BoxIn
Developer Guide

BoxIn

Contents

1 Background

BoxIn is a C++ based to-do list manager and this guide will help to understand how it works. BoxIn design adheres to the following coding standards

SLAP SLAP refers to Single Level Abstraction Principle. Code should be well abstracted so that each function only has one level of function calls. This helps to keep code from becoming convoluted by abstracting away the details of how a function is implemented

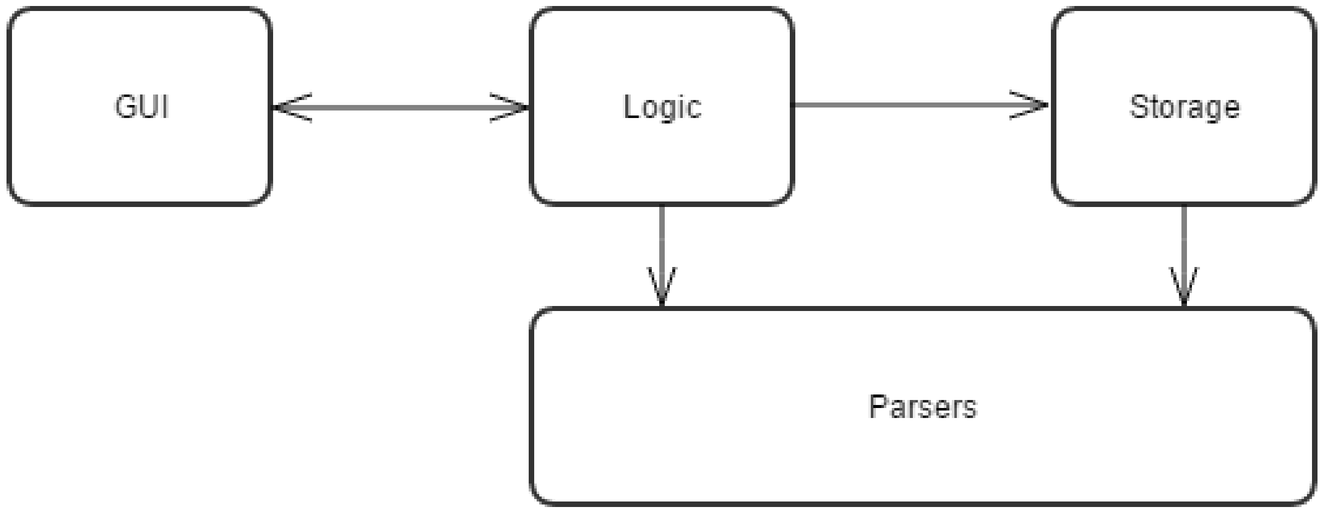
Memory Management Coding in C++ requires efficient use of memory. Memory should be freed if its not in use any longer.

Compatibility Code should be written such that is is cross platform and compatible with different compilers. Compiler specific code should be marked as such.

Coding standard BoxIn code follows the coding standard listed out here: C++ coding standards.

Namespaces All namespaces should be marked out to avoid confusion with the Boost library which is used in many parts of the code. For example, `std::string` rather than simply `string`

2 Anatomy



BoxIn has 4 major components, the GUI, the Logic, the Storage and the Parser components. The components are divided in a way that follows two guiding principles. We also apply the Model View Controller pattern.

2.1 Model View Controller pattern

In BoxIn, the GUI component acts as both the View and the Controller. Users view all events through the GUI and the GUI is also responsible for taking care of all user interaction, including mouse clicks and information sent through the command line. More details are found in Section 3. The Model in the system is the Event class. More details are found in Section 5.

