QWIK.

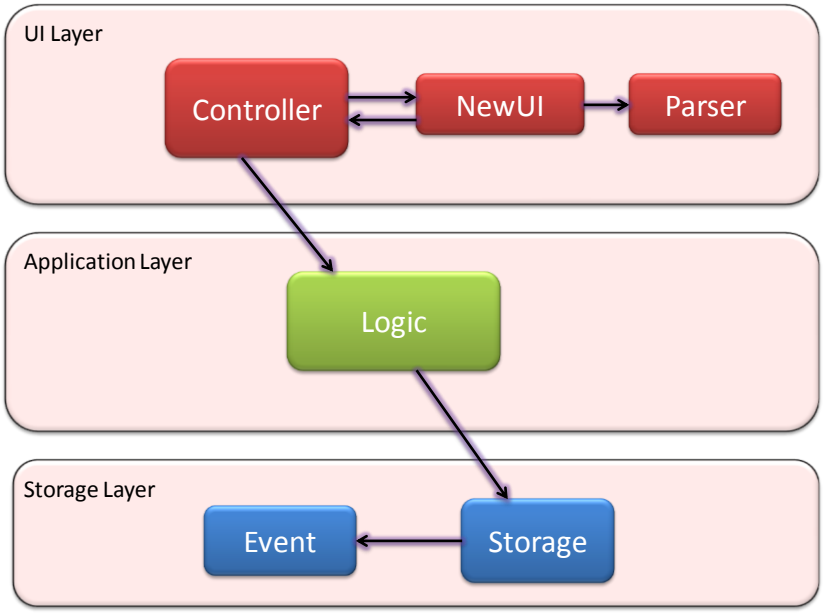
Version 0.2

DEVELOPER'S GUIDE

Table of Contents

- **Software Architecture**
- **Sequence Diagrams & Class Diagrams**
- **Notable Algorithms**
- **Sample Codes**
- **Instructions for testing**

SOFTWARE ARCHITECTURE V0.2



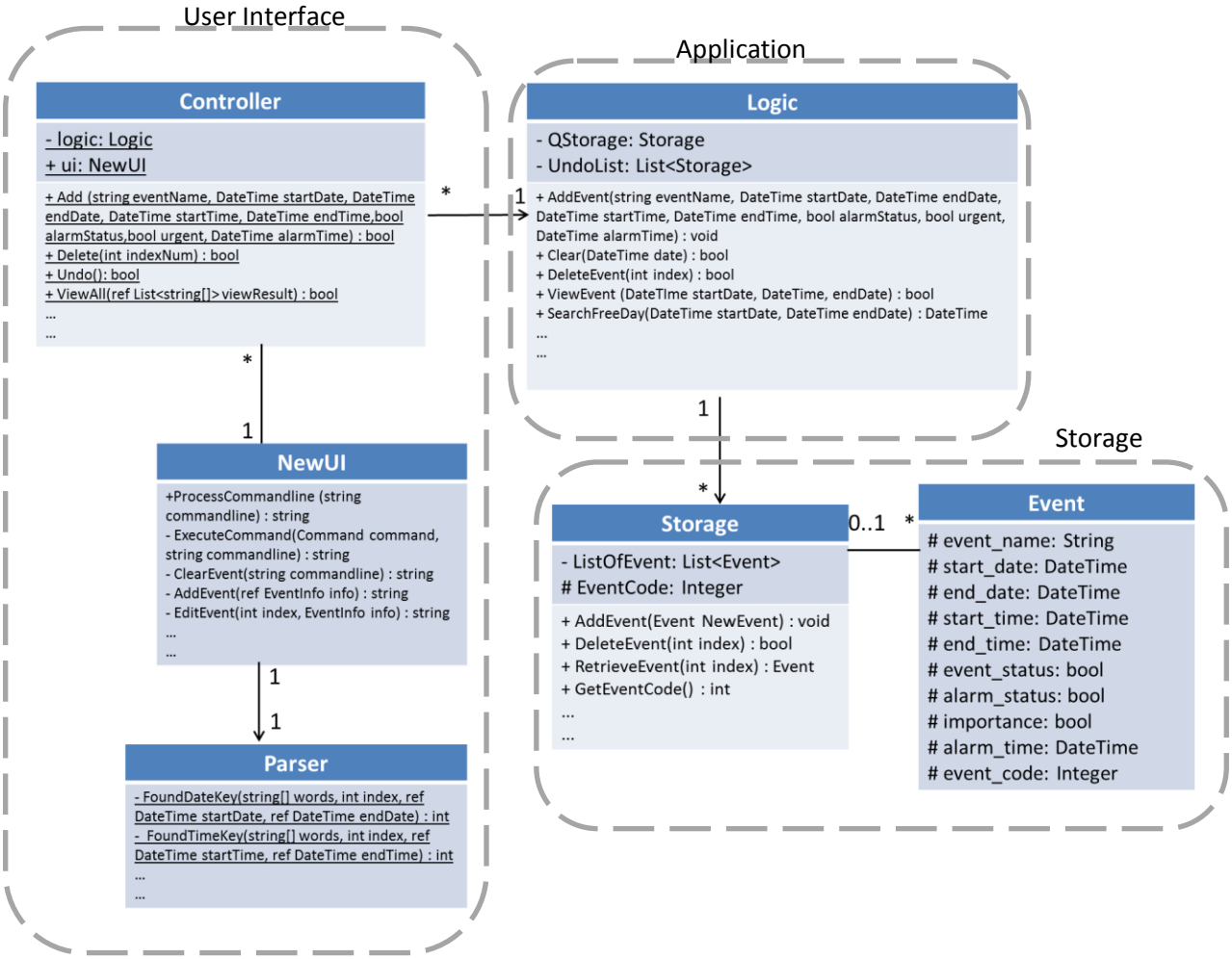
Qwik. is a easy to use software that realises 3 simple layers of architecture as shown in the diagram.

