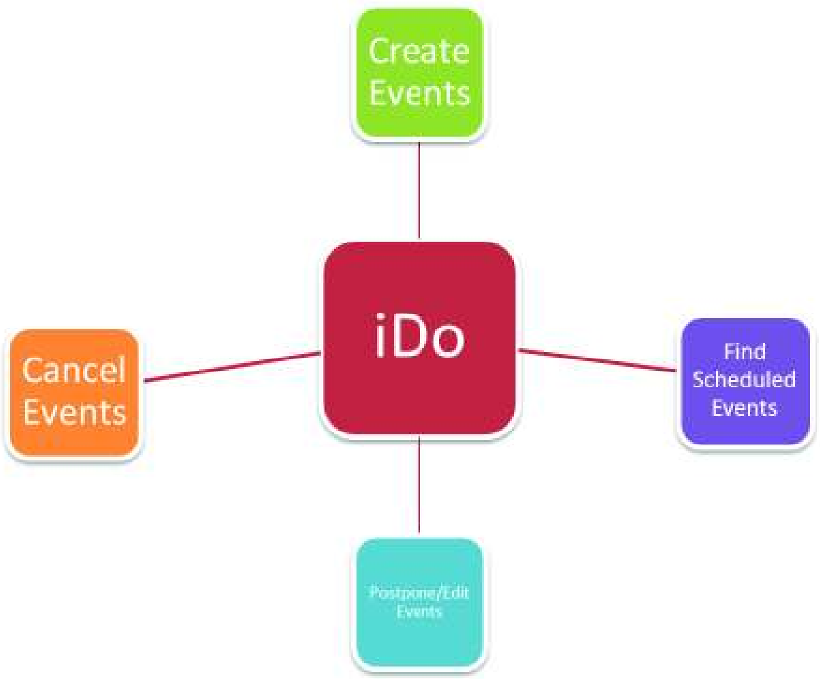
Quick Start Guide

Product Overview

So many things to do, so little time, and so hard to keep track of them all! Looking for a desktop scheduler? Old school and prefer typing everything? Then iDo is the organizer for you – it vows to be your secretary, calendar, to-do list all rolled into one. iDo's can-do attitude can get your hectic work and home life sorted for you. All you need to do is just type in your tasks through simple commands and iDo will show your schedule for the day, keep track of deadlines, and even maintain your bucket list!

What it does for you



The key features of iDo

Starting with iDo

After downloading iDo, you can run it by double clicking the application anytime you wish to. The first time you run it, you will be provided with a help screen to guide you through the application.

Quick tips to perform key operations

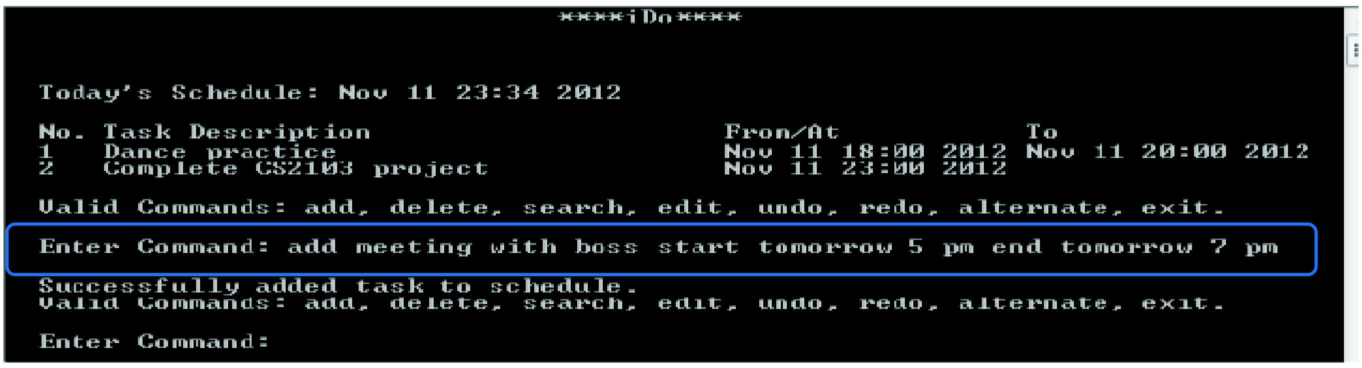
OPERATION	KEYWORD	FORMAT
Add an event	add	<description> start <start time> end <end time>
Delete event(s)	delete	<partial description>
Edit event(s)	edit	edit <partial description> [new] <new description with or without time>
Search events/ to do's	search	<partial description>
Undo the previous operation	undo	
Redo the previous operation	redo	
Create an alternate keyword	alternate	alternate <existing keyword> <new key word>
Exit the application	exit	

***partial description refers to word(s) present consecutively left to right in the description of the event**

How to use iDo?

