*Project Manual*

*ToDo++*

**

October 15, 2012

|  |  |  |  |
| --- | --- | --- | --- |
| myself | Jenna Tay Xiu Li | C:\Users\Jenna\Desktop\396686_10150881413801612_1380408721_n.jpg | __ |
| **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](#_Toc340406306)

[1.1. Introduction 5](#_Toc340406307)

[1.2. At a Glance 5](#_Toc340406308)

[1.3. Quick Start 6](#_Toc340406309)

[Section 2: Creating Your ToDo++ List 7](#_Toc340406310)

[1.1. Types of Tasks 7](#_Toc340406311)

[1.2. Basic Operation 7](#_Toc340406312)

[1.3. Adding a Task 7](#_Toc340406313)

[1.4. Searching & Viewing 8](#_Toc340406314)

[1.5. Updating & Modifying 8](#_Toc340406315)

[1.6. Removing Task(s) 9](#_Toc340406316)

[Section 3: Advanced Features 10](#_Toc340406317)

[3.1. FlexiCommands 10](#_Toc340406318)

[3.1.1. Custom Keywords 10](#_Toc340406319)

[3.1.2. Using Reserved Keywords 11](#_Toc340406320)

[3.2. Important Features for Power Users 12](#_Toc340406321)

[3.2.1. Sorting Tasks 12](#_Toc340406322)

[3.2.2. Marking Task(s) 12](#_Toc340406323)

[3.2.3. Postponing Task(s) 13](#_Toc340406324)

[3.2.4. Scheduling a Task 14](#_Toc340406325)

[3.2.5. Undoing a Command 14](#_Toc340406326)

[3.2.6. Redoing a Command 15](#_Toc340406327)

[3.2.7. Exiting ToDo++ 15](#_Toc340406328)

[3.2.8. Minimizing to TaskBar 15](#_Toc340406329)

[3.2.9. Autorun & Other Settings 15](#_Toc340406330)

[3.2.10. Hotkeys 15](#_Toc340406331)

[Appendix 16](#_Toc340406332)

[COMMAND KEYWORDS 16](#_Toc340406333)

[GENERAL COMMAND KEYWORDS 16](#_Toc340406334)

[QUASI-GENERAL COMMAND KEYWORDS 16](#_Toc340406335)

[SPECIFICALLY ADD COMMAND KEYWORDS 16](#_Toc340406336)

[SPECIFICALLY SCHEDULE COMMAND KEYWORDS 17](#_Toc340406337)

[ADD COMMAND DEFAULT BEHAVIOR 17](#_Toc340406338)

[SEARCH/DISPLAY COMMAND DEFAULT BEHAVIOR 17](#_Toc340406339)

[MODIFY COMMAND DEFAULT BEHAVIOR 17](#_Toc340406340)

[POSTPONE COMMAND DEFAULT BEHAVIOR 17](#_Toc340406341)

[SCHEDULE COMMAND DEFAULT BEHAVIOR 18](#_Toc340406342)

[UNDO COMMAND DEFAULT BEHAVIOR 18](#_Toc340406343)

[HOTKEYS AND KEYBOARD SHORTCUTS 18](#_Toc340406344)

# Section 1: The Basics

## 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.

## At a Glance



## 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 C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png 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

## 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.

## 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.

## 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.*

## 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.

## 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.*

## 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

## 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 |

### 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 C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > FlexiCommands.



*Adding the remove keyword for command delete*

### 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**]**

## Important Features for Power Users

### 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  2. breakfast  3. morning run  4. supper  (User) sort name  (ToDo++) 1. morning run   2. breakfast  3. lunch with colleagues  4. Supper | 11 Nov, 1:00PM 11 Nov, 5:00AM 11 Nov, 3:00AM – 4:30AM 11 Nov, 11:00PM  11 Nov, 3:00AM – 4:30AM 11 Nov, 5:00AM 11 Nov, 1:00PM 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.

### 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 |

### 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:

1. 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.
2. If a duration is not specified, the task will be automatically postponed by the default postpone duration specified in Settings C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > 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.*

### 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:

1. If there is no time slot available within the specified time range, the task will not be scheduled.
2. If the task duration is not specified, the default task duration specified in Settings C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > FlexiCommands > Schedule.
3. 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.*

### 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

### Redoing a Command

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

### 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.

### Minimizing to TaskBar

Simply click on C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\minimizeButton.png 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*

### Autorun & Other Settings

You can get ToDo++ to run automatically on Windows startup by simply checking the Settings C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > Load on Startup option. Additonally, you can also check Settings C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > 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 C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > Stay On Top.

