BoxIn
Developer Guide

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

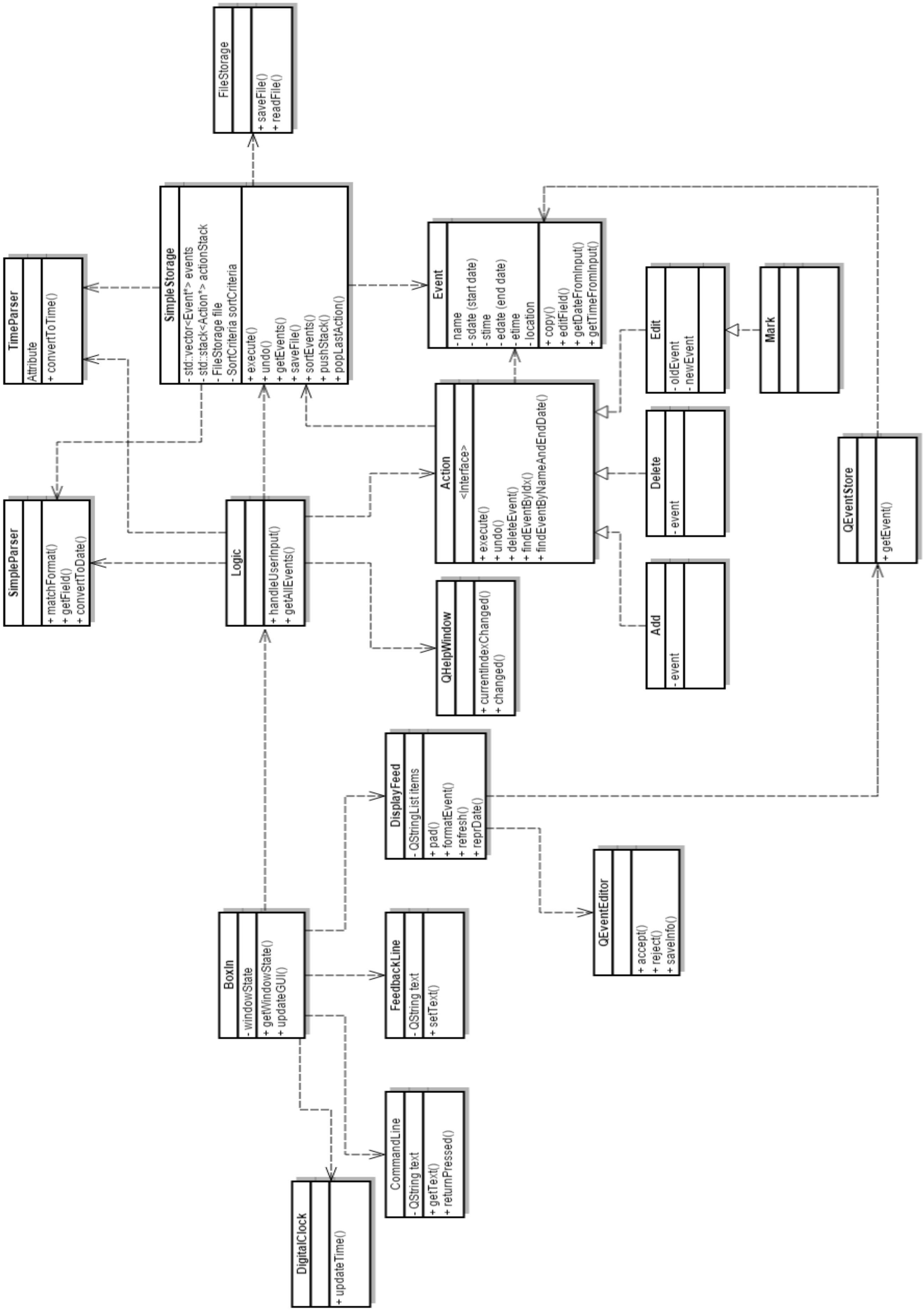
Memory Management Coding in C++ requires efficient use of memory. Memory should be freed if its not in use any longer.

