*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 |

Credits

ToDo++   
User Guide   
For v0.1 software



Table of Contents

[The Basics 5](#_Toc338087949)

[Introduction 5](#_Toc338087950)

[At a Glance 5](#_Toc338087951)

[Quick Start 6](#_Toc338087952)

[Creating Your ToDo++ List 7](#_Toc338087953)

[Types of Tasks 7](#_Toc338087954)

[Basic Operation 7](#_Toc338087955)

[Adding a Task 7](#_Toc338087956)

[Searching & Viewing 8](#_Toc338087957)

[Updating & Modifying 8](#_Toc338087958)

[Moving Your List 10](#_Toc338087959)

[Loading & Saving Your ToDo++ List 10](#_Toc338087960)

[Exporting Your ToDo++ List 10](#_Toc338087961)

[Advanced Features 11](#_Toc338087962)

[FlexiCommands 11](#_Toc338087963)

[Custom Keywords 11](#_Toc338087964)

[Using Reserved Keywords 11](#_Toc338087965)

[Important Features for Power Users 12](#_Toc338087966)

[Undoing a Command 12](#_Toc338087967)

[Minimizing to TaskBar 12](#_Toc338087968)

[Autorun 12](#_Toc338087969)

[Hotkeys 12](#_Toc338087970)

[Appendix 13](#_Toc338087971)

[COMMAND KEYWORDS 13](#_Toc338087972)

[ADD FUNCTION KEYWORDS 13](#_Toc338087973)

[SEARCH/VIEW FUNCTION KEYWORDS 14](#_Toc338087974)

[SEARCH/VIEW FUNCTION DEFAULT BEHAVIOR 14](#_Toc338087975)

[MODIFY FUNCTION KEYWORDS 14](#_Toc338087976)

[MODIFY FUNCTION DEFAULT BEHAVIOR 14](#_Toc338087977)

[DELETE FUNCTION KEYWORDS 14](#_Toc338087978)

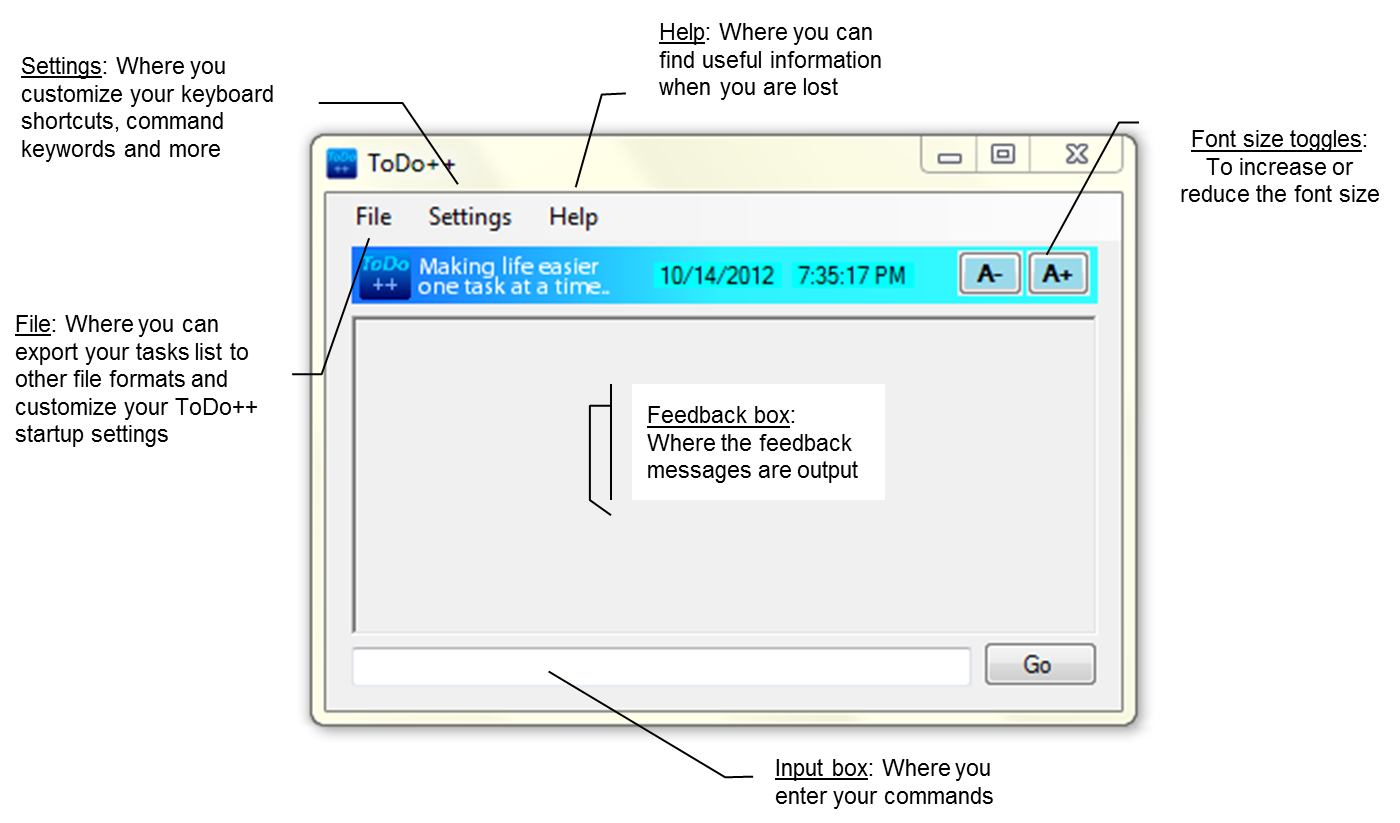
[DELETE FUNCTION DEFAULT BEHAVIOR 15](#_Toc338087979)