1. To Add a task:

add <description> start <start time> end <end time>



```
                              *****iDo*****


 Today's Schedule: Nov 11 23:34 2012

 No. Task Description                        From/At           To
 1   Dance practice                          Nov 11 18:00 2012 Nov 11 20:00 2012
 2   Complete CS2103 project                 Nov 11 23:00 2012

 Valid Commands: add, delete, search, edit, undo, redo, alternate, exit.

 Enter Command: add meeting with boss start tomorrow 5 pm end tomorrow 7 pm

 Successfully added task to schedule.
 Valid Commands: add, delete, search, edit, undo, redo, alternate, exit.

 Enter Command:
```

* You can choose not to enter end time, start time or both start and end time.

** For more **add** command options refer Command Table in pg

2. To Search task(s)

search<description>

*You can search for tasks by partial description

** In order to retrieve all the events search using “ “ (space)

3. To Delete task(s)

delete <description>

In case of multiple matches found for the description entered , you are required to enter the event numbers that you want to delete followed by “ “ (space).,

* You can delete by partial description also.

** In case of only one match found, that task will be deleted automatically.

4. To Edit task(s)

edit <description> [new] <description> start <start time> end <end time>

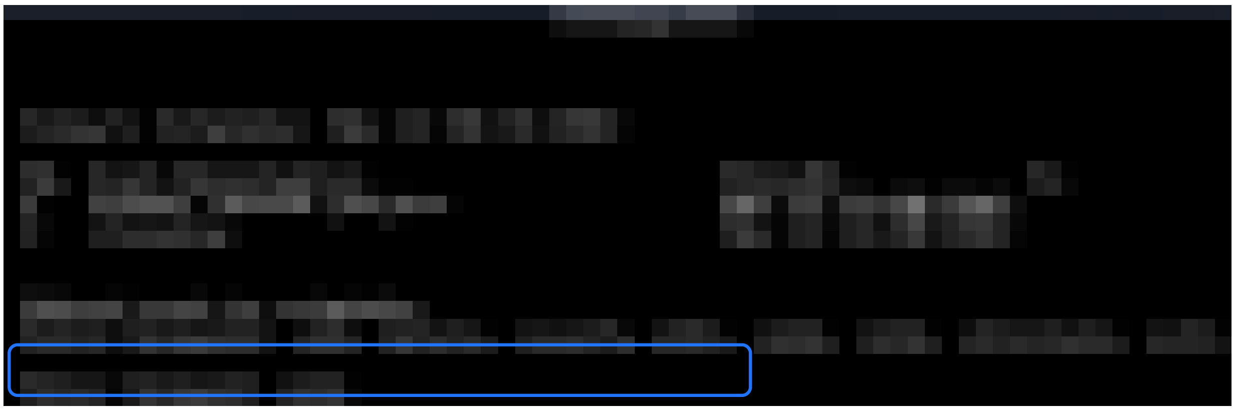
*In case multiple events are found matching the entered description, all the matching events will be displayed to you.
*You can choose the tasks you want to edit by using the serial numbers followed by space.

*All the tasks you enter will be edited according to what you type after your new.

** As before you can choose not to enter start time, end time or both.

5. Undo operation :

undo



* In case no operations are prior operations are performed, undo is not performed

