Project Manual

ToDo++

October 15, 2012

			
Ivan Poon Team lead, logic programming	**Tay Xiu Li** Deadline watcher, documentation	**Yaadhav Raaj** UI programming	**Alice Jiang** Testing

ToDo++

User Guide

For v0.5 software

Table of Contents

Section 1: The Basics **5**
1.1. Introduction 5
1.2. At a Glance 5
1.3. Quick Start 6
Section 2: Creating Your ToDo++ List **7**
1.1. Types of Tasks 7
1.2. Basic Operation 7
1.3. Adding a Task 7
1.4. Searching & Viewing 8
1.5. Updating & Modifying 8
1.6. Removing Task(s) 9
Section 3: Advanced Features **10**
3.1. FlexiCommands 10
3.1.1. Custom Keywords 10
3.1.2. Using Reserved Keywords 11
3.2. Important Features for Power Users 12
3.2.1. Sorting Tasks 12
3.2.2. Marking Task(s) 12
3.2.3. Postponing Task(s) 13
3.2.4. Scheduling a Task 14
3.2.5. Undoing a Command 14
3.2.6. Redoing a Command 15
3.2.7. Exiting ToDo++ 15
3.2.8. Minimizing to TaskBar 15
3.2.9. Autorun & Other Settings 15
3.2.10. Hotkeys 15
Appendix **16**
COMMAND KEYWORDS 16
GENERAL COMMAND KEYWORDS 16
QUASI-GENERAL COMMAND KEYWORDS 16
SPECIFICALLY ADD COMMAND KEYWORDS 16
SPECIFICALLY SCHEDULE COMMAND KEYWORDS 17
ADD COMMAND DEFAULT BEHAVIOR 17

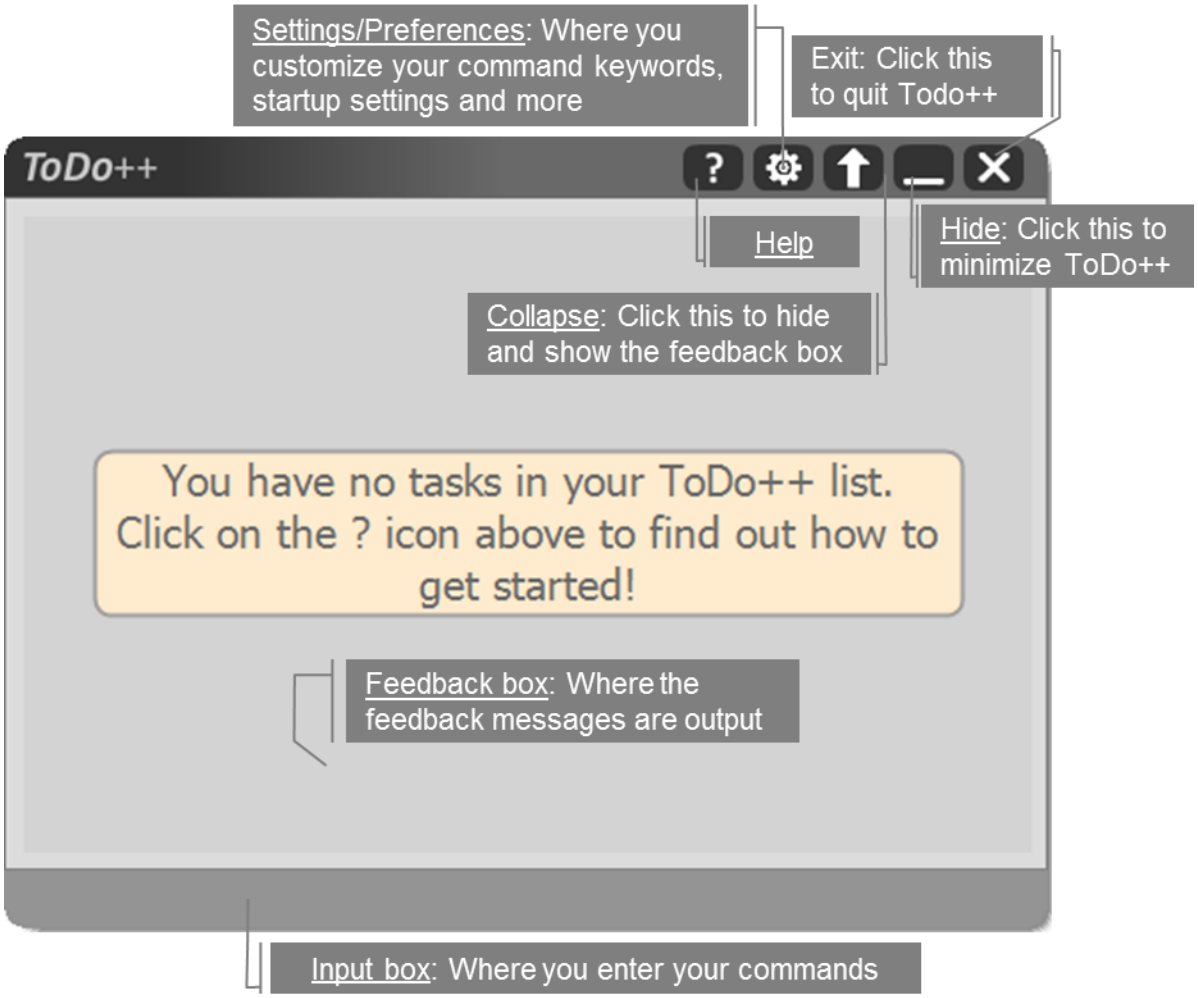
SEARCH/DISPLAY COMMAND DEFAULT BEHAVIOR....17
MODIFY COMMAND DEFAULT BEHAVIOR....17
POSTPONE COMMAND DEFAULT BEHAVIOR....17
SCHEDULE COMMAND DEFAULT BEHAVIOR....18
UNDO COMMAND DEFAULT BEHAVIOR....18
HOTKEYS AND KEYBOARD SHORTCUTS....18

Section 1: The Basics

1.1. Introduction

ToDo++ is the to-do application of your choice. Take control of your life like never before, with keyboard shortcut keys and intuitive natural-language-like text commands. Personalize the way you want to interact with the app. Be alerted of your events, and never lose track of your tasks again.

1.2. At a Glance



1.3. Quick Start

Upon launching ToDo++, you are presented with a minimalistic and intuitive screen. The first time you launch ToDo++, a simple help animation will introduce you to the many wonders of ToDo++! Find it superfluous? Then simply type away! That is all all that you have to do to start creating your to-do list!

- Create your first task/event by typing the keyword “add” followed by your task/event name into the input box, followed by the event time or deadline if any.
- For example, to add your Mom’s birthday brunch this Sunday, simply type in *“add mom’s bday this sunday”*. Hit *Enter* and the item is added!
- To view what you have added, just key in “display” and your list, ordered by date and time (or however you have customized it in the Settings  panel), will be displayed!

Note: If you have a task/event name that includes a date, time, or keyword, such as ‘Thank God It’s Friday' or 'delete ex-girlfriend’s number', use quotation mark or brackets around the task/event name to prevent ToDo++ from messing up your instruction.

You can search for, modify and delete your tasks from the list using these very words as the keywords. Alternative keywords can be found in the *Appendix*.

Remember the order of what you enter does not matter! Take a look at section *3.1. FlexiCommands* to find out more and learn how to customize ToDo++ to your needs.

The following sections will describe basic ToDo++ operations in more detail.

Section 2: Creating Your ToDo++ List

1.1. Types of Tasks

There are three basic types of tasks you can add to your list.

Event tasks **:** items that have a start time (and possibly an end time).
Deadline tasks **:** items that have to be done before a specific time.
Floating tasks **:** items that have no timings attached to them.

As always, there is no fixed way to add a certain type of task. Simply type in what makes sense to you, and ToDo++ will know what type of task to add! See the next section, “Adding an Item” for more information.

1.2. Basic Operation

The order of input of all required fields is flexible. Date input is context sensitive. For more information, please see section *3.1. FlexiCommands*.

For a complete list of default keywords and more details on how to use bracketing, please see section *3.1.2. Using Reserved Keywords*.

You can easily sort your tasks by name and date, schedule your task to be happening at your earliest free time slot and quickly undo a wrongly input command using the “undo” command. For more information, please see section *3.2.5. Undoing a Command*.

The following sections detail the basic operations.

1.3. Adding a Task

Using the default add keyword, you can add all types of items to your list easily in a structured format similar to natural language. The task type will depend on the inclusion or omission of start/end times/deadlines. You can add your tasks in the following ways:

Adding a floating task	Enter “add [task name]” eg. add finish project
Adding an event (timed) task	Enter “add [task name] [start time] {end time} {day/date}“ eg. add max birthday 4pm tomorrow eg. add team meeting 2pm-4pm next wed
Adding a deadline item	Enter “add [task name] by [deadline]“ eg. add do cs2103 CE2 by saturday midnight

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

1.4. Searching & Viewing

Using the default display or search keyword, you can search, filter and view your list of to-do items. ToDo++ has a powerful sort and search algorithm that will help you find your tasks with intuitive commands in consummate ease.

A search request may be made up of more than one search requirement. Below are examples of how you can use this feature to its full potential. Optional commands are in curly braces.

Display all tasks	Enter "display/search"
Search for and display all or some of the tasks scheduled on, before or, after a specific day or date	Enter "display/search [day/date]{before/after}{time}" eg. display 06/09/2012 eg. display Sunday after 1500hrs eg. display 6 sept before 10pm eg. display next Saturday eg. display tomorrow eg. display June
Search for and display tasks by task names or descriptions	Enter "display/search [name]" eg. display buy milk

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

Search results will be displayed in the feedback window as an indexed list. Each task will be given a numbered ID for easy reference. From here, you can do further operations, such as modifying or deleting an item. See the following sections for more information.

1.5. Updating & Modifying

Using the default modify keyword, you can easily modify and edit your task details. For example, you may wish to change the scheduled date for a task or rename it to correct a typographical error.

Note that in order to modify a task name, you must first call for the task by its name and then modify it by calling its reference index. The input should not contain any date/time input. Otherwise, it will be regarded as a request to modify the task's start/end times/deadlines.

Modifying task name	Enter "modify [task index] [new name]" eg, *(User)* modify milk *(ToDo++)* 1. milk Stevv for more info 2. buy more milk *(User)* modify 1 milk Steve for more info
Modifying task time	Enter "modify [task index] [new start time/deadline] {end time} {day/date} " eg, *(User)* modify buy car tomorrow *(ToDo++)* 1. buy toy car, 5pm 2. buy car parking coupon today *(User)* modify 2 tomorrow

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

1.6. Removing Task(s)

Using the default delete keyword, you can remove previously added tasks from your ToDo++ list. You can call for a task and delete it in the 2 following ways:

By task name Call the task by its name.

Note that if more than one task matches the input name, all matching tasks will be reflected in the feedback window. Each task will be given a numbered index for easy reference.

By task index Call the task by its given index, as displayed in the feedback window.

Deleting a single task	Enter "delete [task name/index]" eg. delete 3 eg, delete buy more milk (only one task matching specified task name) eg, *(User)* delete milk *(ToDo++)* 1. milk Steven for more info 2. buy more milk *(User)* delete 2
Deleting all tasks displayed	Enter "delete all"
Deleting all tasks on a specific date	Enter "delete all [day/date]" eg. delete all tmr eg. delete all friday eg. delete all 3 dec

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

Section 3: Advanced Features

3.1. FlexiCommands

ToDo++ is an intelligent software that allows you to input commands in a way that is natural to you. Your input is therefore not case sensitive and not order specific. What this means is that the following commands are both equivalent and valid!

add max's birthday 25th oct	add 25/10 max's birthday	
add movie outing 3pm to 5pm today	3pm – 5pm movie outing add	
add breakfast tmr with family 5am	ADD breakfast with family 5am	*[past 5am today]*
add party lunch tmr afternoon	party lunch tmr 12pm – 5pm add	*[default settings]*
add project REMAKE by midnight	add project REMAKE by 12am	

3.1.1. Custom Keywords

ToDo++ is a to-do list made to be familiar and intuitive to the individual user. If you do not wish to use the default keywords provided, you may easily set your own custom keywords for the various basic operations in Settings > FlexiCommands.