# 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. All you have to do to start creating your list is to just start typing!

* 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 page), 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 *4.1.* *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.

# 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 *4.1.* *FlexiCommands*.

For a complete list of default keywords and more details on how to use bracketing, please see section *4.1.3 Using Reserved Keywords*.

You can quickly undo a wrongly input command using the “undo” command. For more information, please see section *4.2.1 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 check todo++ in 2 hours  eg. add max birthday 4pm tomorrow  eg. add team meting 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 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 for and display all or some of the tasks scheduled on, before or, after a specific day or date | Enter “display [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 |
| Search for and display tasks by task names or descriptions | Enter “display [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. modify visit the vet Sunday eg, *(User)* modify buy car tomorrow  *(ToDo++)* 1. buy toy car on sunday  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 a Task**

Using the default delete or -remove 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 (as above).

|  |  |
| --- | --- |
| Deleting a single task | Enter “delete/-remove [task name/index]” eg. delete 3 eg, *(User)* delete milk  *(ToDo++)* 1. milk Steven for more info  2. buy more milk  *(User)* delete 2 |
| Deleting all or some of the tasks scheduled on, before or, after a specific day/date | Enter delete/-remove [day/date] {before/after} {time}” eg. delete today eg. -remove Sunday after 1500 hours eg. delete 31 december after 10pm |

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

# Moving Your List

## Loading & Saving Your ToDo++ List

By default, when ToDo++ is run for the first time, a new list is created and saved in default.todo.xml file, which can be located in the same directory. Subsequently, whenever ToDo++ is run, this file will be automatically loaded to retrieve the list in its latest state.

Note that manual saving is not required as the file is written to immediately after each command.

## Exporting Your ToDo++ List

To export and save your list in a different format such as a .txt text file, or for Google Calendar, select File > Export To option in the menu bar.

# 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 | add 3pm-5pm movie outing today |
| display tomorrow after 7pm | display after 7pm tomorrow |
| delete after 4pm 6 december | remove 6/12 after 4pm |

### Custom Keywords

ToDo++ is a to-do list made to be familiar and intuitive to the individual user. We realize that you may not want to use the default keywords for your commands and have allowed for you to set custom keywords for the various basic operations. You can find this under Settings > Custom Keywords in the menu bar.