User Interface layer consists mainly of the classes that displays the Windows Form to the user with the exception of the 'Parser' class. This class takes in raw string commands and extracts out the key data for the actual UI to pass into the Application layer. The NewUI class follows a singleton pattern that is, it is only instantiated to one object across the program. This is because there is only one user using the software and hence multiple interface classes are not required.

Application layer consists of 1 class which processes the data and performs actual editing on them. The main Logic class does everything from adding delete event to searching and viewing events. It does not contain any form of storage. The Logic class is follows a singleton pattern. This is useful because only this one object is needed to coordinate actions across the program.

Storage layer consists of 1 main Storage class. Storage maintains a list of Event objects, of which it can also create when asked to by the Logic class in the Application layer.

The following class diagram illustrates the above description further:



The interface layer has 3 main components: The NewUI, Parser and Controller classes.

NewUI: This is the actual interface where the code for the form is. It controls what is displayed to the user and gets the input from the user to process.

Parser: Interprets what the user has entered and returns the decoded message to the NewUI class. The Parser class is important because Qwik allows the user to enter information in any format. For example, to add an event, the user can enter “meeting tmr at 3pm urgent” or “meeting 3pm tomorrow urgent alarm 2 pm”.

Controller: It is the sole link between the interface layer and the application layer. The controller is the class that instantiates and runs the NewUI application. It is the class that allows instantiation of all other classes. The NewUI and Logic classes were made static so that there is only one instance of the form exists and there is only one way to edit the list of tasks.

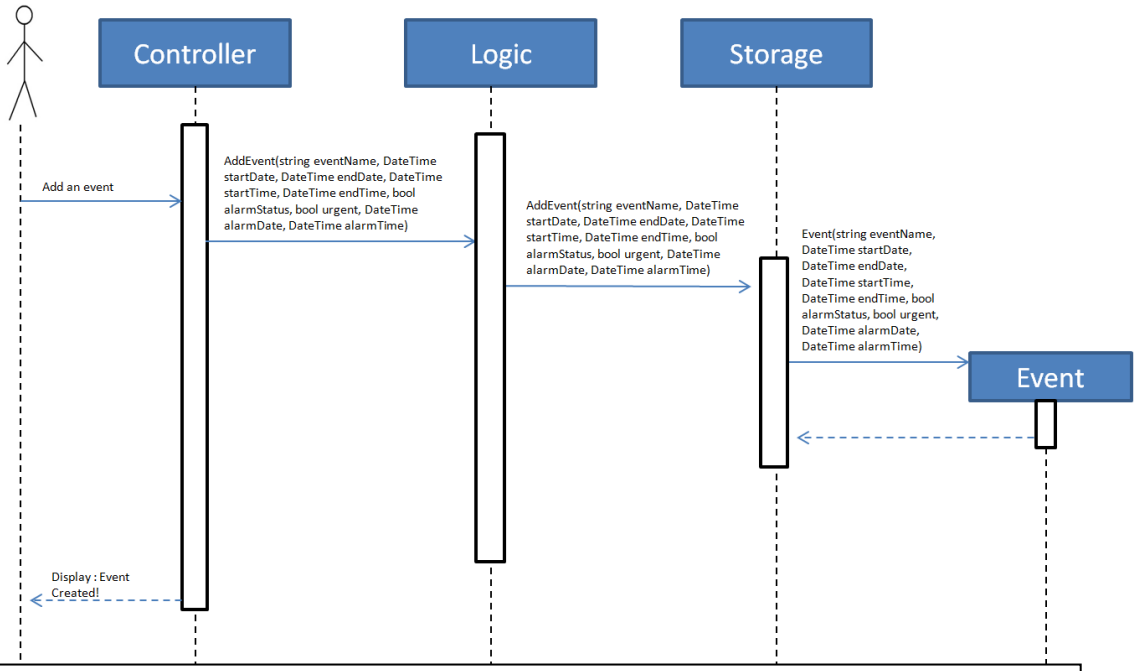
The application layer consists of the Logic class.
Logic: This class is static and does the actual editing and retrieval of tasks that the user has. There is a need to create a storage list in the Logic class because our software has a undo feature

(using the keyboard shortcut ‘Z’ or typing in “undo” in the commandline) which allows for multiple undo.