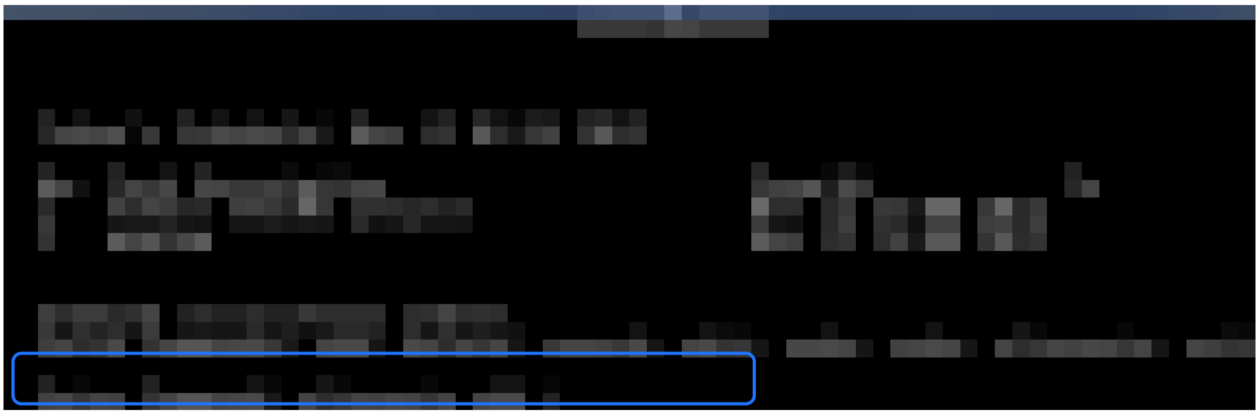
6. Redo operation

redo

*In case no undo operations are performed prior to the redo operation, redo is not performed

7. To create alternate keywords :

alternate <keyword> <new keyword>

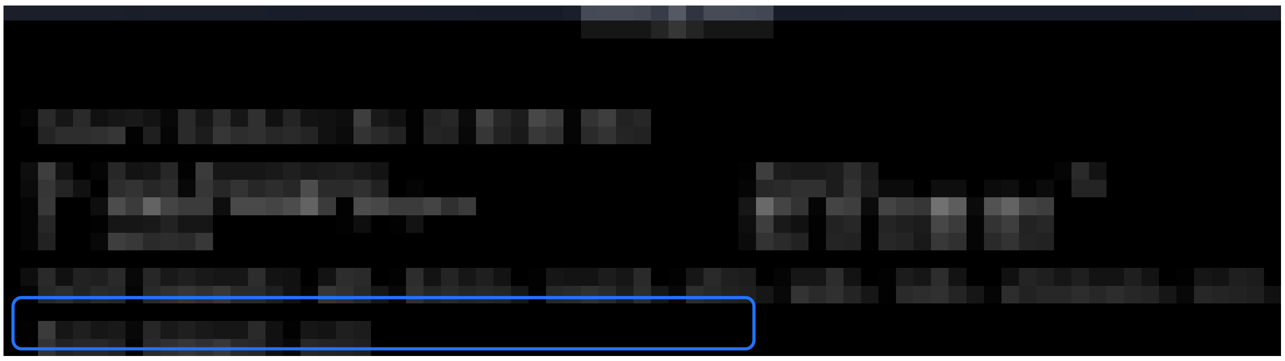


* Alternate keywords are assumed to be single words.

** You cannot enter the same alternate keyword for more than one existing keyword.

8. To exit :

exit



* Exits out of the application after saving all the change made.

POSSIBLE DATE FORMATS[2]

next[1] month hh:mm	next[1] week hh:mm	next[1] year hh:mm
coming month hh:mm	coming week hh:mm	coming year hh:mm
hh:mm[3]	next[1] day after tomorrow hh:mm	next[1] June hh:mm
next[1] Monday hh:mm	coming June hh:mm	coming Monday
dd-mm-yy hh:mm dd-mm-yyyy hh:mm	now	

1. 0 or more number of "next"s is allowed

2. In all formats if time is not given, current time is considered and the separators for day month, year, hours and minutes can be any non-alphabetical or non-numerical character other than white spaces

3. if only time is entered the current date is considered

Developer’s Guide

1. Introduction

a. Development Tools

b. Coding Standards

c. General Team Principles

d. Testing

2. Architecture

a. Major Components

b. How they interact with each other

c. Flow of Control – Activity Diagram

3. Appendix A – Class Index, Class Diagram

4. Appendix B – Logic Class Methods

5. Appendix C – CommandProcessor Methods

We assume that before you come to this guide you’ve read iDo’s user manual – or at least the quick start command table and product overview.

Introduction

1. Development Tools

- **IDE:** We use Microsoft Visual Studio 2010 and develop in C++.
- **Revision control**: Local repository is maintained using Mercurial. We use a Centralized RCS. Remote repository is on Google Code.
- In addition, we use the Issue Tracker on Google Code to: delegate tasks and fix defects, schedule code reviews, meetings.

