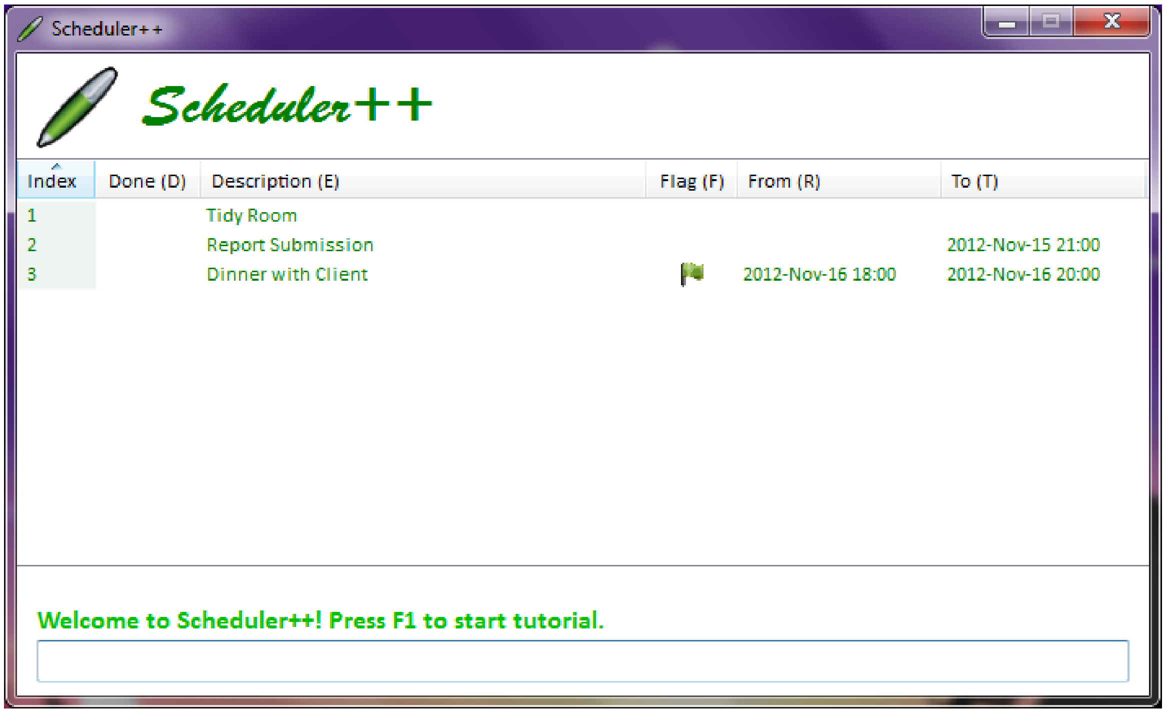
SCHEDULER++ DEVELOPER MANUAL



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	🏳	2012-Nov-16 18:00	2012-Nov-16 20:00

Welcome to Scheduler++! Press F1 to start tutorial.

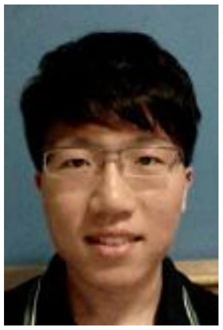
			
Lim Jia Hui, Bryan	Ling Chun Kai	Sudarsan Gopalaswami Padmanabhan	Tan Wei Jian Joash
A0086671W	A0002927L	A0092076A	A0081136M
Logic, LogicTutorial Command	Parser Architecture	Team Leader Data	GUI Writer

Table of Contents

1. INTRODUCTION

Welcome to Scheduler++ Developer's Manual. This manual offers developers insights into the makings of Scheduler++, it will explain the concepts behind the product as well as other software implementations including algorithms, testing, and supporting external libraries. This manual will equip developers with the relevant knowledge in order to start developmental work on Scheduler++.

1.1 CURRENT FEATURES

The current features that Scheduler++ supports are summarized in the following table. It should be noted that the "good-to-have" feature is Flexible-Commands.

Feature	Description	Remarks
Flexible Date Formats	DD MM YYYY	01 01 2013, 1 1 2013
	DD MMM YYYY	01 Jan 2013, 01 January 2013
	DD MMM	01 Jan
	International Holidays	Christmas
Flexible Time Formats	HH:MM	23:59
	HHMM	2359, 830
	HHam/pm	830am, 5pm, 2359pm
Flexible Date Time	Worded Description	Today, Tomorrow, Next Week
	Day of Week	Monday, mon, Tues
Short Cut Keys	Ctrl + U	Undo
	F	Flags a task when in view
	D	Done a task when in view
	Delete	Deletes a task when in view
	PageUp/PageDown	Scrolls view when in command
Tab to Toggle	Tab	Toggle command and view

1.2 CURRENT COMMANDS

The current commands that Scheduler++ supports are summarized in the following table. It should be noted that the commands are case insensitive, unless otherwise stated.

Command	Description	Associated Keywords
Add	Adds a task	from, to, by, at
Delete	Deletion of done tasks	done
Clear YES	YES (case sensitive)	
Edit	Edits a task	rename, start, end
Flag	Flags a task	
List	List overdue tasks, done tasks, flagged tasks	overdue, flag, done
Sort	Sorts alphabetically, date, flag, done	a, t, f, d (respectively)

2. SOFTWARE ARCHITECTURE

This section will briefly introduce the general architecture and the design flow of Scheduler++.

2.1 GENERAL ARCHITECTURE

Scheduler++ is coded in C++ and compiled with C++/CLI. The general architecture of Scheduler++ is illustrated in the following diagram. *A bigger diagram is available in the Appendix.*

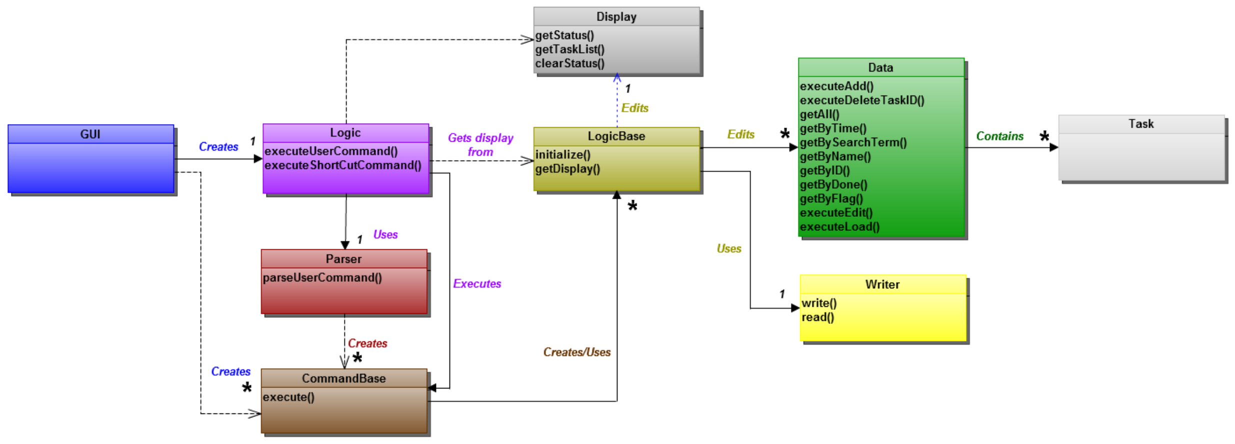


Diagram 2.1: Architecture Diagram

The diagram only displays essential classes that contain most of the software algorithms. Other helper and inherited classes are omitted. Internal architectures as well as detailed implementation will be discussed later (Refer here).

2.2 DESIGN FLOW

The design flow of Scheduler++ is illustrated in the following sequence diagram. *A bigger diagram is available in the Appendix.*

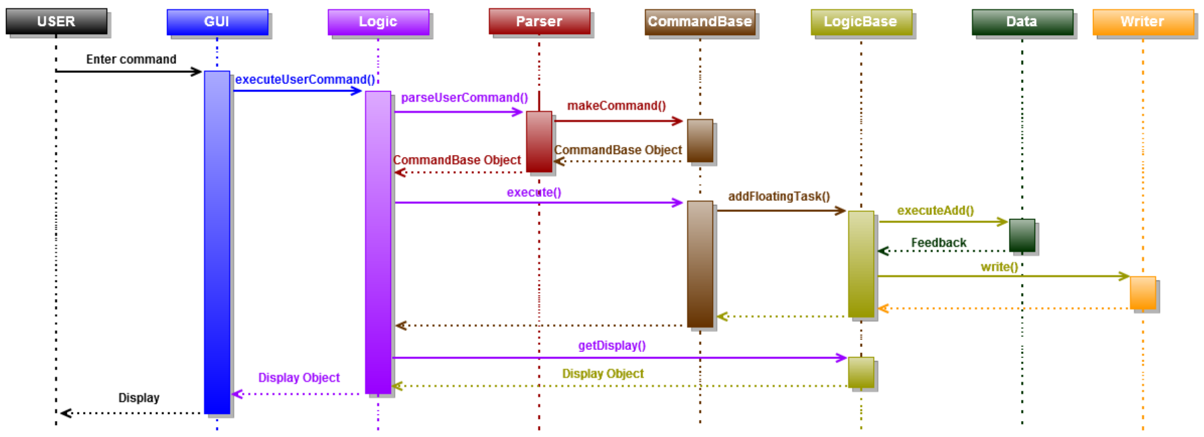


Diagram 2.2: Sequence Diagram of General Design Flow

The sequence diagram illustrates the most typical case when the user enters a command. In this example, the sequence diagram shows the execution of the ADD command. It also briefly shows the relationship between the classes.

2.3 IMPORTANT APIS

The following APIs are key to the functionality of the program

Component	API
Logic	`Display executeTextCommand( string text );`
	`Display executeShortCutCommand( CommandBase* commandClass );`
Writer	`bool write( vector<Task> taskList );`
	`bool read( vector<Task>& taskList );`
Parser	`CommandBase* parseUserCommand(string userCommand);`
Data	*Various mutators which affect a change to locally stored data*
Display	`string getStatus();`
	`vector<Task> getTaskList();`
CommandBase	`void executeCommand();`

3. CLASSES & LIBRARIES

This section will cover most classes in Scheduler++ and give a brief overview of the purpose and function of each class.

3.1 CLASSES

Graphical User Interface (GUI)

The GUI class handles all user interactions with the user. It functions as an interface for the user to input and receive output. The GUI allows enhanced interactions between user and program.