### 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 C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > 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 C:\Users\Jenna\Desktop\Project Repo\ToDo++\Resources\gearButton.png > 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**  **CTRL+Q**  **ALT+H**  **ALT+S**  **ALT+SPACE**  **ALT+UP/DOWN**  **ALT + BACKSPACE**  **CTRL/ALT+LEFT**  **CTRL/ALT+RIGHT**  **UP/DOWN** | select all text in ionput box  exit ToDo++  togle between main window and help panel  toggle between main window and settings panel  return cursor to input box  toggle between expanded and collapsed state  delete previous word or current word till selection position  move to start of curernt word in input field  move to start of next word in input field  retrieve previous/next input |

ToDo++   
Developer Guide   
For v0.5 software



Table of Contents

[An Overview 24](#_Toc340334557)

[Where We Are Now 24](#_Toc340334558)

[Must-have Features 24](#_Toc340334559)

[Extra Feature 24](#_Toc340334560)

[Class Diagram 27](#_Toc340334561)

[User Sequence Diagram 28](#_Toc340334562)

[General Class Descriptions 29](#_Toc340334563)

[Important Attributes 29](#_Toc340334564)

[Main Class 29](#_Toc340334565)

[User Interface Classes 29](#_Toc340334566)

[Logic Class 29](#_Toc340334567)

[Parser Classes 29](#_Toc340334568)

[Token Classes 29](#_Toc340334569)

[Storage Class 30](#_Toc340334570)

[Task Classes 30](#_Toc340334571)

[Operation Classes 30](#_Toc340334572)

[User Interface Classes 31](#_Toc340334573)

[UI 31](#_Toc340334574)

[Summary 31](#_Toc340334575)

[Detailed Description 31](#_Toc340334576)

[Test History 31](#_Toc340334577)

[Important Attributes 31](#_Toc340334578)

[Constructor 32](#_Toc340334579)

[Important API (Private Methods) 32](#_Toc340334580)

[Important API (Protected Method) 32](#_Toc340334581)

[SettingsUI 33](#_Toc340334582)

[Summary 33](#_Toc340334583)

[Detailed Description 33](#_Toc340334584)

[Test History 33](#_Toc340334585)

[Important Attributes 33](#_Toc340334586)

[Important API (Public Methods) 33](#_Toc340334587)

[Settings 35](#_Toc340334588)

[Summary 35](#_Toc340334589)

[Detailed Description 35](#_Toc340334590)

[Test History 35](#_Toc340334591)

[Important API (Public Methods) 35](#_Toc340334592)

[Logic Class 36](#_Toc340334593)

[Summary 36](#_Toc340334594)

[Detailed Description 36](#_Toc340334595)

[Test History 36](#_Toc340334596)

[Important API (Public Methods) 36](#_Toc340334597)

[Parser Classes 37](#_Toc340334598)

[CommandParser 37](#_Toc340334599)

[Summary 37](#_Toc340334600)

[Detailed Description 37](#_Toc340334601)

[Test History 37](#_Toc340334602)

[Important API (Public Method) 37](#_Toc340334603)

[Important API (Private Methods) 37](#_Toc340334604)

[StringParser 39](#_Toc340334605)

[Summary 39](#_Toc340334606)

[Detailed Description 39](#_Toc340334607)

[Test History 39](#_Toc340334608)

[Important Attributes 39](#_Toc340334609)

[Constructor 40](#_Toc340334610)

[Initialization Methods 40](#_Toc340334611)

[Important API (Internal Methods) 40](#_Toc340334612)

[Important API (Private Methods) 41](#_Toc340334613)

[Token Classes 42](#_Toc340334614)

[Token 42](#_Toc340334615)

[Summary 42](#_Toc340334616)

[Detailed Description 42](#_Toc340334617)

[Test History 42](#_Toc340334618)

[Important Attributes 42](#_Toc340334619)

[TokenCommand : Token 42](#_Toc340334620)

[Constructor 42](#_Toc340334621)

[Important API (Internal Methods) 42](#_Toc340334622)