2. Coding Standards

- We adhere to C++ standards from the following resource: https://docs.google.com/document/pub?id=1tJD2XQo3hUb0SZniswLqg0vkmccv7vGBV7_1AhTYK04&amp;embedded=true
- Major software engineering principles: OOP, Top-down design, SLAP – Single Level of Abstraction Principle, Separation of concerns, single responsibility principle (see section 2 for examples and evidence of these).
- Code is supplemented with assertions where relevant as well as extensive exception handling.

4. General Team Principles

- The process model we follow is iterative. Every iteration of product development has consisted of a sequence of activities like: revision control, testing, design & requirements analysis (during initial phases).
- We also draw relevant rules from **agile** process models such as XP and Scrum. Our weekly meetings have a similar agenda to 'Daily Scrum'
- Another example: pair programming and pseudo coding. We have found that this speeds up development, decreases chances of bugs when integrating (and otherwise); it also produces more efficient code.

3. Testing:

- On the developer testing side we do unit testing by using GTest automated testing tool.
- After weekly integration, system testing – manual/exploratory.
- We test each other's code manually; this is black box testing since we may not know others' implementation details.

Architecture & Flow of Control

The design approach that we took to lay down our system architecture is Top-Down.

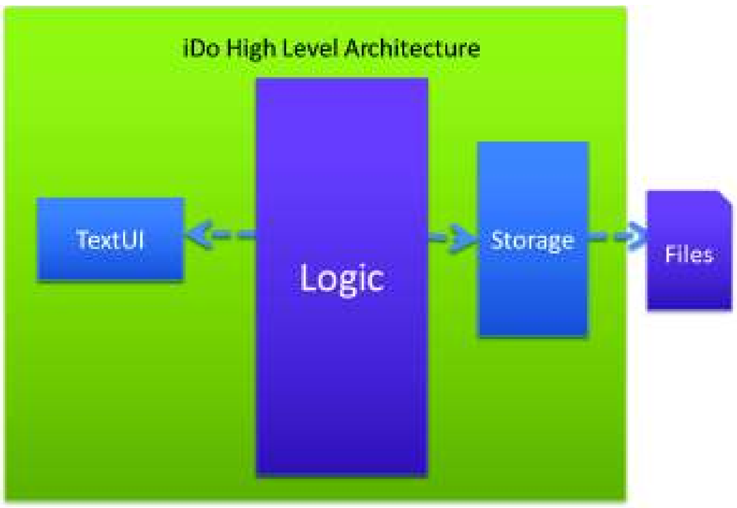
Also, our design is based on the OOP paradigm: each of these major components mentioned can be separated into specific classes and their objects.

1. Major components

TextUI: Interface between user and Logic component.

Logic: iDo's logic component (parsing user input, command execution)

Storage: Search, Sort, File I/O (retrieval of tasks, writing modified tasks to text files)