Task

The Task class contains all the necessary attributes relevant to a task. It functions as a single entity that can be passed between classes.

Item

The Item class is a delegate class. It supports the GUI in the creation of listview items.

Display

The Display class contains all the necessary information to be displayed to the user. It functions as a single entity that the Logic class returns to the GUI class.

Logic

The Logic class controls the flow of information within the architecture. It handles the decision making aspect of Scheduler++.

LogicBase

The LogicBase class contains sub-classes (e.g. LogicAdd) that contain the algorithm to perform a certain operation (add, delete, edit, list, sort, undo, etc.)

LogicTutorial

The LogicTutorial class handles commands related to the tutorial.

CommandBase

The CommandBase class contains sub-classes (e.g. CommandAdd) that determine the execution based on the parsed user input. CommandBase derived classes will execute the appropriate calls to their respective LogicBase derived classes. It functions as a single entity that the Parser class returns to the Logic Class.

Data

The Data class performs the actual operation, which is determined by the logic class. It returns appropriate feedback depending on the operation performed by the logic class.

SchedulerPPException
The SchedulerPPException class is implemented using the Open-closed principle. Developers can extend this class to implement exceptions of their own type.

ParserBase
This is the base class which forms the basis of all Parsers. More detailed information to be found in here.

Parser
This is the main class with which Logic interacts with. It is actually a derived class of *ParserBase*. For more details, refer to here.

ParserDateTime
This is a utility class used for the sole purpose of parsing date and times represented in text format. It is used extensively by the Parser classes, as well as the UI occasionally.

PairDateTime
This class is a class used to store a pair of date and times. These are primarily used to store time periods. Unlike the Boost's TimePeriod class, this class allows the creation of dates where the start date is after the end date.

Writer
The Writer class handles the storage files used for storing data. It functions as an API to the Logic class to read/write to the storage file.

LogicTester
The LogicTester class contains test cases with expected test results. It facilitates automated testing on Scheduler++.

3.2 LIBRARIES

Boost
The Boost Library is primarily used by the Parser class for its parsing methods. It is also used by other classes for its date time functions. For more information, refer to Boost.

ObjectListView
The ObjectListView Library is used by the GUI class in the creation and customization of the ListView display. For more information, refer to ObjectListView.

4. INTERNAL ARCHITECTURE

This section will cover specific concepts and algorithms used in Scheduler++ for various implementations. It will cover in greater detail how helper and inherited classes are used for certain implementations. This section also equips developers with the concepts and knowledge to expand or improve certain implementations.

4.1 GUI INTERNAL ARCHITECTURE

Controls & Tools

Before moving deeper into the GUI implementation, it is important for us to understand the primary controls and GUI tools that make up the Scheduler++ GUI. The GUI is coded using Windows Forms.

TextBox

The TextBox is where the user will enter the command.

ObjectListView

The ObjectListView is the main display for the user where task details are listed.

Status Label

The Status is where the user gets feedback on his last command.

GUI Implementation

The following diagram illustrates the GUI implementation.

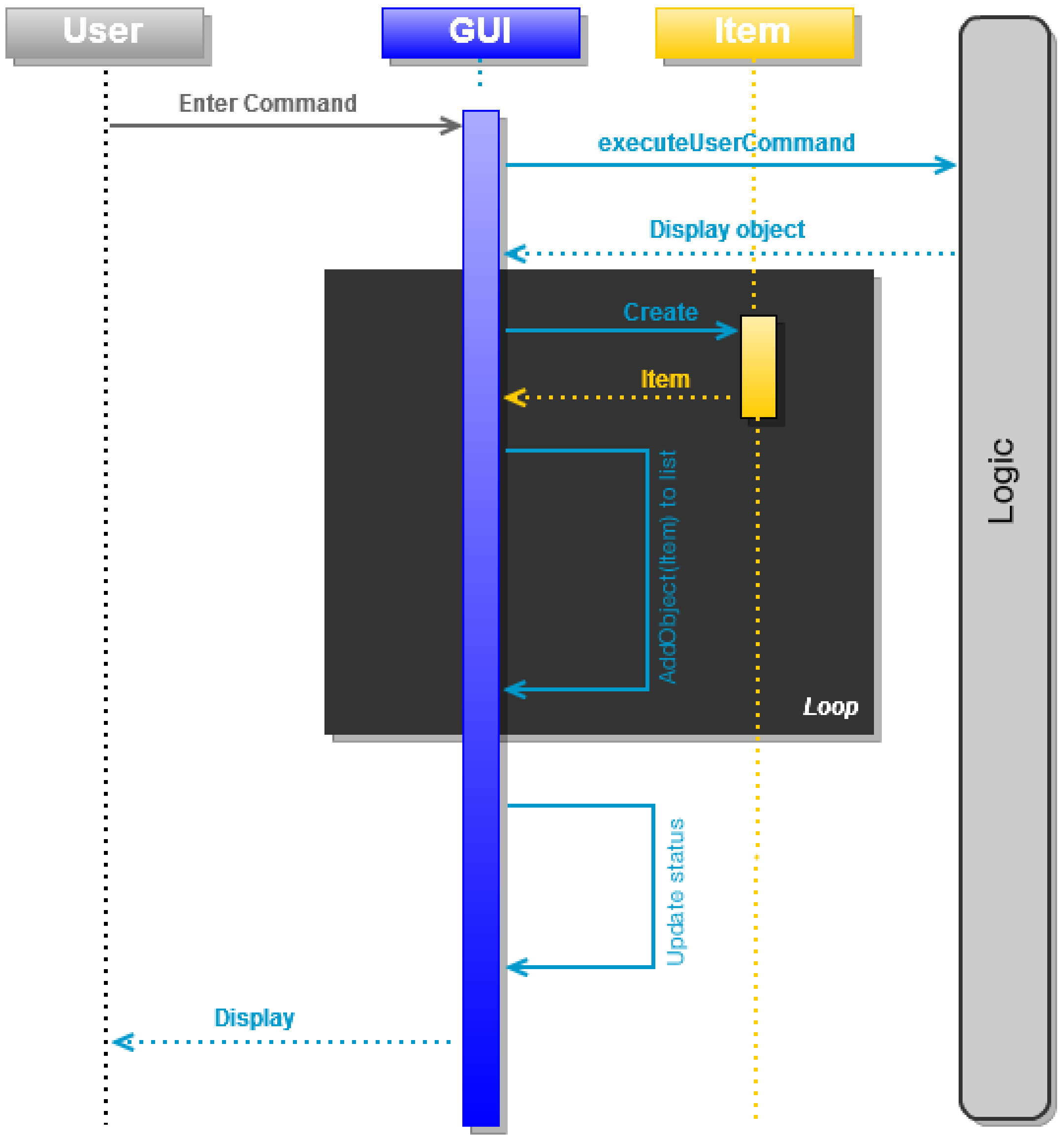


Diagram 4.1: Sequence Diagram of GUI implementation

The sequence diagram above illustrates the case where the user enters a text command. The sequence would be the same for short cut commands except that the Logic API called will be "executeShortCutCommand" instead of "executeTextCommand".

Text Command

An event is triggered when the user presses the Enter key while the textbox is focused. The text command is then passed to the Logic class for processing using the API "executeTextCommand" (Refer here). This will return a Display Object containing all the necessary information to be displayed to the user. The GUI class will then populate the ListView by creating Item objects and update the Status feedback to the user.

Short Cut Command

An event is triggered when the user presses any of the available short cut keys when either the TextBox or the ListView is focused. The GUI class will create a Command object based on the short cut keys that was pressed. The Command object is then passed to the Logic class for processing using the API "executeShortCutCommand". This will return a Display Object containing all the necessary information to be displayed to the user. The GUI class will then populate the ListView by creating Item objects and update the Status feedback to the user.

4.2 LOGIC AND COMMAND INTERNAL ARCHITECTURE

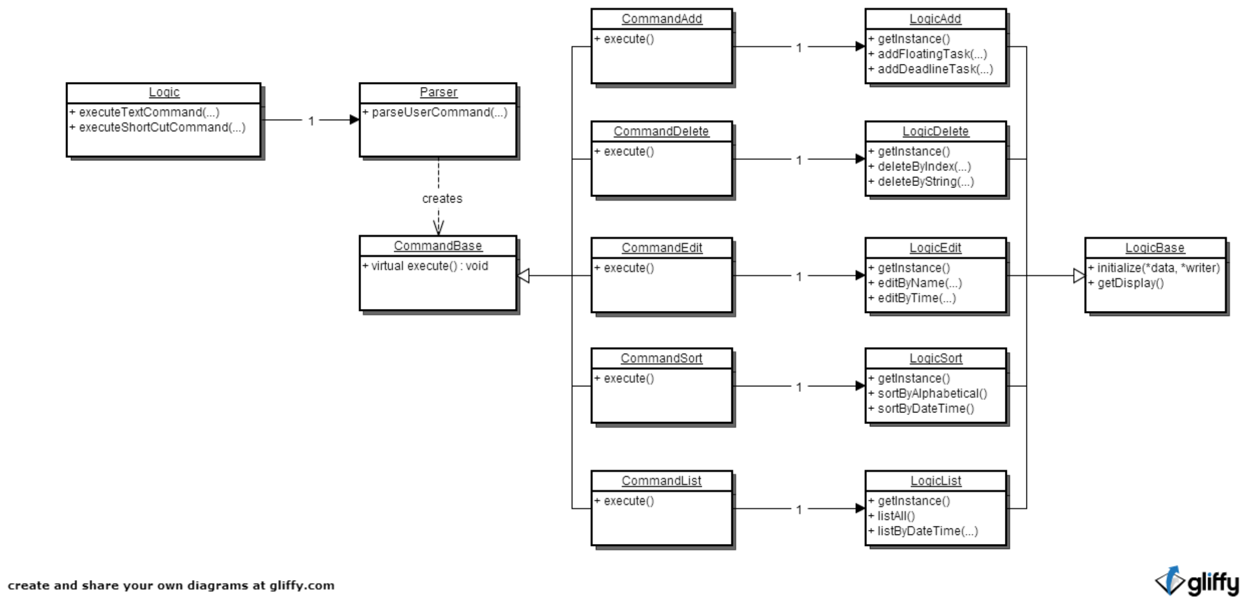


Diagram 4.2 Class Diagram of Logic and Command Internal Architecture

LogicAdd, LogicEdit, etc. are derived classes of LogicBase. CommandAdd, CommandEdit etc. are derived classes of CommandBase. Respective logic sub classes are being invoked by the appropriate command sub classes. Logic sub classes are singletons. (Only 1 object can be created). Each logic sub class contains the relevant methods related an operation. (e.g. LogicAdd allows adding a floating, deadline or timed task).