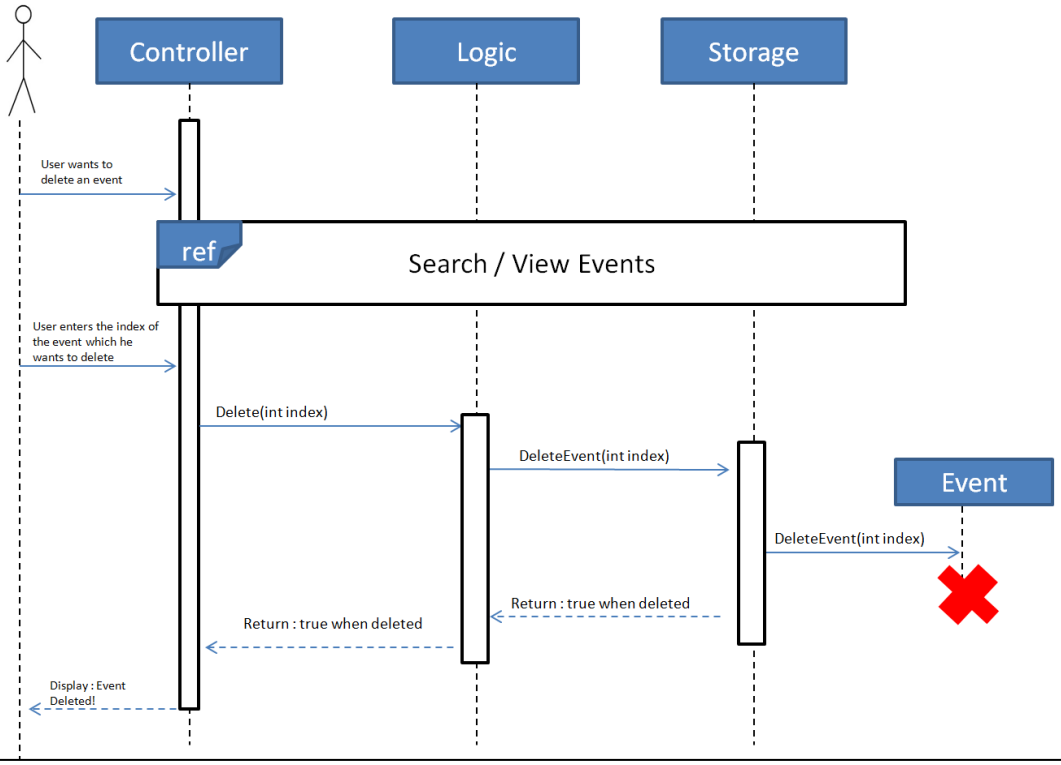
The storage layer consists of the Storage class and the Event class.
Storage: Storage layer implements ISerializable interface so that events in storage can be serialized and written to xml file (readable file storage). This allows us to store the events in a permanent storage, in an xml file.

Event: It is the basic unit of the Storage class.There is an event_code attribute for every event. The event_code is like the barcode of a product and is unique for each event. We use this to differentiate between events in several functions including searching and retrieval of events.