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

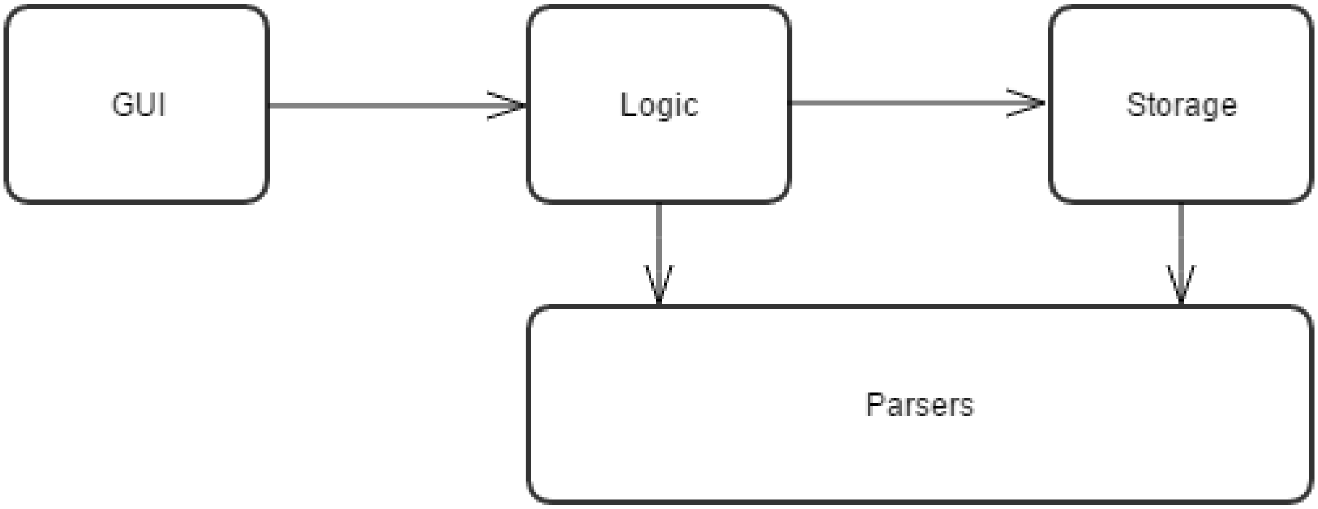
Coding standard BoxIn code follows the coding standard listed out here: C++ coding standards.

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`

1.1 Full class diagram



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.