[TokenDate : Token 43](#_Toc340334623)

[Constructor 43](#_Toc340334624)

[Important API (Internal Methods) 43](#_Toc340334625)

[TokenTime : Token 43](#_Toc340334626)

[Constructor 43](#_Toc340334627)

[Important API (Internal Methods) 43](#_Toc340334628)

[TokenDay : Token 43](#_Toc340334629)

[Constructor 43](#_Toc340334630)

[Important API (Internal Methods) 43](#_Toc340334631)

[TokenContext : Token 43](#_Toc340334632)

[Constructor 43](#_Toc340334633)

[Important API (Internal Methods) 44](#_Toc340334634)

[TokenLiteral : Token 44](#_Toc340334635)

[Constructor 44](#_Toc340334636)

[Important API (Internal Methods) 44](#_Toc340334637)

[Storage Class 45](#_Toc340334638)

[Summary 45](#_Toc340334639)

[Detailed Description 45](#_Toc340334640)

[Test History 45](#_Toc340334641)

[Constructor 45](#_Toc340334642)

[Important API (Public Method) 45](#_Toc340334643)

[Task Classes 46](#_Toc340334644)

[Task 46](#_Toc340334645)

[Detailed Description 46](#_Toc340334646)

[Test History 46](#_Toc340334647)

[Important Attribute 46](#_Toc340334648)

[TaskFloating : Task 46](#_Toc340334649)

[Constructors 46](#_Toc340334650)

[TaskDeadline : Task 46](#_Toc340334651)

[Constructors 46](#_Toc340334652)

[TaskEvent : Task 46](#_Toc340334653)

[Constructors 46](#_Toc340334654)

[Operation Classes 48](#_Toc340334655)

[Operation 48](#_Toc340334656)

[Summary 48](#_Toc340334657)

[Detailed Description 48](#_Toc340334658)

[Test History 48](#_Toc340334659)

[OperationAdd : Operation 48](#_Toc340334660)

[Constructor 48](#_Toc340334661)

[OperationDelete : Operation 48](#_Toc340334662)

[Constructor 48](#_Toc340334663)

[OperationSearch : Operation 48](#_Toc340334664)

[Constructor 48](#_Toc340334665)

[OperationModify : Operation 48](#_Toc340334666)

[Constructor 48](#_Toc340334667)

[OperationHandler 49](#_Toc340334668)

[Summary 49](#_Toc340334669)

[Detailed Description 49](#_Toc340334670)

[Important Attributes 49](#_Toc340334671)

[Constructor 49](#_Toc340334672)

[Important API (Public Methods) 49](#_Toc340334673)

# Section 1: An Overview

## Where We Are Now

### 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

### 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

## 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.

## 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 |

### Logic Class

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

### Response Class

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

### Storage Class

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

### User Interface Classes

|  |  |
| --- | --- |
| **Class** | **Description** |
| UI | Takes in all user input Displays returned feedback |
| Menu | Displays the menu strip and handles all things related |
| SettingsInformation |  |
| SettingsUI | Handles the display of the settings menus and takes in the settings changes |
| Settings | Takes in and implements the settings |

### 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

|  |  |
| --- | --- |
| **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

|  |  |
| --- | --- |
| **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 |