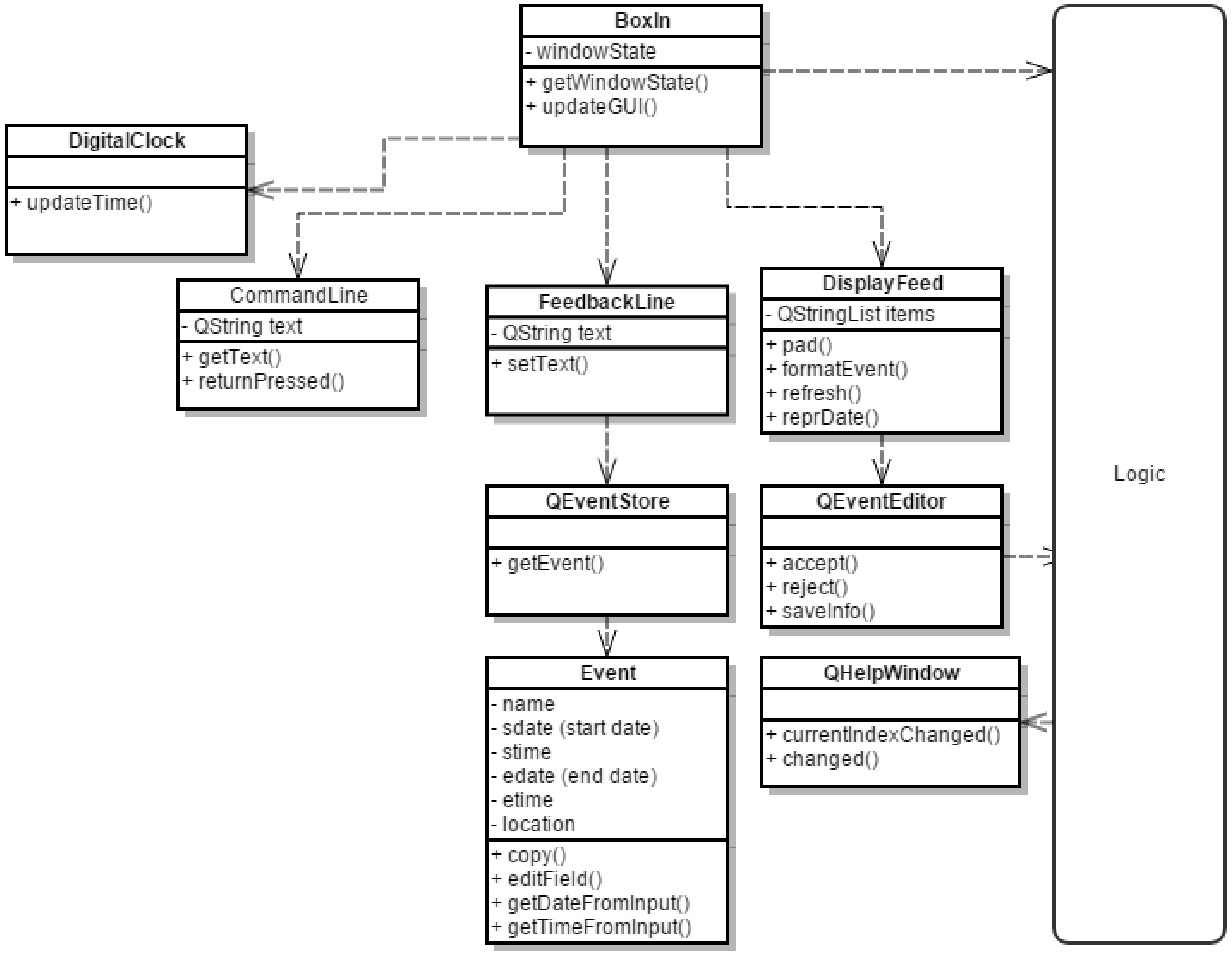
2.2 Separation of Concerns

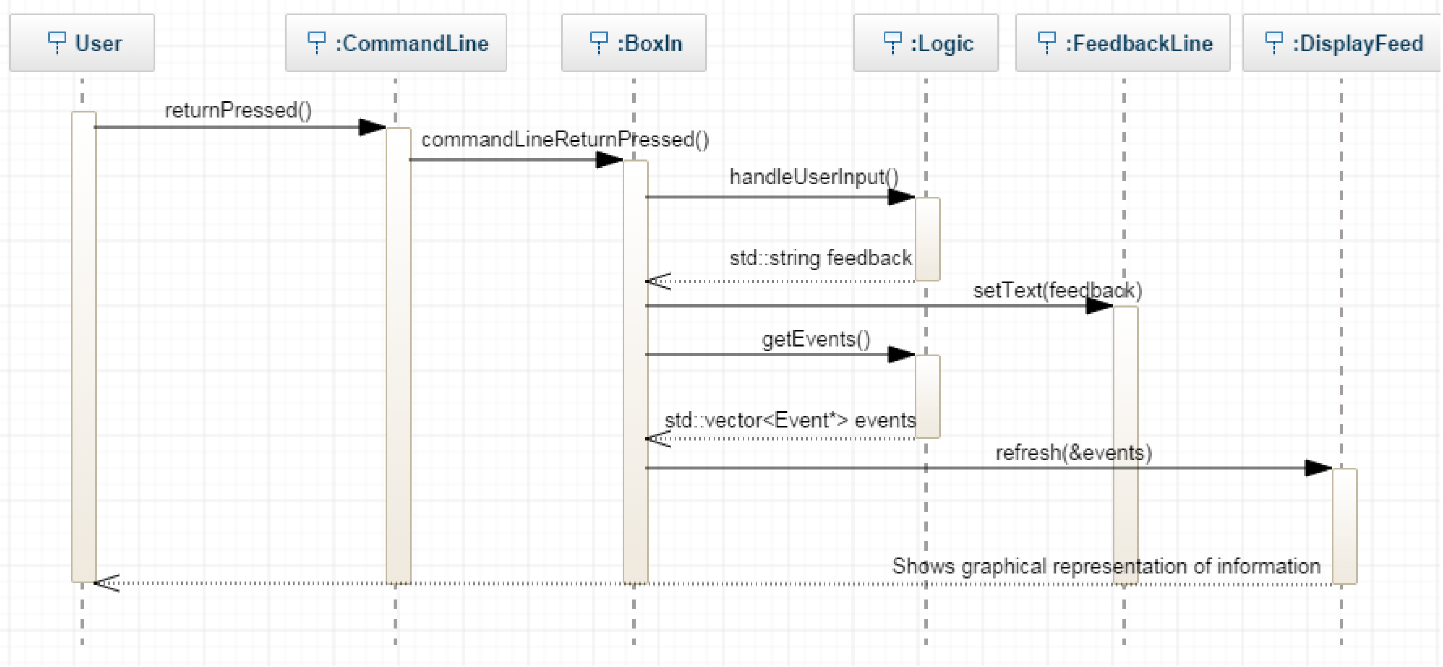
Each major component of the application handles it's own concerns. For example, the GUI or Storage components do not process any command line input, only the Logic and parsers do.