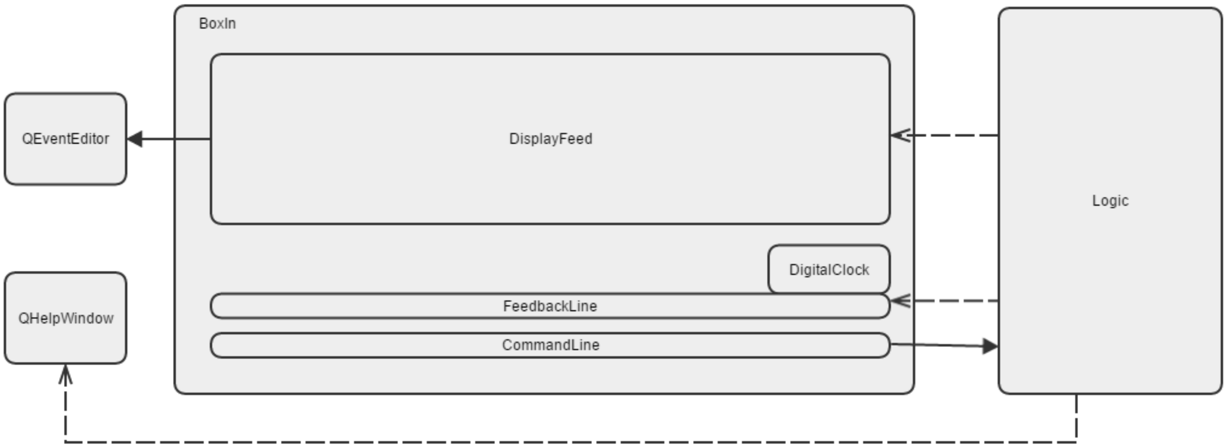
2.2 Separation of Concerns

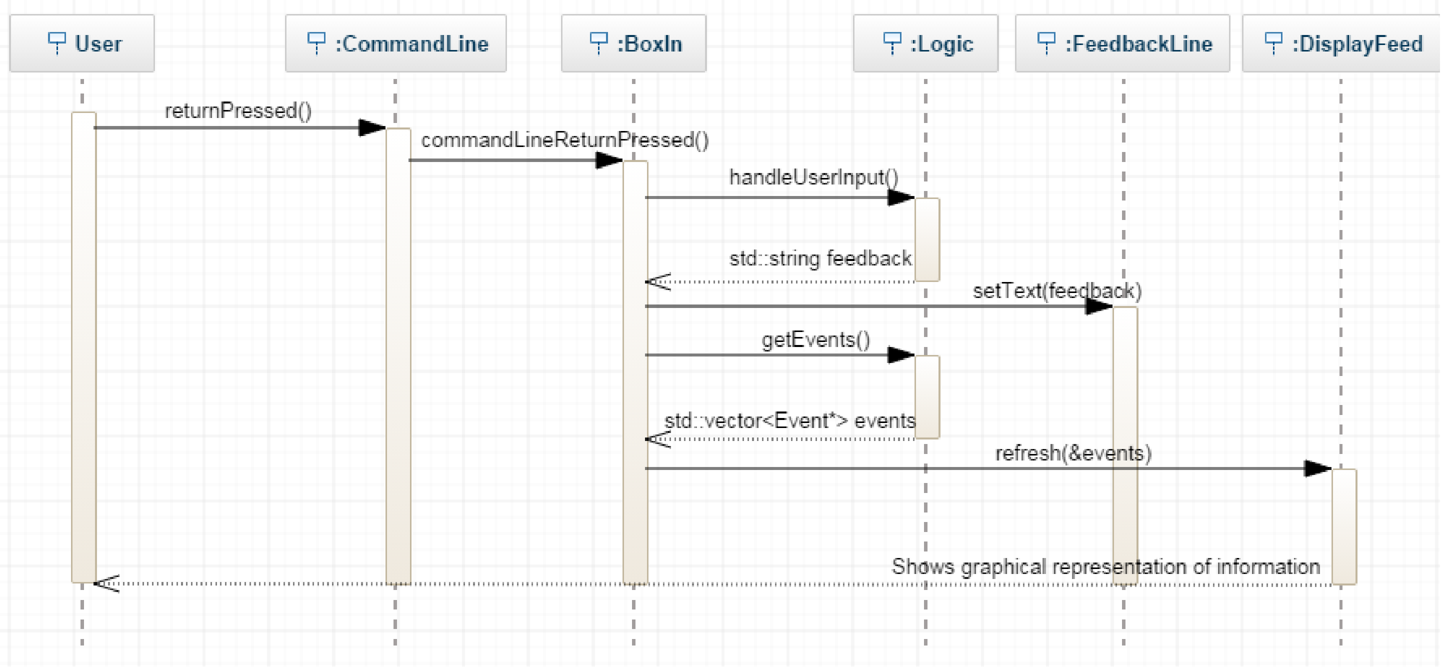
Each major component of the application handles it's own concerns. For example, the GUI or Storage components do not process any command line input, only the Logic and parsers do.

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





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

Private attributes

Attribute type	Name
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

Public methods

Return type	Method
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);

Application icon The entire application has a predefined icon initialized in the constructor of BoxIn

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

Minimize to System Tray BoxIn also has a Windows system tray icon that it can be minimized to. This code is initiated in the constructor and makes use of BoxIn::createActions(), BoxIn::createTrayIcon() and
BoxIn::linkEvents() to achieve

BoxIn::commandLineReturnPressed() This slot captures the signal from the CommandLine and calls the Logic API for processing the user input by passing on a std::string containing the user's input.

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.

DisplayFeed::refresh(&std::vector<Event*>) This is the function called by the BoxIn main window after every user input. refresh(&std::vector<Event*>) basically re-loads the entire vector of events and converts the underlying information into string representations that can be understood by the reader using the QEventStore::repr() function. Furthermore, the QEventStore is linked to the event pointer itself, and emitting the doubleClicked() signal from the DisplayFeed will create a QEventEditor to edit the data

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.

FeedbackLine::setText(QString) Changes the current display text to whatever was in the QString. This is the only method of note for this class.

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.

CommandLine::returnPressed() This is the signal emitted anytime the user presses the enter key with the CommandLine in focus. This is the trigger event for all data processing, and it calls the Logic Controller component to handle whatever input was given by the user.

CommandLine::getText() This is called when the previous signal is emitted, and the text is retrieved and passed on as a QString.

3.5 DigitalClock

`DigitalClock` is simply a digital clock displayed on the main window. It tells the time with a flashing colon, and will be used for further extensions in future.

