computers.

Standalone

iPlanner works as a standalone software; it is not a plug-in to another software. However, extensions might be available in later versions to maintain compatibility and exporting of data with other programs or other existing software.

Non-Installation

iPlanner works without requiring an installer.

Command Line Interface

Command Line Interface is the primary mode of input, and a basic graphic user interface is used to display the output. There is no need to use the mouse for any point-and-click functions.

Data Storage

Data storage is done via text files created when the program is run. Text files will be human-editable, and users are able to manipulate the task list by editing the data file.

Object-Oriented Programming

Most of the iPlanner will follow the object-oriented programming paradigm.

Appendix C: Product Survey

Product: **Remember The Milk**
Documented by: **Lee Joon Fai**

Strengths:

- Easy syncing between smartphone app and desktop interface
- Adding of tasks and assigning details done very conveniently within one command line
- Sort and filter tasks according to labels and priority
- Maintains record of completed tasks
- Option for app to send automated email reminders
- Very flexible and customisable search system
- Geo-tagging of tasks is available
- User can add notes to tasks along the way
- Allows flexible postponement of tasks
- Easy undo-ing of accidental actions
- Can share to-do list with other users of the app

Weaknesses:

- No calendar view, hence difficult for user to visualize longer-term tasks
- No option for adding floating tasks i.e. all tasks must have a deadline
- Smartphone app interface is very different from desktop interface. The switch between devices is not intuitive enough
- The software has remained pretty stagnant over the years, updates are not frequent

Product: **wunderlist**
Documented by: **Yu Young Bin**

Strengths:

- It is easy to post on facebook immediately since it is connected to facebook account.
- It has simple and easy user interface.
- User can customize the background picture.
- User gets notification on important events.
- User can categorise the events and to-do lists.
- User can search for a certain event.
- User can even post pictures related to the event.
- User can sort the events in accordance with the alphabetical order or deadlines etc.
- It is easy to create, delete, and move any events.
- User can print out the to-do list directly from the application
- User can e-mail the to-do list directly from the application

Weaknesses:

- It does not seem quite necessary to create an account and login to use this application.
- There is no user guide for the new users to get familiarized with the product.
- There is no calendar view. It is difficult to visualize the sequence of the upcoming events.
- When a user accidentally presses the tick-box on the to-do list, the event just disappears.

Product: **Microsoft Outlook Calendar**
Documented by: **Ng Chon Beng**

Strengths:

- Can add item into calendar view
- Able to reschedule and delete item
- Able to view calendar offline

- Easy calendar view to see schedule
- Has a reminder function
- Has recurring function
- Able to categorise items with different colours
- Able to change view (e.g. daily, weekly, monthly)
- Has a search function
- Has weather forecast (provided there is internet connection)
- Able to share calendar with others

Weaknesses:

- Unable to see timing at default view
- Unable to mark as done nor archive action which are done
- Unable to quick launch software using shortcut key
- Unable to add item with no specific due date
- Takes multiple actions to add an item

Product: **Google Calendar**
Documented by: **Shri Kishen Rajendran**

Strengths:

- It has a structured and organised look.
- It can be viewed in the format of a day, week or month.
- It allows differentiating tasks by the usage of different colours for different events.
- It has a search function which can also search for files on the user's Google Drive.
- User is a given a reminder of a task half an hour before a scheduled event.
- User is allowed to sync other calendar(.ics) files.
- User can have a task list where additional sub tasks can also be added under tasks.
- User can specify a deadline for a task.

Weaknesses:

- It does not work without internet connection and a Google account.
- It can be difficult to read if there are too many appointments.
- The interface is more useful for planning events than tasks.
- No feature to indicate if an event is complete or not.

Product: **ColorNote**
Documented by: **Andy Soh Wei Zhi**

Strengths:

- Two types of notes are available – free-style text notes and lists
- Notes are simple and intuitive; they only have three properties – title, content, colour
- Sorting of the notes can be by time modified and colour
- Notes can be viewed as a list (with titles only), list (with details), grid, or a large grid, giving users flexibility to decide what works best for them.
- Notes can be backed-up on cloud storage.
- List-styled notes can be dragged easily to re-order items within the notes
- Sticky notes that can appear on home screen for user convenience
- Notes can be edited directly by double-tapping them, again simple and intuitive.
- Accidental deletion of notes is prevented (delete and permanently delete options).
- Notes can be assigned to days via a calendar.
- Time reminders can be set for each note.
- Notes can be locked with password to ensure privacy.

Weaknesses:

- Only platform supported is Android (no iOS or PC).
- Notes are not file compatible with other programmes, i.e. cannot export notes to a Microsoft Excel compatible file or Google Calendar.
- Pictures, photographs, and drawings are not supported.
- High customizability comes with a trade-off – lack of automaton, e.g. manually marking certain items/lines as completed.