2.3 Law of Demeter

This law is particularly effected in the fact that the GUI and the Storage classes have no knowledge of each other - neither calls any functions of the other. Essentially, classes which do not have any direct relation with each other should not be calling each other.

3 GUI


BoxIn
- windowState
+ getWindowState()
+ updateGUI()
DigitalClock
+ updateTime()
CommandLine
- QString text
+ getText()
+ returnPressed()
FeedbackLine
- QString text
+ setText()
DisplayFeed
- QStringList items
+ pad()
+ formatEvent()
+ refresh()
+ reprDate()
QEventStore
+ getEvent()
QEventEditor
+ accept()
+ reject()
+ saveInfo()
Event
- name
- sdate (start date)
- stime
- edate (end date)
- etime
- location
+ copy()
+ editField()
+ getDateFromInput()
+ getTimeFromInput()
QHelpWindow
+ currentIndexChanged()
+ changed()
Logic


User
:CommandLine
:BoxIn
:Logic
:FeedbackLine
:DisplayFeed
returnPressed()
commandLineReturnPressed()
handleUserInput()
std::string feedback
setText(feedback)
getEvents()
std::vector<Event*> events
refresh(&events)
Shows graphical representation of information

The GUI component acts as both the controller and the view in the MVC pattern. The library used to is the Qt library. The documentation for the Qt library is available here: http://qt-project.org/doc/ The above diagrams give a graphical representation of how the `GUI` is designed in both the sequential calls on user actions and as a component. The GUI is divided into 7 components, discussed below

3.1 BoxIn (main window)

The `BoxIn` class is the main window. All sub-components found in this window should also have this window as a parent window. This class inherits from `QWidget`. The `BoxIn` class mainly acts as a container for most of the GUI components
The above sequence diagram shows the generic flow of events within the GUI component everytime the user presses the return key

	Attribute type	Name
Private attributes	Ui::BoxInClass	ui
	Logic	logic
	QAction*	minimizeAction
	QAction*	restoreAction
	QAction*	quitAction
	DigitalClock	clock
	QLabel*	nameLabel
	QLabel*	placeLabel
	QLabel*	startLabel
	QLabel*	idxLabel
	QSystemTrayIcon*	trayIcon
	QMenu*	trayIconMenu
	DisplayFeed*	displayFeedIdx
	QLineEdit*	commandLine

	Return type	Method
Public methods	void	displayFeedback(QString feedback)
	void	clearCommandLine()
	QString	readCommandLine()
	void	setVisible(bool visible)
	void	updateGUI()
	void	createComponents()
	void	setComponentSizes()
	void	setComponentColors()
	void	linkEvents()
	void	createTrayIcon()
	void	createActions()
	void	iconActivatd()
	void	changeEvent(QEvent *event);

Key API The following are the key methods that deal with functionality of the GUI

Method	Description
returnPressed()	This signal is by the Qt framework whenever the user presses return with the command line in focus
linkEvents()	This method is a setup method linking all the relevant signals to the respective slots for processing
commandLineReturnPressed()	The slot connected to the returnPressed() signal mentioned above. This starts the chain of events which result in event processing, by reading the line and sending it to the Logic component
readCommandLine()	Returns the string held by the command line. Called exclusively by commandLineReturnPressed()
clearCommandLine()	Removes any text in the commandLine. Also called exclusively by commandLineReturnPressed()
displayFeedback(QString feedback)	Calls setText() on the FeedbackLine to display feedback

Application icon The entire application has a predefined icon initialized in the constructor of BoxIn and packaged together with the application. The entire program can be minimized to the System Tray

Method	Description
createActions()	Creates the actions achievable by right-clicking the system tray icon. The actions supported are Minimize, Restore and Quit. These actions are then connected to the relevant slots to apply them
createTrayIcon()	Creates the icon itself and the supported menu, adding items in
setVisible()	Handles the minimize / maximize actions
iconActivated()	Handles the double-click event from the user

Fixed Size The `BoxIn` main window is of a fixed size (1000 x 600). This size is implemented as the constants `WIDTH_WINDOW` and `HEIGHT_WINDOW`

3.2 DisplayFeed

The `DisplayFeed` inherits from `QListWidget`. This widget contains data members of type QEventStore, which make up the View component of the MVC design pattern. `DisplayFeed` is designed in its' own constructor, without a `.ui` file. `DisplayFeed`'s purpose is to display all events the user wishes to view. At the moment, it simply displays everything.

	Return type	Method
Public methods	void	addItem(QListWidgetItem* item) (Inherited)
	void	setBorder()
	void	refresh(std::vector<Event*> *thingsToInclude)
	void	setItemColors()
	std::string	pad(std::string str, int spaces)
	std::string	reprDate(std::string date)
	std::string	formatEvent(Event* event)

Key API The DisplayFeed uses the following key methods to display input

Method	Description
refresh()	This is the function call made by BoxIn's commandLineReturnPressed(). This sets of the chain of other methods used to display the input. It creates a QEventStore pointer for each item to display and adds them to its' internal display
formatEvent()	This method takes an event, extracts its' data and turns into a equally spaced string representation of the event
setItemColors()	Changes the text color for the items - Red for past and undone items, purple for the latest change and the rest alternate between black and grey so as to differentiate rows
pad()	Adds whitespace or truncates overly long strings to give even sizing
reprDate()	Replaces dates with Today / Tomorrow for the matching dates

3.3 FeedbackLine

The FeedbackLine inherits from QLabel and is a simple instant feedback system for the user. It simply displays messages coming from the Logic component regarding the success or failure of user commands.