Parser will process the user input and will create/construct the appropriate command sub class with the required information for execution (Refer here). Upon completion, a CommandBase object will be returned to Logic. Logic will then invoke the execute command, which will perform the necessary operations (Addition, deletion, etc.).

A Display object containing the list of tasks and a status message is being returned to the GUI class, which will populate the graphical screen with the appropriate details (Refer here).

Important algorithms

In order to properly implement the undo feature, multiple Data objects are stored by Logic. They are pushed and popped of a stack as and when required.

4.3 PARSER INTERNAL ARCHITECTURE

Overview

The Parser is based on a chain of responsibility design pattern. This is implemented by the handler *ParserBase*. The client sends in a command to the Parser by calling parseUserCommand(). If this particular parser can parse the user command successfully, it will return a pointer to a *CommandBase* object to the client. If not, it will reroute the request to the next parser associated with it (its successor). The Parser will eventually return NULL if there was no appropriate handler found. **Note: It is the caller's responsibility to free memory allocated for the *CommandBase* object created. Failure to do so may result in a memory leak.**

A simplified class diagram is illustrated below.

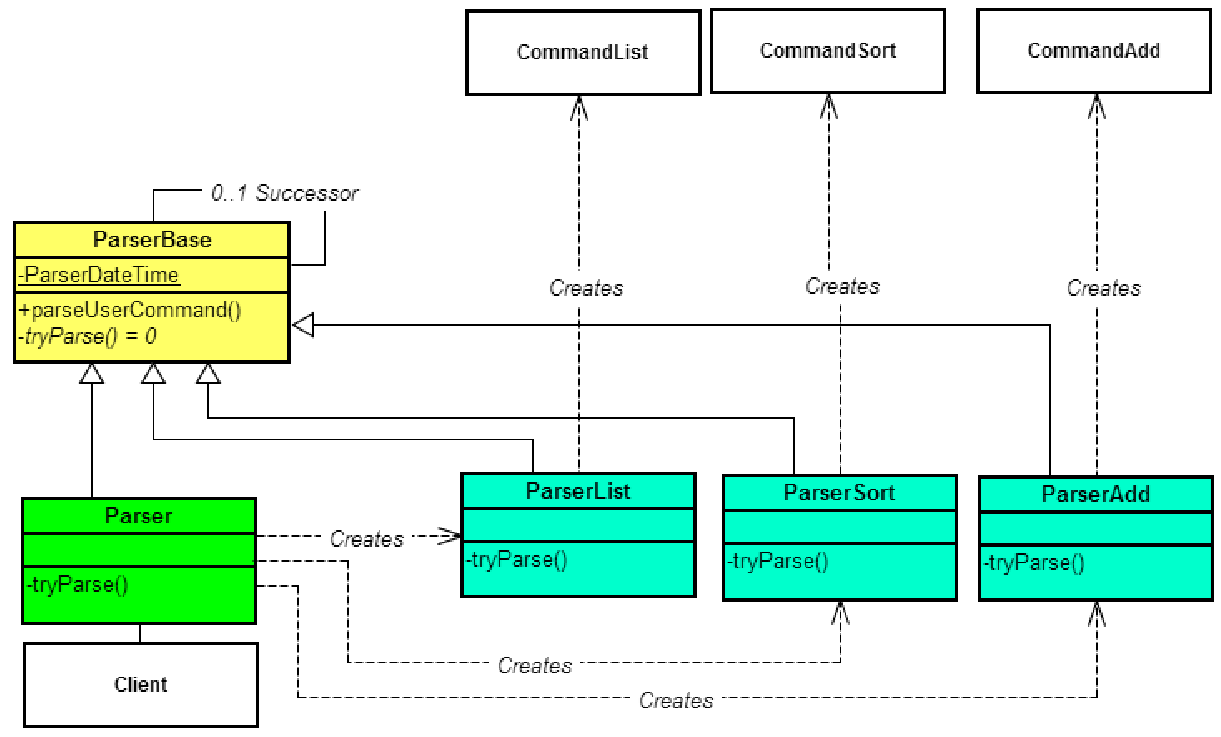


Diagram 4.3.1 Class Diagram of Parser Internal Architecture

As shown above, each parser should have 0 or 1 successors. If there are no successors, the command is not parsed and NULL will be returned. In reality, each subclass of *ParserBase* will have its own specific set of parsing rules and syntax. As such, there are virtually 10 or more parsers who try to parse a user command in turn. Do also note that there is no requirement for each *ParserXXX* derived class to handle exactly one type of command each. For example, should developers wish to add a new esoteric syntax to be supported for the command *Add,* they are free to create a new subclass in the chain of responsibility to handle these cases. However, as of the time of writing, almost all *CommandXXX* classes have exactly one *ParserXXX* handler. This is for simplicity in debugging and organization.

Construction is done by the class *Parser*, which is also derived from *ParserBase*. It creates all instances of other parsers, and “links” the classes together via successor associations. It is also responsible for the clean destruction of all instances of *ParserBase* objects it creates.

In addition, there is a specific subclass named *ParserCatchAll*. This is placed at the end of the successor chain, and is responsible for generating filling NULL return values which inform the client that the command was completely invalid – the parser was not even able to classify the type of command entered.

Errors

In general, there are two types of parsing errors.

a) There was some form of format recognized, but some small portions could not be parsed. For example, the command "sort qqq" was clearly *intended* as a sort command. However qqq is not a valid parameter for the sort instruction. In such cases, **a *ParserException* is thrown, with the error stored inside in the format of a string.**
b) Other types of unparseable commands – for example "RandomCommand". No handler exists for these – thus a NULL pointer is returned.

Some errors are particularly tricky to resolve – for example, 31-Feb-2012 was clearly intended to be a date, despite 31 Feb being an invalid date. In such cases, the behavior would depend on the underlying instruction (add, delete, etc).

Adding Additional Commands/Interpretations/Syntax

Create a new subclass of *ParserBase*. Ensure that the tryParse() method is implemented correctly – ie. it returns a *CommandBase** if the parsing was successful[1], and NULL otherwise. Next, modify the *Parser* class to ensure that the "successor links" are appropriately set up during the construction of Parser.

Note that because of this architecture, if there are commands which satisfy the parsing criteria of more than one *ParserXXX*, the first *ParserXXX* in the chain of responsibility will be the one who handles it. It is advisable for the developer to understand the existing supported syntax before implementing additional syntax. This is to avoid potential conflicts in syntax being supported.

Editing Existing Commands

Most of the current parsing is done with the aid of regular expressions. For most implementations, the Parser first checks if the instruction (usually the first word), matches a list of predefined terms. If not, it passes control over to its successor. If it does, it proceeds to attempt to classify which type of command it is (eg. add timed task, add deadline tasks etc.) Following which, it will proceed to create the relevant *CommandXXX* object and return it.

Date and Time Parsing

As there are numerous occasions where date and times need to be parsed, a specialized static utility class *ParserDateTime* was created. Developers may employ the methods within – namely parseDateTime() and parseTime() for whichever commands they add/modify, although its primary purpose is to service the *ParserXXX* classes.

[1] Successful refers to the successful matching of what the user typed to some command. Invalid arguments will still result in a *ParserException* being thrown.

The primary APIs for the *ParserDateTime* are parseDateTime(), parseDateTimePeriod(), parseDate() and parseTime(). Several arguments are in place so as to allow clients to specify the type of parsing required.

Automated Testing

There is a list of tests available for automated testing. This is done in Google Test, and should be readily available together with the source code. The tests for the Parser are separated into two portions, 1) Syntax testing for individual commands, and 2) Date and Time format testing. Additional details may be found in the appendix. Due to the large number of potential tests for date and time testing, each test is associated with a 4-digit code. Details are found in the test cases proper.