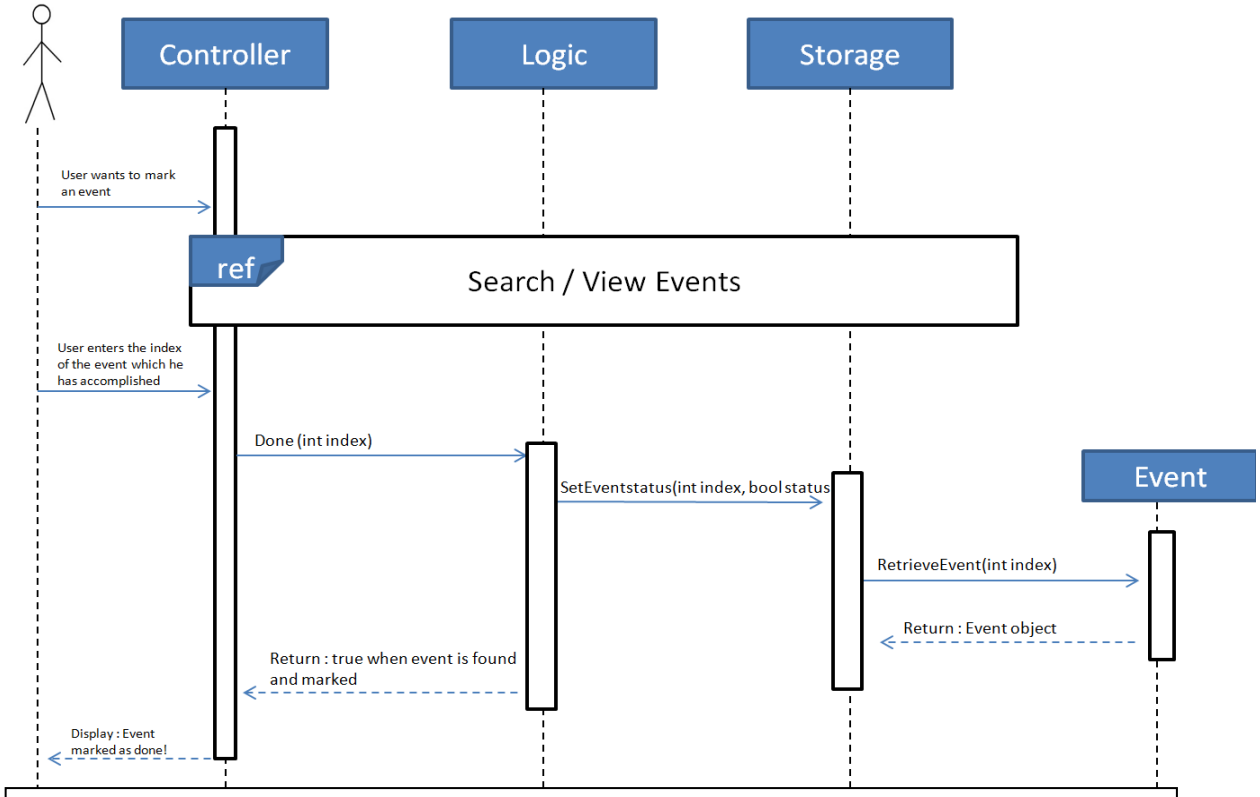
SEQUENCE DIAGRAMS



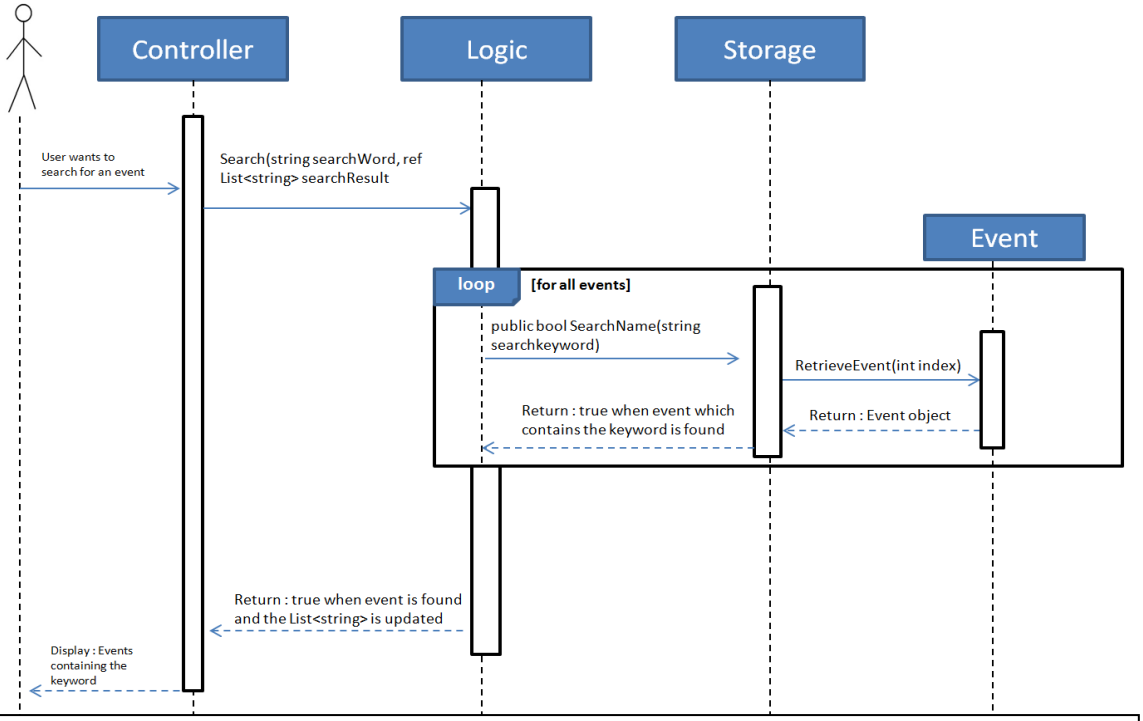
AddEvent SD: The user wanted to add an event and he key in the event with its particular including date and time.
The Controller class process the string of words and call the add event function.
The Logic class then calls the AddEvent function.
The Storage class calls the constructor of Event class.
An Event object has been created and it is’ stored’ inside the storage.



