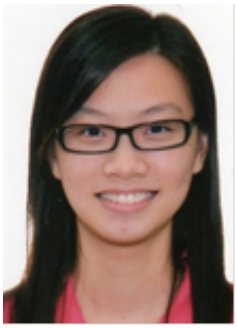
[t17-2c][V0.1]

TimeMaster

```
                              15 March 2015
                                  Sunday

1. 0800 - 1000 Meeting with CEO
2. 1000 - 1400 Revise price list
3. 2200 - 2230 Check email

                              16 March 2015
                                  Monday

4. By today:   Submit report
5. 1000 - 1200 Write evaluation
6. 1200 - 0100 Lunch
7. 1200 - 0200 Department meeting

                              17 March 2015
                                 Tuesday

8. By 1100:    Submit finalized price list
9. 0800 - 1000 Organize excel sheet for item description
10.1000 - 1100 Interview with potential employee
11.1700 - 1900 Dinner with family
```

Take control of your life

Supervisor: Fazli Sapuan Extra feature: FlexiCommands

			
Ng Wei Ern Team lead, Documentation	Gordy Adiprasetyo Documentation, Deliverables and Deadlines, Scheduling and Tracking	Ng Xian Loon Documentation, Code Quality, Testing	Teo Kia Meng Documentation, Visual Studio expert, Integration

TimeMaster's Basic Commands

Here are TimeMaster's basic commands to manage your tasks. For a complete explanation of the commands, please refer to the complete user guide below.

#	Command	Description
1	**add** <task>	Adds tasks into TimeMaster. The user may specify deadlines, dates and times. **Example:** add print report *(adds task "print report")*
2	**delete** <task number>	Deletes task which task number is specified by the user **Example:** delete 1 *(deletes task 1)*
3	**show** <date/day of the week/task type>	Shows the tasks of the specified type that are scheduled to be done on the specified day or date **Example:** show 160315 *(shows the tasks scheduled on 16 March 2015)*
4	**edit** <task number> <new task>	Edits task of the specified task number **Example:** edit 1 buy lunch *(changes task 1 to "buy lunch")*
5	**search** <keyword>	Searches for tasks containing the keyword **Example:** search lunch
6	**undo**	Undo your last action
7	**complete** <task number>	Marks the task of the specified task number as complete
8	**incomplete** <tasks number>	Marks the task of the specified task number as incomplete
9	**exit**	Exits TimeMaster

Additional Feature

FlexiCommands

User is able to enter their commands in the form of natural language. With the exception of the command keyword which should always be the first word of the command, the task description, dates and times can be in any desired order.

The Complete User Guide

Command format explanation

The commands will be explained in the following order:

Format

- Describes how a user should execute the command
- The following syntax notation is used to denote:
    - Essential information: words encapsulated in <>
    - Optional information: words in {}
    - Command keywords: **command**
- Command keywords are required to be the first word of the user's entry

Description

- Tells you the result and the behavior of the command

Examples

- Examples of the command

Equivalent

- Words that can be used in place of the standard keywords

Valid days, dates and times

If the date is not specified by the user, the date of the task will be today's date if the time has not yet passed, otherwise it will be tomorrow. If a time is not specified, the end time for a task with deadline will be 23:59 and the start time for a timed task will be 00:00. Examples of valid dates and times are as follows:

Days	`monday`
	`mon`
	`today`
	`tomorrow`
	`next mon`
	`next Monday`

Dates	`16 mar`
	`16 march`
	`16 mar 2015`
	`160315`
	`16032015`

Times	`1600`
	`2pm`
	`2am`

	230am 1230am

1. Adding a task

Format

add <task description> {{<time markers> <time/time period>} {<date markers> <date/date period>}/{day}}

Description

Purpose and behaviour

This command is used to add tasks into task list in chronological order. The new task will compared against each task in the list before being added into its rightful position where its start time and start date is later than that of the task above it and earlier than that of the task below it. (what about in the case where they have the same start time and end time? And in the case where they have the same start time but different end time?) A **task number**, which is the position of the task in the list, will be assigned to the task as a reference to be used in other commands. The **task description** is required. Failing to enter a task description would result in an error message.

Time markers

A time marker is required to be entered right before the time or time period should the user desire to add a task with deadline or tasks that will take place at or during a specific time. Time markers signal to TimeMaster that the time or time period would be trailing behind them so that TimeMaster will be able to extract the information and assign it to the task. Time markers consist of the words **at**, **before** and **from**.

Time marker **at** must be placed before the **start time** of the task. The **end time** of the task will be empty. The **end date** will be the same as the **start date**. If the **start date** is not specified, it will be assumed to be today's date if the time has not yet passed or else, it will be tomorrow's date.