### Using Reserved Keywords

There may be times when you wish to use a keyword within an event name. Simply enclose your event name within any of the following delimiters to do so.

* Quotation marks
  + add **“**add girl on facebook**”**
  + remove **‘**delete error**’**
* Brackets/Braces
  + modify 2 **{**reserve dinner for 3**}**
  + add **(**go to 2am Bar**)** 2am tomorrow
  + display **[**21st birthday**]**

As shown above, the same goes for days, dates and times. For operations where indexing is possible (i.e. modify, delete), floating numbers should also be enclosed in delimiters.

## Important Features for Power Users

### Undoing a Command

In order to undo a mistyped command, simply enter undo to revert to the state before the last entered command. Alternatively, you can also use the *ALT-Z* keyboard shortcut.

### Minimizing to TaskBar

Simply enter hide to minimize the ToDO++ window to the system tray. You will stil be able to see its icon in the taskbar notification area while it runs in the background.

To restore the ToDO++ window, simply double-click on the icon or right-click and select *Restore*. Alternatively, you can use the *ALT-Q* keyboard shortcut to toggle between the minimized and restored states.

Untitled-2*Icon minimized to notification area*

### Autorun

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

### Hotkeys

Keyboard shortcuts can be changed and customized under Settings > Hotkeys in the GUI menu ribbon. A list of default hotkeys 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, -new

**SEARCH/VIEW:** display, view, search

**MODIFY:** modify, -update

**DELETE:** delete, -remove

**UNDO:** undo

**LOAD:** -load

**EXPORT:** -export

**MINIMIZE:** -min

#### ADD FUNCTION KEYWORDS

**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]{st/nd/rd/th}{MONTH} {valid year}

*\* date must be in the future to be valid*

**TIME:** [1-12] am/pm, [0000-2359] hrs/hours

*\* spaces are optional*

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

**SIGNIFY EVENT (TIMED) TASK:** {from} [DATE/DAY/TIME] {– / to} {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]

ADD FUNCTION DEFAULT BEHAVIOR

* 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 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 event task will cause ToDo++ to assume that the numbers refer to month by default.

#### SEARCH/VIEW FUNCTION KEYWORDS

**DAY/MONTH/DATE/TIME:** *same as ADD. see above*

**DISPLAYING BY EVENT/DEADLINE TIMES:** *same as ADD. see above.*

**INDEX:** all integer numbers are reserved if and only if the previous command caused indexed results to be displayed

#### SEARCH/VIEW FUNCTION 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 FUNCTION KEYWORDS

**DAY/MONTH/DATE/TIME:** *same as ADD. see above*

**MODIFYING EVENT/DEADLINE TIMES:** *same as ADD. see above.*

**INDEX:** all integer numbers are reserved if and only if the previous command caused indexed results to be displayed

#### MODIFY FUNCTION DEFAULT BEHAVIOR

* When the modify operation is called with an input that includes a date/tday/time, it will be automatically considered to be a request for modification of the start/end times/deadline of the relevent 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.

#### DELETE FUNCTION KEYWORDS

**DAY/MONTH/DATE/TIME:** *same as ADD. see above*

**INDEX:** all integer numbers are reserved if and only if the previous command caused indexed results to be displayed

#### DELETE FUNCTION DEFAULT BEHAVIOR

* When more than one task in the list is found to match the input, the delete operation functions like a search/view operation and returns a list of matching hits.

ToDo++   
Developer Guide   
For v0.1 software



Table of Contents

[The Basics 5](#_Toc338088565)

[Introduction 5](#_Toc338088566)

[At a Glance 5](#_Toc338088567)

[Quick Start 6](#_Toc338088568)

[Creating Your ToDo++ List 7](#_Toc338088569)

[Types of Tasks 7](#_Toc338088570)

[Basic Operation 7](#_Toc338088571)

[Adding a Task 7](#_Toc338088572)

[Searching & Viewing 8](#_Toc338088573)

[Updating & Modifying 8](#_Toc338088574)

[Moving Your List 10](#_Toc338088575)

[Loading & Saving Your ToDo++ List 10](#_Toc338088576)