# Section 3: Application Programming Interface (API)

# 3.1 UI Classes

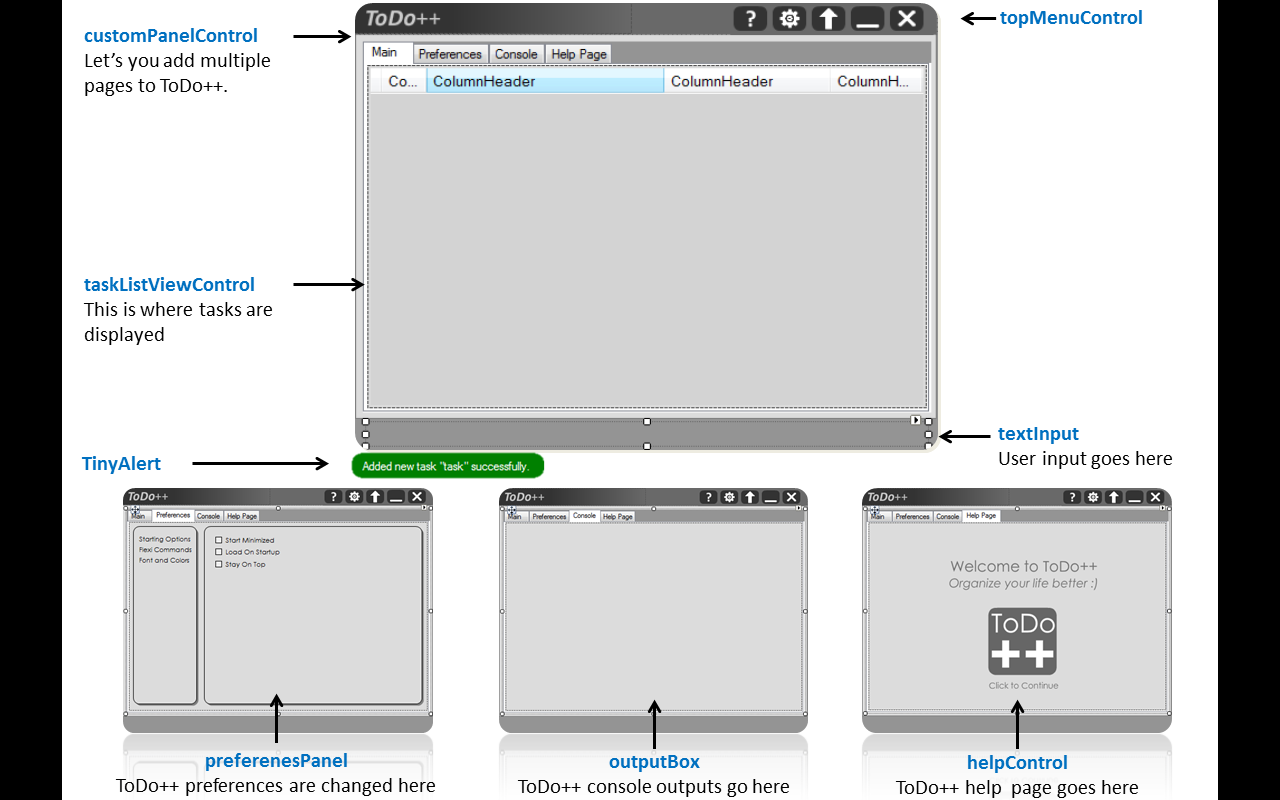
## UI

### 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 |

## 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.

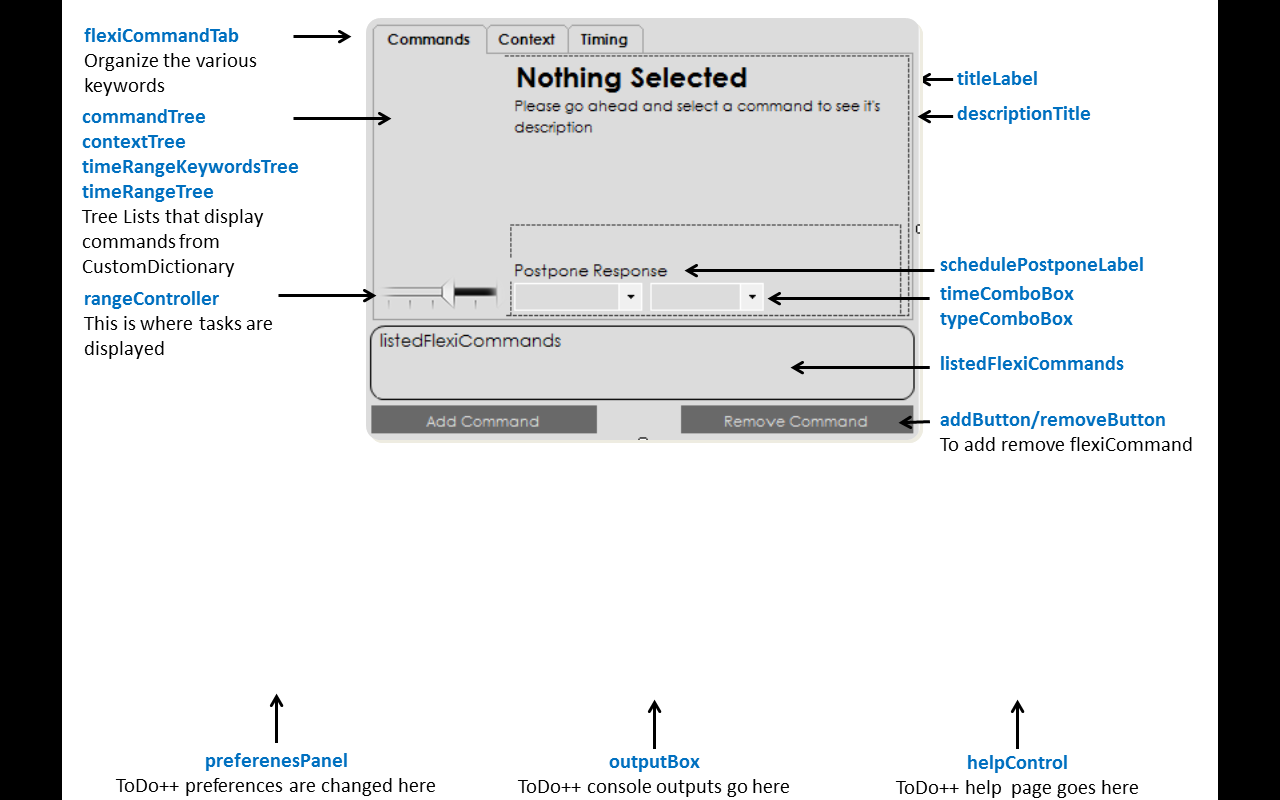
## FlexiCommandsControl

### 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 |

## 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 subTitle) | Shows the UserInputBox with the title and subtitle set |
| 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 |

## Settings Classes

### 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 |

### 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 |

# 3.2 Logic & Parser Classes

## 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 for processing and executing the commands.  
Provides the event handler for the UpdateSettings event.

Methods: ProcessCommand, ParseCommand, ExecuteCommand, PromptUser\_CreateNewTaskFile, UpdateSettings, UpdateSettingsFile, GetDefaultView, UpdateLastDisplayedTasksList

### 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.

### 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 (Private Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| Operation ParseCommand (String input); | Uses a CommandParser to parse the input string and returns the corresponding Operation. |
| Response ExecuteCommand (Operation operation); | Executes the input operation and returns the Response returned from the execution as feedback. |

## CommandParser

### Summary

This class parses string commands into Operations, which describes the derived meaning of the user inputted string command.

### Detailed Description

Methods: **ParseOperation**, GenerateOperation, CreateOperation \*, CombineDateAndTime, GenerateNewTask, GetDateFromDay, GetDaysToAdd, WarnUsersOfMultipleCommands, GetPositionsOfDelimiters, RemoveBadIndexes, SortIndexes, CompareBasedOnZerothIndex