Time marker **from** must be placed before a **time period**. The **start time** and the **end time** of the time period has to be separated by a **-** character or the word **to**. If the **start date** and

end date is not specified, it will be assumed to be today's date if the time period has not yet passed or else, it will be tomorrow's date.

Time marker **before** must be placed before the time by which the task has to be completed. It is used to indicate that the task to be added has a **deadline**. The specified time will be assigned to the **end time** of the task while the **start time** and **start date** will be left empty. If the **end date** is not specified, it will be assumed to be today's date if the deadline has not yet passed or else, it will be tomorrow's date.

Date markers

A date marker is required to be entered right before the date or the date period should the user desire to add a task with deadline or tasks that will take place on a day or over a period of days. Date markers signal to TimeMaster that the date or date period would be trailing behind them so that TimeMaster will be able to extract the information and assign it to the task. Date markers consists of the words **on**, **before** and **from.**

Date marker **on** must be placed before the **start date** of the task. The **end date** and **end time** of the task will be left empty. Unless specified by the user, the **start time** is assumed to be 00:00.

Date marker **from** is used to indicate the start of a **date period**. The **start date** and the **end date** of the date period has to be separated by a **-** character or the word **to**. Unless specified by the user, the **start time** and **end time** will be 00:00 and 23:59 respectively.

Date marker **before** must be placed before the date by which the task has to be completed. It is used to indicate that the task to be added has a **deadline**. The specified date will be assigned to the **end date** of the task while the **start time** and **start date** will be left empty. If the **end time** is not specified, it will be assumed to be 23:59 of the day before the specified date.

Days

Instead of keying in a date for the task. The user may enter the day of the week in place of the actual date should the start date or end date of the task fall within the current week or the week after.

Examples

To add tasks with deadlines

```
add submit report before 2359 on 13022015 or
add submit report before 2359 on Friday
```

To add tasks with specific time(s)

```
add presentation from 2pm – 4pm on 13022015
add presentation at 2pm on 13022015
```

To add tasks that have neither time nor deadline

```
add take a break
```

Equivalent

```
before = by
- = to
add = a
```

2. Deleting a task

Format

```
delete <task number>
```

Description

Purpose and behaviour

This command is used to remove tasks that were previously added. Each task will be given a running number (explained in the following section). This number can be seen at the default TimeMaster UI screen, or using the **search** function. The user inputs the command followed by this number and it will be removed from the program's memory.

Task number

Every task that is added to TimeMaster will be given a **task number**. This number is running, meaning the number will be updated as new tasks are added or other changes are made. Users need to input the current number attached to the desired task after the command "delete" in order to remove the task.

Examples

```
delete 1
delete 3
```

Equivalent

```
delete = d
```

3. Showing tasks listed on a specific date or day or by task type

Format

```
show <date/day of the week/task type>
```

Description

Purpose and behaviour

This command is used to show all tasks that are scheduled on a specific **date** or **day** of the week. TimeMaster will display the tasks of the entered day or date in chronological order. Refer to the table of valid days, dates, and times for the accepted formats.

This command can also be used to show all tasks of a specific type. (tasks with deadlines or undated tasks). The user can input the word **undated** or **deadline** after the command keyword **show**, and a list of the desired task type will be displayed.

Lastly, the user can also use this function to show available timeslot on a specific day. The user can do this by the typing the word **free** followed by the date or day after the command keyword show. A list of free timeslots, from 0800 hours to 0000 hours will be displayed.

Examples

```
show friday
```

```
show 16 mar

show 16032015
```

Equivalent

```
show = sh
```

4. Editing tasks

Format

```
edit <task number> <task component> <new task description or timing or
completion status or confirmation status>
```

Description

Purpose and behaviour

This command is used to edit previously added tasks. Like the **delete** function, the user inputs task number after the word **edit**, followed by the new task description. This function will then edit the details of the specified task.

Adding/removing information

New task description and/or timing can be added or removed from a specific task. To change description, simply enter the new task description desired. To change timing, enter keywords as stated in the examples of the **add** function, followed by the change desired. To remove timing, enter the command ***"-time"***.

Task component

Possible inputs

Task description: *name, desc*

Time: *time, timing*

Completion status: *complete, completion, com*

Confirmation status: *confirm, confirmation, con*

Completion status/confirmation status

User enters “0” or “no” to indicate that the task is not completed or confirmed.

User enters “1” or “yes” to indicate that the task is completed or confirmed.

Examples

```
edit 1 name buy lunch
edit 2 timing by sunday
edit 3 time from 16032015 to 20 mar 2015
edit 4 -time
edit 5 complete yes
edit 6 confirm 0
```

Equivalent

```
edit = e
```

5. Searching tasks

Format

```
search <keyword(s)>
```

Description

Purpose and behaviour

This command is used to search tasks with specific keyword(s) in their task description. TimeMaster will then return the desired tasks, sorted chronologically. Note that if a user wishes to search for tasks scheduled on a specific date, he needs to use ***show*** function instead of search.