[Exporting Your ToDo++ List 10](#_Toc338088577)

[Advanced Features 11](#_Toc338088578)

[FlexiCommands 11](#_Toc338088579)

[Important Features for Power Users 12](#_Toc338088580)

[Appendix 13](#_Toc338088581)

[An Overview 19](#_Toc338088582)

[Where We Are Now 19](#_Toc338088583)

[Class Diagram 19](#_Toc338088584)

[User Sequences Diagram 19](#_Toc338088585)

[General Class Descriptions 20](#_Toc338088586)

[User Interface Classes 22](#_Toc338088587)

[UI 22](#_Toc338088588)

[SettingsUI 24](#_Toc338088589)

[Settings 26](#_Toc338088590)

[Logic Class 27](#_Toc338088591)

[Parser Classes 28](#_Toc338088592)

[CommandParser 28](#_Toc338088593)

[StringParser 30](#_Toc338088594)

[Token Classes 33](#_Toc338088595)

[Token 33](#_Toc338088596)

[TokenCommand : Token 33](#_Toc338088597)

[TokenDate : Token 34](#_Toc338088598)

[TokenTime : Token 34](#_Toc338088599)

[TokenDay : Token 34](#_Toc338088600)

[TokenContext : Token 34](#_Toc338088601)

[TokenLiteral : Token 35](#_Toc338088602)

[Storage Class 36](#_Toc338088603)

[Task Classes 37](#_Toc338088604)

[Task 37](#_Toc338088605)

[TaskFloating : Task 37](#_Toc338088606)

[TaskDeadline : Task 37](#_Toc338088607)

[TaskEvent : Task 37](#_Toc338088608)

[Operation Classes 39](#_Toc338088609)

[Operation 39](#_Toc338088610)

[OperationAdd : Operation 39](#_Toc338088611)

[OperationDelete : Operation 39](#_Toc338088612)

[OperationSearch : Operation 39](#_Toc338088613)

[OperationModify : Operation 39](#_Toc338088614)

[OperationHandler 40](#_Toc338088615)

# An Overview

## Where We Are Now

### Must-have Features

Most of the must-have features have been implemented and verified to be working in good order. The CRUD (create/add, read/display, update/modify, delete) tasks support is available and a simple search is also possible using the display command.

### Extra Feature

The focus good-to-have feature FlexiCommand has been more or less fully implemented but may require additional refinement to ensure full working functionality and catch exceptions. A simple GUI has also been implemented such that keyboard shortcuts/hotkeys and user customization of command keywords and shortcuts are also available.

## Class Diagram

[diagram]

## User Sequences Diagram



## General Class Descriptions

### Important Attributes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| enum CommandType | Enumerates the various command types (add, delete, display, sort, search, modify, undo, redo, invalid) |
| enum ContextType | Enumerates the various context types (starttime, endtime, deadline, current, next, following) |
| enum Month | Enumerates the 12 months, starting from index 1 (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec) |

### Main Class

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

### User Interface Classes

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

|  |  |
| --- | --- |
| **Class** | **Description** |
| CommandParser | Takes in and parses command string Returns an Operation object |
| StringParser | Takes in and parses the command string into tokens Returns a List of generated Tokens |

### 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 |
| TokenDate : Token | Stores the details pertaining to the date information of a task |
| TokenTime : Token | Stores the details pertaining to the time information of a task |
| TokenDay : Token | Stores the details pertaining to the day information of a task |
| TokenContext : Token | Stores the details pertaining to the context information of a task |
| TokenLiteral : Token | Stores the details pertaining to the literal information of a task |

### Storage Class

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

### Task Classes

|  |  |
| --- | --- |
| **Class** | **Description** |
| 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

|  |  |
| --- | --- |
| **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 |
| OperationDisplay : Operation \* | Stores the operation details for a display operation |
| OperationSearch : Operation | Stores the operation details for a search operation |
| OperationModify : Operation | Stores the operation details for a modify operation |
| OperationUndo : Operation | Stores the operation details for an undo operation |
| OperationHandler | Takes in an Operation object and executes it by calling the relevant methods |