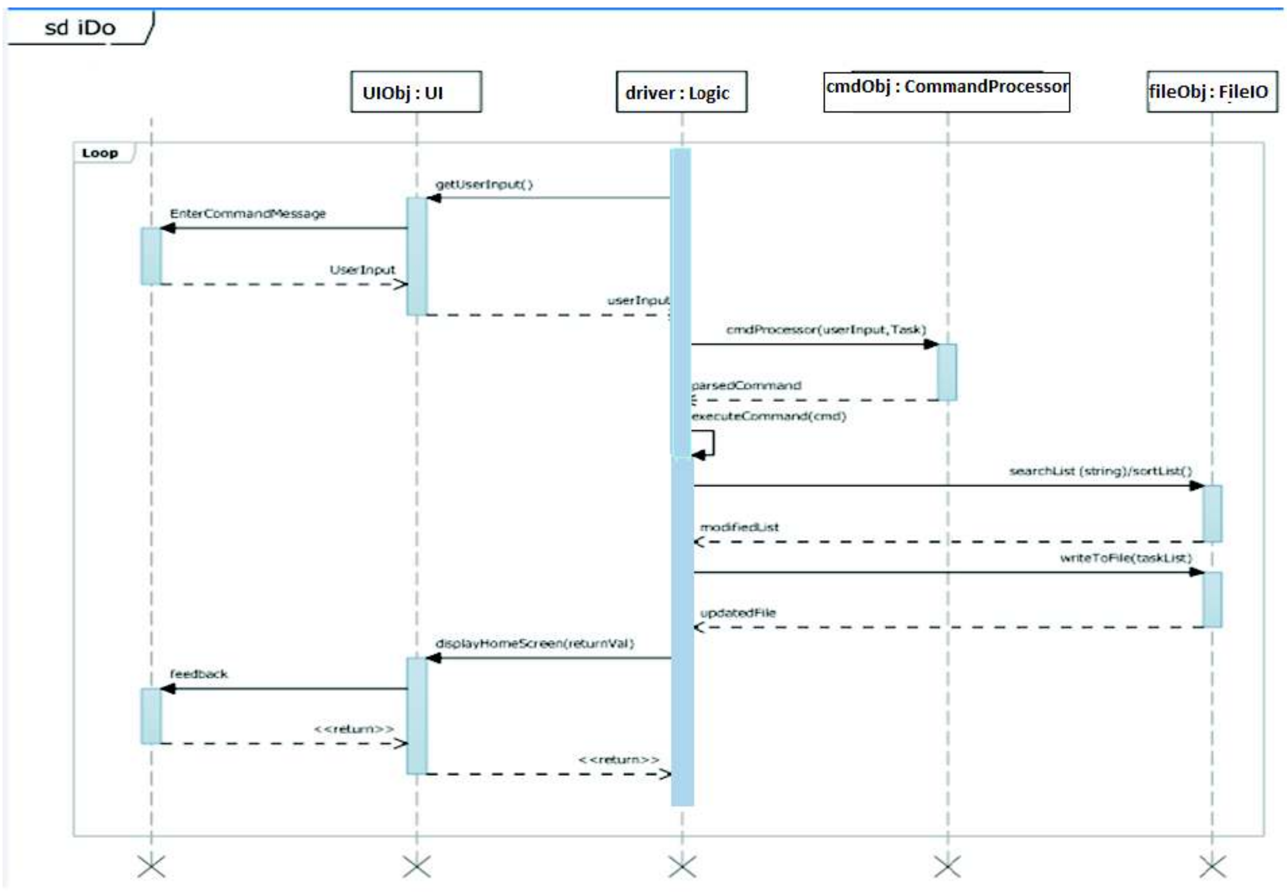


Note the dependency arrows. Logic and its component classes *need* to have the TextUI to operate; Logic depends on component TextUI. Similarly, since Logic components have objects that represent the Storage component of our system, Logic is dependent on Storage.

Logic controls the system. It talks to the user through TextUI (the external view) and coordinates with the Storage component (model). This pattern allows our design to be "low coupling". From the above you can also see how Logic component is a façade class between the TextUI and the internals (Storage).