Adding the remove keyword for command delete

3.1.2. Using Reserved Keywords

There may be times when you wish to use a keyword (a command, day, date or time keyword) as part of your task name. Simply enclose the keyword within any of the following delimiters to do so:

- Quotation marks
    - add **“**add hot girl on facebook**”**
    - delete **‘***delete* issue**’**
- Brackets/Braces
    - modify 2 **{***remove* hot girl from facebook**}**
    - add **(**go to *2am* Bar**)** 2am tomorrow
    - display **[***21st* birthday**]**

3.2. Important Features for Power Users

3.2.1. Sorting Tasks

Using the default sort keyword, you can sort the items in your list easily in the following ways:

By task date	Enter "sort date" eg. (User) search tomorrow (ToDo++) 1. lunch with colleagues 11 Nov, 1:00PM 2. breakfast 11 Nov, 5:00AM 3. morning run 11 Nov, 3:00AM – 4:30AM 4. supper 11 Nov, 11:00PM (User) sort name (ToDo++) 1. morning run 11 Nov, 3:00AM – 4:30AM 2. breakfast 11 Nov, 5:00AM 3. lunch with colleagues 11 Nov, 1:00PM 4. Supper 11 Nov, 11:00PM
By task name	Enter "sort name" eg. (User) display (ToDo++) 1. peter asked me out! DATE! 2. richard asked me out! DATE! 3. simon asked me out! DATE! 4. david asked me out! DATE! (User) sort name (ToDo++) 1. david asked me out! DATE! 2. peter asked me out! DATE! 3. richard asked me out! DATE! 4. simon asked me out! DATE!

Note that "date" and "name" are keywords that must be used in conjunction with the sort command.

3.2.2. Marking Task(s)

[SIMILAR TO THE DELETE AND POSTPONE COMMAND]

Using the default done and undone keyword, you can easily mark the tasks/items in your to-do list as complete (aka done) or incomplete (aka undone). Your task will then be displayed as [DONE] and [UNDONE] respectively.

You can call for a task and mark it as done or undone in the 2 following ways:

By task name Call the task by its name.

Note that if more than one task matches the input name, all matching tasks will be reflected in the feedback window. Each task will be given a numbered index for easy reference.

By task index Call the task by its given index, as displayed in the feedback window.

Marking a single task	Enter "done/undone [task name/index]" eg. done 3 eg, done buy more milk

	(only one task matching specified task name) eg, *(User)* done milk *(ToDo++)* 1. milk Steven for more info 2. buy more milk *(User)* done 2
Marking all tasks displayed	Enter "done/undone all"
Marking all tasks on a specific date	Enter "done/undone all [day/date]" eg. done all today eg. undone all tomorrow eg. done all june

3.2.3. Postponing Task(s)

[SIMILAR TO THE DELETE AND DONE COMMAND]

Using the default postpone keyword, you can easily postpone your deadline or timed event task by a duration specified in hours, days or weeks. For example, you may wish to postpone a dinner date by an hour due to unforeseen heavy traffic or unplanned overtime.

Some points to note:

i. A task with no specific time cannot be postponed by a number of hours and a task with no specific date cannot be postponed by a number of days.
ii. If a duration is not specified, the task will be automatically postponed by the default postpone duration specified in Settings  > FlexiCommands > Postpone.

You can call for a task and postpone it in the 2 following ways:

By task name Call the task by its name.

Note that if more than one task matches the input name, all matching tasks will be reflected in the feedback window. Each task will be given a numbered index for easy reference.

By task index Call the task by its given index, as displayed in the feedback window.

Postponing a single task	Enter "postpone [task name/index] {duration}" eg. postpone 3 1 hour eg, postpone buy more milk 2 days (only one task matching specified task name) eg, *(User)* postpone milk *(ToDo++)* 1. milk Steven for more info 2. buy more milk *(User)* postpone 2 2 days	 11 Nov, 1:00PM 11 Nov
Postponing all tasks displayed	Enter "postpone all"	
Postponing all tasks on a specific date	Enter "postpone all [day/date] {duration}" eg. postpone all tmr 1 hour eg. postpone all Friday 1 day eg. postpone all 3 dec eg. postpone all june, 1 week	

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

3.2.4. Scheduling a Task

Using the default schedule keyword, you can schedule your event task to be automatically allocated your earliest free time slot within your specified time range. For example, you may wish to schedule a short 1 hour dental appointment on Friday but find it a pain to look through your Friday schedule for a suitable time slot.

Tasks duration may be specified in hours, days, weeks, or months. For example, a task may be 3 hours long or 5 days long in duration. Schedule time ranges may be specified by dates (days or months) or keywords like morning, afternoon, evening and night.

Some points to note:

i. If there is no time slot available within the specified time range, the task will not be scheduled.
ii. If the task duration is not specified, the default task duration specified in Settings  > FlexiCommands > Schedule.
iii. If no time range is specified, the task will be scheduled at your earliest fitting free time slot.

Scheduling an event task	Enter "schedule [task name] {task duration} {time range}" eg. schedule dental appointment 1 hour Friday afternoon eg. schedule dental appointment Friday afternoon eg. schedule dental appointment eg. schedule chalet 3 days, june

Note: Optional keywords are in curly braces. See Appendix for a full list of keywords and defaults.

3.2.5. Undoing a Command

In order to undo a mistyped command, simply enter undo to revert to the state before the last undoable entered command.

Note that commands such as search and sort cannot be undone.

eg, *(User)* display
(ToDo++) 1. milk Steven for more info
2. buy more milk
3. buy car coupon
4. visit supermarket to enter lucky draw ticket
(User) delete 4
(ToDo++) 1. milk Steven for more info
2. buy more milk
3. buy car coupon
(User) sort name
(ToDo++) 1. buy car coupon
2. buy more milk
3. milk Steven for more info
(User) undo *[late undoable commandwas delete]*
(ToDo++) 1. buy car coupon
2. buy more milk

3. milk Steven for more info
4. visit supermarket to enter lucky draw ticket

3.2.6. Redoing a Command

In order to redo an undone command, simply enter redo to revert to the state before the last undone.

3.2.7. Exiting ToDo++

To exit ToDo++, you can simply click on the cross in the top right hand corner or enter exit. Alternatively, you can also use the *CTRL+Q* keyboard shortcut to exit the program.

3.2.8. Minimizing to TaskBar

Simply click on  to minimize ToDo++ to the system trya.Alternatively, you can also use the *ALT+Q* keyboard shortcut. You will stil be able to see the ToDo++ icon in the taskbar notification area while it runs in the background.

To restore the ToDO++ window, simply double-click on the icon or use the same minimizing *ALT+Q* keyboard shortcut.

Icon minimized to notification area

3.2.9. Autorun & Other Settings

You can get ToDo++ to run automatically on Windows startup by simply checking the Settings  > Load on Startup option. Additonally, you can also check Settings  > Start Minimized if you wish for ToDo++ to start running in the minimized state.

If you wish for ToDo++ to always be placed floating on top of all your other programs, simply check Settings  > Stay On Top.

3.2.10. Hotkeys

A list of default hotkeys such as *ALT-Q* to toggle between the minimized and restored states can be found in the *Appendix*.

Appendix

Note: Optional keywords are in {curly braces}. Inputs in [square brackets] must be valid in order for the keywords to work.

COMMAND KEYWORDS

ADD: add
SEARCH/VIEW: display, search
MODIFY: modify
DELETE: delete
SORT: sort
DONE: done
UNDONE: undone
POSTPONE: postpone
SCHEDULE: schedule
UNDO: undo
REDO: redo
EXIT: exit

GENERAL COMMAND KEYWORDS

DATE SEPARATOR: hyphen (-), period (.), forward slash (/)

DAY: mon, monday, tues, tuesday, wed, wednesday, thurs, thursday, fri, friday, sat, saturday, sun, sunday, today, tomorrow

MONTH: jan, feb, mar, apr, may, jun, jul, aug, sep, sept, oct, nov, dec, january, february, march, april, may, june, july, august, september, october, november, december

DATE: {1-31}[DATE SEPARATOR]{1-12}[DATE SEPARATOR]{valid year}
DATE: {1-31}{st/nd/rd/th}{MONTH}{valid year}
** accepted partial dates must be a combination of suffixed day, day and month or month and year (see below for more information)*

TIME: [1-12] am/pm
** spaces are optional*
TIME: midnight, noon
TIME RANGES: morning, afternoon, evening, night

CONDITIONAL ADJECTIVES: [next/following] [DAY/MONTH]

QUASI-GENERAL COMMAND KEYWORDS

INDEX: all integer numbers are reserved if and only if it is used with a relevant command that may require an index
** such as modify, delete, postpone, done and undone*

ALL: the all keyword is reserved if and only if it is used with a relevant command
** such as modify, delete, postpone, done and undone*

DURATION: hr, hrs, wk, wks, hours, hours, day, days, week, weeks
duration keywords are reserved if and only if used with relevant commands
** such as postpone and schedule* ; *must be used with a preceding integer*

SPECIFICALLY ADD COMMAND KEYWORDS

TIME: [0000-2359] hrs/hours
** spaces are optional*

EVENT SEPARATOR: hyphen (-), to

SIGNIFY EVENT (TIMED) TASK:
{from} [DATE/DAY/TIME] {EVENT SEPARATOR} {DATE/DAY/TIME}
** omission of time/month keyword for start time is valid as long as it is present in the end time (e.g. 2-4pm)*

SIGNIFY DEADLINE TASK: by [DATE/DAY/TIME]

SPECIFICALLY SCHEDULE COMMAND KEYWORDS

DURATION: mth, mths, month, months
these duration keywords are reserved if and only if used with the schedule command
** must be used with a preceding integer*

ADD COMMAND DEFAULT BEHAVIOR

- A task must be given a valid name that consists of at least non-space character.
- Not specifying any fields in date or time will cause the software to assume the most upcoming date which fits the entered fields. For example, “add event 2nd" will set the event to be on 2nd October if today’s date is 3rd September but 2nd September if today’s date is 1st September.
- The valid partial date inputs include inputs consisting of only the
  (1) day with suffixes i.e. 15th
  (2) day and month i.e. 15/10
  (3) month and year i.e. 10/2012 (requires full year input of YYYY)
- Not specifying the by keyword will create an event (timed) task instead of a deadline task even if only one date/time is specified. The date/time specified will be the event’s start time.
- Omission of both time and month keyword when creating an event task will cause ToDo++ to assume that the numbers refer to month by default.

SEARCH/DISPLAY COMMAND DEFAULT BEHAVIOR

- All possible results will be displayed when keywords are omitted. For more specific results, try to enter a more specific search string.
- When searching by date or day, as long as the specified search date or day falls within the time window of an event task, the event task will also be displayed as a search result.

MODIFY COMMAND DEFAULT BEHAVIOR

- When the modify operation is called with an input that includes a date/day/time, it will be automatically considered to be a request for modification of the start/end times/deadline of the relevant task. If there is only one task found to match the input, its start/end times/deadline will thus be automatically modified. Otherwise, the modify operation functions like a search/view operation and returns a list of matching hits.
- In order to modify a task name, the task must be called by its index. Otherwise, the request will not be recognized.

POSTPONE COMMAND DEFAULT BEHAVIOR

- In order to postpone a deadline or timed event task by a specific duration type, the task must be accordingly time-specific.

- If a duration is not specified, the task will be automatically postponed by the default postpone duration specified in Settings  > FlexiCommands > Postpone.

SCHEDULE COMMAND DEFAULT BEHAVIOR

- If there is no time slot available within the specified time range, the task will not be scheduled.
- If the task duration is not specified, the default task duration specified in Settings  > FlexiCommands > Schedule.
- The tasssssk duration must be specified in full i.e. with a specified amount and valid duration type.
- If no time range is specified, the task will be scheduled at your earliest fitting free time slot.
- A scheduled task will only begin at the start of every hour.
- If a time is specified in conjunction with a time range keyword such as morning and afternoon, it will be taken to be the end time and will override the time range's end time. Therefore, by default, "schedule task tomorrow morning 1pm" will attempt to schedule the task some time tomorrow from 5am to 1pm.
- If both the start time and end time are specified in conjunction with a time range keyword such as morning and afternoon, the specified times will override the time range's start time and end time. Therefore, by default, "schedule task tomorrow morning 4am to 1pm" will attempt to schedule the task some time tomorrow from 4am to 1pm.