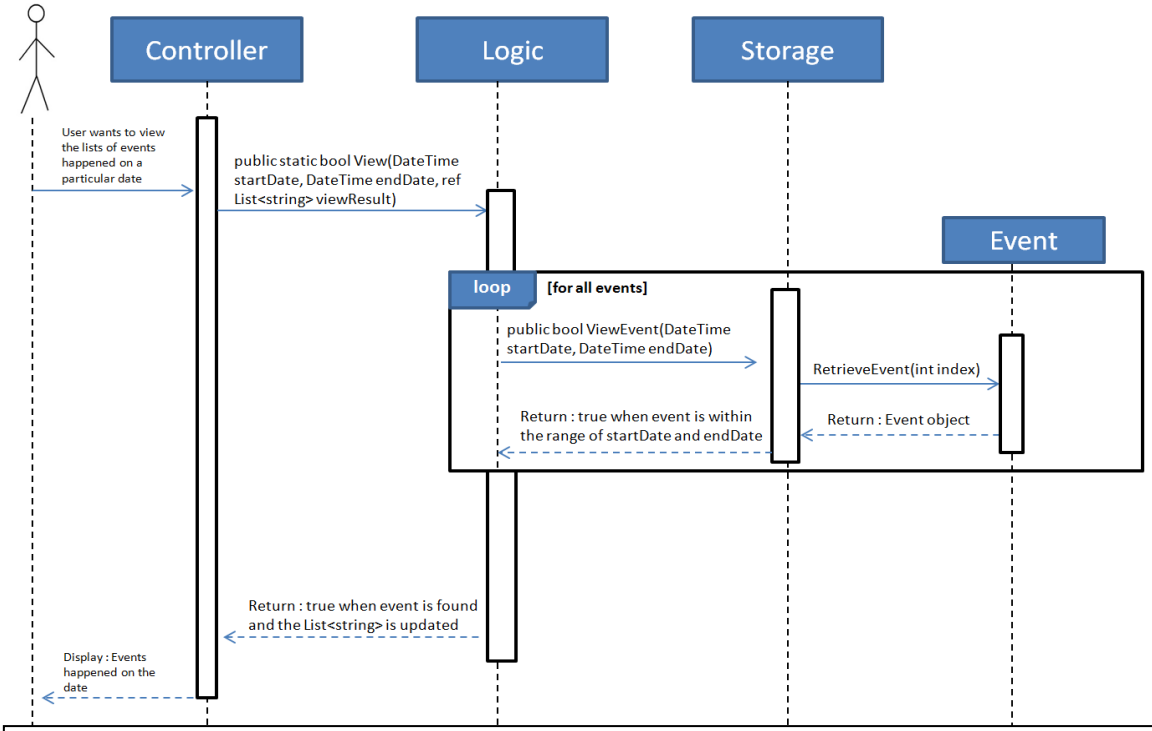
DeleteEvent SD: The user wants to delete an event.
The user will search or view the events, or by default, which shows list of events on the panel of QWIK.
The user enters the index of the event which he wants to delete.
The Controller class calls the Delete function, followed by Logic class calling the DeleteEvent function.
The Storage class then calls the DeleteEvent (which calls the destructor of the Event class), and update the storage.



