# Quicklyst Developer’s Manual

## Architecture

Quicklyst adopts an n-tier architectural style where higher level components make use of the services from lower level components. Hence higher level components are dependent on lower level components while lower level components are independent of higher level components.

The GUI component is at the highest level and is the only component that interacts with the user. It uses APIs provided Logic component to carry out the user’s commands. The Logic component implements the different functionalities of Quicklyst through the help of a few sub-components which will be further elaborated. Finally the Storage and Google Calendar component are at the lowest level and allows data to be loaded and stored. Figure 1 illustrates the architecture of Quicklyst.



## GUI Component

### Class diagram

The Graphical User Interface (GUI) provides an interactive and visual indication for the user. By handling the command entered (e.g. add, delete, sort), GUI will update three main fields (task list, overview and feedback) accordingly. Figure 1 below shows the structure of the GUI component and its dependency.



### Sequence diagram

A sequence diagram shown in *Figure 2* will demonstrate the interaction between the user and the GUI.



## Logic Component

### Class diagram

The class diagram in Figure \_\_ shows the relationship of the classes that are relevant to the Logic component. The Logic component consists of three sub-components- QLLogic, CommandParser and DateHandler, and handles Task objects.



### Task Object

A Task object is instantiated from the Task Class and its represents a typical real life task. Apart from typical instance methods such as accessors and modifiers, Task contains some special methods shown below.

|  |  |  |
| --- | --- | --- |
| Method | Parameters | Description |
| clone(): Task |  | Returns a new instance of a Task with identical attributes as this Task |

### DateHandler Class

The DateHandler Class handles anything that deal with dates. It is used by QLLogic when it needs to interpret a date, and also Task when it needs to set and modify its dates.

|  |  |  |
| --- | --- | --- |
| Method | Parameters | Description |
| convertToDateCalendar(  String: dateString): Calendar | **dateString:** a string in the form of ‘DDMM’, ‘DDMMYYYY’, ‘TDY’ or ‘TMR’. | Converts a String of date into a Calendar of date and returns it. |

### CommandParser Class

The CommandParder Class handles commands that are keyed in by the user. It is used by QLLogic when it needs to interpret a command.

|  |  |  |
| --- | --- | --- |
| Method | Parameters | Description |
| splitActionAndFields(  String: command): String[] | **command**: the command string that is typed in by the user. | Split the command into ‘action’ and ‘fields’. ‘action’ is the type of operation such as add, delete, edit, etc. ‘fields’ are the fields of a Task that the action needs to apply on. Returns a String array of size = 2 where the first element is the ‘action’ and second element is the ‘fields’. |
| processFieldLine(String:fieldLine):  LinkedList<String> | **fieldLine**: a string that may contain some fields. | Extracts the individual fields and returns a LinkedList of fields. If there are no fields, returns an empty list. |
| extractTaskName(  String: fieldLine): String | **fieldLine**: a string that may contain a task name. | Extracts and returns a potential task name. |
| extractTaskNumberString(  String: fieldLine): String | **fieldLine**: a string that may contain a task number. | Extracts and returns a potential task number. |
| getSortingCriteria(  LinkedList<String>: fields): LinkedList<char[]> | **fields:** a list of fields that may contain the sorting criteria in which higher level sorting criteria appears first in the list. | Interpret each field in the list and determine its soring criterion. Returns a list of char array of size 2. The first element in the char array is the field type and second element is the sorting order. |

## Storage Component & Google Integration