UNDO COMMAND DEFAULT BEHAVIOR

- The last undoable command will be undone. This means that previously entered commands that did not result in any changes as well as search and sort commands will simply be disregarded.

HOTKEYS AND KEYBOARD SHORTCUTS

CTRL+A	select all text in ionput box
CTRL+Q	exit ToDo++
ALT+C	toggle between feedback panel and console panel
ALT+H	toggle between main window and help panel
ALT+S	toggle between main window and settings panel
ALT+SPACE	return cursor to input box
ALT+UP/DOWN	toggle between expanded and collapsed state
ALT + BACKSPACE	delete previous word or current word till selection position
CTRL/ALT+LEFT	move to start of curernt word in input field
CTRL/ALT+RIGHT	move to start of next word in input field
UP/DOWN	retrieve previous/next input

ToDo++

Developer Guide

For v0.5 software

Table of Contents

1.1. Where We Are Now 28
1.1.1. Basic Features 28
1.1.2. Extra Feature 28
Section 2: Architecture & Implementation 29
2.1 Architecture 29
2.1.1. Architecture Diagram 29
2.1.2. Class Diagram 30
2.1.3. User Sequence Diagram 31
2.2. General Class Descriptions 32
Program Class 32
Logger Class 32
Response Class 32
User Interface Classes 32
Settings Classes 32
Logic Class 32
Parser Classes (Parsers) 32
Token Classes (Tokens) 33
Task Classes (Tasks) 33
Operation Classes (Operations) 33
Storage Class 34
Section 3: Application Programming Interface (API) 35
3.1. UI Classes 35
3.1.1. UI 35
Summary 35
Detailed Description 35
Adding or Displaying Pages (#PanelSwitching) 35
Adding Keyboard Shortcuts (#Hotkeys) 35
Internal Design Functions (#InternalDesignFunctions) 36
Constructor 36
Important API (Public Method) 36
Important API (Private Methods) 36
3.1.2. PreferencesPanel 37
Summary 37

Detailed Description 37
Constructor 37
Important API (Public Method) 37
Important API (Private Methods) 37
3.1.3. FlexiCommandsControl 38
Summary 38
Detailed Description 38
Constructor 38
Important API (Public Method) 38
Important API (Private Methods) 38
3.1.4. Custom Message Boxes 40
FontDialogBox 40
Summary 40
Static Methods 40
UserInputBox 40
Summary 40
Static Methods 40
AlertBox 41
Summary 41
Static Methods 41
TinyAlert 41
Summary 41
Static Methods 41
3.2. Settings Classes 42
3.2.1. SettingsInformation 42
Summary 42
Default Values 42
Adding new settings 42
Constructor 42
Settings 42
Important API (Public Methods) 42
3.2.2. Settings 43
Summary 43
Adding Settings Methods 43

Constructor 43
Important API (Private Method) 43
Important API (Public Methods) 43
3.3. Logic & Parser Classes 45
3.3.1. Logic Class 45
Summary 45
Sequence Diagram 45
Detailed Description 46
Test History 46
Important Attributes 46
Constructor 46
Important API (Public Methods) 46
Important API (Internal Methods) 46
Important API (Private Methods) 46
3.3.2. CommandParser 48
Summary 48
Detailed Description 48
Test History 48
Constructor 48
Important API (Public Method) 48
Important API (Private Methods) 48
3.3.3. StringParser 49
Summary 49
Detailed Description 49
Test History 49
Important Attributes 49
Important API (Public Method) 49
3.3.4. OperationGenerator 51
Summary 51
Detailed Description 51
Test History 53
Important Attributes 53
Constructor 53
Important API (Public Methods) 53

Important API (Internal Methods) 53
3.3.5. TokenGenerator 55
Summary 55
Detailed Description 55
Test History 55
Important API (Public Methods) 55
3.3.6. CustomDictionary 57
Summary 57
Detailed Description 57
Test History 57
Constructor 57
Important Attributes 57
Important API (Public Methods) 59
3.4. Token Classes 60
3.4.1. Token 60
Summary 60
Detailed Description 60
Test History 60
Important API (Internal Methods) 60
3.4.2. TokenCommand : Token 61
Summary 61
Detailed Description 61
Constructor 61
Important API (Internal Methods) 61
3.4.3. TokenContext : Token 62
Summary 62
Detailed Description 62
Constructor 62
Important API (Internal Methods) 62
3.4.4. TokenDate : Token 63
Summary 63
Detailed Description 63
Constructor 63
Important API (Internal Methods) 63

3.4.5. TokenDay : Token 64
Summary 64
Detailed Description 64
Constructor 64
Important API (Internal Methods) 64
3.4.6. TokenIndexRange : Token 65
Summary 65
Detailed Description 65
Constructor 65
Important API (Internal Methods) 65
3.4.7. TokenLiteral : Token 66
Summary 66
Detailed Description 66
Constructor 66
Important API (Internal Methods) 66
3.4.8. TokenSortType : Token 67
Summary 67
Detailed Description 67
Constructor 67
Important API (Internal Methods) 67
3.4.9. TokenTime : Token 68
Summary 68
Detailed Description 68
Constructor 68
Important API (Internal Methods) 68
3.4.10. TokenTimeRange : Token 69
Summary 69
Detailed Description 69
Constructor 69
Important API (Internal Methods) 69
3.5. Task Classes 70
3.5.1. Task 70
Detailed Description 70
Test History 70

Important Attribute 70
3.5.3. TaskFloating : Task 71
Summary 71
A task with no date or times attached to it. The simplest type of task. 71
Detailed Description 71
Constructor 71
Important API (Public Method) 71
3.5.4. TaskDeadline : Task 72
Summary 72
A task with a single date and time representing the deadline of the task. Date and time may be ambiguous. 72
Detailed Description 72
Constructor 72
Important API (Public Method) 72
Important API (Private Method) 72
3.5.5. TaskEvent : Task 73
Summary 73
A task with one or more date and times attached to it. Represents the start and end times of the task. If there is only one date/time, it means the task does not have an end time. 73
Detailed Description 73
Constructor 73
Important API (Public Method) 73
Important API (Private Method) 73
3.6. Operation Classes 75
3.6.1. Operation 75
Summary 75
Detailed Description 75
Test History 75
Constructor 75
Important API (Public Method) 75
3.6.2. OperationAdd : Operation 77
Summary 77
Detailed Description 77
Constructor 77

Important API (Public Method) 77
3.6.3. OperationDelete : Operation 78
Summary 78
Detailed Description 78
Constructor 78
Important API (Public Method) 78
3.6.4. OperationDisplayDefault : Operation 79
Summary 79
Detailed Description 79
Constructor 79
Important API (Public Method) 79
3.6.5. OperationMarkAsDone : Operation 80
Summary 80
Detailed Description 80
Constructor 80
Important API (Public Method) 80
3.6.6. OperationMarkAsUndone : Operation 81
Summary 81
Detailed Description 81
Constructor 81
Important API (Public Method) 81
3.6.7. OperationModify : Operation 82
Summary 82
Detailed Description 82
Constructor 82
Important API (Public Method) 82
3.6.8. OperationPostpone : Operation 83
Summary 83
Detailed Description 83
Constructor 83
Important API (Public Method) 83
3.6.9. OperationRedo : Operation 84
Summary 84
Detailed Description 84

Constructor 84
Important API (Public Method) 84
3.6.10. OperationSchedule : Operation 85
Summary 85
Detailed Description 85
Constructor 85
Important API (Public Method) 85
3.6.11. OperationSearch : Operation 86
Summary 86
Detailed Description 86
Constructor 86
Important API (Public Method) 86
3.6.12. OperationSort : Operation 87
Summary 87
Detailed Description 87
Constructor 87
Important API (Public Method) 87
3.6.13. OperationUndo : Operation 88
Summary 88
Detailed Description 88
Constructor 88
Important API (Public Method) 88
3.7. Storage Class 89
Summary 89
Detailed Description 89
Constructor 89
Important API (Public Methods) 89
Important API (Internal Methods) 89

Section 1: An Overview

1.1. Where We Are Now

1.1.1. Basic Features

The basic features of ToDo++ include support for 3 types of tasks, as follows:

Event tasks **:** items that have a start time (and possibly an end time).
Deadline tasks **:** items that have to be done before a specific time.
Floating tasks **:** items that have no timings attached to them.

They include the following:

1. Add, Display/Search, Modify, Delete (CRUD)
2. Undo
3. Redo
4. Schedule
5. Postpone
6. Mark (as done/undone)
7. Sort

1.1.2. Extra Feature

The focus good-to-have feature is FlexiCommand. This means to say that the user is allowed high flexibility in his command format; highly ordered/structured input is not required. A simple GUI has also been implemented such that keyboard shortcuts/hotkeys and user customization of command keywords are also available.

Section 2: Architecture & Implementation

2.1 Architecture

This section will provide you with a top-down model of ToDo++.

2.1.1. Architecture Diagram

The following describes the general architectural overview of our software.

2.1.2. Class Diagram

The following class diagram captures the basic relationships between all the important classes of the software.

~ UpdateLastDisplayedTasksList(displayedList : List<Task>)

2.1.3. User Sequence Diagram

The following is the sequence diagram describing the steps involved when a user inputs a command into the GUI's input box.

2.2. General Class Descriptions

Program Class	
Class	**Description**
Program (static)	The main entry point for the application

Logger Class	
Class	**Description**
Logger	Static class which allows logging to be done

Response Class	
Class	**Description**
Response	Container which stores all necessary details for a UI to feedback the command results to the user.

User Interface Classes	
Class	**Description**
UI	Takes in all user input Displays returned feedback
PreferencesPanel	
FlexiCommandsControl	
FontAlertBox	
AlertBox	
TinyAlert	

Settings Classes	
Class	**Description**
SettingsInformation	
Settings	Takes in and implements the settings

Logic Class	
Class	**Description**
Logic	Takes in and processes user input Executes the command Returns feedback

Parser Classes (Parsers)	
Class	**Description**
CommandParser	Takes in and parses command string Returns an Operation object
CustomDictionary	
OperationGenerator	
StringParser	Takes in and parses the command string into tokens

	Returns a List of generated Tokens
TokenGenerator	

Token Classes (Tokens)	
Class	**Description**
Token (abstract)	An abstract class from which all the other Token classes inherit from
TokenCommand : Token	Stores the details pertaining to the command information of a task
TokenContext : Token	Stores the details pertaining to the context information of a task
TokenDate : Token	Stores the details pertaining to the date information of a task
TokenDay : Token	Stores the details pertaining to the day information of a task
TokenIndexRange : Token	Stores the details pertaining to the index(es) information of a task
TokenLiteral : Token	Stores the details pertaining to the literal information of a task
TokenSortType : Token	Stores the details pertaining to the sort type information of a task
TokenTime : Token	Stores the details pertaining to the time information of a task
TokenTimeRange : Token	Stores the details pertaining to the time range information of a task (specified by duration keywords)

Task Classes (Tasks)	
Class	**Description**
Specificity	Stores the date and time specificity of a task
Task (abstract)	An abstract class from which all the other Task classes inherit from
TaskFloating : Task	Stores the task details of a floating task
TaskDeadline : Task	Stores the task details of a deadline task
TaskEvent : Task	Stores the task details of an event (timed) task

Operation Classes (Operations)	
Class	**Description**
Operation (abstract)	An abstract class from which all the other Operation classes inherit from
OperationAdd : Operation	Stores the operation details for an add operation
OperationDelete : Operation	Stores the operation details for a delete operation
OperationDisplayDefault: Operation	Stores the operation details for the default display operation
OperationMarkAsDone : Operation	Stores the operation details for a mark as done operation
OperationMarkAsUndone : Operation	Stores the operation details for a mark as undone operation
OperationModify : Operation	Stores the operation details for a modify operation
OperationPostpone : Operation	Stores the operation details for a postpone operation
OperationRedo : Operation	Stores the operation details for a redo operation
OperationSchedule : Operation	Stores the operation details for a schedule operation