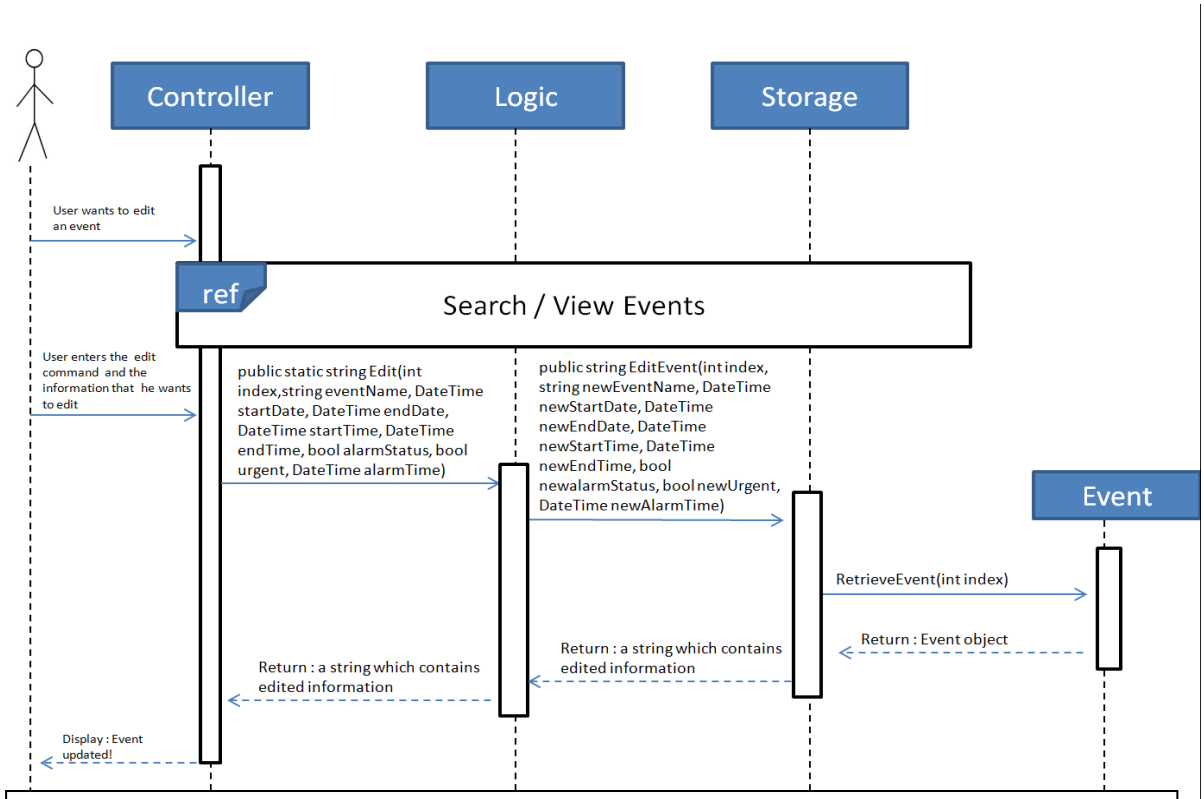
Mark Event SD: The user wants to mark an event after he has accomplished his task.
The user will search or view for the events, or by default, which shows list of events on the panel of QWIK.
The user enters the index of the event which he wants to mark.
The Controller class calls the Done function, followed by Logic class calling the SetEventStatus function.
The Storage class then calls the RetrieveEvent function, and the event with the particular index is returned back to the storage, hence modifying the 'status' attribute of the event object.



Search SD: The user wants to search for a particular event, knowing/guessing the keyword contained.
The Controller class will call the Search function.
The Logic class will search for all the events that are stored in the list.
For every iteration, Storage class will retrieve an Event object.(in our context is to go through the element 1by1 in the List.)
The Logic class will compare the keywords keyed in by user with the EventName of the Event object.
The Logic class will return a true(if at least one event is found) to the Controller class, and update the List<string>
The Controller class will display the search results to the user.



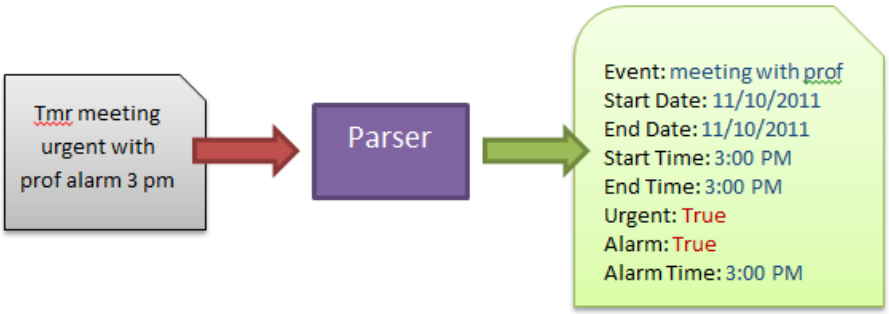
View Event SD: The user wants to key in the date that he wants to view.
The Controller class will call the View function.
The Logic class will call the ViewEvent function.
For every iteration, Storage class will retrieve an Event object.(in our context is to go through the element 1by1 in the List.)
The Logic class will search for all the events by comparing the starting dates and ending dates of the events.
The Logic class will return a true(if at least 1 event lies on the date) to the Controller class, and update the List<string>
The Controller class will display the all the events(can be none) of the particular date to the user.



Edit Event SD: The user wants to edit an event, but the events are chosen from the list either from default or results of search and view.
The user chooses the event that he wanted to edit, and he key in the information that he wants to edit.
Controller class calls the Edit function, and Logic class calls its Edit function.
Storage class will call the RetrieveEvent function, and the particular Event object is returned.
The attributes(desired by the user) of the Event object is modified, and string of edited information is passed from Storage class to Logic class, and from Logic class to Controller class.
Events updated information is displayed.