Examples

```
search buy
search lecture
```

Equivalent

```
search = se
```

6. Undoing an action

Format

undo

Description

Purpose and behaviour

This command is used to undo user's last action should the user wishes to do so. This function can undo user's action up to first successfully executed command after TimeMaster begins to run.

Examples

```
Undo
```

Equivalent

```
undo = u
```

7. Marking a task as done or not done

Format

complete <task number>

incomplete <task number>

Description

Purpose and behaviour

This command is used to indicate whether a task has been completed or not. The tasks that are completed will be shown in yellow text, while those not done in white.

Examples

```
incomplete 1
complete 2
```

Equivalent

```
complete = c
incomplete = i
```

8. Exiting TimeMaster

Format

exit

Description

Purpose and behaviour

This command is used to close TimeMaster. All changes made to user’s schedule will then be saved.

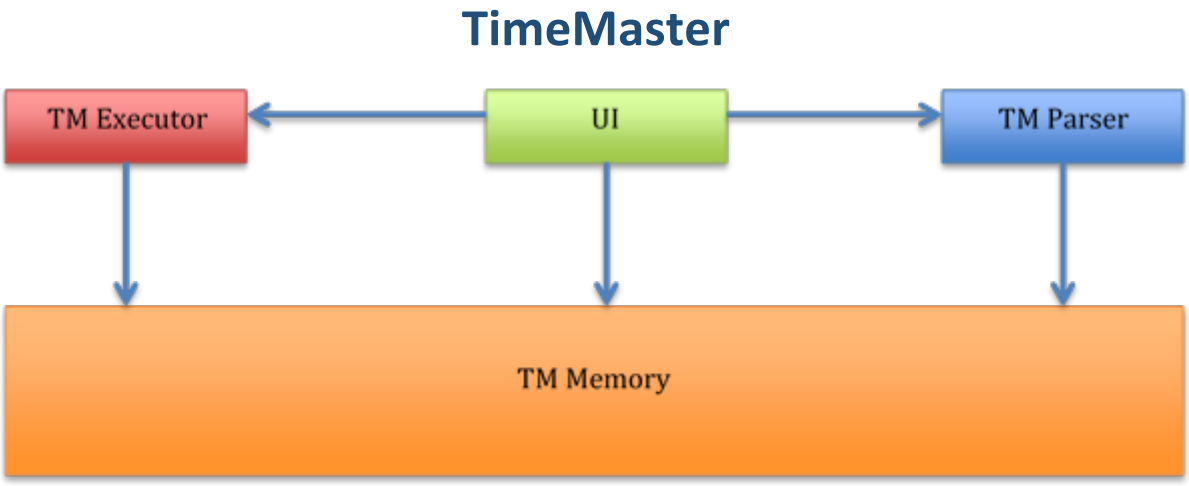
Examples

```
Exit
```

Equivalent

The Developer Guide

Architecture



The architecture of TimeMaster consists of the following components:

UI

The component through which the user interacts with the system. It displays information to the user and decides which command to be executed based on the user input.

Parser

Parser sieves out the command and the information associated to the command.

Executor

Carries out the intended actions of the user by calling the API of TM Memory.