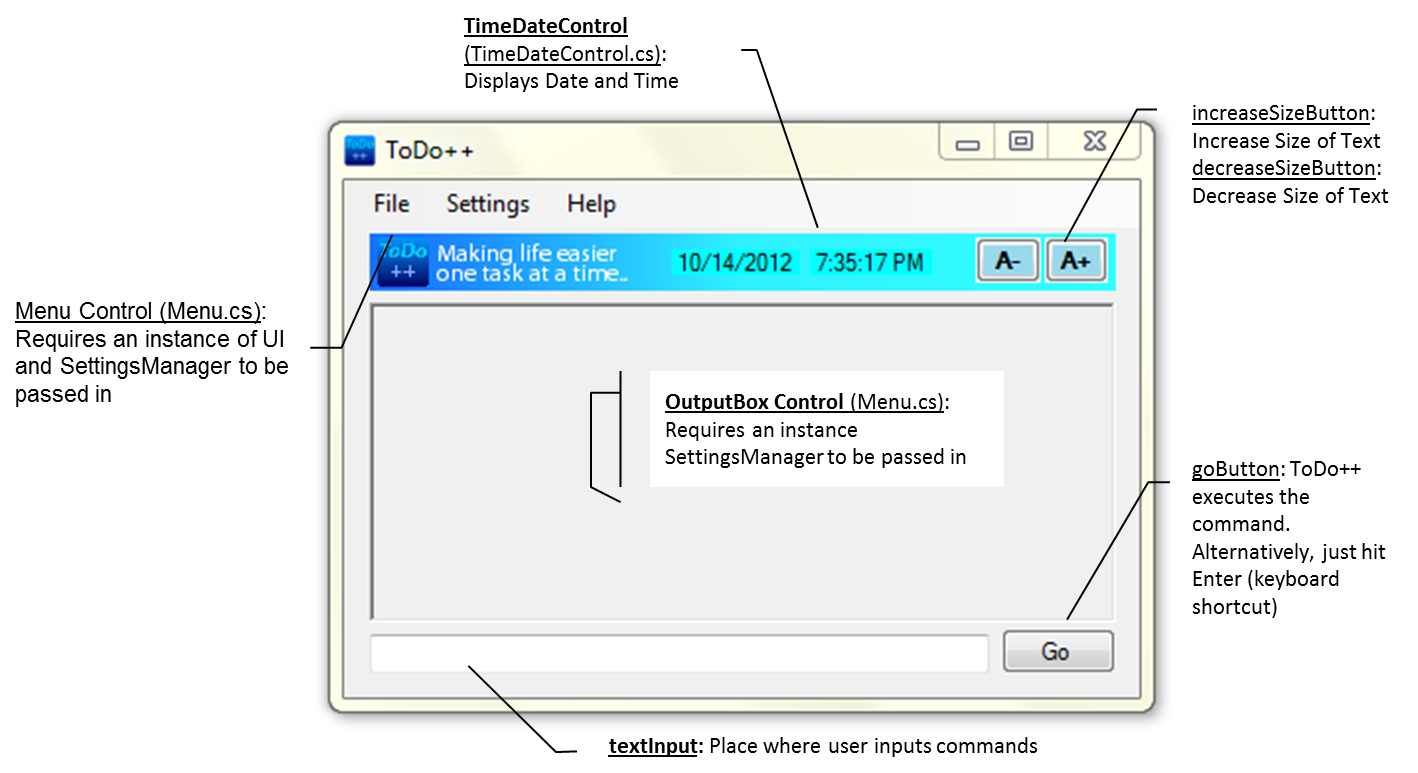
*\* not yet implemented / only partially implemented*

# User Interface Classes

## UI

### Summary

High-level class that handles all the user interaction and allows access to the customization settings menus, displays feedback and calls on the other classes to handle the command processing



### Detailed Description

Provides the method for user customization of keyboard shortcuts  
Handles all direction interaction with the user

Methods: PrepareSystemTray, PrepareSettingsManager, PrepareMenu, PrepareOutputBox, PrepareLogic, MinimiseMaximiseTray, RegisterInStartup, MinimiseToTrayWhenChecked, RegisterLoadOnStartupWhenChecked, ProcessText, *ProcessCmdKey*

### Test History

Some testing has been done with regards to the testing user interaction (testing of hotkeys, keyboard shortcuts, user customized settings etc) with little overall system testing (testing of input and feedback etc).