NOTABLE ALGORITHMS

To make our "Qwik" shortcut User Interface user-friendly and intelligent, a powerful Parser class is created to effectively extract various data from a single command line entered by user.



Firstly, we split up the sentence into words (string array).

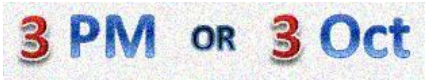
We navigate through the whole line of command using array indices.

2 ways to identify different types of data

1. Search for keywords

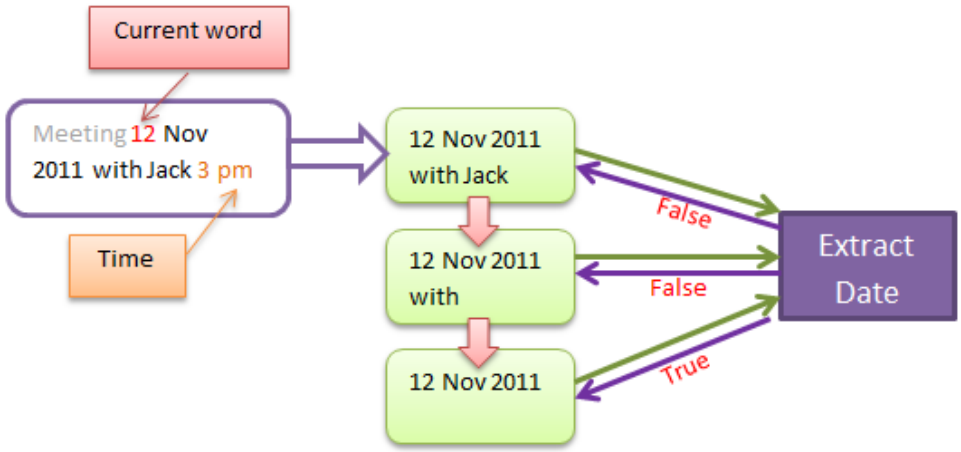
```
private static string[] dateKey = { "by", "before", "due", "on" ,"to","from"};
private static string[] timeKey = { "at", "from", "to", "by", "am", "pm" };
private static string[] otherKey = { "urgent", "alarm", "this", "next", "week" };
private static string[] days = { "mon", "tue", "wed", "thur", "fri", "sat", "sun", "tmr", "monday", "tuesday",
                                 "wednesday", "thursday", "friday","saturday", "sunday", "today", "tomorrow" };
```

2. First letter is digit shows possibility of being Date or Time



Peek into previous or next word in string array to determine if the current word is a part of date or time.