OperationSearch : Operation	Stores the operation details for a search operation
OperationSort : Operation	Stores the operation details for a sort operation
OperationUndo : Operation	Stores the operation details for an undo operation

Storage Class	
Class	**Description**
Storage	Handles the storage of tasks information Takes in a Task object and writes it to an XML file

Section 3: Application Programming Interface (API)

3.1. UI Classes

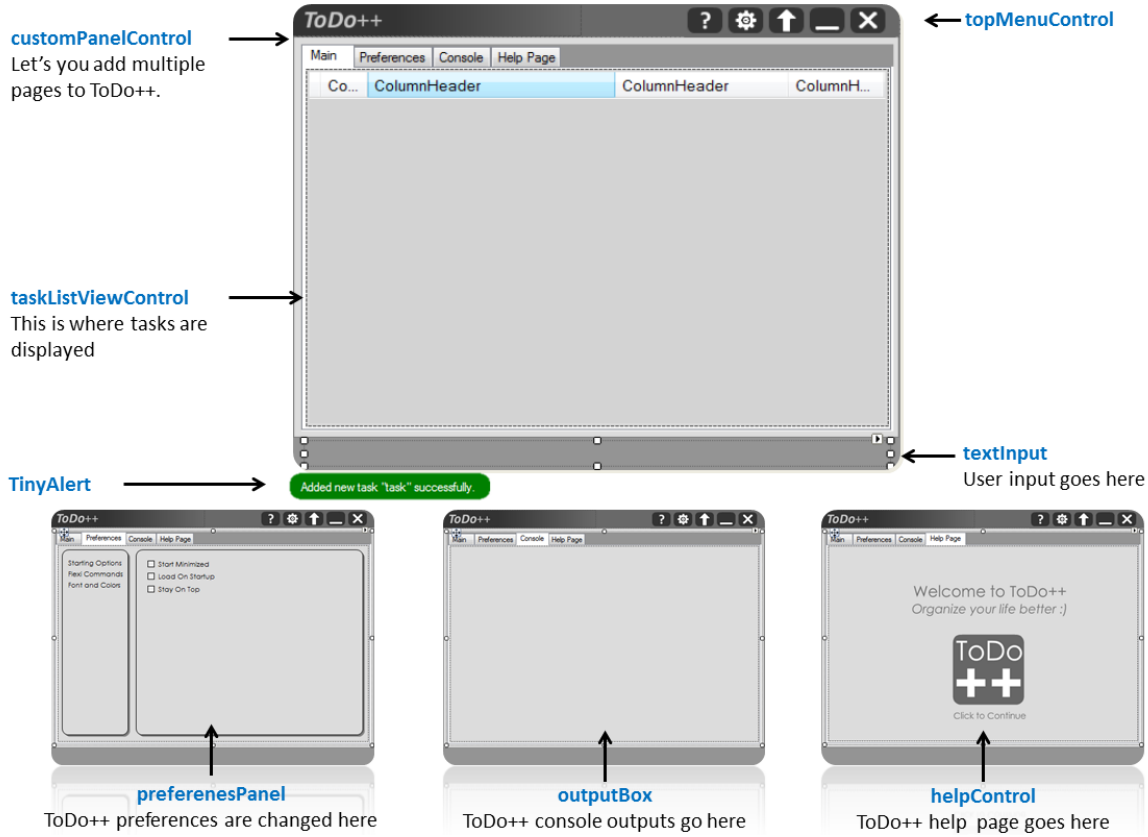
3.1.1. UI

Summary

The class that contains all user interface components and interacts directly with Logic and Settings, displaying the relevant output to the user

Detailed Description

UI contains the following components. They have been labeled in the images below



Adding or Displaying Pages (#PanelSwitching)

UI employs a developer friendly component called CustomPanelControl that allows you to add multiple pages/panels. These panels can be switched easily by modifying the SelectedIndex property. Implementation can be seen in the #region PanelSwitching

Adding Keyboard Shortcuts (#Hotkeys)

UI contains a function ProcessCmdKey that lets developers add new hotkeys as long as you are within the scope of ToDo++. This function can be found in #Hotkeys. Adding global hotkeys that are accessible outside ToDo++ however is a Win32 Function that can be found in the section below

Internal Design Functions (#InternalDesignFunctions)

This region contains code for animations such as collapsing and expanding of form (#CollapseExpand), Fading in and Out (#FormFadeInOut), Minimizing to the TaskBar (#SystemTray), and other Win32 based functions such as loading on startup, shadows and rounded edges.

Constructor

UI(Logic logic);	Starts by initializing all designer components, including Logic and MainSettings. References of these are then passed into some components such as preferencesControl

Important API (Public Method)

Method	Description
ToggleHelpToDoPanel()	Toggles between Help and TaskListView Panel
ToggleToDoPreferencesPanel()	Toggles between Preferences and TaskListView Panel
ToggleConsolePanel()	Toggle between TaskListView and Console Panels

Important API (Private Methods)

Method	Description
InitializeSettings()	Checks if ToDo++ should be minimized to tray when started and ensures that ToDo++ should load on startup depending on user settings
RegisterInStartup(bool isChecked)	Adds a registry entry to ensure that ToDo++ opens when started up
MinimiseMaximiseTray()	Toggles between minimizing and maximizing ToDo++ from the system tray
ProcessText()	Takes in the user input and processes it via logic, displaying the task list in taskListViewControl

3.1.2. PreferencesPanel

Summary
Manages all Preferences Controls

Detailed Description
PreferencesPanel uses a CustomPanelControl to switch between components. This means you can add as many preference controls as long as space permits. The controls found here directly modify MainSettings, and the settings are saved immediately.

Constructor

PreferencesPanel()	Loads preference names and all preference controls and components

Important API (Public Method)

Method	Description
InitializeWithSettings(Settings settings)	Loads MainSettings into this class. The controls cannot function without a reference to this

Important API (Private Methods)

Method	Description
LoadPreferencesTree()	You can set the preference titles here for your controls and modify the event handler for selecting preferences

In this developer manual, we will only be looking at the FlexiCommandControl component, as that is likely the component you will be interacting with as a developer.

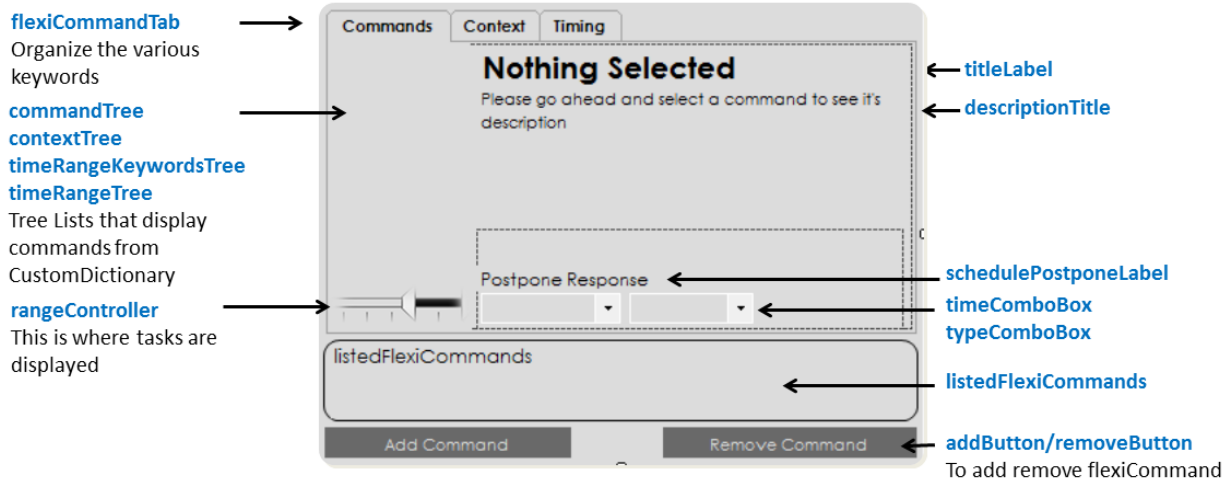
3.1.3. FlexiCommandsControl

Summary

This is where the user can modify the commands and keywords

Detailed Description

FlexiCommandsControl is a user interface wrapper that calls and modifies the flexicommand properties of MainSettings.



Constructor

PreferencesPanel()	Loads all keywords from CustomDictionary and displays them

Important API (Public Method)

Method	Description
InitializeFlexiCommands(Settings settings)	Loads MainSettings into this class. This control cannot function without a reference to this

Important API (Private Methods)

Method	Description
CommandType ConvertStringToCommand(string command)	Automatically converts a string into a CommandType
ContextType ConvertStringToContext(string context)	Automatically converts a string into a ContextType
TimeRangeKeywordsType ConvertStringToTimeRangeKeyword(string rangeKeyword)	Automatically converts a string into a TimeRangeKeywordType
TimeRangeType ConvertStringToTimeRange(string timeRange)	Automatically converts a string into a TimeRangeType
LoadCommandList()	Automatically load all CommandType from CustomDictionary

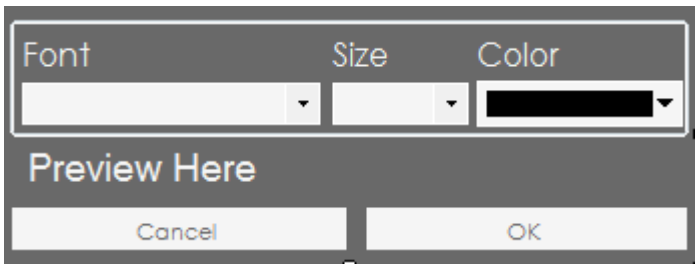
LoadContextList()	Automatically load all ContextType from CustomDictionary
LoadTimeKeywordRangeList()	Automatically load all TimeRangeKeywordsType from CustomDictionary
LoadTimeRangeList()	Automatically load all TimeRangeType from CustomDictionary
ClearSelectedCommands()	Clears commands from listedFlexiCommands
ShowUserInputBox()	Shows the UserInputBox for user to add a new flexi command
UpdateFlexiCommandList()	Updated listedFlexiCommands with all the latest flexiCommands from the selected item
UpdateTimeRangeUI()	Updates the rangeController with the modified time ranges
AddFlexiCommandToSettings(string flexiCommand)	Adds a flexiCommand by calling the function in settings
RemoveFlexiCommandToSettings(string flexiCommand)	Removes the selected flexiCommand by calling the function in settings
UpdateDescription()	Updates the description of the selected flexiCommand to descriptionLabel and descriptionTitle
UpdateTimeRangeDescription()	Updates the description of time ranges
UpdateSchedulePostponeLabel()	Updates description of schedule and postpone default time ranges
UpdateTabDescription()	Updates description of the selected tabs in flexiCommandTab

3.1.4. Custom Message Boxes

Custom Message Boxes contain custom built pop-ups you can call to get display alerts, change fonts, or get user input. They are designed to fit the style of ToDo++, and are static classes that can be called at any place.

FontDialogBox

Summary
Get and Set Font Size, Color and name.