Key API This object only implements one important method

Method	Description
setText(QString feedback)	Sets the text on the feedback line to the given input

3.4 CommandLine

The `CommandLine` component of the GUI is the controller for majority of the system. Since the target audience prefers to use a command line style input, this becomes the main input interface. This component inherits from `QLineEdit`. It's key API is discussed at a wider level with BoxIn above

3.5 DigitalClock

`DigitalClock` is simply a digital clock displayed on the main window. It tells the time with a flashing colon. This object inherits from QLCDNumber

3.6 QEventStore

`QEventStore` is the wrapper class for the `Event` class implemented. This class allows `Event` objects to be added to the `DisplayFeed` so that a direct association is kept between the objects in the `DisplayFeed` and the `Event` objects themselves. `QEventStore` inherits from `QListWidgetItem`

Key API This class only implements one key method.

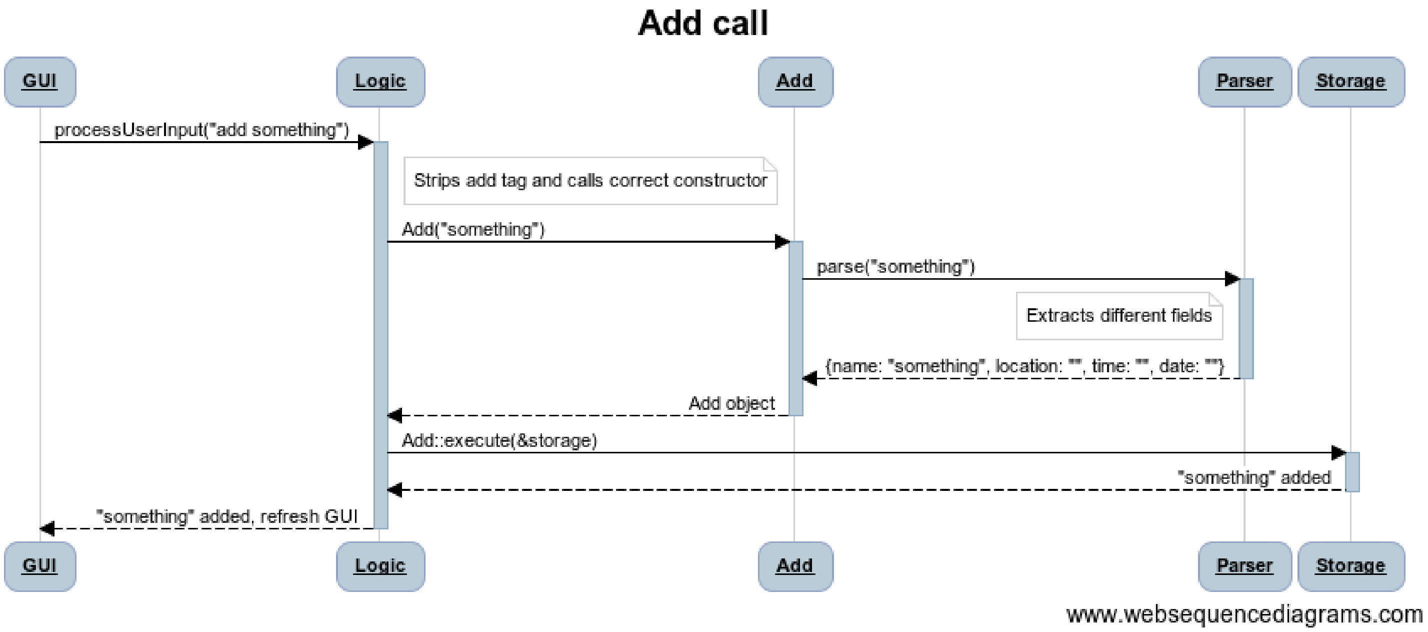
Method	Description
getEvent()	This function takes any information available from the stored event and returns a `QString` representation of it.

3.7 QHelpWindow

This window provides an interface for the user to view examples and various help regarding the usage of BoxIn. It is created by the `Logic` component when the user passes in the command `help`.

The QHelpWindow contains a QComboBox which the user uses to select a function he wishes to view help for, and the currentIndexChanged() signal is emitted and caught by the QHelpWindow to change the text contained in the QTextEdit

4 Logic


Add call
GUI
Logic
Add
Parser
Storage
processUserInput("add something")
Strips add tag and calls correct constructor
Add("something")
parse("something")
Extracts different fields
{name: "something", location: "", time: "", date: ""}
Add object
Add::execute(&storage)
"something" added
"something" added, refresh GUI
GUI
Logic
Add
Parser
Storage
www.websequencediagrams.com

The logic of the system is explained by the above sequence diagram as an example call to the add functionality of the system. The user input triggers the add command and the add command is passed to the storage to be executed.