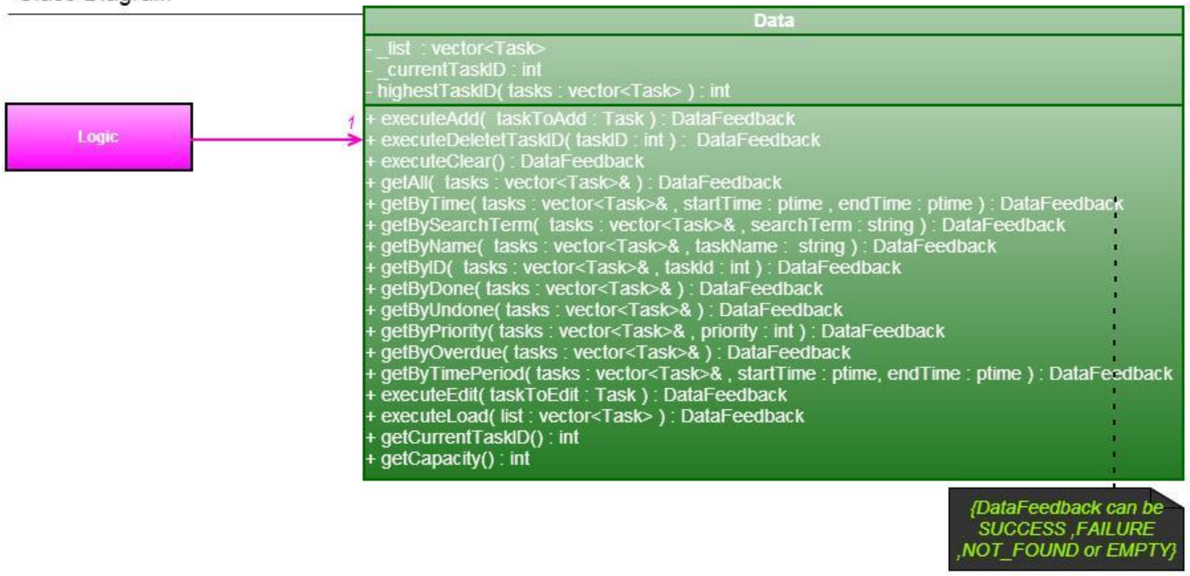
Notable algorithms/methods

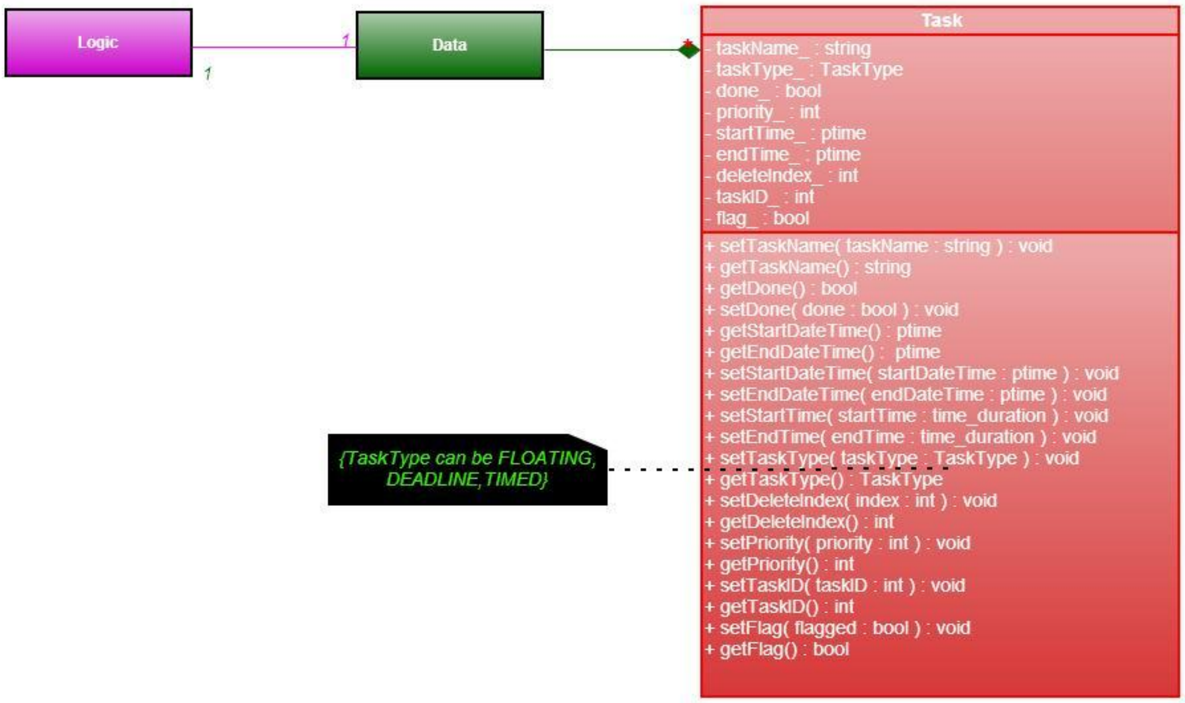
The methods extractDateTimeFront(), extractDateTimeBack(), extractTimePeriodFront(), and extractTimePeriodBack() within *ParserBase* allow subclasses to easily extract date/times/time periods existing within a string. However, due to repeated iterations across the input string, this search takes a long time to complete. As such, developers are advised to avoid frequent calls to these methods unless necessary.

4.4 DATA INTERNAL ARCHITECTURE

The Data class provides appropriate feedback depending on the command being executed which is determined by Logic. Task objects which are stored in a vector are manipulated only by data. Vectors in function arguments are populated by Data which help logic to perform appropriate operations.

Class Diagram


Data
- _list : vector<Task>
- _currentTaskID : int
- highestTaskID(tasks : vector<Task>) : int
+ executeAdd(taskToAdd : Task) : DataFeedback
+ executeDeletetTaskID(taskID : int) : DataFeedback
+ executeClear() : DataFeedback
+ getAll(tasks : vector<Task>&) : DataFeedback
+ getByTime(tasks : vector<Task>& , startTime : ptime , endTime : ptime) : DataFeedback
+ getBySearchTerm(tasks : vector<Task>& , searchTerm : string) : DataFeedback
+ getByName(tasks : vector<Task>& , taskName : string) : DataFeedback
+ getByID(tasks : vector<Task>& , taskId : int) : DataFeedback
+ getByDone(tasks : vector<Task>&) : DataFeedback
+ getByUndone(tasks : vector<Task>&) : DataFeedback
+ getByPriority(tasks : vector<Task>& , priority : int) : DataFeedback
+ getByOverdue(tasks : vector<Task>&) : DataFeedback
+ getByTimePeriod(tasks : vector<Task>& , startTime : ptime, endTime : ptime) : DataFeedback
+ executeEdit(taskToEdit : Task) : DataFeedback
+ executeLoad(list : vector<Task>) : DataFeedback
+ getCurrentTaskID() : int
+ getCapacity() : int
Logic
1
{DataFeedback can be SUCCESS ,FAILURE ,NOT_FOUND or EMPTY}


Logic
1
1
Data
*
Task
- taskName_ : string
- taskType_ : TaskType
- done_ : bool
- priority_ : int
- startTime_ : ptime
- endTime_ : ptime
- deleteIndex_ : int
- taskID_ : int
- flag_ : bool
+ setTaskName(taskName : string) : void
+ getTaskName() : string
+ getDone() : bool
+ setDone(done : bool) : void
+ getStartDateTime() : ptime
+ getEndDateTime() : ptime
+ setStartDateTime(startDateTime : ptime) : void
+ setEndDateTime(endDateTime : ptime) : void
+ setStartTime(startTime : time_duration) : void
+ setEndTime(endTime : time_duration) : void
+ setTaskType(taskType : TaskType) : void
+ getTaskType() : TaskType
+ setDeleteIndex(index : int) : void
+ getDeleteIndex() : int
+ setPriority(priority : int) : void
+ getPriority() : int
+ setTaskID(taskID : int) : void
+ getTaskID() : int
+ setFlag(flagged : bool) : void
+ getFlag() : bool
{TaskType can be FLOATING, DEADLINE,TIMED}

4.5 EXPANDABILITY

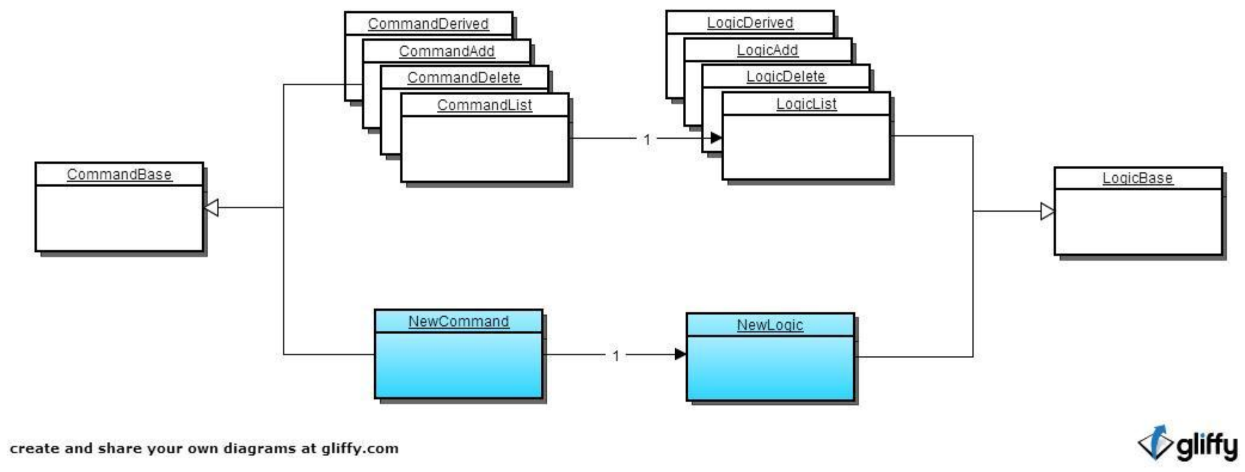
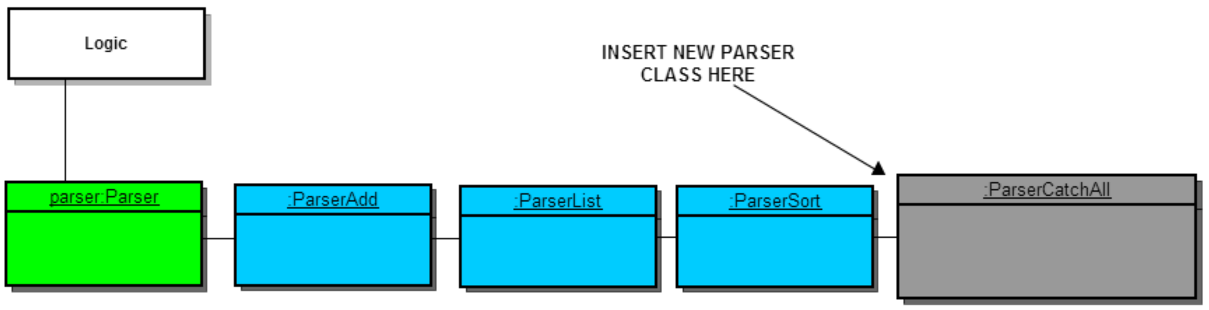


Diagram 4.5 Class Diagram of Logic and Command

To add new features into the application, add in the derived classes highlighted in blue.



Due to the usage of command patterns, adding new features into the application only requires new derived classes. They are highlighted in the diagram above in blue. The relevant parser class catered for the new feature is also required (Refer here). This can be done in a similar fashion.

In addition, create or modify an existing *ParserXXX* class to allow it to create the new *CommandXXX* object introduced. In general, it is advisable to create a new *ParserXXX* class for this purpose in order to maintain separation of concerns.

Point to note on expandability

This architecture allows ease of features expansion. To expand on features, developers can create a new derived class of CommandBase. A new derived class of LogicBase is to be created to cater to the functionality of the added feature.

In addition, its strength lies in maintaining the concept of Open-Closed Principle to a large extent. Adding a new command base derived class, together with logic base derived class allows for extension, instead of modification to the existing code.

It is also important to note the existence of the Task class and the attributes it contains. The Task class essentially represents a task object that can be added.

5 TESTING

This section will give a brief overview on how automated testing is done on Scheduler++. This will help developers to eliminate regression while development is done on Scheduler++.

The automated testing is configured with Google Test. For a guide on how to set up Google Test, you may refer to our Project Tutor Mark's Website here. Once the setup is done, the test files are ready to be used.

There are various test files for individual components. The following table shows the relevant test files for their respective components.

