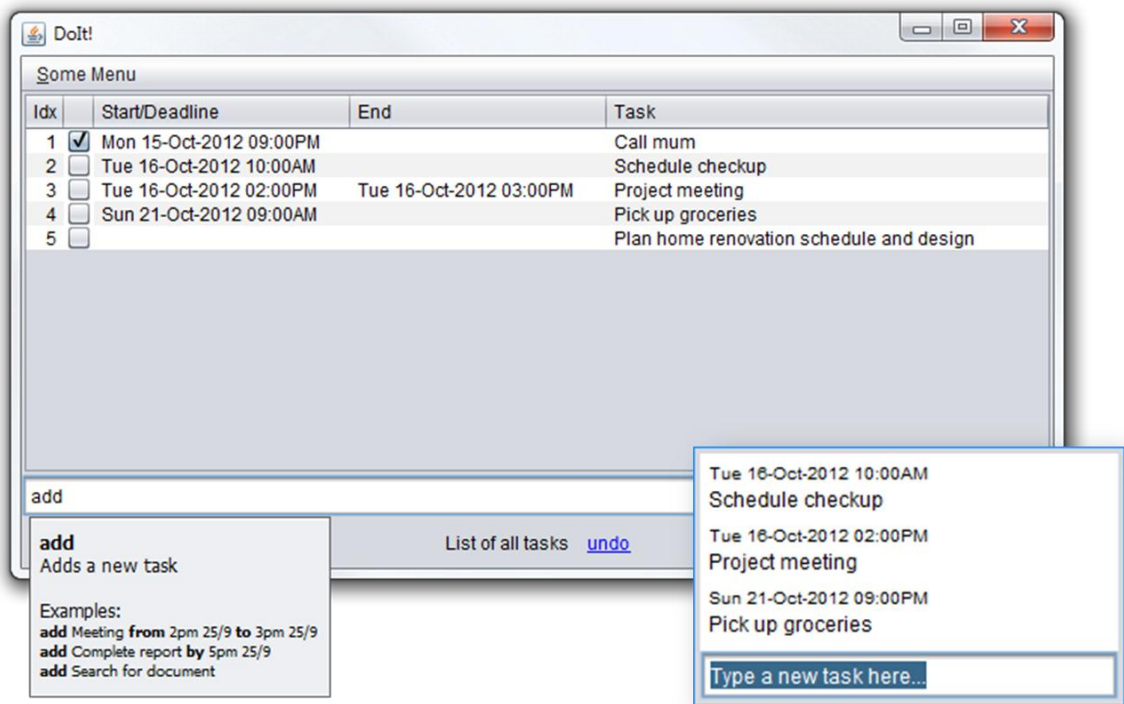

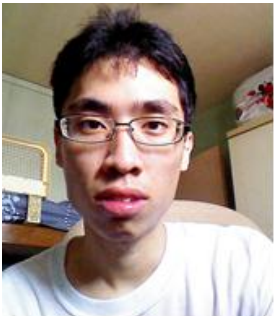
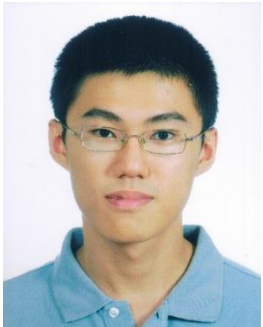


# Dolt! Manual



 <p>Nguyen Hien Linh</p> <p>Team lead, architect, testing</p>	 <p>Yeo Kheng Meng</p> <p>Deadline watcher, lead programmer</p>	 <p>Yeow Kai Yao</p> <p>GUI design, documentation</p>
--	--	---

## Credits

Dolt! would not be possible without the following software libraries:

- Joda-Time date and time handler <http://joda-time.sourceforge.net/>
- Natty natural language date parser <http://natty.joestelmach.com/>
- JLine console library <https://github.com/jline/jline2>