### 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 an 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. |
| Operation CreateOperation (CommandType commandType, …); | Create operation based on derived values and whether they have been used by creating the relevant Task and adding it to an instance of the relevant Operation class |
| DateTime? CombineDateAndTime (string taskName, …); | Returns a combined DateTime consisting of date and time inputs If only the time is defined, the DateTime returned is specified to be tomorrow If only the date is defined, the default DateTime constructed is returned |
| Task GenerateNewTask (string taskName, …); | Creates and returns the new relevant Task object based on the input startTime and endTime, which are optional |
| DateTime GetDateFromDay (…); | Takes in a day of the week and returns the corresponding date depending on what the preposition is |
| int GetDaysToAdd (…); | Calculates the number of days to add to the given day of the week in order to return the next occurrence of the desired day of the week. |
| int[] GetPositionsOfDelimiters (string input); | Combs the input string for all delimiters Returns the positions of all matching delimiters |
| void RemoveBadIndexes(ref List<int[]> indexOfDelimiters); | Checks each pair of indexes removes those that overlaps with the previous pair |

## 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.

Attributes: CommandType enum, ContextType enum, Month enum, delimitingCharacters, commandKeywords Dictionary, contextKeywords Dictionary, dayKeywords Dictionary, monthKeywords Dictionary, timeSpecificKeywords Dictionary, timeGeneralKeywords Dictionary, timeSuffixes Dictionary and regexes (see Attributes)