Component	**Tester File Name**
Parser	ParserTester.h
Logic	LogicTester.h
Scheduler++	Tester.h

These test cases aim to achieve as much coverage as possible.

Test Fixtures are used to SetUp() and TearDown() the environment for the individual test cases. This saves coding time and allows code reuse on the same configuration of environment for different test cases.

6 CREDITS

Boost

http://www.boost.org/

ObjectListView

http://objectlistview.sourceforge.net/cs/index.html

Gliffy

http://www.gliffy.com/

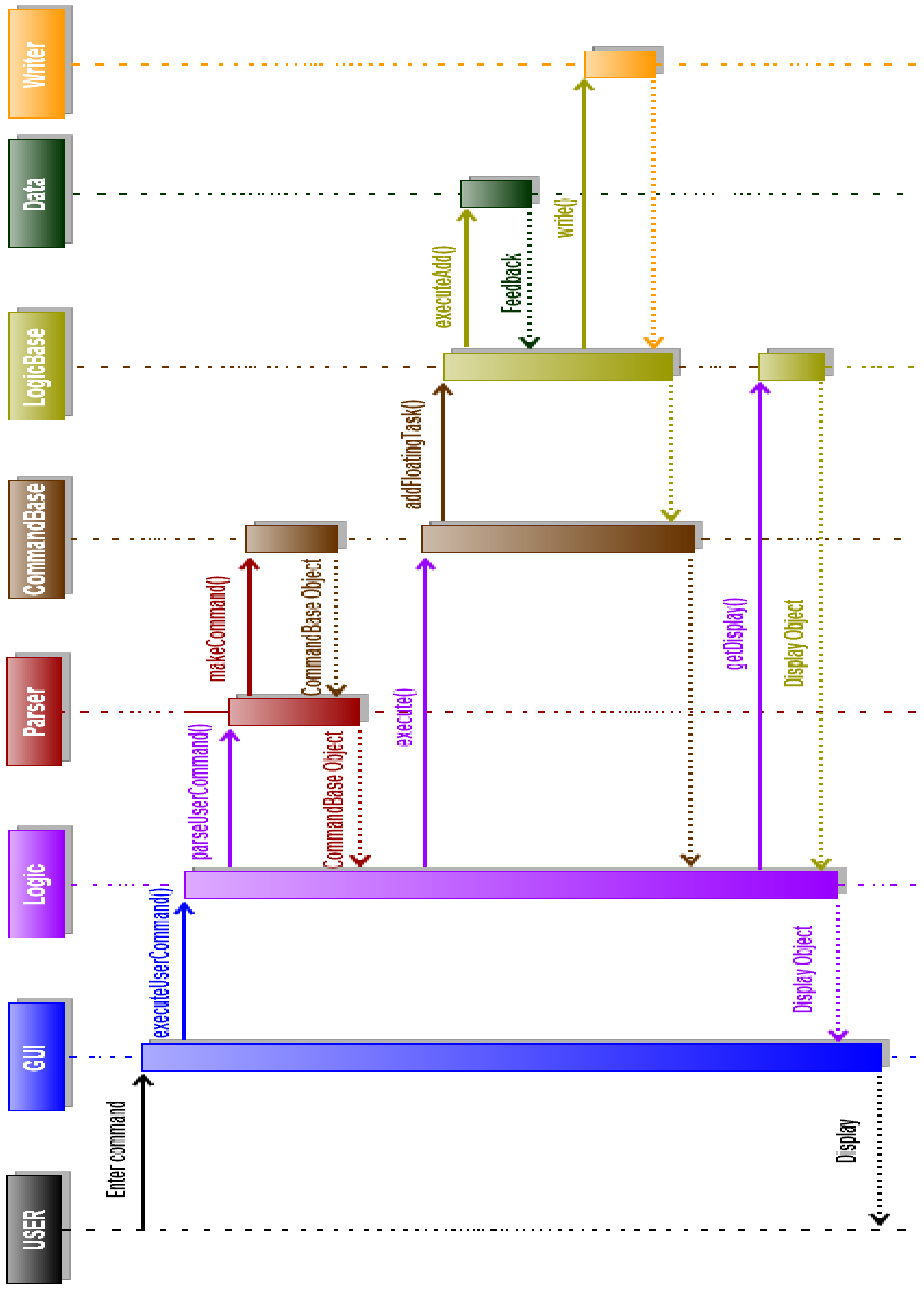
IconArchive

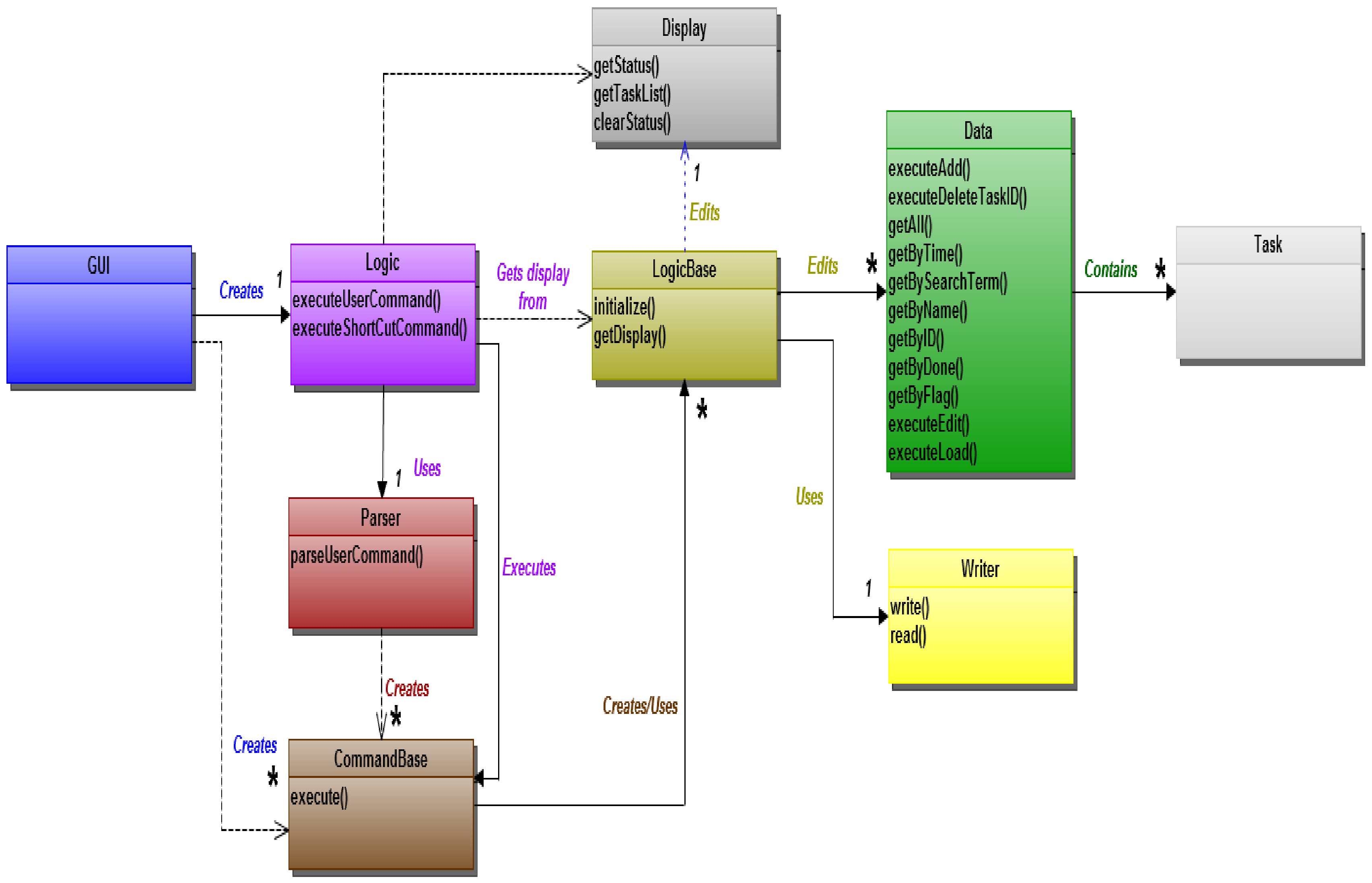
http://www.iconarchive.com/

DryIcons

http://dryicons.com/free-icons/

APPENDIX

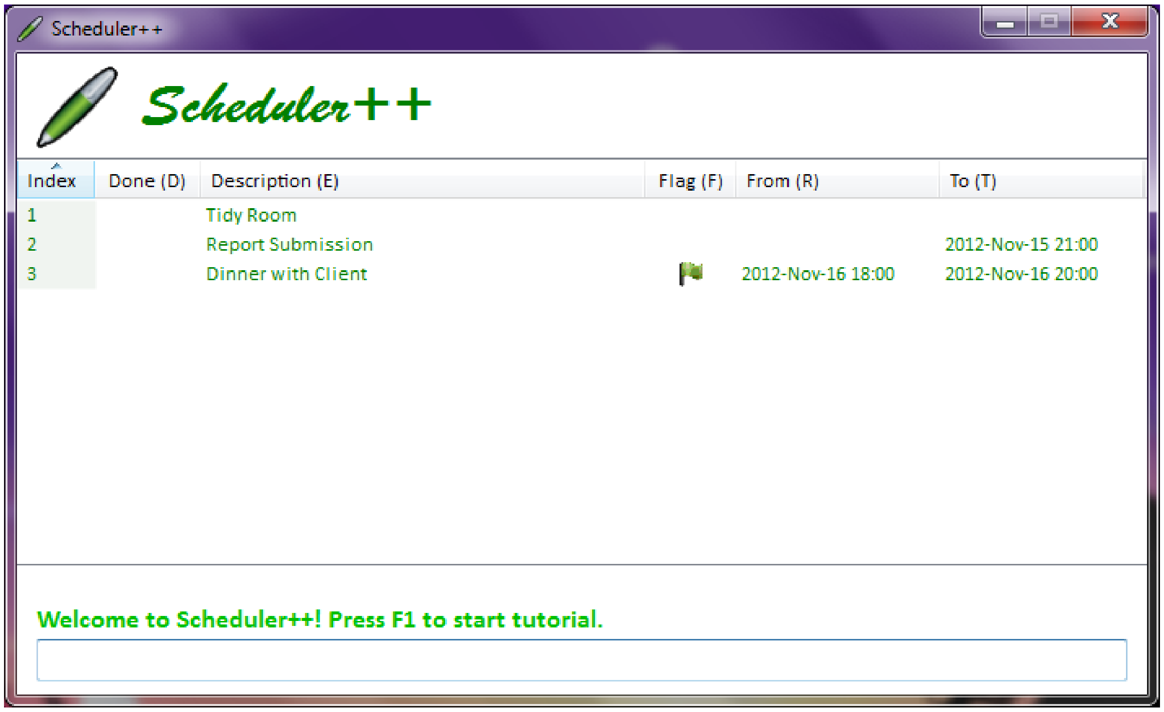

USER
GUI
Logic
Parser
CommandBase
LogicBase
Data
Writer
Enter command
executeUserCommand()
parseUserCommand()
makeCommand()
CommandBase Object
CommandBase Object
execute()
addFloatingTask()
executeAdd()
Feedback
write()
getDisplay()
Display Object
Display Object
Display