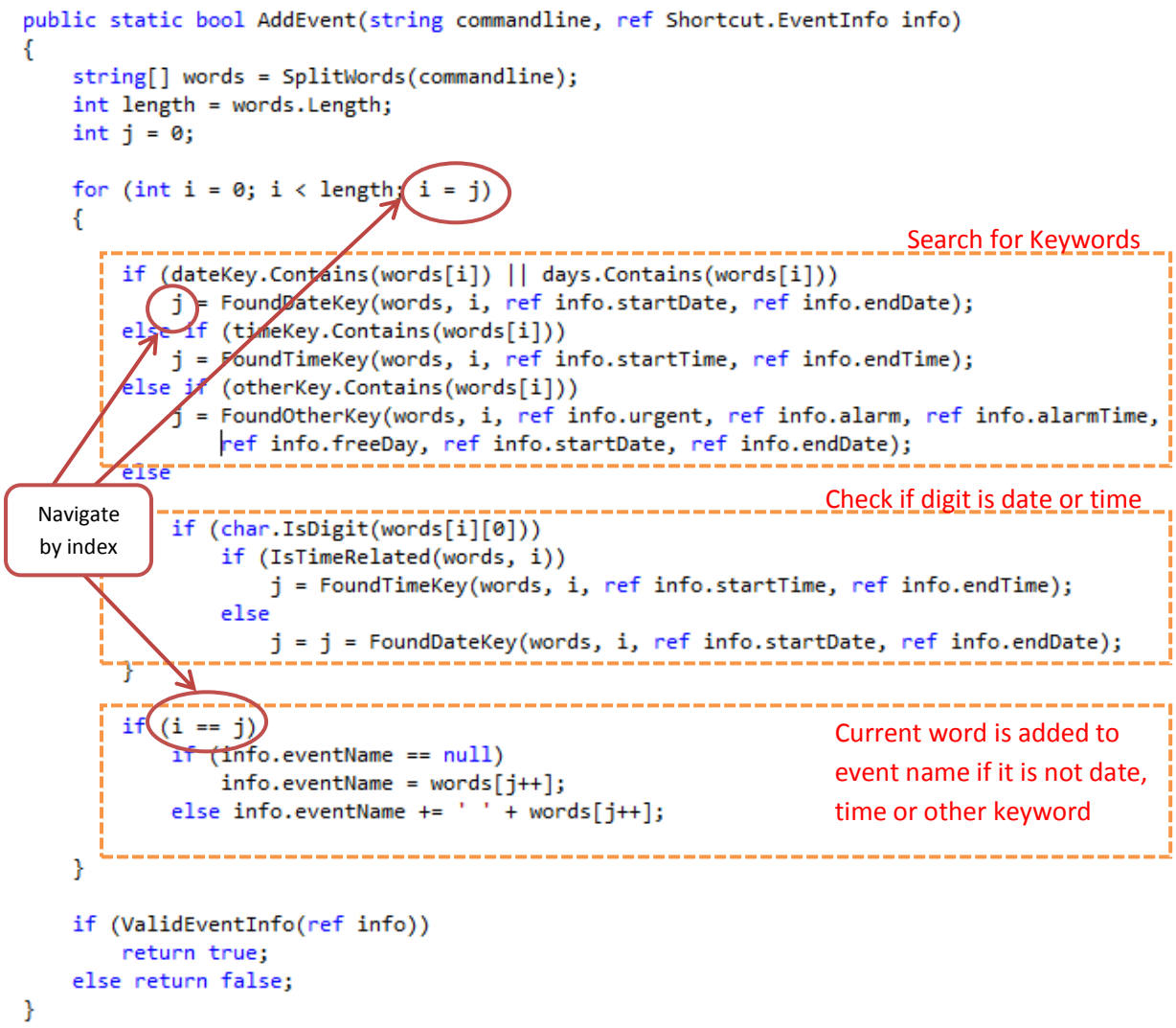
Pseudocode for extracting data



1. Check type of current_word
2. WHILE type of current_word = type of previous_word
    2.1 add to substring
3. WHILE substring.length > 0
    3.1 IF extractDateOrTime = false THEN
        3.1.1 Remove last word from substring
    3.2 ELSE
        3.2.1 data is obtained
        3.2.2 BREAK

Code segment for Parser.AddEvent

The code below shows the AddEvent method in Parser class which extracts information of an event from command line.



Ways to Extract Date and Time

1. DateTime.Parse functions to help us convert some phrases to Dates or Time.
2. Switch case to identify days of the week such as Monday, fri, Sun, today, tomorrow, etc.

Automated Testing

We have implemented unit testing using Visual Studio Unit Testing Framework. Using unit testing, individual units of source code are tested to determine if they are fit for use. This means each API in each class can be tested separately and test cases are independent of each other. This ensures that each method meets its design and behaves as intended. All methods, including private methods can be tested.

Regression testing is made easy by automated unit testing which can be run anytime the code is changed.

Code for testing NewUi.ViewEvent

```
/// <summary>
///A test for ViewEvent
///</summary>
[TestMethod()]
[DeploymentItem("ShortcutUIv1.exe")]
public void ViewEventTest()
{
        NewUi_Accessor target = new NewUi_Accessor();
        string[] date = {null,"Urgent","Done","4/2/12","1 jan 12 to 3jan 12","from 11 dec
        11 to 13 dec 11"};
        string[] expected = { "No events.\n", "No events urgent.\n", "No events done.\n",
        "No events from 4/2/2012 to 4/2/2012.\n", "No events from 1/1/2012 to 3/1/2012.\n",
        "No events from 11/12/2011 to 13/12/2011.\n"};

        string[] actual = new string[date.Length];
        for(int i=0; i< date.Length;i++)
           actual[i] = target.ViewEvent(date[i]);
        for (int i = 0; i < date.Length; i++)
           Assert.AreEqual(expected[i], actual[i]);
}
```

Private accessor for class under test (NewUi) is created by the framework to access all public and private methods in the class. The API under test is then executed and result is obtained. We compare the results with expected results by using assertions. Passed assertions will result in passed test cases while failed assertions will fail the test case.

Choosing Test Cases

Specification	Partitions
ClearEvent(string commandline): string Return string to show to user according to cases	[null] [empty space] [“all”] [any valid date] [any other string]
ProcessDeleteEvent(string commandline): string Return string to show to user according to cases	[null] [“all”] [MIN_INT, 0] [0, MAX_INT] [any other string]
ProcessDoneEvent(string commandline): string Return string to show to user according to cases	[null] [“all”] [MIN_INT, 0] [0, MAX_INT] [any other string]
SearchEvent (string commandline) : string Return search result to show to user	[null] [empty space] [any string]
ViewEvent (string commandline) : string Return view result to show to user	[null] [“all”] [“urgent”] [“done”] [any valid date] [any valid period]