### Important Attributes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Hotkeys.GlobalHotKey ghk; | Global Hotkey to minimize to the system tray |
| SettingsManagers mainSettingsManager; | Instance of SettingsMangers that stores all settings data, including the user customized commands (FlexiCommands) |
| Logic logic; | Instance of Logic that handles the data structure and file operations; needed for use by text processing |

### Constructor

|  |  |
| --- | --- |
| UI(); | Calls the PrepareSystemTray, PrepareSettingsManager, PrepareMenu, PrepareOutputBox and PrepareLogic methods, which loads the minization code, initial settings, menu strip and output box respectively |

### Important API (Private Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| void PrepareSystemTray(); | Registers the hotkey (Alt+Q) to be used for minimizing/maximizing to/from System Tray |
| void PrepareSettingsManager(); | Creates an Instance of Settings Manager |
| void PrepareMenu(); | Prepares the menu bar by passing in an instance of SettingsManager and loading the relevant settings |
| void PrepareOutputBox(); | Prepares the output box by passing in an instance of SettingsManager and loading the relevant settings |
| void PrepareSettingsManager(); | Creates an Instance of Logic |
| void MinimizeMaximizeTray(); | Minimize/Maximizes ToDo++ from system tray |
| void RegisterInStartup(bool isChecked); | Detects the user setting and determines if ToDo++ is to be open at startup or not |

### Important API (Protected Method)

|  |  |
| --- | --- |
| **Method** | **Description** |
| override bool ProcessCmdKey(ref Message msg, Keys keyData) | Processes the user customized keyboard shorcut to override the default/current keyboard shortcuts Takes note that ALT+Q is reserved for minimization |

## SettingsUI

### Summary

The class that handles the input of all user settings changes/modifications regarding ToDo++ Startup options and user customized command keywords (FlexiCommands) and calls on the Settings class to implement and store the user defined settings

### Detailed Description

Methods: **LoadPersonalSettingsTab**, **LoadFlexiCommandTab**

### Test History

Testing has been done to verify that the user is able to customize and define additional command keywords and start ToDo++ on Startup minimized.



### Important Attributes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Settings settings; | Contains the instance of settings passed in |
| CommandType currentCommand; | Specifies the type of command that is currently selected |
| Settings tempSettings; | Temporary copy of the user defined settings during customization that are copied from when Apply or OK are hit |

### Important API (Public Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| void LoadPersonalSettingsTab(); | Loads the Personal Settings Tab |
| void LoadFlexiCommandTab(); | Loads the Flexi Commands Tab |

## 

## Settings

### Summary

The class that implements, stores and manages all user defined settings

### Detailed Description

Stores the following public attributes: booleans loadOnStartup and startMinimized, integer textSize, Lists of Strings customKeywords\_ADD, customKeywords\_DELETE, customKeywords\_MODIFY, customKeywords\_UNDO and customKeywords\_REDO

Provides the methods to change the font size  
Provides the methods to add and remove user defined command keywords

Methods: **SetTextSize**, **GetTextSize**, **IncreaseTextSize**, **DecreaseTextSize**, **AddCommand**, **RemoveCommand**, **GetCommandList**, **AddCommandsToStringParser**, **OpenFile**, **WriteToFile**, **CopyUpdatedCommandsFrom**, **CloneObj**

### Test History

Testing has been done to verify that the user is able to customize and define additional command keywords and start ToDo++ on Startup minimized.