Display
getStatus()
getTaskList()
clearStatus()
GUI
Creates
1
Logic
executeUserCommand()
executeShortCutCommand()
Gets display from
LogicBase
initialize()
getDisplay()
1
Edits
Edits
*
Data
executeAdd()
executeDeleteTaskID()
getAll()
getByTime()
getBySearchTerm()
getByName()
getByID()
getByDone()
getByFlag()
executeEdit()
executeLoad()
Contains
*
Task
*
Uses
1
Uses
Parser
parseUserCommand()
Executes
1
Writer
write()
read()
Creates
*
Creates
*
CommandBase
execute()
Creates/Uses

USER GUIDE

Scheduler++



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	⚑	2012-Nov-16 18:00	2012-Nov-16 20:00

Welcome to Scheduler++! Press F1 to start tutorial.

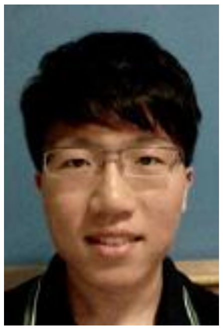
			
Lim Jia Hui, Bryan	Ling Chun Kai	Sudarsan Gopalaswami Padmanabhan	Tan Wei Jian Joash
A0086671W	A0002927L	A0092076A	A0081136M
Logic	Parser	Group Leader Data	GUI Writer

Table of Contents

Introduction

Ever wanted a **Convenient** and **Accessible** way to **Manage** your calendar?
Scheduler++ offers users a painless way to tackle the complexities of life!

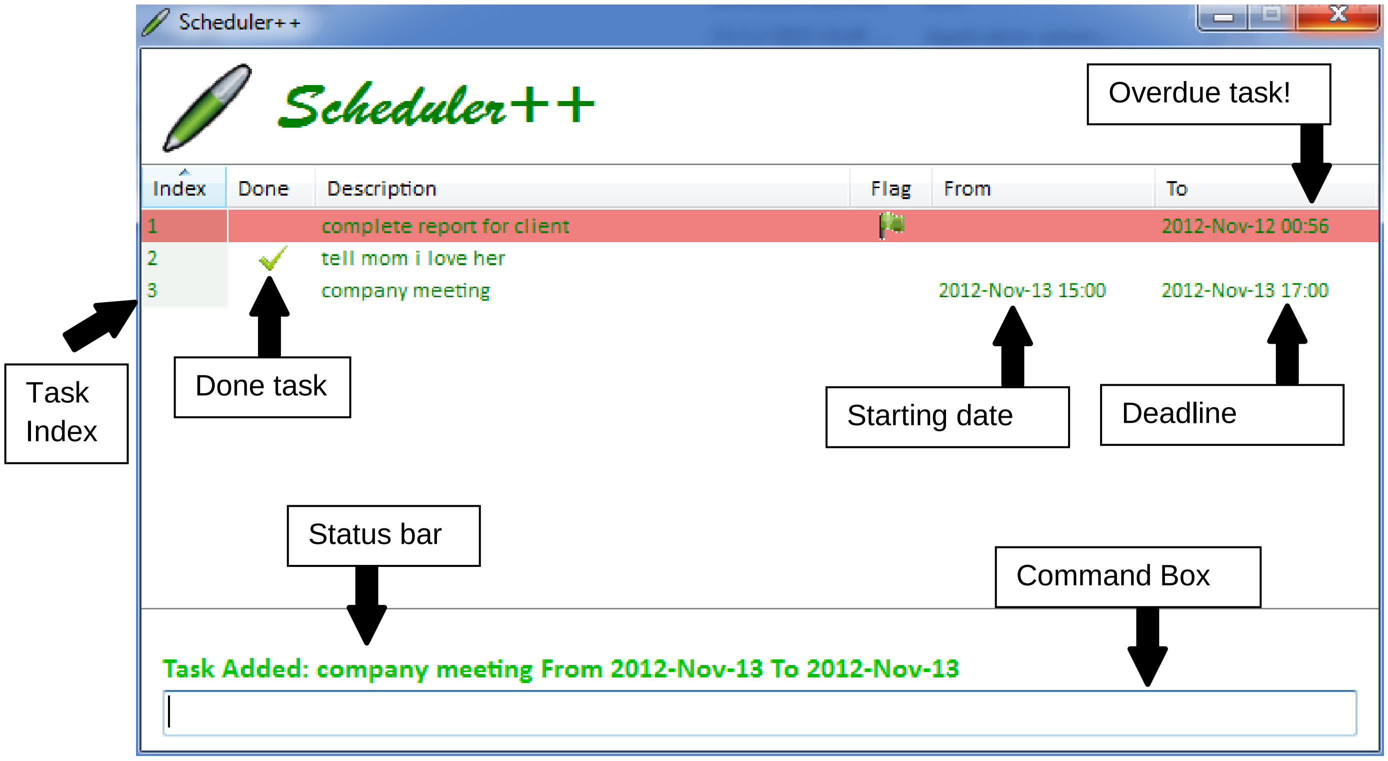
- Forget about complicated user forms and plunge straight in with our **Simple User Interface** and **Intuitive User Commands**.
- Easily **Review** and **Prioritize** tasks for easy classification with our flagging and searching capabilities.

Fasten your seat belt and get ready to zip through life with Scheduler++**!**

Getting Started

We shall assume that you have downloaded a copy of Scheduler++ and have unzipped it into your preferred directory.

Please ensure that ObjectListView.dll is present in the same directory as Scheduler++.exe. Once that is done, launch Scheduler++ by double clicking on it! You should get a similar picture as what is shown below (albeit with minor differences). At this point, we strongly suggest you hit <F1> and complete the in-program tutorial before proceeding.



Index	Done	Description	Flag	From	To
1		complete report for client			2012-Nov-12 00:56
2	✓	tell mom i love her			
3		company meeting		2012-Nov-13 15:00	2012-Nov-13 17:00

Task Added: company meeting From 2012-Nov-13 To 2012-Nov-13

Scheduler++ offers u a simple way of viewing all your tasks at a glance. However, let us first orientate ourselves. The main bulk of the screen belongs to a list containing several tasks. There are 3 broad categories of tasks, namely:
Deadline tasks: Tasks which have to be completed by a certain date
Floating tasks: Tasks which have no deadline – they can be completed at any time
Timed tasks: Tasks which have both a starting and ending date

Tasks may be **done(completed)** and/or **flagged**. Overdue tasks are highlighted red automatically by the program.

You may key in your commands into the **Command Box** and register them by hitting the <Enter> key. Any feedback would be displayed in the **Status bar**.

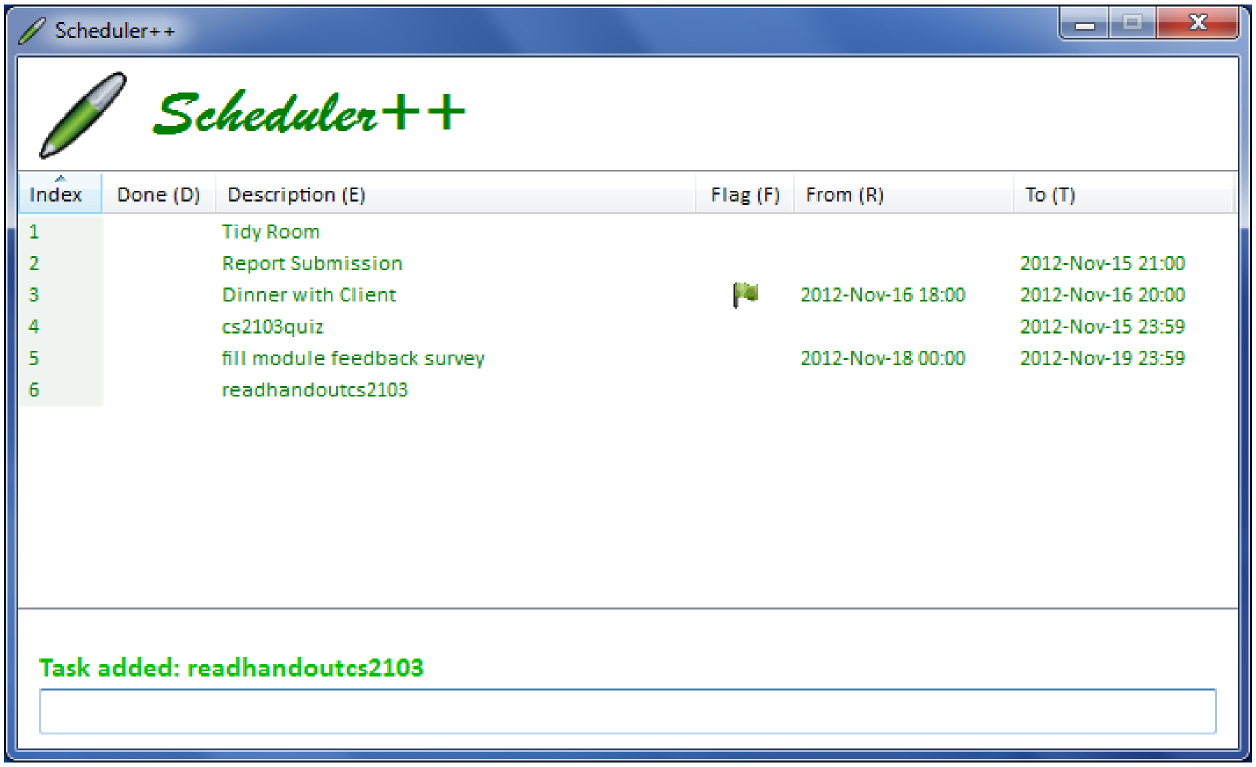
Quick Start Tutorial

1. Adding Tasks to Scheduler++ : use the add keyword

   1.1 First we shall add a task which has to be completed by a specific time.
   The syntax for deadline task: add cs2103quiz 15 nov