4.1 Key API

The only call to the `Logic` component is made by the `GUI` when the `CommandLine::returnPressed()` signal is emitted. The function `Logic::handleUserInput(std::string)` will then proceed to process the input internally.

4.2 Controller

The controller(Logic component) is responsible for creating and executing commands. The user input is received by the controller and passed to the parser. The controller then receives the details of the user command from the parser and performs the action required (add, delete, edit, etc) detailed in the use cases in the appendix.

Method	Return Type
create(string input)	pointer
execute(string input)	pointer

4.3 Parser

The parser deciphers user input and creates the relevant command based on the user input. It these sends the command to the controller for execution.

Method	Return Type
parse(string input)	pointer

1 All handlers(add, delete, etc) must use the parse() method and the argument must accept the string parameter.

2 The parser should not modify the storage. If a task is supposed to be added, the parser should only generate the necessary fields of the task.

3 All arguments which have an index, the parser must obtain the relevant information of the task which is then returned to the handlers.

4 The exceptions thrown by the parser should be caught by the associated handler methods.

4.4 Commands

BoxIn currently recognizes the following user commands //add a class diagram here

Sort Within the Command class, the information will be sorted out so it would be easier to parse out later.

add	edit	delete
undo	view	

5 Parsers

The parsers for BoxIn deal with extracting information out of a user-given string. There are two parsers used in BoxIn - the SimpleParser (for generic items) and the TimeParser, which deals exclusively with times

5.1 SimpleParser

The SimpleParser deals more with dates and basic parsing. The following date formats are accepted: `DDMMYY, YYYYMMDD, YYYY/Jan/DD, monday, tuesday etc, today, tomorrow`.

	Type Name	Purpose
Type Definitions	InfoType	Determines the information to be extracted
	DateFormat	Matches the date format to the correct parsing algorithm

	Attribute type	Name
Private attributes	std::map<InfoType, std::string>	keywordMap
	std::map<std::string, std::string>	monthMap
	std::map<std::string, boost::date_time::weekdays>	dayMap

	Return Type	Method
Public methods	std::string	getField(std::string input, InfoType info)
	void	setupMaps()
	bool	isKeyword(std::string word)
	bool	isInteger(std::string text)
	boost::gregorian::date	convertToDate(std::string date)
	DateFormat	matchFormat(std::string date)
	bool	isNumericalFormat(std::string date)
	bool	isDayOfWeek(std::string day)
	bool	isToday(std::string day)
	bool	isTomorrow(std::string day)
	std::string	removeEscapeChar(std::string word)
	std::string	removeWhitespace(std::string text)

Key API The `SimpleParser` implements the following key API to extract data. Many of the functions are used to match dates

Method	Description
getField()	Retrieves the information matching the InfoType provided by the caller
isKeyword()	Returns `true` if the word given is a keyword
removeEscapeChar()	Returns the word removing the escape character **.**
removeWhitespace()	Trims trailing whitespace on a string
convertToDate()	Converts a string into a boost::gregorian::date object by matching formats using the other functions. If a match is not found, returns boost::gregorian::not_a_date_time
matchFormat()	Returns the format which matches the string it was given. If an appropriate format is not found, returns `FormatNotRecognised`