2. How these components interact

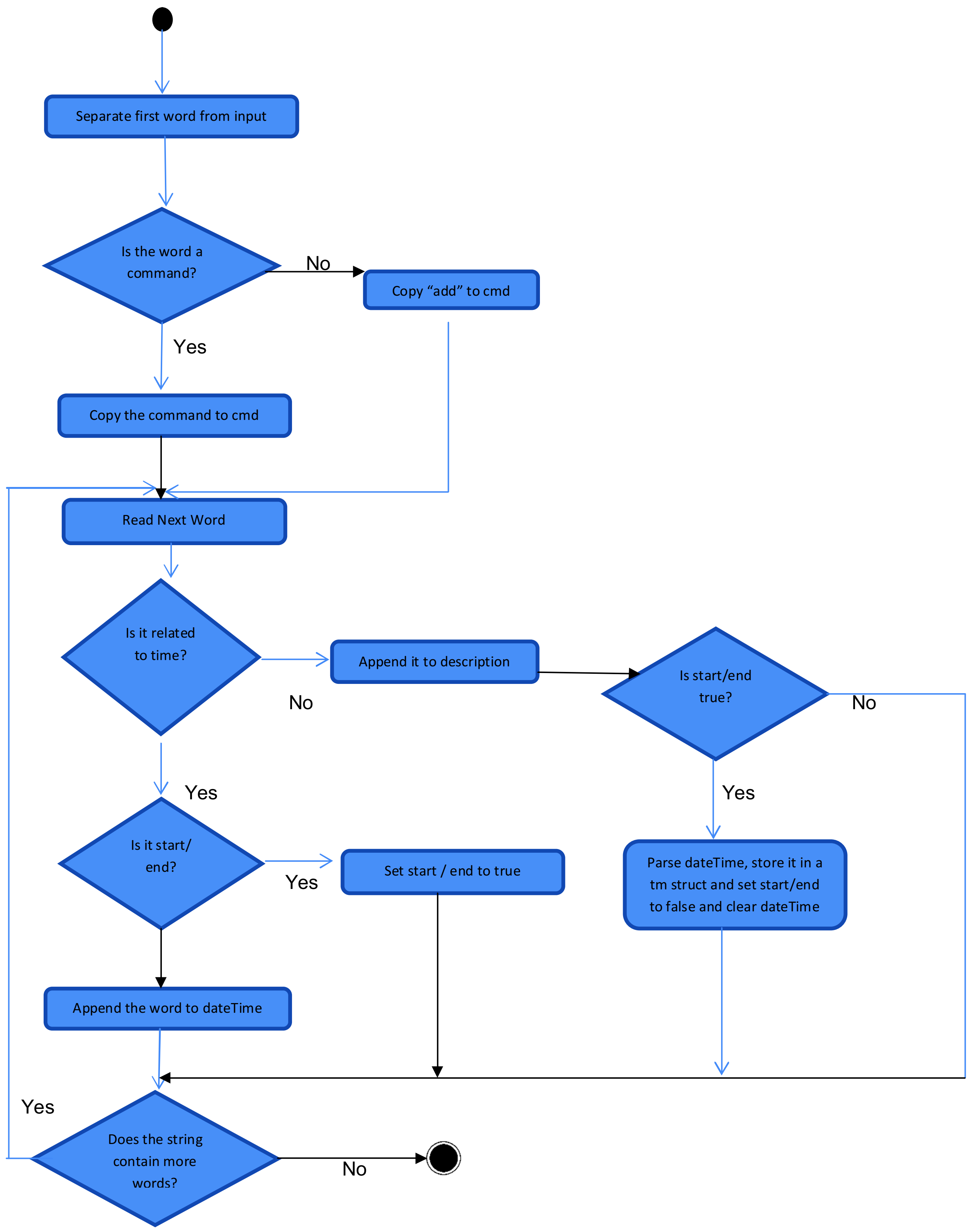
When the user enters a command, the UI simply reads this and passes it on to Logic. Logic processes this user input into parts that can be easily stored (as a Task) and changed. The command is performed and response is given to user through UI. This interaction is shown by the sequence diagram below.



3. Flow of Control

As we have seen, the general sequence and interaction between the classes in our major components. The processing that happens in cmdObj of CommandProcessor class is crucial to our product; especially since we aim to provide flexibility to user in command format. The activity diagram on the next page illustrates the steps involved in this processing.