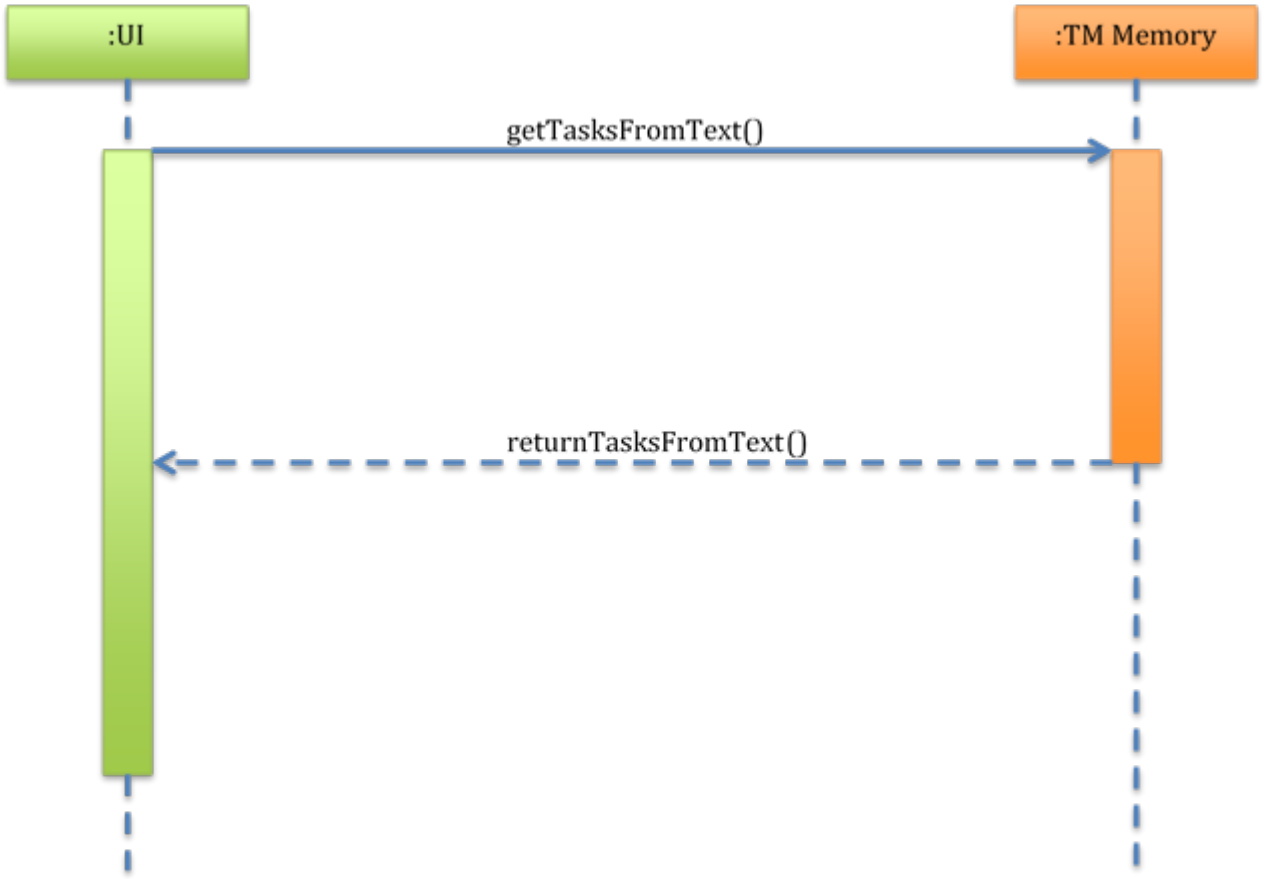
Memory

TM Memory contains the functions related to storing or removing user input into containers, displaying tasks, and saving and loading the tasks into text file.

High-level sequence diagrams

Communication between each component follows a façade design pattern. From a high-level point-of-view, the interactions between the main components can be summed up in two sequence diagrams:

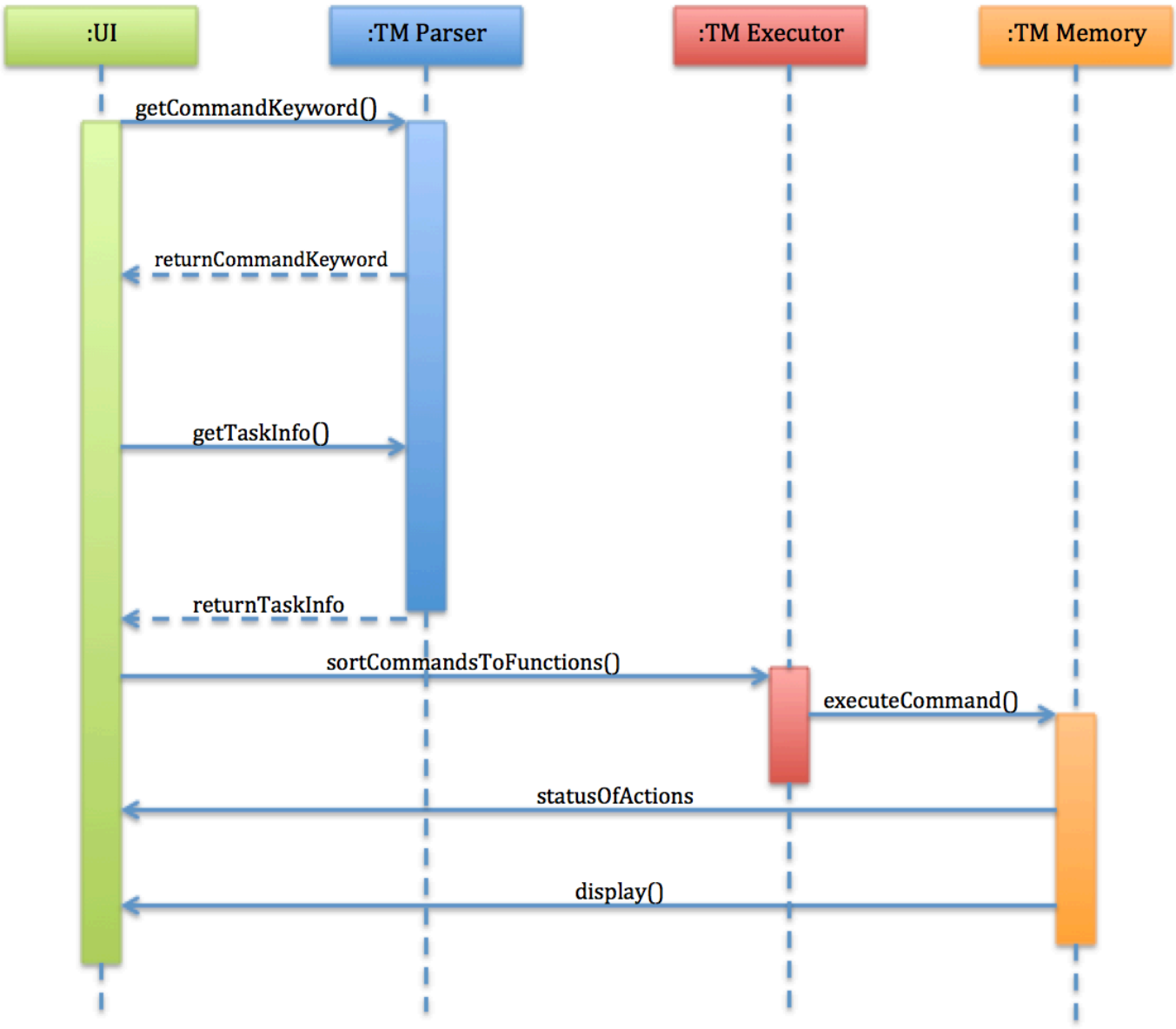
Phase I: Upon Initialization



The sequence diagrams above describe the order in which each component interacts with each other upon running TimeMaster. The order of interaction is as follows:

(1) After the user runs the program, UI will call the function getTasksFromText() in TM Memory to load the information stored in the text file.
(2) TM Memory will then return tasks stored in the text file to UI. UI will the store the information from the text into TMTaskList object.

Phase II: Normal Runtime



The sequence diagrams above describe the order in which each component interacts with each other in normal runtime. The order of interaction is as follows:

(1) UI prompts the user for an entry. And after getting the entry, UI will call TM Parser to extract and return the command keyword.
(2) Upon receiving the command, UI will decide which information to be extracted based on the command.
(3) UI will then pass the relevant information to TM Executor.
(4) TM Executor will call on the relevant functions in TM Memory to execute the command.
(5) After completion of the action, TM Memory will display the result of the action, whether it is successful or not. After which, it will display the updated task list in UI.

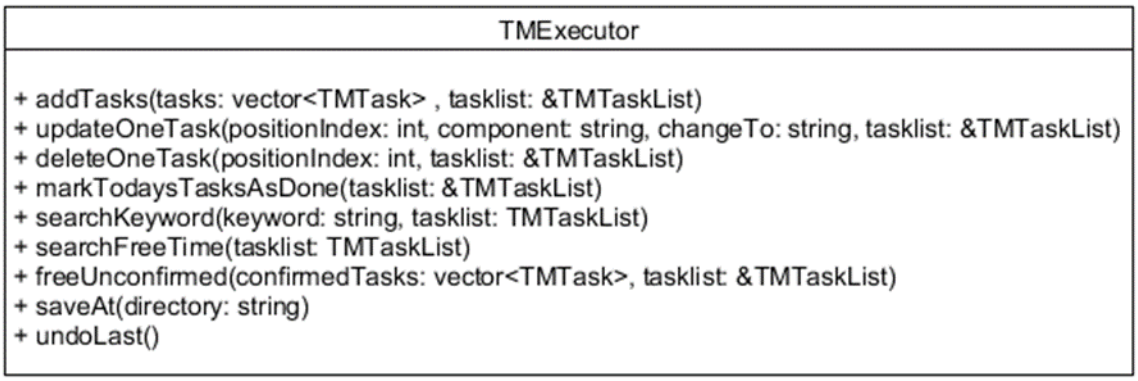
TM Parser

Main API:

Methods	Description
`vector<string> getTokenizedUserEntry(string)`	Converts a user's entry of string variable into a vector of string(s). Each word is extracted and treated as a string. Phrases that are encapsulated in open and closed inverted commas are treated as a string. Returns a vector of string(s) upon completion of the function.
`string extractCommand(string)`	Returns the intended command keyword from the user's entry
`vector<TMTask> parseTaskInfo(string)`	Extracts the task description of the task and if available the entered date(s) and time(s). After which, it stores these pieces of information into a TMTask object before pushing it into a vector

	of TMTask(s). Returns the vector upon completion.
`vector<int> parseTaskPositionNo(string)`	Converts the task position number from a string to an integer before pushing it into a vector of integer. Returns the vector upon completion.