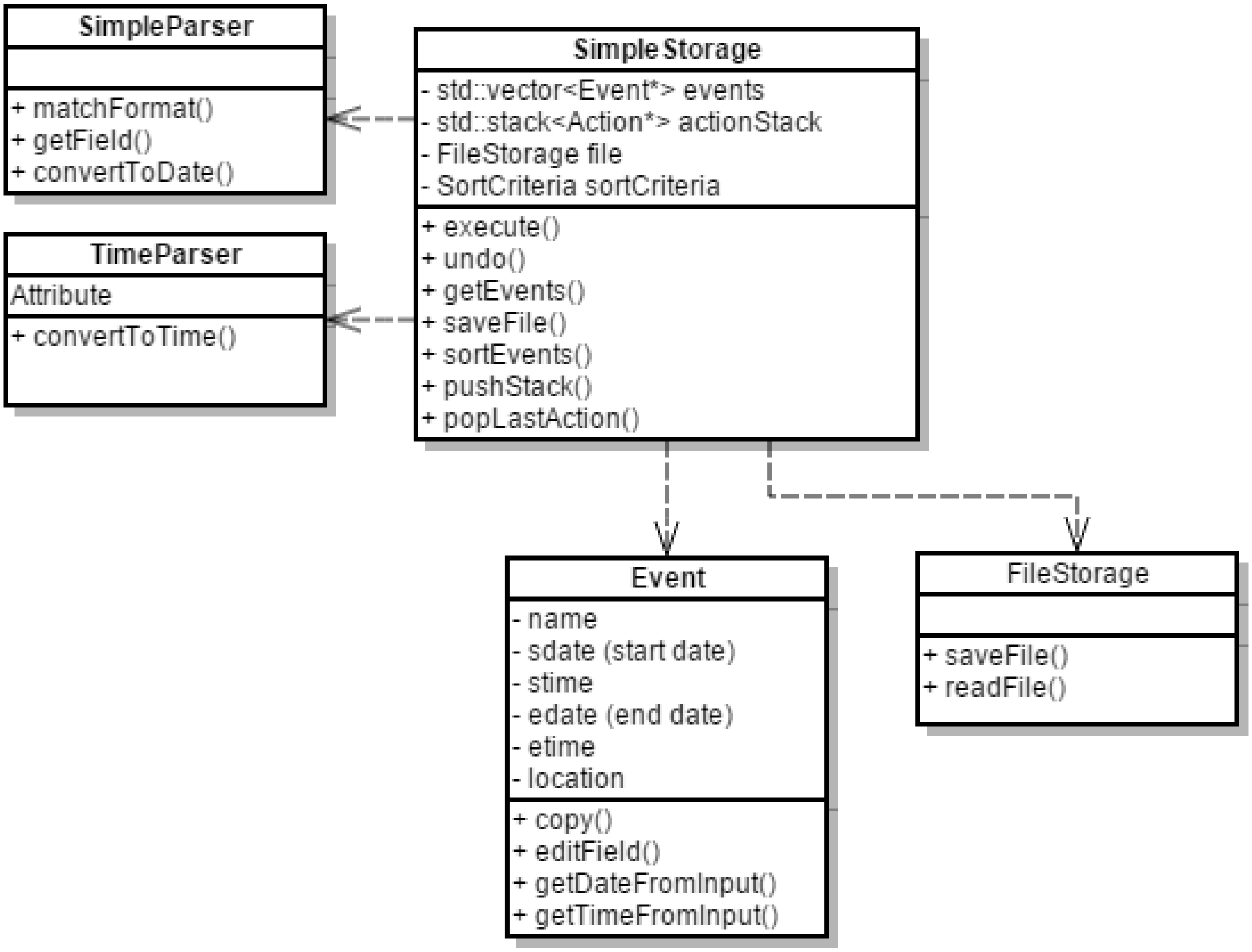
5.2 TimeParser

The TimeParser deals in particular with the parsing of times from user strings. The following formats are accepted: `HHMM, HH:MM`

Key API The TimeParser only implements one key method

Method	Description
convertToTime()	Converts a string into a boost::posix_time::ptime object by identifying it by length. Returns boost::date_time::not_a_date_time

6 Storage



The Storage class keeps both the internal representaion and the json representation of the Event classes. There are 3 major components, discussed below

6.1 SimpleStorage

SimpleStorage is the highest level structure of the Storage component of BoxIn. It handles all the interactions with the Logic component. The json storage file is declared as a constant here as `BoxInData.json`

Type Definitions

Type Name	Purpose
SortCriteria	Determines the sorting method

Private attributes

Attribute type	Name
std::vector<Event*>	events
std::stack<Action*>	actionStack
FileStorage	file
SortCriteria	criteria

	Return Type	Method
Public methods	std::vector<Event*>	getEvents()
	void	pushStack(Action* action)
	Action*	popLastAction()
	void	sortEvents()
	std::string	execute(Action* action)
	std::string	undo(Action* action)
	void	saveFile()

Key API SimpleStorage implements the following key API to store and undo events. These are called from the Logic component. In addition, in the initializer, the FileStorage is read.

Method	Description
getEvents()	Returns all events found in Storage
pushStack()	Adds an executed Action to the stack of things done
popLastAction()	Removes the most recent Action and returns it. Used in `undo`
sortEvents()	Arranges all the events in `events` by their end date
execute(Action* action)	Executes action onto `events` (see Action class for details)
undo(Action* action)	Undo-es action from `events`
saveFile()	Calls the FileStorage class to write all events to the json file

6.2 Event

The Event class stores data of one Event. It is the model class being passed around the entire system and is therefore included in almost every component.

	Attribute Type	Name
Private attributes	std::map<std::string, Field>	fieldMap (used in generic edit)
	std::string	name
	boost::gregorian::date	sdate (start date)
	boost::gregorian::date	edate (end date)
	boost::posix_time::ptime	stime
	boost::posix_time::ptime	etime
	std::string	nonformattime
	std::string	location
	SimpleParser	parser
	TimeParser	timeParser
	int	idx
	bool	recent
	bool	done

Public methods

Return Type	Method
Event*	copy()
std::map<std::string, Field>	setupMap()
std::string	getName()
std::string	getStartDate()
std::string	getEndDate()
std::string	getStartTime()
std::string	getEndTime()
std::string	getLocation()
int	getIdx()
bool	isRecent()
boost::posix_time::ptime	getPosixStartTime()
boost::posix_time::ptime	getPosixEndTime()
void	editField(std::string field, std::string newValue)
void	setName(std::string newName)
void	setStartDate(std::string newDate)
void	setEndDate(std::string newDate)
void	setStartTime(std::string newTime)
void	setEndTime(std::string newTime)
void	setLocation(std::string newLocation)
void	setIdx(int newIdx)
void	removeRecent()
void	setDone(bool newValue)
bool	getDone()
boost::gregorian::date	getDateFromInput(std::string date, std::string time)
boost::gregorian::date	getDateFromInput(std::string date, std::string time, std::string preDate)
boost::posix_time::ptime	getTimeFromInput(boost::gregorian::date date, std::string time)
std::string	repr()

Key API The Event class implements the following key API which helps it store data accurately.