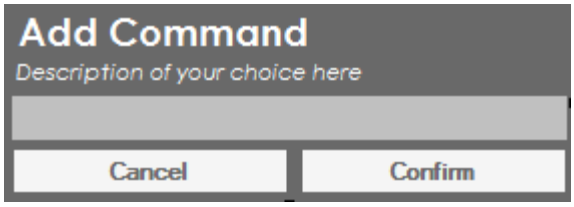


Static Methods

Method	Description
InitializeOptions(string font, int size, Color color)	Set the initial options so the preview label is displayed with these settings
int GetSize()	Gets size selected by user
string GetFont()	Gets Font selected by user
Color GetColor()	Gets Color Selected by user
bool ConfirmHit()	Checks if the Okay Button was Hit or Not
Show(bool font, bool size, bool color)	Displays the Font Dialog Box with whichever controls that need to be enabled or disabled
OnTop(bool val)	Set this to be on top of other forms

UserInputBox

Summary
A input box to get and set user input



Static Methods

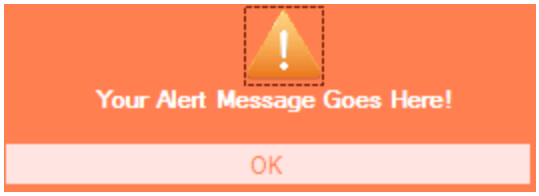
Method	Description
Show(string title,string	Shows the UserInputBox with the title and subtitle set

subTitle)	
bool ValidData()	Check if Confirm was hit or not
string GetInput()	Gets the user inpit
OnTop(bool val)	Set this to be on top of other forms

AlertBox

Summary

Shows an alert message. Alternative for MessageBox.



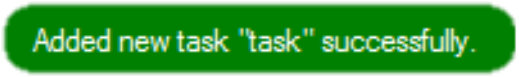
Static Methods

Method	Description
Show(string alertText)	Shows the alert with stated text
OnTop(bool val)	Set this to be on top of other forms

TinyAlert

Summary

TinyAlert is where the response from Logic is displayed. It normally flashes green when successful, orange when a warning is issued and red if a command has failed



Static Methods

Method	Description
SetUI(UI uiPass)	Pass an instance of UI in so TinyAlertView knows its position
Show(StateTinyAlert state, string response)	Flashes TinyAlert for the pre-set number of seconds with the state and response
SetLocation()	Sets the location of TinyAlert
SetTiming(int time)	Sets how long TinyAlert should stay until it fades away
DismissEarly()	Dismisses TinyAlert before it's preset timing

3.2. Settings Classes

3.2.1. SettingsInformation

Summary
This is the class that stores all settings information, and is what is actually written to file

Default Values
Default values for the various settings are modifiable here. If no settings file exists, or one is loading ToDo++ for the first time, these values will be loaded.

Adding new settings
To add new settings, you have to create a default value for your setting, add your setting to the MiscSettings struct and modify it's constructor to load the default value, and finally, create a Property for it. The settings file will automatically accommodate all new settings without any issue.

Constructor

SettingInformation()	Initializes default settings. These settings can later be modified

Settings

Variable	Description
MiscSettings misc;	Contains all Miscellaneous settings you may wish to add
Dictionary<string, CommandType> userCommandKeywords;	Contains user flexi commands for CommandType Keywords
Dictionary<string, ContextType> userContextKeywords;	Contains user flexi commands for ContextType Keywords
Dictionary<string, TimeRangeKeywordsType> userTimeRangeKeywordsType;	Contains user flexi commands for TimeRangeKeywordsType Keywords
Dictionary<string, TimeRangeType> userTimeRangeType;	Contains user flexi commands for TimeRangeType Keywords
Dictionary<TimeRangeKeywordsType, int> userTimeRangeKeywordsStartTime;	Contains the start time for TimeRangeKeywords
Dictionary<TimeRangeKeywordsType, int> userTimeRangeKeywordsEndTime;	Contains the end time for TimeRangeKeywords

Important API (Public Methods)

Method	Description
bool ContainsFlexiCommandKeyword(string userKeyword, Enum flexiCommandType)	Checks if such a flexi command already exists in the relavent type that is passed in

3.2.2. Settings

Summary
Contains an instance of SettingsInformation, and acts a wrapper, modifying the values safely

Adding Settings Methods
You can add new getters and setters for your settings added to SettingsInformation

Constructor

Settings()	Calls InitializeSettings which loads settingInfo with all default values

Important API (Private Method)

Method	Description
InitializeSettings()	Initializes settingInfo of type SettingsInformation with default values. You can load a new instance of SettingsInformation by calling the method UpdateSettings()
UpdateDictionaryPostponeSchedule()	Modifies CustomDictionary by setting the schedule and postpone length and type

Important API (Public Methods)

Method	Description
UpdateSettings(SettingInformation updatedInfo)	Completely wipes and re-updates Settings Data with the instance of SettingsInformation passed in
Other Settings	
bool GetFirstLoadStatus()	Gets whether this is the first time loading ToDo++. Once gotten, it is set to false
SetTextSize(int size)	Set default text size of Task View
Int GetTextSize()	Get the text size of Task View
SetLoadOnStartupStatus(bool status)	Sets the load on startup status
bool GetLoadOnStartupStatus()	Get the load on startup status
SetStartMinimized(bool status)	Set start minimized status
bool GetStartMinimizeStatus()	Get the start minimized status
SetStayOnTop(bool status)	Set stay on top status
bool GetStayOnTopStatus()	Get stay on top status
SetFontSelection(string font)	Set Task View font
string GetFontSelection()	Gets Task View font
Task Color Settings	
SetTaskDoneColor(Color col)	Set task done color
Color GetTaskDoneColor()	Get task done color

SetTaskMissedDeadlineColor(Color col)	Set task missed deadline color
Color GetTaskMissedDeadlineColor()	Get task missed deadline color
SetTaskNearingDeadlineColor(Color col)	Set task nearing deadline color
Color GetTaskNearingDeadlineColor()	Get task nearing deadline color
SetTaskOverColor(Color col)	Set task over color
Color GetTaskOverColor()	Get task over color
Time Range	
SetDefaultScheduleTimeLength(int length)	Set default time length for Command SCHEDULE
int GetDefaultScheduleTimeLength()	Get default time length for Command SCHEDULE
SetDefaultPostponeDurationLength(int length)	Set default duration length for Command POSTPONE
int GetDefaultPostponeDurationLength()	Get default duration length for Command POSTPONE
SetDefaultScheduleTimeLengthType(TimeRangeType timeRange)	Set default time length type (HOUR,DAY etc.) for Command SCHEDULE
TimeRangeType GetDefaultScheduleTimeLengthType()	Get default time length type (HOUR,DAY etc.) for Command SCHEDULE
SetDefaultPostponeDurationType(TimeRangeType timeRange)	Set default duration type (HOUR,DAY etc.) for Command POSTPONE
TimeRangeType GetDefaultPostponeDurationType()	Get default duration type (HOUR,DAY etc.) for Command POSTPONE

3.3. Logic & Parser Classes

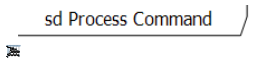
3.3.1. Logic Class

Summary

The main logic layer of this program. It is used to process and execute the user input from the UI as well as to update necessary settings. It also acts as a facade to control calls between classes.

Sequence Diagram

The following is a sequence diagram describing how Logic will operate when a command string is passed in from the UI.



Generate Operation fragment can be found in OperationGenerator class.

Detailed Description

Provides the methods: processing and executing the commands.
Provides the event handler for the UpdateSettings event.

Getter and setter methods: MainSettings	
Public method	ProcessCommand
Internal methods	SetUI, GetDefaultView
Private methods	ParseCommand, ExecuteCommand, UpdateLastDisplayedTasksListPromptUser_CreateNewTaskFile, UpdateSettings, UpdateSettingsFile

Test History

Testing for this class was done during integration testing, after v0.4.0. The tests can be found in the *Integration Tests* project under the *LogicTest* class. The tests were primarily to ensure that all the components worked correctly together and the appropriate feedback was received.

Important Attributes

Variable
Settings MainSettings

Constructor

Logic();

Important API (Public Methods)

Method	Description
Response ProcessCommand (string input);	Processes an input string command and returns the processed Response which contains the result of the operation which can be displayed to the user. Returns a Response object containing the list of tasks to be displayed and the result of the operation.

Important API (Internal Methods)

Method	Description
void SetUI (UI ui);	Sets up a UI with logic for two-way communication.
Response GetDefaultView();	Executes the DisplayDefault operation so that the Response object given by the operation can be returned. Returns the default view.

Important API (Private Methods)

Method	Description
Operation ParseCommand (String input);	Uses a CommandParser to parse the input string and returns the corresponding Operation.
Response	Executes the input operation and returns the Response returned

ExecuteCommand (Operation operation);	from the execution as feedback.

3.3.2. CommandParser

Summary

This class parses string commands into Operations, which describes the derived meaning of the user inputted string command. Parses the input command string into tokens and then generates an Operation object containing the relevant task object based on these tokens.

Detailed Description

Parses the input command string into tokens and then generates an Operation object containing the relevant task object based on these tokens.

Public method	ParseOperation
Private method	GenerateOperation

Test History

Black box testing has been employed in setting up test methods to test and validate ParseOperation method. The various following situations/cases have been tested: addition of valid deadline task, addition of invalid task, addition of valid timed event task with only one single specified start time, addition of valid timed event task with both start and end times specified. While certain test have been deprecated as the product evolved, the final tests for this class are contained in *CommandParserTest* with 4 test cases.

Constructor

CommandParser();

Important API (Public Method)

Method	Description
Operation ParseCommand (string input);	Parses a input string and returns the Operation that can be executed. Returns an operation object representing the input command.

Important API (Private Methods)

Method	Description
Operation GenerateOperation (List<Tokens> tokens);	This method uses the given list of tokens to generate a corresponding Operation. Returns the generated operation object.

3.3.3. StringParser

Summary

This class processes an input string and uses a CustomDictionary to parse them into meaningful substrings.

Detailed Description

String parsing is done by first either taking whitespaces or delimiting characters to defined substrings. Relevant substrings with the same meaning for eg. “2” “pm” are then merged as a single substring. A CustomDictionary is to derive if substrings correspond to a keyword or meaning. The substrings are returned as a list of string, each string containing a meaningful word which can be converted into Tokens by a TokenGenerator.

Public method	ParseStringIntoWords, MarkWordsAsAbsolute, UnmarkWordsAsAbsolute
Private methods	SplitStringIntoSubstrings, MergeNumericalRangeWords, AdjacentCharsAreNumerical, MergeTimeRangeWords, MergeDateAndTimeWords, MergeTimeWords, MergeWord_IfValidTime, MergeDateWords, MergeWord_IfValidAlphabeticDate, FindPositionOfDelimiters, RemoveBadIndexes

Test History

Unit testing has been employed in setting up test methods to test and validate all date and time parsing methods (MergeDateWords, MergeWord_IfValidAlphabeticDate, GenerateDateTokens etc.), including the testing of all the date and time regexes.

Invalid date inputs such as 33 Feb are currently ignored; they do not flag or call exceptions to notify the user of the erroneous date input.

StringParserTest class contains 8 unit tests for this class.

Important Attributes

Const Variable	Description
int START_INDEX	0
int END_INDEX	1
char[] delimitingCharacters	Specifies all delimiting characters ([,], (,), {, }, ‘, “)

Important API (Public Method)

Method	Description
List<string> ParseStringIntoWords (string input);	This method parses a string of words into a list of substrings determined by their meaning, by spacing, or by delimiting characters. Returns the list of tokens.
string MarkWordsAsAbsolute (string absoluteSubstr);	This method marks each and every word within the input string (as absolute) with a pair of inverted commas at the start and end of the word.

	Returns the marked string of words.
string UnwarkWordsAsAbsolute (string absoluteSubstr);	This method unmarks each and every word within the input string. The words were originally marked by a pair of inverted commas. Returns the unmarked string of words.