3.6 QEventStore

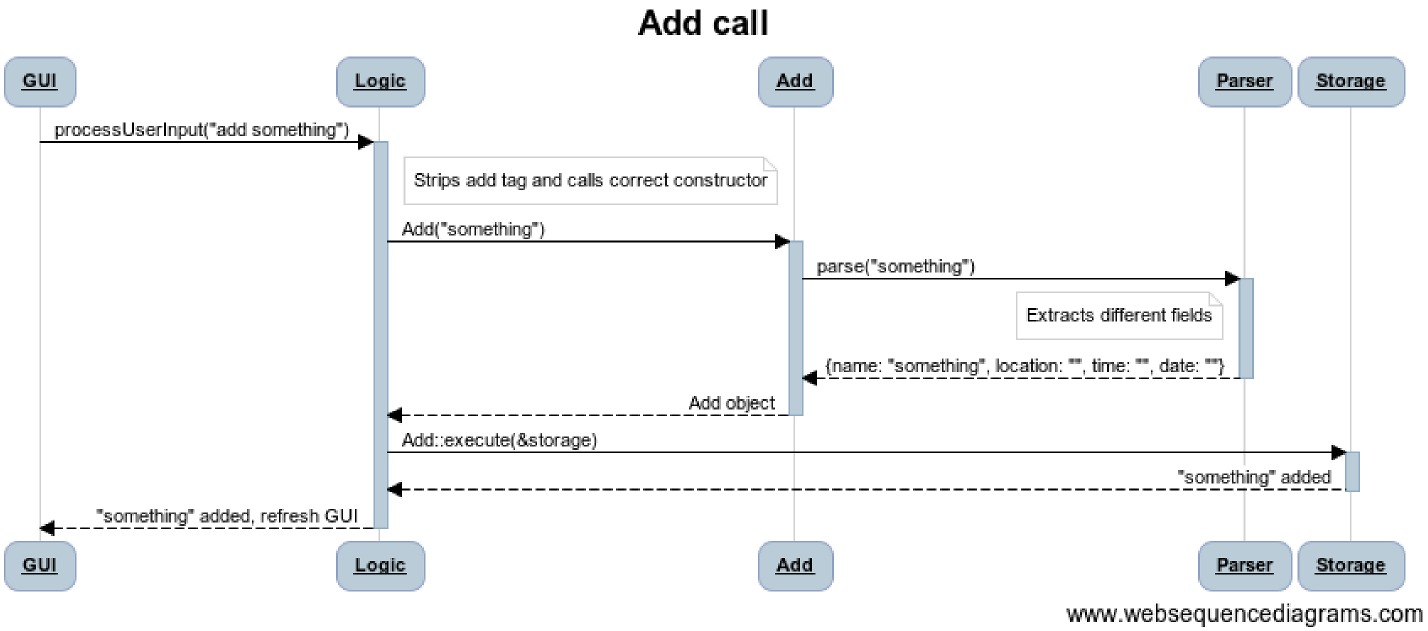
`QEventStore` is the wrapper class for the `Event` class implemented. This class allows `Event` objects to be added to the `DisplayFeed` so that a direct association is kept between the objects in the `DisplayFeed` and the `Event` objects themselves. `QEventStore` inherits from `QListWidgetItem`

QEventStore::repr() 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. The GUI will process the user input into the logic, and it will parse to the controller for the add which later stores it into the storage.

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 Storage

The Storage class has only one purpose: to write all data stored in the Tasklist class into a file, and to retrieve the data in the file for BoxIn to use in the next session.

5.1 Event

The Event class stores data created by the user. All of the data created by the user will be included here.

set-parts The different set functions are vectors that stores the different sections of the user's data. `setName` for example, sets the name of the event to be inside a name folder. This allows the parser to use the data later on in a more efficient way.

setFinish The `setFinish` function of the Event class saves the data completely when the user puts in all the parameter that is needed. It uses a vector as the structure to store all the information the user needs.

5.2 EventList

The EventList implements a list based on the content the user inputs as data. It also returns the functions such as returning the data as vectors and sorting the data.

eventList::contains(Events event) This method under event list returns a boolean of true or false and it works with logic to provide feedback if the event class is stored inside storage or not. If the event is already in the list, it can be use to edit later on.

eventCompare The function of eventCompare allows the event storage class to determine which data goes to which category, it allows the data to sort based on date, location, time to store the data in a specific order.

5.3 ActionStack

The ActionStack Class saves the different events into files as well as returns the data when it is needed to be edited.

undo When the user wants to undo a mistake that they have, they can undo using the undo method. This allows the stack to be popped off and be erased from the storage data. The stack will be popped off.

Return Type	Method
Vector<Event>	add(stringName)
Bool	delete(vector<Event> event)

6 Appendix A

6.1 Use cases

7 Appendix B:setting up

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

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

7.1 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

7.2 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`

7.3 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.

7.4 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!

8 Appendix C: Testing Instructions

8.1 Unit Tests

8.2 System Tests

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	-