Method	Description
editField()	Identifies the correct field to edit and from there calls the appropriate function. This allows for a more generic edit call
repr()	Gives a textual representation of the data in the event. Used for testing purposes, but not in the main program
getDateFromInput()	Makes use of the SimpleParser to convert a date to the appropriate date, failing which will put the default date (today if none is specified) if there is a time associated to the date. Used in the constructor, and overloaded for the case where there is no default date
getTimeFromInput()	Returns the appropriate boost::posix_time::ptime object that tallies with the associated date and time given. Used in the constructor

6.3 FileStorage

The FileStorage class deals exclusively with reading and writing files to and from a json file.

Private attributes

Attribute Type	Name
std::string	filename

Public Methods

Return Type	Method
void	saveFile(std::vector<Event*>)
void	writeEvent(json_spirit::Array &eventArray, Event* event)
std::vector<Event*>	readFile()
Event*	readEvent(const json_spirit::Object& obj, unsigned int idx)

Key API All methods are critical to the correct reading and writing of the json file.

Method	Description
saveFile()	Takes in a vector of events and writes all of them to the json file
writeEvent()	Takes in an json_spirit Array and a single Event pointer and adds that pointer to the Array
readFile()	Returns a vector of Event pointers stored in the json file
readEvent()	Reads and creates a single Event pointer from the json file as a json_spirit Object

7 Appendix A

7.1 Use cases

Name	UC01:Add a new task
Description	To add a new task
Precondition	BoxIn is currently running
Basic course of event	1. User indicates the event that they want to add (Name, Date, Time, Place) and it has to be in this specific order 2. BoxIn will give feedback indicating that the event has been added
Alternative path	1. One of the parameter is missing : 1a. BoxIn responds that a parameter is missing and ask the user to try again
Post Condition	A new event is added and saved.

Name	UC02: Delete a task
Description	To delete an existing task.
Pre Condition	BoxIn is already running.
Basic Course of Event	1. User types the command to delete an already existing task. 2. The program deletes the task as per the user's command.
Alternative Path	1. If the task does not exist, the program displays the relevant message. 2. Prompts the user to re-enter the command.
Post Condition	The task is updated

Name	Edit a task
Description	To edit an existing task
Pre Condition	BoxIn is already running
Basic Course of Event	1. User types the command to edit an already existing task and specifying the relevant fields to be changed 2. The program edits the task as per the user's command
Alternative Path	1. If the task does not exist, the program displays the relevant message. 2. Prompts the user to re-enter the command.
Post Condition	The task is updated

Name	UC04: Undo action
Description	To undo the previous command
Pre Condition	BoxIn is already running.
Basic Course of Event	1. User types the command to undo the previous command.
Alternative Path	1. If the previous action does not exist, the program displays the relevant message. 2. Prompts the user to re-enter the command.
Post Condition	The task is deleted.

Name	UC05: Search task
Description	To search a task
Pre Condition	BoxIn is already running.
Basic Course of Event	1. User types the command to search for a task. 2. The result is displayed.
Alternative Path	1. If the syntax does not match, prompts the user to re-enter the command. 2. If the task does not exist, relevant message is displayed.
Post Condition	The task is undone.

Name	UC06: Sort task
Description	To sort tasks
Pre Condition	BoxIn is already running.
Basic Course of Event	1. User types the command to sort the tasks. 2. The program displays the task in sorted order.
Alternative Path	1. Sort criteria is not specified and tasks are sorted using the default order.
Post Condition	-

Name	UC07: Display task
Description	To display a task
Pre Condition	BoxIn is already running.
Basic Course of Event	1. User types the command to display the task. 2. The program displays the task.
Alternative Path	1. The relevant task does not exist. 2. Program prompts the user to re-enter command.
Post Condition	-

8 Appendix B: Setting up

To set up, you will need Windows Operating System, VS2012, Boost, and Git.

8.1 Qt 5.3.1

The Visual Studio plugin for Qt. You can find it at http://qt-project.org/downloads. Scroll to the bottom of the page and look for qt-vs-addin-1.2.3-opensource.exe. Then open Visual Studio. The top bar should show QT5 -> QT Options. Make sure that the correct version of QT is selected. Install Qt to C:/Qt

8.2 Boost

Boost libraries - version 1.57, vc2012 (vc11.0), 32 bit. You can find it at
`http://tinyurl.com/BoxInDevBoost`
and install to `C:/Boost`

8.3 3.Visual Studio plugin

The Visual Studio plugin for Qt. You can find it at http://qt-project.org/downloads. Scroll to the bottom of the page and look for qt-vs-addin-1.2.3-opensource.exe. Then open Visual Studio. The top bar should show QT5 -> QT Options. Make sure that the correct version of QT is selected.

8.4 4.Git

You can download git from Github.com and register as a member, then clone the software and open the file and it should work out!