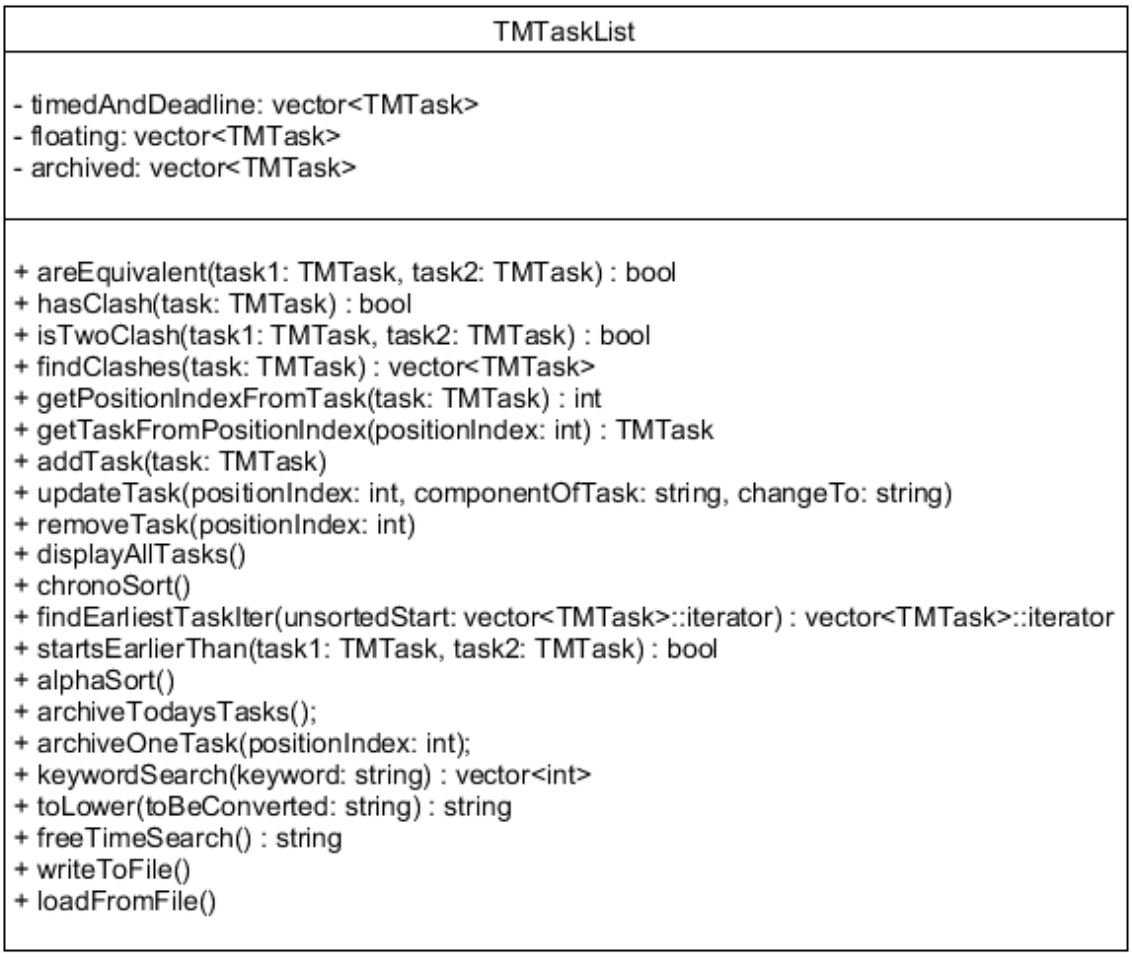
Executor



Main API:

Methods	Description
`addTasks(vector<TMTask>, TMTaskList &tasklist)`	Adds tasks into tasklist including confirmed and unconfirmed tasks.
`updateOneTask(int positionIndex, string component, string changeTo, TMTaskList &tasklist)`	Alters the information of component of the task represented by positionIndex on the display into the change as desired by the user.
`deleteOneTask(int positionIndex, TMTaskList &tasklist)`	Removes the task task represented by positionIndex fom tasklist.
`markTodaysTasksAsDone(TMTaskList tasklist)`	Mark all tasks having end date today as completed and relocate them into the archive vector in tasklist.
`searchKeyword(string keyword, TMTaskList tasklist)`	Search the task description of all tasks for the keyword. Note the search is not case sensitive.
`searchFreeTime(TMTaskList tasklist)`	Searches tasklist for all available slots which are not occupied by any task.
`freeUnconfirmed(vector<TMTask> confirmedTasks, TMTaskList &tasklist)`	Removes all tasks which are temporarily blocked by the user once he confirmed the time slot for a particular task
`saveAt(string directory)`	Saves all tasks and its information in a text file at the directory specified by the user
`undoLast()`	Undo the user's last action