3.3.4. OperationGenerator

Summary

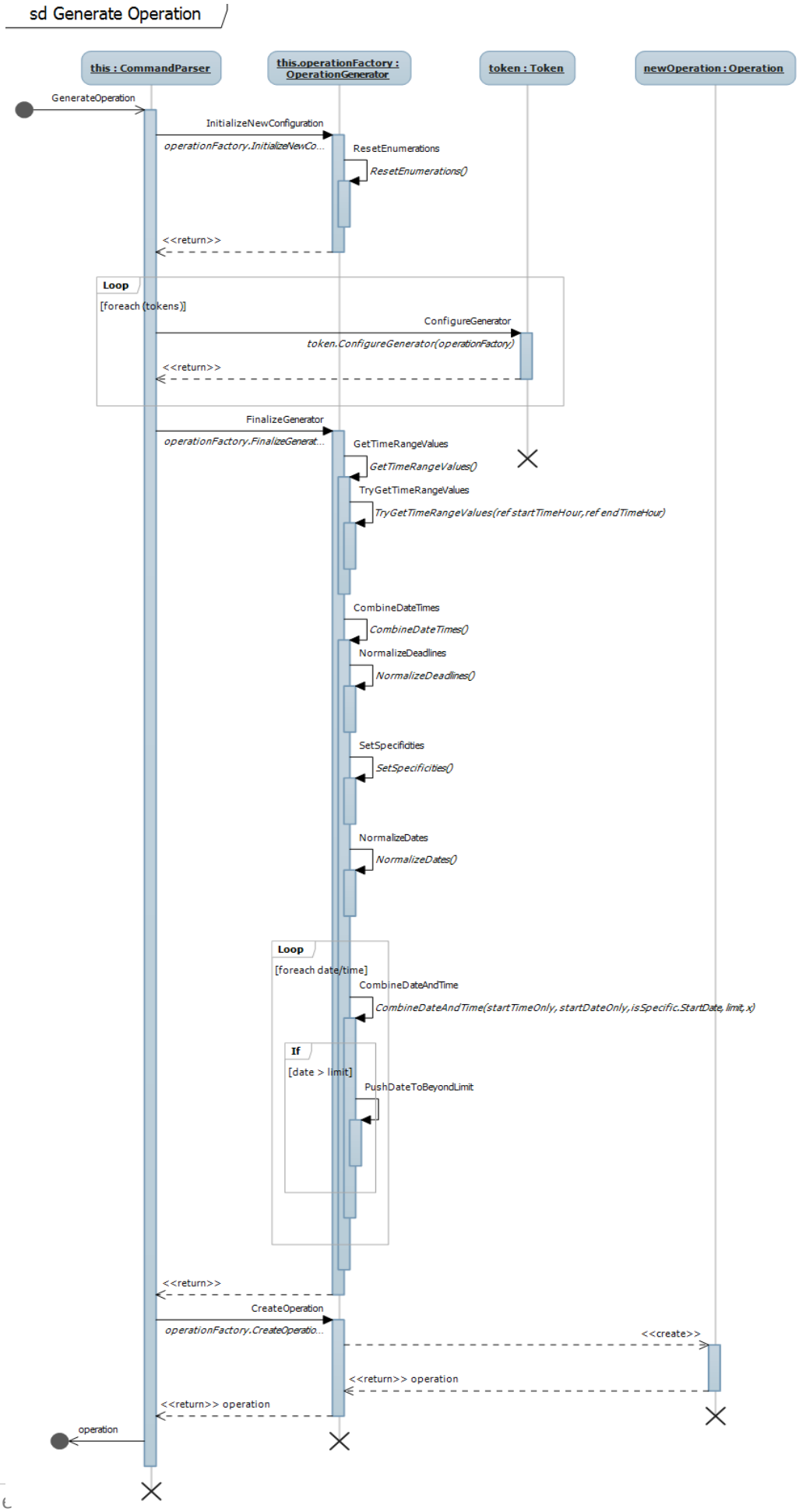
This class is a factory class for creating Operations. It must be first configured by Tokens representing the requested operation in order to be able to produce a meaningful result.

Detailed Description

The OperationGenearator can be configured by passing it into a Token's ConfigureGenerator method. By setting the relevant properties of the generator to fit the keywords or meanings they represent, the OperationGenerator can generate an appropriate Operation representing the user's desired operation. The tokens must configure the generator in the same order as their string representation's position within the input command. Once all tokens representing a single command have configured the generator, the FinalizeGenerator method must be called before the CreateOperation method which generates an operation based on the configured settings.

The following sequence diagram on the following page describes how an operation is generated.

Getter and setter methods: commandType, taskName, taskRangeIndex, isSpecific, timeRangeType, timeRangeOne, timeRangeTwo, timeRangeIndex, currentSpecifier, currentMode	
Setter methods: sortType, searchType, rangeIsAll	
Public method	CreateOperation
Internal methods	SetConditionalEndTime, SetConditionalEndDate
Private methods	ResetEnumerations, CommandIsSearchableType, FinalizeSearchTime, IsOnlyStartTimeSet, IsOnlyStartDateSet, ExtendEndSearchDate, ExtendEndMonthOrYear, ExtendEndDay, FinalizeSchedulingTime, FinalizeScheduleStartDate, GetTimeRangeValues, TryGetTimeRangeValues, IsSpecificTimeSupplied, RetrieveFinalStartAndEndTimes, IsStartTimeWithinTimeRange, IsStartAndEndTimeWithinTimeRange, CombineDateTimes, NormalizeDates, NormalizeDeadlines, SetSpecificities, CombineDateAndTime, PushDateToBeyondLimit, DateIsAmbiguous, IsDayOfWeekSet


sd Generate Operation
this : CommandParser
this.operationFactory : OperationGenerator
token : Token
newOperation : Operation
GenerateOperation
InitializeNewConfiguration
operationFactory.InitializeNewCo...
ResetEnumerations
ResetEnumerations()
<<return>>
Loop
[foreach (tokens)]
ConfigureGenerator
token.ConfigureGenerator(operationFactory)
<<return>>
FinalizeGenerator
operationFactory.FinalizeGenerat...
GetTimeRangeValues
GetTimeRangeValues()
TryGetTimeRangeValues
TryGetTimeRangeValues(ref startTimeHour, ref endTimeHour)
CombineDateTimes
CombineDateTimes()
NormalizeDeadlines
NormalizeDeadlines()
SetSpecificties
SetSpecificities()
NormalizeDates
NormalizeDates()
Loop
[foreach date/time]
CombineDateAndTime
CombineDateAndTime(startTimeOnly, startDateOnly, isSpecific.StartDate, limit, x)
If
[date > limit]
PushDateToBeyondLimit
<<return>>
CreateOperation
operationFactory.CreateOperatio...
<<create>>
<<return>> operation
<<return>> operation
operation

Test History

Both exploratory and automated unit testing have been employed since the inception of this class.

The final test cases are found in *OperationUnitTest*. 8 test cases are employed currently.

Important Attributes

Private Variables
CommandType commandType
string taskName
DateTimeSpecificity isSpecific
TimeRangeType timeRangeType
TimeRangeKeywordsType timeRangeOne
TimeRangeKeywordsType timeRangeTwo
SortType sortType
SearchType searchType
int[] taskRangeIndex
int timeRangeIndex
bool rangeIsAll
TimeSpan? startTimeOnly, endTimeOnly
DateTime? startDateOnly, endDateOnly
bool startDayOfWeekSet, endDayOfWeekSet
ContextType currentSpecifier
ContextType currentMode
DateTime? startDateTime, endDateTime
bool crossDayBoundary

Constructor

OperationGenerator();	Constructor for the generator which initializes it's settings to the default values.

Important API (Public Methods)

Method	Description
void FinalizeGenerator ();	Finalizes the generator so that it can begin generating operations with the correct time ranges.
Operation CreateOperation ();	This operation generates an operation based on how this generator has been configured. Returns the generated operation object.

Important API (Internal Methods)

Method	Description
void SetConditionalEndTime (TimeSpan Value, bool IsSpecific);	Sets the configured end time to the specified time and specificity. Moves the end time to the start time if necessary.

void SetConditionalEndDate (TimeSpan Value, bool IsSpecific);	Sets the configured end date to the specified date and specificity. Moves the end date to the start date if necessary.

3.3.5. TokenGenerator

Summary

This class is a factory class for creating Tokens. It can operate on a list of strings, each of them representing words and create a list of the requested type of Token, or all Tokens using the GenerateAllTokens method. The words must have a well-defined meaning set by the CustomDictionary; otherwise they will act as a “literal” string and be converted as such.

Detailed Description

The TokenGenerator class requires a static CustomDictionary class to process the meaning of the input words. It can generate all appropriate Tokens regardless the Tokens order or the order of execution of its *Generate* calls. However, some Tokens require a list of already generated Tokens (such as GenerateLiteralTokens) in order to not generate Tokens where they will not be meaningful.

Test History

Unit testing has been employed on this class. The tests can be found within the TokenGeneratorTest class. There are 9 test cases in this test class.

Important API (Public Methods)

Method	Description
List<Token> GenerateAllTokens (List<string> input);	This method searches an input list of strings and generates the relevant tokens representing the meaning of each string. Returns the list of matched phrases as tokens.
List<TokenCommand> GenerateCommandTokens (List<string> input);	This method searches an input list of strings against the set list of command keywords and returns a list of tokens corresponding to the matched command keywords. Returns a list of the generated command tokens.
List<Token> GenerateCommandTokens (List<string> input, List<TokenCommand> commandTokens);	This method checks an input list of strings for index range words and generates a list of tokens based on the found index range words. Returns a list of the generated index raommannge tokens.mandp
List<Token> GenerateSortTypeTokens (List<string> input, List<TokenCommand> commandTokens);	This method checks an input list of strings for sort type keywords (name or date) and generates a list of tokens based on the found sort type keywords. Returns a list of the generated sort type tokens.
List<Token> GenerateTimeRangeTokens (List<string> input, List<TokenCommand> commandTokens);	This method checks an input list of strings for time range words and generates a list of tokens based on the found time range words. Returns a list of the generated time range tokens.
List<Token> GenerateDayTokens (List<string> input);	This method searches an input list of strings against the set list of day keywords and returns a list of tokens corresponding to the matched day keywords

	Returns a list of the generated day tokens.
List<Token> GenerateDateTokens (List<string> input);	This method searches an input list of strings for all valid dates and generates a list of date tokens corresponding to all the found matched date strings using regexes. Returns a list of the generated date tokens.
List<Token> GenerateTimeTokens (List<string> input);	This method searches an input list of strings for all valid times and generates a list of time tokens corresponding to all the found matched time strings using regexes. Returns a list of the generated time tokens.
List<Token> GenerateContextTokens (List<string> input, List<Token> parsedTokens);	This method searches an input list of strings against the set list of context keywords and returns a list of tokens corresponding to the matched context keywords. Returns a list of the generated context tokens.
List<Token> GenerateLiteralTokens (List<string> input, List<Token> parsedTokens);	This method compares an input list of strings against a list of parsed Tokens, and returns a list of Tokens representing all strings which have not been been parsed as Tokens. The purpose of this method is to assign all unparsed strings as LiteralTokens. Returns a list of the generated literal tokens.

3.3.6. CustomDictionary

Summary

Contains the bulk of all keywords found in the Appendix of the user manual.

Detailed Description

This class is a dictionary containing all the keywords available and their meanings. The keywords can be modified using an UpdateSettings method.

Getter methods: commandKeywords, contextKeywords, timeRangeKeywords, timeRangeType, sortTypeKeywords, timeRangeKeywordsStartTime, timeRangeKeywordsEndTime	
Private initialization methods	InitializeCommandKeywords, InitializeContextKeywords. InitializeMonthKeywords, InitializeDateTimeKeywords, InitializeTimeRangeKeywords, InitializeSortTypeKeywords
Public comparison methods	IsToday, IsTomorrow, IsValidDate, IsValidTime, IsValidNumericDate, IsValidAlphabeticDate, IsValidMonthWord, IsTimeRange, CheckIfWordIsTimeSuffix, IsTimeRangeOverDayBoundary
Public method for Flexi-Command:	UpdateDictionary

Test History

This class was originally part of the StringParser class, and TDD was originally employed to write the methods of this class. However, as the API for the classes changed, most tests have been deprecated. The original tests can be found in the repository at rev. 122. The currently used unit tests are found in CustomDictionaryTest test class.