### Important API (Public Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| List<string> GetCommandList(CommandType commandType); | Returns a list of all added/available user commands |
| void AddCommandsToStringParser(); | Pushes new set of FlexiCommands into the StringParser |
| void CopyUpdatedCommandsFrom(Settings passedSettings); | Copies the new user defined command keywords from a copy of the stored temporary settings |
| Settings CloneObj(); | Clones an instance of Settings (deep copy) |

# Logic Class

### Summary

High-level class that calls the other appropriate class methods to process and execute the commands as well as update the settings; where the user input is passed in and processed

### Detailed Description

Provides the methods for processing and executing the commands  
Provides the methods for calling the command parser to parse the commands and for calling the operation handler to execute the commands  
Provides the method to update settings \*

Methods: **ProcessCommand**, **UpdateSettings \***, **ExecuteCommand**, ParseCommand

### Test History

No testing has been done for the Logic class as yet.

### Important API (Public Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| string ProcessCommand (string input); | Processes (parses and executes) the commend passed in by the UI Return feedback string to the UI. |

# Parser Classes

## CommandParser

### Summary

Calls the parsing methods to make sense of the command string and create the relevant operation object

### Detailed Description

Provides the method to parse the command string into tokens and then generate an operation object containing the relevant task object based on these tokens

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.

### Important API (Public Method)

|  |  |
| --- | --- |
| **Method** | **Description** |
| Operation ParseCommand (string input); | High-level method that first calls the GetPositionsOfDelimiters method, then an instance of StringParser to parse the string into tokens and finally, the GenerateOperation method |

### Important API (Private Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| Operation GenerateOperation (List<Tokens> tokens); | Takes in a List of Tokens and generate an operation based on the tokens |
| 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

The class that does most of the actual command parsing

### Detailed Description

Provides the method for thorough command parsing and the generation of tokens, which are then subsequently used to create the actual task object in the CommandParser class  
Provides the method for addition of user customized keywords

Stores 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

White box 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 31 Feb are currently just ignored; they do not flag or call exceptions to notify the user of the erroneous date input.

### 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) |

### 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 |
| List<Token> GenerateTokens(List<string> input); | Searches an input list of strings and generates the relevant command, day, date, time, context and literal tokens of all the relevant matching strings by calling the various token generation methods GenerateCommandTokens, GenerateDayTokens, GenerateDateTokens, GenerateTimeTokens, GenerateContextTokens, GenerateLiteralTokens Returns a sorted list of the generated Tokens |

# Token Classes

## Token

### Summary

Abstract class that cannot be instantiated

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 |

# 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

Some testing has been done to verify that the tasks information can be successfully written in a fixed formatting to a standard XML file.

### Constructor

|  |  |
| --- | --- |
| Storage(); | No argument is needed |

### 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 |

# Task Classes

## Task

Abstract class that cannot be instantiated

Subclasses: TaskFloating, TaskDeadline, TaskEvent

### Detailed Description

All subclasses inherit the attribute taskname.

Each subclass object stores the details of a relevant task.  
For example, an TaskFloating object stores the details of a floating task that has no date and time specification.

### Test History

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

### Important Attribute

|  |  |
| --- | --- |
| **Variable** | **Description** |
| string taskname | Stores the name of the task |

## 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 |

# 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 |

## OperationHandler

### Summary

The class that handles the operations execution

### Detailed Description

Provides the method for operation execution given an Operation object

Stores feedback strings  
Creates and stores lastListedTask (list of Task objects), undoStack (stack of Operation objects and storageXML (Storage object)

Methods: **OperationHandler**, **Execute**, Add, Delete, DisplayAll, TrackOperation

### Important Attributes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| List<Task> lastListedTasks; | List of the most recent tasks for ready retrieval when required |
| Stack<Operation> undoStack; | Stack of Operation objects that store the latest executed operations for ready retrieval when user chooes to undo an operation |
| Stack<Operation> redoStack; | Stack of Operation objects that store the undone operations for ready retrieval when user chooes to redo an operation |
| Storage storageXML; | An instance of storage |

### Constructor

|  |  |
| --- | --- |
| OperationHandler(); | Creates a List of Tasks, Stack of Operations and an instance of Storage |

### Important API (Public Methods)

|  |  |
| --- | --- |
| **Method** | **Description** |
| string Execute(Operation operation, ref List<Task> taskList) | Processes command from UI Returns the feedback string |