Storage

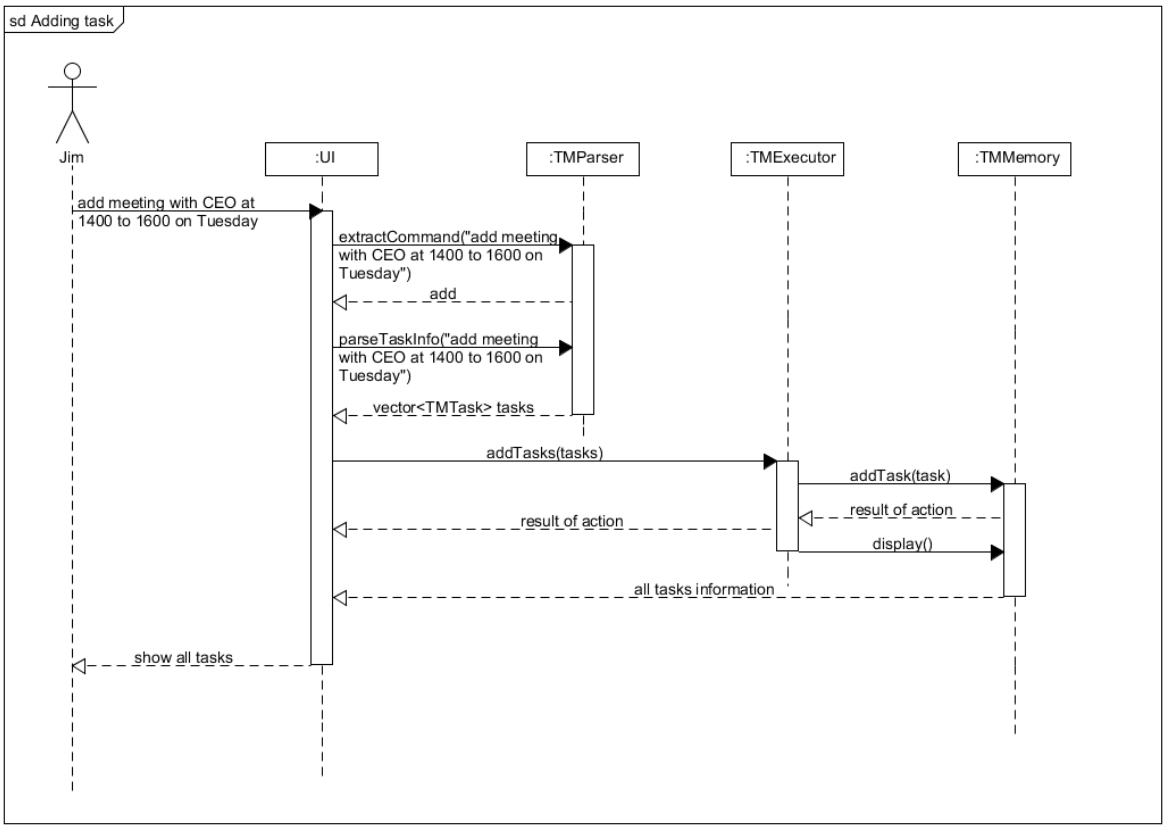


Main API:

Methods	Description
`addTask(TMTask)`	add task added by the user into appropriate container.
`removeTask(int)`	remove task specified by the user from the appropriate container.
`updateTask(int)`	update specified component(s) of task specified by the user.
`displayAllTask()`	display all task that is added previously. displayAllTask will split the task into two sections, the first will be dated task (sorted chronologically) and undated task (sorted by entry).
`keywordSearch(string)`	searches all task descriptions containing keyword desired by the user and displays them on a list.
`writeToFile()`	copies containers containing tasks into a .txt file. This function is called after every successful addTask and upon exit.
`loadFromFile()`	loads from text whatever writeToFile() saved to the same

text and put them into appropriate containers.