Processing Input



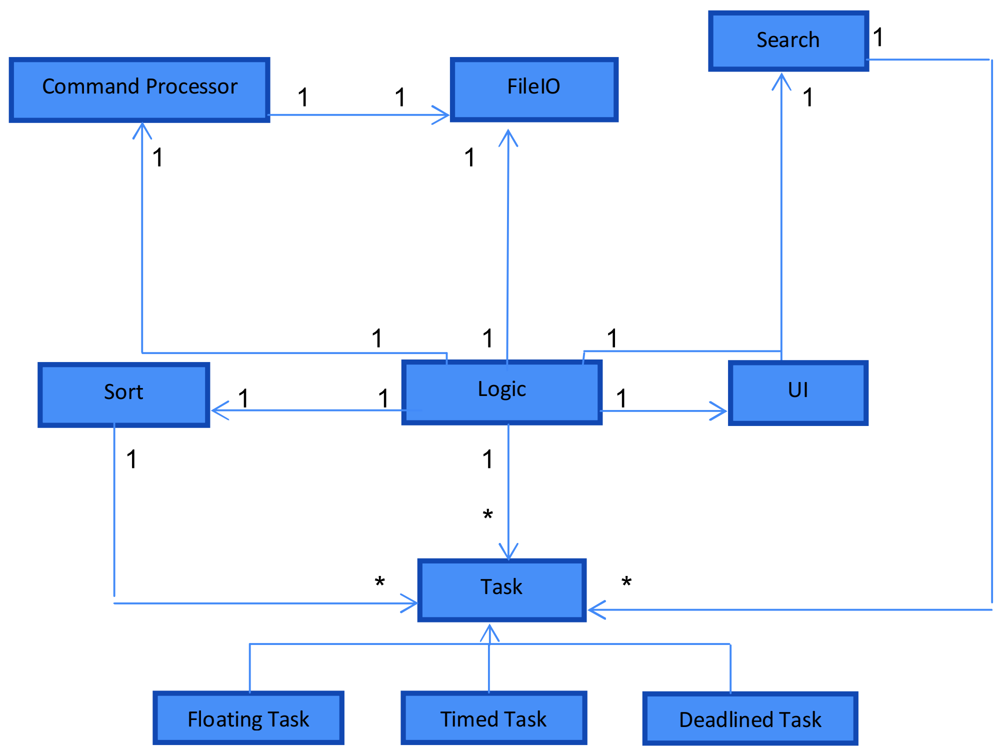
Appendix A – Class Index

Our appendices summarize all of the classes we use as well as the public methods of the classes in Logic component. This should be sufficient to get you started with developing iDo!

Class	Summary
UI	View class; displays data and reads user input but 'when' is decided by the controller.
Logic	Is the controller component (MVC pattern) and is also the façade class as described in section 2 of this guide.
CommandProcessor	Used to manipulate the string that the user enters by parsing it into the correct command type and the fields of the task to be created.
Sort	Sorts the main list based on time or descriptions in ascending or descending order.
Search	Contains the list to be searched on, the search results, and the indices of the search result in the main vector. There are 2 search functions to enable search based on description or time.
FileIO	It has a task list which stores the tasks to be read from files and the tasks to be written on to files and getters and setters for the same.
Task	Abstract class to represent a task entered by the user and its various fields such as start time, end time and description.
TimedTask	Inherits from Task; implements it specifically for Tasks with a start time and an end time, i.e. timed tasks.
FloatingTask	Inherits from Task; implementation class for Tasks with no end time and start time.
DeadlinedTask	Inherits from Task, concrete class for Tasks with either only start time OR end time, i.e. tasks with a deadline only.

Appendix A – Class Diagram

This shows the structure of our system at a lower (class) level. It describes the relations between the classes (navigability, multiplicity).



Appendix B - Logic Class Methods

Need-To-Know Public Method	Summary
logicMain( )	Uses a UI object to read and display data from/to the user. Uses a CommandProcessor object to parse the user input to give you the command type (add/delete etc.) Calls the **execute()** function and returns the feedback to the UI.
execute( )	Executes the new task as per the command entered by the user after determining command type. Based on the type, it calls the relevant function within a switch case. Returns feedback to logicMain (passes on the feedback from the function it called).
bool addTask(Task *)	Gets the existing list of tasks by using FileIO. Appends the task passed as argument to the list. Calls the sort function using the sort object. Sends the sorted list back to FileIO to be stored in the relevant file and returns control to execute.
bool search(Task*)	Returns the search results based on the user input. Takes the description entered by the user as argument. Executes search using search object Displays results using UI object. Returns control to execute.
void deleteTask(int)	Deletes a task from the task list. Obtains the index of task(s) to be deleted from the findToDelete method of Logic. Deletes the task existing in the above obtained index. Removes the task from the vector list of tasks.
void editTask(int)	Edits a task in the task list. Obtains the index of task to be edited from the findToEdit method of Logic. Deletes the task existing in the above obtained index. Adds the new task back into the relevant index.
bool createAlternateKeyword(Task *)	Task pointer passed as argument has the original keyword and its user entered alternate keyword in description. This method calls commandProcessor to again process this description, and then appends to file of alternate commands as well as to a character array of alternates. Cannot be undone/redone.
bool undoTask( )	Pops the latest command from the undoStack data member and undoes the changes made to the list of tasks during the command. The current limitation of the project is that the user can undo an add operation only if a single task description matches his previously added task description.
bool redoTask( )	Redoes the previously undone command.

Appendix C – CommandProcessor Class Methods

Need-To-Know Public Methods	Summary
string cmdProcessor (string, Task*&, Task*&)	cmdProcessor is invoked when the preliminary user command needs to be processed.
vector<int> intProcessor (string)	intProcessor processes the serial numbers of the tasks entered by the user for multiple edit or delete
void descProcessor (string, Task*&)	descProcessor processor the description entered for editing tasks
void editProcessor(string, Task*&, Task*&)	editProcessor processes only the edit command (for one line edits)
bool actualKeyWord(char userCmd[MAX_COMMAND_SIZE])	actualKeyword returns the actual keyword represented by its synonym (i.e alternate keyword)