Constructor

CustomDictionary();	Calls all private initialization methods to initialize class attributes

Important Attributes

Variable	Description
int defaultScheduleTimeLength	Specifies the default schedule time length
TimeRangeType defaultScheduleTimeLengthType	Specifies the default schedule time length type i.e.hour, day, week etc.
int defaultPostponeDurationLength	Specifies the default postpone time length
int defaultPostponeDurationType	Specifies the default postpone time length type i.e.hour, day, week etc.
Dictionary<string, CommandType> commandKeywords	Specifies the relevant strings to be the keys for various relevant CommandType i.e. add, delete etc.
Dictionary<string, ContextType> contextKeywords	Specifies the relevant strings to be the keys for the various ContextType i.e. by, on, from etc.
Dictionary<string,	Specifies the relevant strings to be the keys for the

TimeRangeKeywordsType > timeRangeKeywords	various TimeRangeKeywordsType i.e. morning, afternoon etc.
Dictionary< string, TimeRangeType> timeRangeType	Specifies the relevant strings to be the keys for the various TimeRangeType i.e. hour, day etc.
Dictionary< TimeRangeKeywordsType, int> timeRangeKeywordsStartTime	Specifies the relevant starting hours to be the keys for the various TimeRangeType i.e. hour, day etc.
Dictionary< TimeRangeKeywordsType, int> timeRangeKeywordsEndTime	Specifies the relevant ending hours to be the keys for the various TimeRangeType i.e. hour, day etc.
Dictionary< string, int> timeSpecificKeywords	Specifies the relevant strings to be the keys for the various time specific keywords i.e. noon, midnight etc.
Dictionary<string, Month> monthKeywords	Specifies the relevant strings to be the keys for the various Month i.e. jan, january, feb, february etc.
Dictionary<string, DayOfWeek> dayKeywords	Specifies the relevant strings to be the keys for the various DayOfWeek i.e. mon, monday, tues, weekend etc.
Dictionary<string, SortType> sortTypeKeywords	Specifies the relevant strings to be the keys for the various SortType i.e. name, date_type etc.
List<string > timeSuffixes	Specifies the various time suffixes i.e. am, pm, hr
List<string > todayKeywords	Specifies the various today keywords i.e. today
List<string > tomorrowKeywords	Specifies the various tomorrow keywords i.e. tmr, tomorrow etc.
List<string > rangeAllKeywords	Specifies the various all keywords i.e. all
Regex time_24HourFormat	Find all time inputs in the 24 hour format
Regex time_12HourFormat	Find all time inputs in the 12 hour format
Regex date_numericFormat	Find all date inputs in the numeric format i.e. DD/MM/YYYY, MM/DD/YYYY
Regex date_alphabeticFormat	Find all date inputs in the numeric format i.e. DD/MMM/YYYY, MMM/DD/YYYY
Regex date_daysWithSuffixes	Find all date inputs that only consist of days with suffixes
Regex date_alphabeticMonth	Find all spelled out month inputs
Regex isNumericalRange	Find all index and indexes pair inputs
Regex isTimeRange	Find all time range inputs that consist of an index/amount and/or type

Important API (Public Methods)

Method	Description
bool IsWordTimeSuffix (string word);	Checks if a word is a time keyword and returns a boolean indicating whether it is. Returns true if the word is a time keyword, false if otherwise.
bool IsTimeRangeOverDayBoundary (string word);	Checks if a time range crosses the day boundary and returns a boolean indicating whether it does. Returns true if positive; false if otherwise.
void UpdateDictionary();	Updates the CustomDictionary keywords with new Dictionaries (for when user updates the settings).

3.4. Token Classes

3.4.1. Token

Summary

A Token is a representation of a word that is part of a user's input command. It contains the derived meaning and position within the input string, among other information.

The base class is an abstract class that cannot be instantiated. It must be derived.

Subclasses: TokenCommand, TokenDate, TokenTime, TokenDay, TokenContext, TokenLiteral

The various token objects from the different subclasses are differentiated with the attribute type.

Detailed Description

Enumerates the various token types, which have differing storage details information of tokens according to the type. All subclasses inherit the attributes position and type.

Each subclass object stores the details of a relevant task.
For example, a TokenCommand object stores the details pertaining to the command information of a task an operation is to execute.

Test History

No testing has been done for the Token class and subclasses as yet.

Important API (Internal Methods)

Method	Description
abstract void ConfigureGenerator (OperationGenerator attrb);	The base method which should be overridden by derived classes. It allows the token to configure an OperationGenerator to create an appropriate Operation.
virtual bool AcceptsContext ();	Gets a flag indicating if the token accepts a context token at the position before it. Returns true if it uses a context token; false if otherwise. Returns false by default.

3.4.2. TokenCommand : Token

Summary

Token representing a command such as add, delete, etc.

Detailed Description

Internal overriding methods	ConfigureGenerator,
Other internal methods	RequiresIndexRange, RequiresTimeRange

Constructor

TokenCommand(int position, CommandType val) : base(position)	Sets the command type and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)
bool RequiresIndexRange();	This method checks if the command is of a type that accepts index ranges i.e. delete Returns true if positive; false if otherwise.
bool RequiresTimeRange();	This method checks if the command is of a type that accepts time ranges i.e. schedule Returns true if positive; false if otherwise.

3.4.3. TokenContext : Token

Summary

Token representing a context keyword such as by, to, etc.

Detailed Description

Internal overriding methods	ConfigureGenerator, AcceptsContext

Constructor

TokenContext(int position, ContextType val) : base(position)	Sets the context type and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)
override bool AcceptsContext ();	Overriding method of the base class's base method (see above). Returns true by default.

3.4.4. TokenDate : Token

Summary

Token representing a date.

Detailed Description

Internal overriding methods	ConfigureGenerator, AcceptsContext

Constructor

TokenDate(int position, DateTime date, Specificity isSpecific) : base(position)	Sets the date value, information regarding the date specificity and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)
override bool AcceptsContext ();	Overriding method of the base class's base method (see above). Returns true by default.

3.4.5. TokenDay : Token

Summary

Token representing a day of the week.

Detailed Description

Internal overriding methods	ConfigureGenerator, AcceptsContext
Private methods	GetDateFromDay, GetDaysToAdd

Constructor

TokenDay(int position, DayOfWeek val) : base(position)	Sets the day value and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)
override bool AcceptsContext ();	Overriding method of the base class's base method (see above). Returns true by default.

3.4.6. TokenIndexRange : Token

Summary

Token representing an index range.

Detailed Description

Internal overriding methods ConfigureGenerator

Constructor

TokenIndexRange(int position, int[] val, bool isAll) : base(position)	Sets the indexes pair value, information regarding whether the keyword ‘all’ was detected and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class’s base method (see above)

3.4.7. TokenLiteral : Token

Summary

Token representing a literal string.

Detailed Description

Internal overriding methods ConfigureGenerator

Constructor

TokenLiteral(int position, string val) : base(position))	Sets the literal string value and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)

3.4.8. TokenSortType : Token

Summary

Token representing a sort type.

Detailed Description

Internal overriding methods ConfigureGenerator

Constructor

TokenLiteral(int position, SortType val) : base(position))	Sets the sort type value and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)

3.4.9. TokenTime : Token

Summary

Token representing a time.

Detailed Description

Internal overriding methods	ConfigureGenerator, AcceptsContext

Constructor

TokenTime(int position, TimeSpan val, Boolean specific_flag) : base(position)	Sets the time value, information regarding its specificity and token index within the input string.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)
override bool AcceptsContext ();	Overriding method of the base class's base method (see above). Returns true by default.

3.4.10. TokenTimeRange : Token

Summary

Token representing a time range.

Detailed Description

Internal overriding methods ConfigureGenerator

Constructor

TokenContext(int position, int val, TimeRangeType type) : base(position)	Sets the time range amount and type and token index within the input string i.e. 3 days. Time range keyword type is set to none.
TokenContext(int position, TimeRangeKeywordsType range) : base(position)	Sets the time range keyword type and token index within the input string i.e. morning. Time range amount and type are set to 0 and default.

Important API (Internal Methods)

Method	Description
override void ConfigureGenerator (OperationGenerator attrb);	Overriding method of the base class's base method (see above)

3.5. Task Classes

3.5.1. Task

A Task is an abstract representation of a user defined task containing details and information relevant to that task such as the task's name. Derived classes can inherit from the base class to represent a certain type of task that the user has created or wishes to create. The base class cannot be instantiated.

Subclasses: TaskFloating, TaskDeadline, TaskEvent

Detailed Description

All derived classes inherit the id, taskName and doneState attribute. Each derived class has to override all abstract methods and can contain additional information specific to that type of Task

For example, a TaskFloating object stores the details of a floating task that has no date and time specification while a TaskEvent stores

Test History

No testing has been done for the Task class and subclasses.

Important Attribute

Variable	Description
int id	The task's unqiue ID.
string taskName	Stores the name of the task
bool doneState	Flag indicating whether the task has been marked as done by the user or not

3.5.3. TaskFloating : Task

Summary

A task with no date or times attached to it. The simplest type of task.

Detailed Description

Public overriding methods	ToXElement, IsWithinTime, CopyDateTimes

Constructor

TaskFloating(string taskName, Boolean isDone = false, int forceID = -1) : base(taskName, isDone, forceID)	Constructor for floating tasks. The task's done state. Is set to false by default. The task's ID. Is set to -1 by default for the base constructor to generate a new ID.

Important API (Public Method)

override XElement ToXElement();	Overriding method of the base class's base method (see above)
override bool IsWithinTime(DateTime? start, DateTime? end);	Overriding method of the base class's base method (see above)
override void CopyDateTimes (ref DateTime? startTime, ref DateTime? endTime, ref DateTimeSpecificity specific);	Overriding method of the base class's base method (see above)

3.5.4. TaskDeadline : Task

Summary

A task with a single date and time representing the deadline of the task. Date and time may be ambiguous.

Detailed Description

Getter methods: isSpecific, endDateTime

Public overriding methods ToXElement, IsWithinTime, CopyDateTimes

Constructor

TaskDeadline(string taskName, DateTime endTime, Boolean isDone = false, int forceID = -1) : base(taskName, isDone, forceID)	Constructor for deadline tasks. The task's done state. Is set to false by default. The task's ID. Is set to -1 by default for the base constructor to generate a new ID.

Important API (Public Method)

override XElement ToXElement();	Overriding method of the base class's base method (see above)
override bool IsWithinTime(DateTime? start, DateTime? end);	Overriding method of the base class's base method (see above)
override void CopyDateTimes (ref DateTime? startTime, ref DateTime? endTime, ref DateTimeSpecificity specific);	Overriding method of the base class's base method (see above).
override string GetTimeString();	Overriding method of the base class's base method (see above)
override bool Postpone(TimeSpan postponeDuration);	Overriding method of the base class's base method (see above)

Important API (Private Method)

void ExtendStartSearchRange (ref DateTime startCompare);	Extends the given start search time to the appropriate start of day/month/year depending on the specificity of this task.
void ExtendEndSearchRange (ref DateTime startCompare);	Extends the given end search time to the appropriate start of day/month/year depending on the specificity of this task.

3.5.5. TaskEvent : Task

Summary

A task with one or more date and times attached to it. Represents the start and end times of the task. If there is only one date/time, it means the task does not have an end time.

Detailed Description

Getter methods: isSpecific, endDateTime, startDateTime;

Public overriding methods ToXElement, IsWithinTime, CopyDateTimes

Constructor

TaskEvent(string taskName, DateTime startTime, DateTime endTime, Boolean isDone = false, int forceID = -1) : base(taskName, isDone, forceID)	Constructor for event tasks. The task's done state. Is set to false by default. The task's ID. Is set to -1 by default for the base constructor to generate a new ID.

Important API (Public Method)

override XElement ToXElement();	Overriding method of the base class's base method (see above)
override bool IsWithinTime(DateTime? start, DateTime? end);	Overriding method of the base class's base method (see above)
override void CopyDateTimes (ref DateTime? startTime, ref DateTime? endTime, ref DateTimeSpecificity specific);	Overriding method of the base class's base method (see above).
virtual bool CanBeScheduledOver();	Overriding method of the base class's base method (see above).
override string GetTimeString();	Overriding method of the base class's base method (see above)
override bool Postpone(TimeSpan postponeDuration);	Overriding method of the base class's base method (see above)

Important API (Private Method)

void ExtendStartSearchRange (ref DateTime startCompare);	Extends the given start search time to the appropriate start of day/month/year depending on the specificity of this task.
void ExtendEndSearchRange (ref DateTime startCompare);	Extends the given end search time to the appropriate start of day/month/year depending on the specificity of this task.
bool	Checks for the specificity of the DateTimes and returns a

IsDateTimesSpecificEnough (ref TimeSpan postponeDuration));	boolean indicating if they are specific enough to postpone for the given timespan. Returns true if the task is specific enough; false if otherwise.