   1.2 We shall add a task which has to be completed within a specific period of time.
   The syntax for timed task: add fill module feedback survey 18 nov to 19 nov

   1.3 Now let us add a task which can be completed at any time you want.
   Floating Tasks can be entered using the following syntax: add readhandoutcs2103



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	🚩	2012-Nov-16 18:00	2012-Nov-16 20:00
4		cs2103quiz			2012-Nov-15 23:59
5		fill module feedback survey		2012-Nov-18 00:00	2012-Nov-19 23:59
6		readhandoutcs2103			

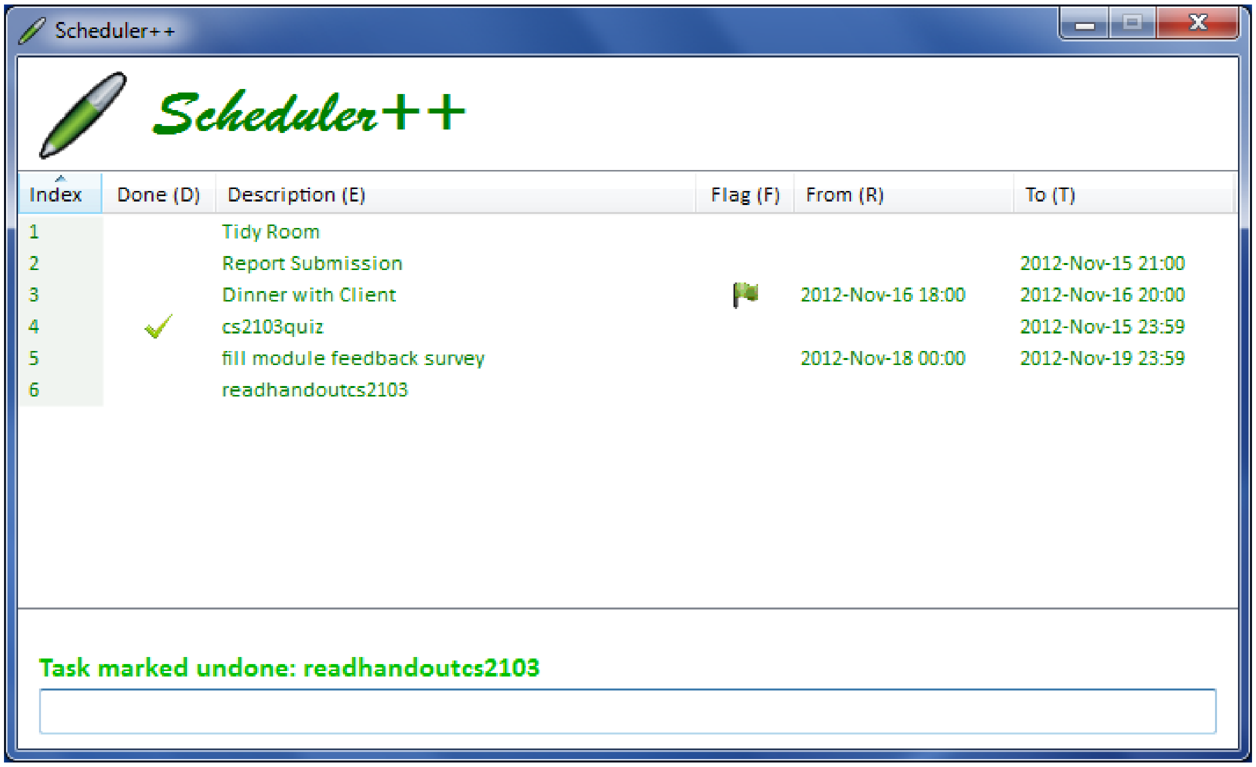
Task added: readhandoutcs2103

2. Mark task as done or un-done : use the done keyword

2.1 After you have completed a task you can mark the task as done using the done keyword. The tasks can be marked done using the index which is displayed on the screen.
Syntax: done 4

2.2 After completing a particular task you could also type a part of the description to mark a particular task as done.
Syntax: done handout

2.3 If you had marked a wrong task as done or you want to mark the task as not done then use the done keyword with either the syntax used in 2.1 or 2.2.
Syntax: done 6



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	⚑	2012-Nov-16 18:00	2012-Nov-16 20:00
4	✓	cs2103quiz			2012-Nov-15 23:59
5		fill module feedback survey		2012-Nov-18 00:00	2012-Nov-19 23:59
6		readhandoutcs2103			

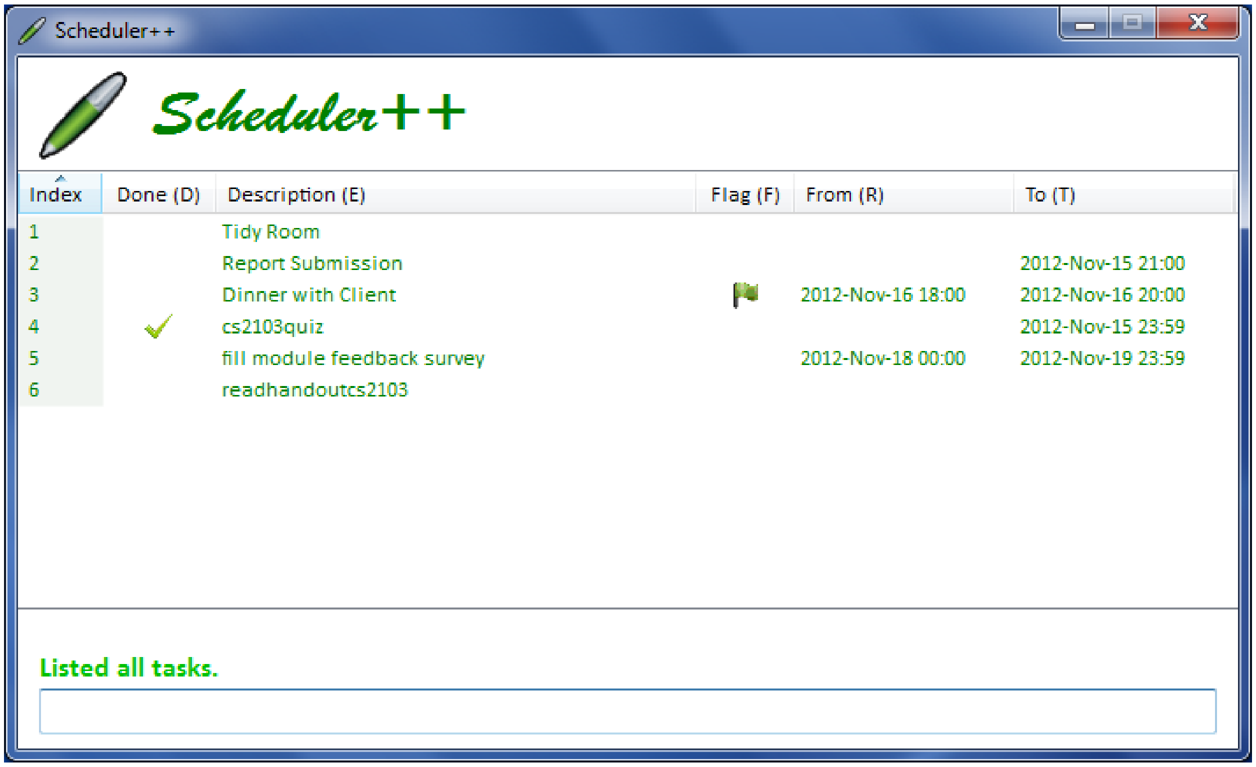
Task marked undone: readhandoutcs2103

3. List the tasks entered: use list keyword

3.1 Tasks that you have marked done can be listed separately using the list keyword.
Syntax: list done

3.2 Tasks can also be listed by part of description. The tasks containing the string you entered will appear on the screen.
Syntax: list 03q

3.3 All the tasks that you have entered can be displayed using only the list keyword.
Syntax: list



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	⚑	2012-Nov-16 18:00	2012-Nov-16 20:00
4	✓	cs2103quiz			2012-Nov-15 23:59
5		fill module feedback survey		2012-Nov-18 00:00	2012-Nov-19 23:59
6		readhandoutcs2103			

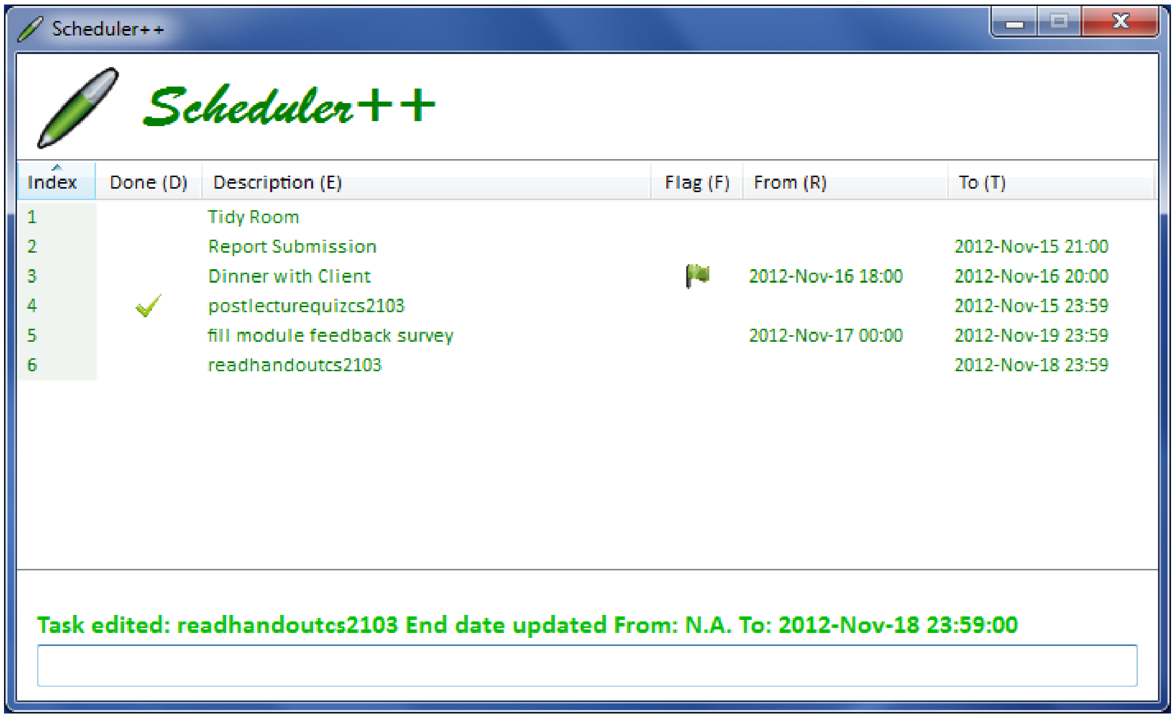
Listed all tasks.