9 Appendix C: Testing Instructions

Both unit tests and system tests are held in the same project file `(UnitTest)` and all code is contained in `unittest.cpp`, with headers in `unittest.h`

Setting up Change the directory of the include directory to match that of the project directory

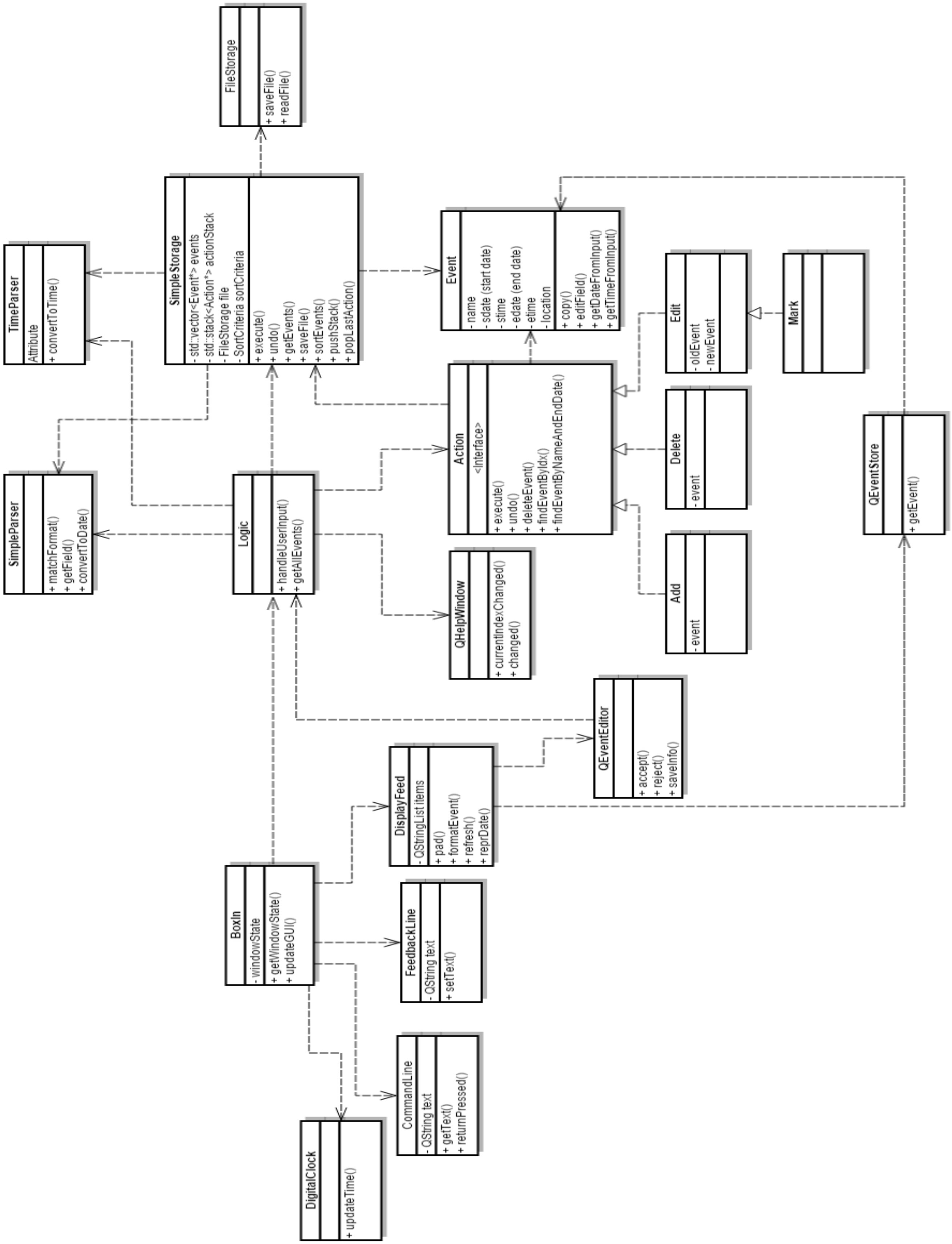
Prerequisites to testing To create any unit test that would include any GUI components, it is necessary to first start a QApplication. Refer to the following code example. `foo` is a defined function used to create the integer reference, but has no further use. The application need not be used further.

```
int& argc = foo();
char** argv;
QApplication app(argc, argv);
```

Testing policy All newly added code should come together with a series of unit tests to prove it works for both the general and the borderline cases. Also, before committing any new code, it must clear all system level tests.

Note on Qt It is possible to get the system to run events by using the emit function to generate te signals that normally would be generated from user activity

10 Appendix D: Full Class Diagram


FileStorage
+ saveFile()
+ readFile()
SimpleStorage
- std::vector<Event*> events
- std::stack<Action*> actionStack
- FileStorage file
- SortCriteria sortCriteria
+ execute()
+ undo()
+ getEvents()
+ saveFile()
+ sortEvents()
+ pushStack()
+ popLastAction()
TimeParser
Attribute
+ convertToTime()
Event
- name
- sdate (start date)
- stime
- edate (end date)
- etime
- location
+ copy()
+ editField()
+ getDateFromInput()
+ getTimeFromInput()
Edit
- oldEvent
- newEvent
Mark
Action
<Interface>
+ execute()
+ undo()
+ deleteEvent()
+ findEventByIdx()
+ findEventByNameAndEndDate()
Delete
- event
Add
- event
QEventStore
+ getEvent()
SimpleParser
+ matchFormat()
+ getField()
+ convertToDate()
Logic
+ handleUserInput()
+ getAllEvents()
QHelpWindow
+ currentIndexChanged()
+ changed()
QEventEditor
+ accept()
+ reject()
+ saveInfo()
DisplayFeed
- QStringList items
+ pad()
+ formatEvent()
+ refresh()
+ reprDate()
BoxIn
- windowState
+ getWindowState()
+ updateGUI()
FeedbackLine
- QString text
+ setText()
CommandLine
- QString text
+ getText()
+ returnPressed()
DigitalClock
+ updateTime()