Methods: **StringParser**, AddUserCommand, ResetCommandKeywords, FindPositionOfDelimiters, ParseStringIntoTokens, SplitStringIntoSubstrings, MergeCommandAndIndexKeywords, MergeDateAndTimeWords, MergeTimeWords, MergeWord\_IfValidTime, MergeDateWords, MergeWord\_IfValidAlphabeticDate, GenerateTokens, GenerateCommandTokens, GenerateDayTokens, GenerateDateTokens, GetDateMatch, GetMatchTagValues, ConvertMatchTagValuesToInts, ConvertToNumericMonth, RemoveSuffixesIfRequired, GenerateTimeTokens, GenerateContextTokens, GenerateLiteralTokens, AddLiteralToken, CompareByPosition, IsValidTime, IsValidNumericDate, IsValidAlphabeticDate, IsValidDate, GetTokenAtPosition

### 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

|  |  |
| --- | --- |
| **Variable** | **Description** |
| char[] delimitingCharacters | Specifies all delimiting characters ([, ], (, ), {, }, ‘, “) |
| Dictionary<string, CommandType> commandKeywords | Specifies the relevant strings to be the keys for various relevant CommandTypes i.e. add, delete etc. |
| Dictionary<string, ContextType> contextKeywords | Specifies the relevant strings to tbe keys for the various ContextTypes i.e. by, on, from etc. |
| Dictionary<string, DayOfWeek> dayKeywords | Specifies the relevant strings to tbe keys for the various DayOfWeeks i.e. mon, monday, tues, weekend etc. |
| Dictionary<string, Month> monthKeywords | Specifies the relevant strings to tbe keys for the various Months i.e. jan, january, feb, february etc. |
| 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 |

### Constructor

|  |  |
| --- | --- |
| StringParser(); | Calls the InitializeDefautKeywords method (see below) |

### 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. |

### Initialization Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| void InitializeDefaultKeywords (); | Calls all the other initialization methods |
| void InitializeCommandKeywords (); | Sets up the commandKeywords Dictionary |
| void InitializeDateTimeKeywords (); | Sets up the dayKeywords Dictionary Creates String Lists storing time specific keywords, time general keywords and time suffixes |
| void InitializeMonthKeywords (); | Sets up the monthKeywords Dictionary |
| void InitializeContextKeywords (); | Sets up the contextKeywords Dictionary |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| void AddUserCommand (string userCommand, CommandType commandType) | Adds a user defined/customized keyword to the list of defualt operation keywords |
| List<int[]> FindPositionsOfDelimiters (string input); | Searches the input string against the set delimiters' and return the positions of the delimiters as a list of integer pairs |
| List<Token> ParseStringIntoTokens (string input. List<int[]> = null); | Parses a string of words into a list of tokens, each containing a token representing the meaning of each word or substring By inputting a list of integer pairs to mark delimiting characters, multiple words can be taken as a single absolute substring (word). |

### Important API (Private Methods)

|  |  |
| --- | --- |
| **Methods** | **Description** |
| List<string> SplitStringIntoSubstrings (string input. List<int[]> indexOfDelimiters); | Splits a string and returns a list of substrings, each containing either a word delimited by a space, or a substring delimited by positions in the parameter indexOfDelimiters |
| List<string> MergeCommandAndIndexKeywords (List<string> words); | Checks to see if the command is followed by an index and if positive, merges the command with the index |
| List<string> MergeDateAndTimeKeywords (List<string> words); | Detects and merges all the date and time words into a single string while keeping the other words separate and unmerged by calling the MergeTimeWords and MergeDateWords methods i.e. input "add", "task", "friday", "5", "pm", "28", "sept", "2012" returns "add", "task", "friday", "5pm", "28 sept 2012" |
| bool MergeWord\_IfValidTime (ref List<string> output, List<string> input, int position); | Checks if the indicated word in a list of string is part of a time phrase and if positive, merges it with the other words constituting the time phrase into one string |
| bool MergeWord\_IfValidAlphabeticDate(ref List<string> output, List<string> input, int position, …); | Checks if the indicated word in a list of string is part of an alphabetic date phrase and if positive, merges it with the other words constituting the date phrase into one string |

## 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.



### 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.

### Constructor

|  |  |
| --- | --- |
| OperationGenerator(); |  |

### 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. |

## 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 range tokens. |
| 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. |