4. Edit the task entered: use edit keyword

4.1 You can also change the description of the task you have entered.
Syntax: edit 4 rename postlecturequizcs2103

4.2 You can also change the existing start time for a timed task task.
Syntax: edit 5 start 17 nov

4.3 You can also change end time of deadline or timed task or to set the end time for floating task.
Syntax: edit 6 end 18 nov



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	⚑	2012-Nov-16 18:00	2012-Nov-16 20:00
4	✓	postlecturequizcs2103			2012-Nov-15 23:59
5		fill module feedback survey		2012-Nov-17 00:00	2012-Nov-19 23:59
6		readhandoutcs2103			2012-Nov-18 23:59

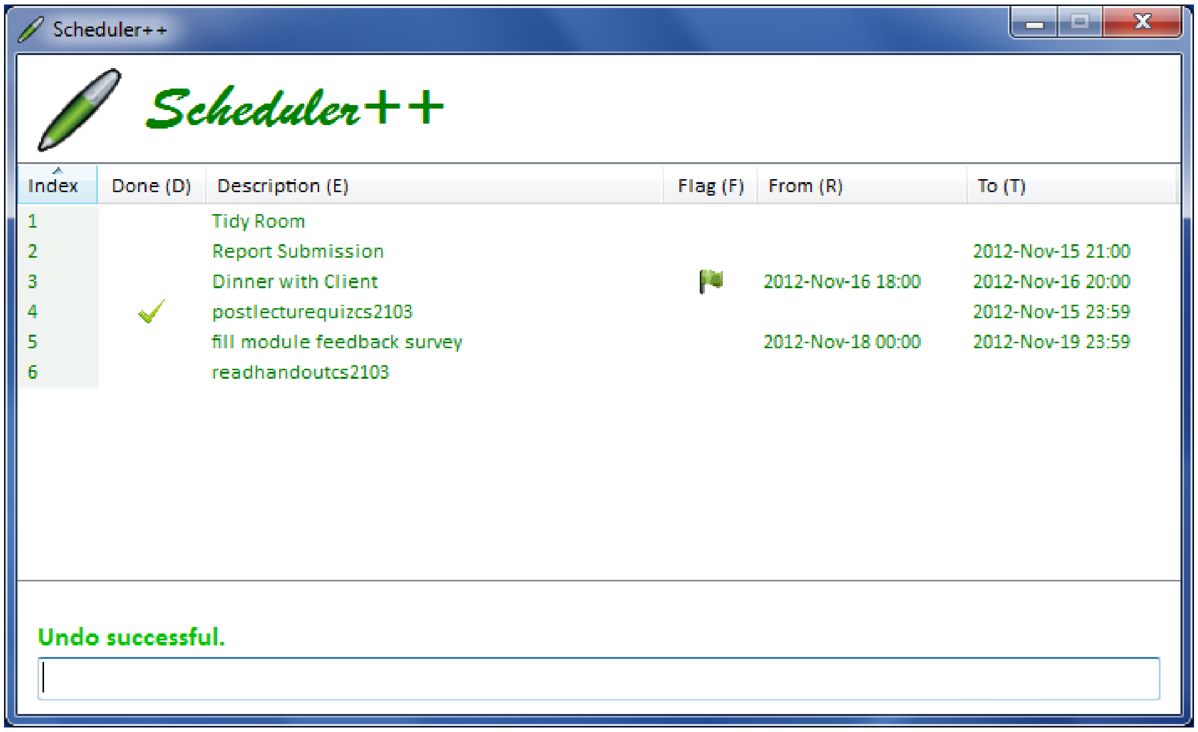
Task edited: readhandoutcs2103 End date updated From: N.A. To: 2012-Nov-18 23:59:00

5. Undo any command operation performed: use undo keyword

If you had entered a sequence of not useful commands use undo till you get the desired display.

Syntax: undo

undo



Index	Done (D)	Description (E)	Flag (F)	From (R)	To (T)
1		Tidy Room			
2		Report Submission			2012-Nov-15 21:00
3		Dinner with Client	⚑	2012-Nov-16 18:00	2012-Nov-16 20:00
4	✓	postlecturequizcs2103			2012-Nov-15 23:59
5		fill module feedback survey		2012-Nov-18 00:00	2012-Nov-19 23:59
6		readhandoutcs2103			

Undo successful.

User Guide Format

Words in **bold** are Command Keywords
(You may refer to the List of Command Keywords on next page)
Words underlined are shortcuts for Keywords

- Bulleted points explain what the Keyword does

```
Instructions are shown in this box.
Text written in italics denote fields which you need to fill up accordingly.
```

```
Examples are shown as the following format:
user input
: displayed output  (appears in Status Bar of Scheduler++)
```

Date and Time Format

Scheduler++ allows for a large variety of date and time formats. Some of them are listed below. Please note that they are not exhaustive. Feel free to enter your favourite date and time formats!

```
Standard Formats: day-month-year, time / month-day-year, time
Shortcut Formats: day-month / month-day
Flexible Formats: tomorrow, next month, this coming Monday, next week
Special Days/Events: eg. Christmas, new year

Note: if fields are left out the program will make the most appropriate
choice based on the users' commands.
```

```
12 Sep 2012, 2330
12-9-12, 1130pm
12 Sep 12, 23:30
Sept 12 2012, 2330
Tmr @ noon
Next monday, 8am
Christmas at midnight
```

Command Keywords Help List

Add

- Adds a task to your Scheduler++
- Tasks may be floating, deadlined, or timed
- The task will be a floating task if no end date or time is provided
- The task will be a deadline task if an end date or time is provided
- The task will be a timed task if a start and end date or time is provided

```
add description
add description date
add description [>/by/on] date
add description date to date
add description [>/from] date to date
add date description
```

```
add Tidy Room
: Task added: Tidy Room
add Report Submission Jan 1
: Task added: Report Submission By 2013-Jan-01
add Camp next mon, 6pm to next wed, 11:45
: Task Added: Camp From 2012-Dec-17 To 2012-Dec-19
add Roast Turkey on Christmas, 630pm
: Task Added: Roast Turkey By 2012-Dec-21
```

- There are many other formats not listed that are supported. Type around and explore!

Delete

- Deletes the task with the specified name
- Deletes the task with the task index as displayed in the task list
- Deletes all tasks which have been marked as done
- Deletes all tasks containing a search term designated by the user. If there is more than one task with the matching search term, the command will then function similar to "list", allowing the user to select an appropriate task index.
- Deletes listed tasks on display.
- You can use the undo command if a mistake was made!

```
delete task index
delete task description
delete search term
delete done
delete listed
```

```
delete 2
: Task deleted: girlfriend
delete campfire
: Task deleted: campfire @ pulau ubin
delete done
: All completed tasks deleted.
delete listed
: All tasks on display deleted.
```

Clear

- ⚠ Use with caution!
- Clears all tasks

```
clear YES
```

```
Clear YES
: All tasks cleared.
```

List

- Lists down all tasks if no search term is provided
- Lists down tasks in the current month/week/day
- Lists down tasks in the indicated month/week/day
- Lists completed/uncompleted tasks
- Lists overdue tasks
- Lists down all tasks containing the particular search term. Users may enclose their search term with quotation marks if the search term contains keywords normally associated with dates.

```
list
list [month/day/week]
list stipulated [day/month/week]
list search term
list overdue
list [done/undone/not done]
```

```
list
: Listed all tasks.
list day
: Listed tasks from 2012-Dec-10 to 2012-Dec-10
list next week
: Listed tasks from 2012-Dec-10 to 2012-Dec-17
list 21 Dec 2012
: Listed tasks from 2012-Dec-10 to 2012-Dec-21
list CS2103
: Listed tasks containing: CS2103
list done
: Listed completed tasks.
list undone
: Listed uncompleted tasks.
```

Edit

- Edit task
- Omit unchanged details

```
edit task index start date + time(optional)
edit task index end date + time(optional)
edit task index start time
edit task index start time
edit task index rename new description
```

```
edit 2 end 2359
: Task edited: Lab report End time updated From: 23:00:00 To: 23:59:00
edit 1 start 23 Dec
: Task edited: purchase tickets From: 2012-Dec-22 To: 2012-Dec-23
edit 3 rename Make money
: Task renamed: From: Make monkey To: Make money
```

Flag

- Adds a flag to task

```
flag task index
flag task description
```

```
flag 3
: Task flagged: Vacation to USA
flag vacation
: Task unflagged: Vacation to USA
```

Done

- Marks the task as done

```
done task index
done task description
```

```
done 1
: Task marked done: Dinner with client
done Walk the dog at park
: Task marked undone: Walk the dog at park
```

Sort

- Sorts the tasks currently listed
- Sorts Alphabetically / Flag / Time / Done

```
sort type
```

```
sort a
: Sorted by alphabetical order.
sort f
: Sorted by flagged tasks on top.
sort t
: Sorted by date and time.
sort d
```

```
: Sorted by completed tasks on top.
```

Undo

- Undo a previous command
- Up to 10 undos

```
Undo
```

```
undo
: Undo successful.
```

Short-Cut Keys

Key	Control	Function
F1	All	*Toggles between tutorial and normal mode*
Tab	All	*Toggles control between command box and list*
Escape	All	*Closes Scheduler++*
PageUp	Command Box	*Scrolls list up*
PageDown	Command Box	*Scrolls list down*
ArrowUp	Command Box	*Scrolls up previous commands*
ArrowDown	Command Box	*Scrolls down previous commands*
D	List	*Toggles selected task Done*
F	List	*Toggles selected task Flag*
E	List	*Edits selected task Description*
R	List	*Edits selected task From*
T	List	*Edits selected task To*
Delete	List	*Deletes selected task*