Algorithm of adding a task



Appendix A: User stories. As a user, …

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)	Criteria
addFloating	add a task by specifying a task description only	can record tasks that I want to do some day	
addBeforeSpecific	Add a task by specifying both task description and the deadline to meet	Can record tasks that has to be done by certain deadlines	Specify bfr {day/date}
addTimePeriod	Add a task by specifying both task description and the start and end times	Can record tasks that elapses over a period of time	Specify {start time} – {end time}
Glossary	Refer to a glossary of hotkeys	Can key in the commands faster	
sortTask	Classify different tasks into their respective categories (school/work/etc.) if I want to	Can refer to them easily when making a decision to swap events	Specify –sort {category}
attachFile	Attach files/documents that I'll have to work on as specified in the to-do (if any)	I can access them directly instead of having to locate them	
countdownToTask	be constantly reminded of tasks at a preset interval or at an interval of my desire	I will have enough time to prepare for that task	
searchFreeDates	Search for days that I am free by days or a certain period (week(s)/month(s)/customized)	So that I can add new appointments	Search {date/week/month/period}
setPriority	Set the priorities for tasks	I can make comparisons and decide which should	Set {high/med/low}

		be worked on first	
deleteTask	Delete a particular task should I choose to do so	I do not have to keep track of tasks that are done/that I do not want to follow anymore	CRUD (delete)
undoEntry	Undo entries	I can cancel my latest entries	Undo
searchEntry	Search for entries using keywords	I can find desired tasks faster	Search
repeatTask	Add tasks which repeats on daily, weekly or monthly basis with just one line of input	I can save more time on other tasks instead of repeatedly entering the same repetitive task	

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)	Criteria
passwordProtect	Protect my program with a password	My privacy is ensured	

Appendix B: Non-functional requirements

Platform

TimeMaster works on Windows 7 and above, on both 32-bit and 64-bit versions.

No internet access required

TimeMaster runs both in the presence and absence of internet connectivity

Command-line interface

User interacts with TimeMaster by entering commands in the form of text

Interoperability

TimeMaster is able to load information from text files and store information from the program into text files as well.

Appendix C: Product survey

Product: Apple IPhone Calendar **Documented by**: Ng Wei Ern

Strengths:

- Allows user to zoom out into full calendar view (whole year) or narrow down into a monthly view
- User can generate a list of to-do(s) which are categorized under their respective days and date
- able to segregate calendars into their different categories (emails/work/school)
- Today button available to return to today's date and view the to-do for the day (today)
- A to-do entry can be edited
- Information of the task can be categorized as follows:
  - title
  - location
  - all-day(Y/N)
  - start and end time
  - repeat
  - Invitees
  - Alert (Time before event)
  - can be classified under different calendars
  - Show as (Busy/Free)
  - URL
  - Notes

Weaknesses:

- Users are only allowed to search by the title of the task and not by date
- Users are not allowed to export calendar to other platforms

Product: iCal **Documented by:** Gordy

Strengths:

- Day/Month/Week/Year view
- Search function
- Timetable view down to the nearest minute

- Events automatically stacked if they clashes
- Different icons, different colors for different calendar source/categories, makes it easier for user to organize calendar.
- Filter function so that user can decide events from what category/source to show.
- Default current and next month view on the sidebar to make it easier for user to navigate through different weeks.
- Quick add function to add event to today.
- Can be synchronized with Google calendar, Exchange, etc.
- Numerous useful details can be added to an event. (Attachments, URL, Location, Repeated or not)
- Built-in reminder to show events today and tomorrow. Friendly language makes user feel comfortable to use.

Weakness:

- Syncing is problematic at times, very internet dependent

Product: iStudiez Pro (iOS) **Documented by**: Teo Kia Meng

Strengths:

- Provides very good support for any students. Allows the adding of student specific concepts such as
    - Semesters, Courses (modules) in Semester, Instructor, Venues
- Allows user to view different timescales (1 day, 1 week or 1 month).
- Homework tasks are also implemented in this program as Assignments.
- Allows cloud syncing between mobile and desktop apps.
- Allows viewing of past semesters and courses taken.
- Data export/ Data backup
- Notifications can be set to trigger at a specified time before tasks.
- Allows the importing of other online calendars so that multiple calendars can be viewed as once.
- Recurring tasks are implemented able to be scheduled across semesters
- Holidays can be manually added to indicate that there should be no class on a particular day

Weaknesses:

- Desktop app involves a lot of clicking to get things done, would be good if there was a command line interface
- Only good at student-centric tasks, not able to replace all other organizers, causing redundancy and wasted effort by having to use more than 1 app.

Product: My Calendar **Documented by**: Ng Xian Loon

Strengths:

- Presence of human-editable data file to manipulate the contents other from the program's interface.
- It is a desktop software
- No internet connection required to operate the program
- Tasks can be sorted within various parameters: start time, due date, subject name, category, completion %, priority.
- There is a completion bar to help keep track of task progress. User can choose to delete task after it is 100% completed.
- Able to export the data to a text file for reference.

Weaknesses:

- Unable to enter tasks without a specific due date.

Requires several clicking instead of keyboard usage in order to create an event.