## CustomDictionary

### Summary

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

### Detailed Description

### 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.

# 3.3 Token Classes

## 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 Attributes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| enum TokenType | Enumerates the various token types (command, date, time, day, context, literal) |
| int position | Stores the index of the token in the input List of Strings |
| TokenType type | Stores the token type |

## TokenCommand : Token

### Constructor

|  |  |
| --- | --- |
| TokenCommand(int position, CommandType val, int taskIndex = 0) : base(position) | Sets the command type, token index and task index (The task index will be the same for all tokens generated for the same task) |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| CommandType Value | Getter method retrieves the command type stored |
| int TaskIndex | Getter method retrievs the task index |

## TokenDate : Token

### Constructor

|  |  |
| --- | --- |
| TokenDate(int position, DateTime date, Boolean specific\_flag) : base(position) | Sets the date value, information regarding the date specificity and token index |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| DateTime Value | Getter method retrieves the date information stored |
| bool IsSpecific | Getter method retrieves the specificity information of the stored date |

## TokenTime : Token

### Constructor

|  |  |
| --- | --- |
| TokenTime(int position, TimeSpan val) : base(position) | Sets the time value and token index |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| TimeSpan Value | Getter method retrieves the time information stored |

## TokenDay : Token

### Constructor

|  |  |
| --- | --- |
| TokenDay(int position, DayOfWeek val) : base(position) | Sets the day value and token index |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| DayOfWeek Value | Getter method retrieves the day information stored |

## TokenContext : Token

### Constructor

|  |  |
| --- | --- |
| TokenContext(int position, ContextType val) : base(position) | Sets the context type and token index |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| ContextType Value | Getter method retrieves the context type stored |

## TokenLiteral : Token

### Constructor

|  |  |
| --- | --- |
| TokenLiteral(int position, string val) : base(position)) |  |

### Important API (Internal Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| string Value | Getter method retrieves the literal information stored |

# 3.4 Task Classes

## 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 |

## TaskFloating : Task

### Constructors

|  |  |
| --- | --- |
| TaskFloating(); | Default constructor that requires no argument |
| TaskFloating(string TaskName); | Sets name of task as TaskName |

## TaskDeadline : Task

### Constructors

|  |  |
| --- | --- |
| TaskDeadline(); | Default constructor that requires no argument |
| TaskDeadline(string TaskName, DateTime EndTime) | Sets name of task as TaskName and task deadline as EndTime. |

## TaskEvent : Task

### Constructors

|  |  |
| --- | --- |
| TaskDeadline(); | Default constructor that requires no argument |
| TaskDeadline(string TaskName, DateTime StartTime, DateTime EndTime) | Sets name of task as TaskName, starting time of the task as StartTime and ending time of the task as EndTime, thereby specified the time range of the event (timed) task |

# 3.5 Operation Classes

## Operation

### Summary

Abstract class that cannot be instantiated

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

### 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.

### 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.

## OperationAdd : Operation

### Constructor

|  |  |
| --- | --- |
| OperationAdd(Task setTask); | Sets the new task to be setTask |

## OperationDelete : Operation

### Constructor

|  |  |
| --- | --- |
| OperationDelete(int index); | Stores the index of the task to be deleted in index |

## OperationSearch : Operation

### Constructor

|  |  |
| --- | --- |
| OperationSearch(string searchString); | Sets the search string to be searchString By default, the stored search string is an empty string |

## OperationModify : Operation

### Constructor

|  |  |
| --- | --- |
| OperationModify(int Previous, Task Revised); | Stores the previous task index in oldTaskIndex and the new revised task in newTask |

# 3.6 Storage Class

### Summary

The class that handles the storage and retrieval of tasks information

### Detailed Description

Provides the method for writing of tasks information to an XML file for storage

For floating tasks, only the task name will be recorded and stored  
For deadline and event (timed) tasks, both the task name and relevant specified time information will be recorded and stored

Methods: **WriteXML**, AddTask \*, RemoveTask \*

### Test History

Exploratory testing has been employed to verify that the task and settings information can be successfully written in a fixed formatting to a standard XML file.

### Constructor

|  |  |
| --- | --- |
| Storage(String taskFileName, String settingsFileName); | Initializes the controller to operate on two files. taskFileName is the file which |

### Important API (Public Method)

|  |  |
| --- | --- |
| **Method** | **Description** |
| bool WriteXML(Task task); | Writes task information to an XML file Returns true if writing process succeeded and false if otherwise |