3.6. Operation Classes

3.6.1. Operation

Summary

Abstract class that cannot be instantiated

Subclasses: OperationAdd, OperationDelete, OperationDisplay, OperationSearch, OperationModify, OperationUndo

This class contains the necessary information representing a user's requested operation. It can be executed by providing a list of Tasks to execute the command on, as well as a Storage controller to store necessary data.

Detailed Description

Each subclass object stores the details of a relevant operation to be executed. For example, an OperationAdd object stores the details of the new task to be added in a Task object.

Getter and setter methods: StartDate, EndDate, StartTime, EndTime	
Public method	UpdateCurrentListedTasks, Execute, Undo, Redo, AllowSkipOver
Protected methods	SetMembers, AddToOperationHistory, AddTask, DeleteTask, MarkTaskAs, SearchForTasks, ExecuteBySearch, DisplaySearchResults, ExecuteByIndex, CheckIfIndexesAreValid, GenerateStandardSuccessResponse, GenerateXMLFailureResponse, SetArgumentsForSearchFeedbackString, IsValidString, HasValidTime
Private methods	FilterByTaskTime, FilterByTaskName, FilterBySearchType, TaskIsInvalid, AddToOperationHistory, ExecuteAllBySearch, ExecuteOnAll, TrySearchNonExact, AddTaskToParameters, InvokeAction

Test History

The various operation subclasses have been validated by the verification of the ParseOperation method as auxilliary storage objects used in the testing methods.

Constructor

Operation();	Initializes the static variables used by all Operations.
Operation(SortType sortType);	Initializes the neccesary variables for all Operation.

Important API (Public Method)

void UpdateCurrentListedTasks (List<Task> tasks);	Sets the currently displayed list of tasks shared by all Operations to the input list of tasks.
abstract Response Execute(List<Task> taskList,	Base method to execute this Operation. Must be overriden

Storage storageIO);	by all derived Operations. Returns a Response object indicating the result of the operation execution.
virtual Response Undo (List<Task> taskList, Storage storageIO);	Base Undo method. All undoable operations must override this method. This base method will throw an assertion if called without being overriden and debug mode is on. Returns a Response object indicating the result of the undo operation.
virtual Response Redo (List<Task> taskList, Storage storageIO);	Base Redo method. All undoable operations must override this method. This base method will throw an assertion if called without being overriden and debug mode is on. Returns a Response object indicating the result of the redo operation.
virtual bool AllowSkipOver (Response response);	Indicates whether the Operation should allow a multiple-task execution to continue if one of the tasks execute unsuccessfully. This method can be overriden to specify when this condition should be allowed. If it is not overriden, it will return false by default.

3.6.2. OperationAdd : Operation

Summary

Operation used to add a new task.

Detailed Description

Public overriding methods Execute, Undo, Redo

Constructor

OperationAdd(Task addTask, SortType sortType) : base(sortType)	Derived constructor to create an Add Operation.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the global operation history.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)

3.6.3. OperationDelete : Operation

Summary

Operation used to delete one or more tasks.

Detailed Description

Public overriding methods	Execute, Undo, Redo

Constructor

OperationDelete(string taskName, int[] indexRange, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, bool isAll, SearchType searchType, SortType sortType) : base(sortType)	This is the constructor for the Delete operation which accepts argumentsto define the way this operation will be executed. If a valid index range is specified or the isAll set to true, the operation will be carried out those corresponding indicies or all displayed tasks respectively. If search parameters are specified instead, a search operation will be carried out instead. The operation will be carried out on the search results if the isAll flag is true.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the operation history. This operation tries to delete one or more tasks using the given parameters. If an index exist, it will delete all tasks by index. If not, it will perform a search, deleting tasks immediately if the isAll flag is set.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)

3.6.4. OperationDisplayDefault : Operation

Summary

Operation used to show the default view.

Detailed Description

Public overriding method	Execute

Constructor

OperationDisplayDefault (); OperationDisplayDefault (SortType sortType) : base(sortType)

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)

3.6.5. OperationMarkAsDone : Operation

Summary

Operation used to mark one or more tasks as done.

Detailed Description

Public overriding methods Execute, Undo, Redo, AllowSkipOver

Constructor

OperationMarkAsDone(string taskName, int[] indexRange, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, bool isAll, SearchType searchType, SortType sortType) : base(sortType)	This is the constructor for the MarkAsDone operation which accepts arguments to define the way this operation will be executed. If a valid index range is specified or the isAll set to true, the operation will be carried out those corresponding indicies or all displayed tasks respectively. If search parameters are specified instead, a search operation will be carried out instead. The operation will be carried out on the search results if the isAll flag is true.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the operation history. This operation tries to mark one or more tasks as done using the given parameters. If an index exist, it will mark all tasks by index. If not, it will perform a search, marking tasks immediately if the isAll flag is set.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
virtual bool AllowSkipOver (Response response);	Indicates whether the Operation should allow a multiple-task execution to continue if one of the tasks execute unsuccessfully. This method can be overriden to specify when this condition should be allowed. If it is not overriden, it will return false by default.

3.6.6. OperationMarkAsUndone : Operation

Summary

Operation used to mark one or more task as undone

Detailed Description

Public overriding methods	Execute, Undo, Redo, AllowSkipOver

Constructor

OperationMarkAsUndone (string taskName, int[] indexRange, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, bool isAll, SearchType searchType, SortType sortType) : base(sortType)	This is the constructor for the MarkAsUndone operation which accepts arguments to define the way this operation will be executed. If a valid index range is specified or the isAll set to true, the operation will be carried out those corresponding indicies or all displayed tasks respectively. If search parameters are specified instead, a search operation will be carried out instead. The operation will be carried out on the search results if the isAll flag is true.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the operation history. This operation tries to mark one or more tasks as undone using the given parameters. If an index exist, it will mark all tasks by index. If not, it will perform a search, marking tasks immediately if the isAll flag is set.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
virtual bool AllowSkipOver (Response response);	Overriding method of the base class's base method (see above)

3.6.7. OperationModify : Operation

Summary

Operation used to modify a task's name or date or both.

Detailed Description

Public overriding methods	Execute, Undo, Redo
Private methods	MultipleTasksSelected, ModifyTask

Constructor

OperationModify(string taskName, int[] indexRange, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, bool isAll, SearchType searchType, SortType sortType) : base(sortType)	This is the constructor for the Modify operation. It will modify the task indicated by the index range to the new parameters specified by the given arguments. If an argument is left empty or null, that parameter will remain unchanged.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the operation history. Modifies the task indicated by the index range to the new parameters in this operation. If a parameter is left empty or null, that parameter will remain unchanged in the new task.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
virtual bool AllowSkipOver (Response response);	Indicates whether the Operation should allow a multiple-task execution to continue if one of the tasks execute unsuccessfully. This method can be overriden to specify when this condition should be allowed. If it is not overriden, it will return false by default.

3.6.8. OperationPostpone : Operation

Summary

Operation used to postpone one or more tasks by a certain duration.

Detailed Description

Public overriding methods	Execute, Undo, Redo, AllowSkipOver
Private methods	PostponeTask

Constructor

OperationPostpone(string taskName, int[] indexRange, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, bool isAll, SearchType searchType, TimeSpan postponeDuration, SortType sortType) : base(sortType)	This is the constructor for the Postpone operation which accepts arguments to define the way this operation will be executed. If a valid index range is specified or the isAll set to true, the operation will be carried out those corresponding indicies or all displayed tasks respectively. If search parameters are specified instead, a search operation will be carried out instead. The operation will be carried out on the search results if the isAll flag is true..

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation and adds it to the operation history.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
virtual bool AllowSkipOver (Response response);	Overriding method of the base class's base method (see above)unsuccessfully. This method can be overriden to specify when this condition should be allowed. If it is not overriden, it will return false by default.

3.6.9. OperationRedo : Operation

Summary

Operation used to redo the last undone operation.

Detailed Description

Public overriding methods	Execute
Private methods	GetLastRevertedOperation

Constructor	
OperationRedo(SortType sortType) : base(sortType)	Derived constructor to create a Redo Operation.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see , Executes the operation and adds it to the operation history.

3.6.10. OperationSchedule : Operation

Summary

Operation used to schedule a task. Can accept a time range to schedule the task in, as well as a variable duration for the task length.

Detailed Description

Public overriding methods	Execute, Undo, Redo
Private methods	SetTimeRange, IsTaskDurationWithinRange, TryScheduleTask, GetNumberOfLoops, IsTimeSlotFreeOfTasks, ScheduleTaskAtSlot

Constructor

OperationSchedule (string taskName, DateTime startTime, DateTime? endTime, DateTimeSpecificity isSpecific, int timeRangeAmount,, TimeRangeType timeRangeType, SortType sortType) : base(sortType)	This is the constructor for the Schedule operation. This operation accepts a time range and tries to schedule a task for the specified time period within the time range at the earliest possible point on execution.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see , Executes the operation and adds it to the operation history. This operation tries to schedule a task within the given parameters.
override Response Undo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)
override Response Redo (List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above)

3.6.11. OperationSearch : Operation

Summary

Operation used to search for tasks matching a given criteria.

Detailed Description

Public overriding methods	Execute

Constructor

OperationSearch (string searchString, DateTime? startTime, DateTime? endTime, DateTimeSpecificity isSpecific, SearchType searchType, SortType sortType) : base(sortType)	This is the constructor for the Search operation. This operation takes in the various parametersto define the search filters to be used when the operation is executed.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation according to this operation's parameters.

3.6.12. OperationSort : Operation

Summary

Operation used to sort the currently displayed list of tasks.

Detailed Description

Public overriding methods	Execute

Constructor

OperationSort (SortType sortType) : base(sortType)	This is the sonstructor for the Sort operation.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see above) Executes the operation according to this operation's parameters.

3.6.13. OperationUndo : Operation

Summary

Operation used to Undo an operation.

Detailed Description

Public overriding methods	Execute
Private methods	GetLastOperation

Constructor

OperationUndo(SortType sortType) : base(sortType)	Derived constructor to create a Redo Operation.

Important API (Public Method)

override Response Execute(List<Task> taskList, Storage storageIO);	Overriding method of the base class's base method (see , Executes the operation and adds it to the operation history.

3.7. Storage Class

Summary

Storage class which controls file creation, I/O. Maintains all necessary information on disk as XML files.

Detailed Description

Public methods	LoadTasksFromFile
Internal methods	CreateNewTaskFile, LoadSettingsFromFile, WriteSettingsToFile, AddTaskToFile, RemoveTaskFromFile, UpdateTask, MarkTaskAs
Private methods	ValidateTaskFile

Constructor

Storage(string taskStorageFile,, string settingsFile);	Constructs a Storage I/O handler class, creating two XML files for task storage and settings storage using the specified taskStorageFile and settingsFile as their respective filenames.

Important API (Public Methods)

Method	Description
List<Task> LoadTasksFromFile ();	Loads all tasks from the task file into a list. Returns the loaded list of tasks.

Important API (Internal Methods)

Method	Description
bool CreateNewTaskFile ();	Creates a new task XML file. Returns true if operation was successful; false if otherwise.
SettingInformation LoadSettingsFromFile ();	Load all settings from the settings file. Returns the settings information loaded from the file.
bool WriteSettingsToFile (SettingInformation settingInfo);	W rite all settings to an XML file. Returns true if operation was successful; false if otherwise.
bool AddTaskToFile (Task taskToAdd);	Appends a task to the task file. Returns true if operation was successful; false if otherwise.
bool RemoveTaskFromFile (Task taskToDelete);	Deletes a task from the task file. Returns true if operation was successful; false if otherwise.
bool UpdateTask (Task taskToUpdate);	Updates a task from the task file. Returns true if operation was successful; false if otherwise.
bool MarkTaskAs (Task taskToMarkAsDone, bool done);	Marks a task from the task file. Returns true if operation was successful; false if otherwise.