# Contents

Credits .....	2
<b>User Manual</b>	
1 System Requirements .....	4
2 Running Dolt! .....	4
3 Using Dolt! .....	4
3.1 Dolt! Main .....	4
3.2 Quick Add .....	4
3.3 Command-line Interface .....	5
4 Supported Task Types .....	5
5 Command Reference .....	5
<b>Developer Guide</b>	
1 Introduction .....	8
2 Architecture .....	8
3 Understanding the command flow .....	9
4 Components .....	10
4.1 UI .....	10
4.1.1 Classes .....	10
4.1.2 Important APIs of the UI Abstract Class .....	10
4.1.3 GuiMain Subclass .....	11
4.1.4 GuiQuick Subclass .....	11
4.1.5 Cli Subclass .....	11
4.1.6 CliWithJline Subclass .....	11
4.2 Logic .....	12
4.2.1 Important API .....	12
4.2.2 Sequence Diagram .....	13
4.3 Storage .....	16
4.3.1 Database class .....	16
4.4 Shared Components .....	17
4.4.1 Task .....	17
4.4.2 SearchTerms .....	18
4.4.3 LogicToUi .....	19

# Dolt! User Manual

Welcome to Dolt!

Dolt! is an easy-to-use task organization program. Just type in what you have on your mind and Dolt! will put it on your schedule, making it easy for you to manage your life!

## 1 System Requirements

Dolt! requires the following:

- Windows XP or later
- Java runtime environment 7. Download it from <http://www.java.com/download>

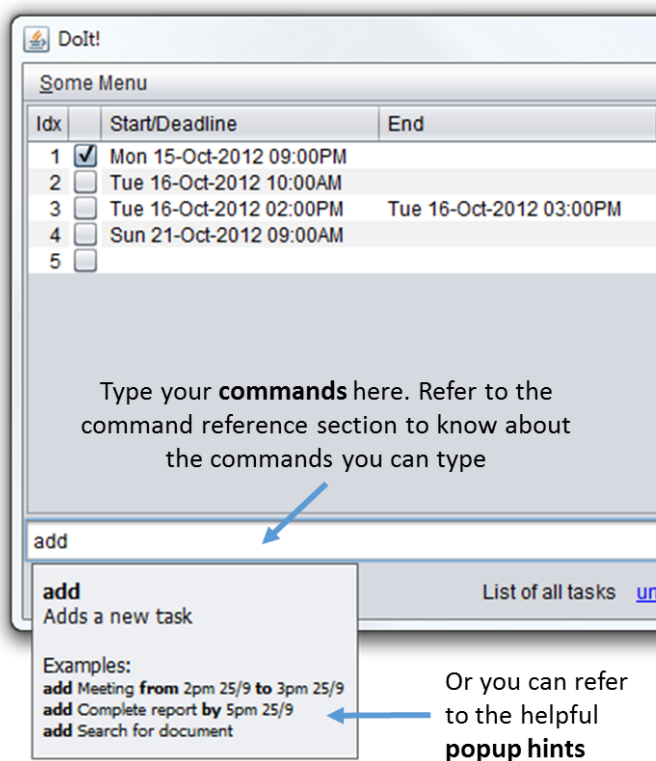
## 2 Running Dolt!

No installation is required. Just double click on the downloaded program's icon to launch the program.

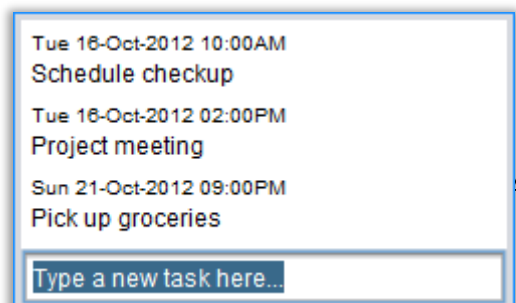
Dolt also supports a complete command-line based interface. Run Dolt! with the `-cli` argument to do so.

## 3 Using Dolt!

### 3.1 Dolt! Main



### 3.2 Quick Add



Pressing Ctrl-Alt-Z (you can change that) will launch the quick add dialog box. Simply type your new task into the box to add it to your task list.

The syntax follows the add command, but you don't have to specify the command word add. Type `full` to launch the full Dolt! user interface.

### 3.3 Command-line Interface

Launch Dolt! in a terminal with the `-cli` argument. Then just type your commands into the prompts.

Dolt's command-line interface supports some handy shortcuts: press `tab` to have Dolt! complete your commands, and press `up/down` to view your command history.

## 4 Supported Task Types

Dolt! supports 3 distinct types of tasks:

1. Floating tasks: Tasks that have no specific time. E.g: Write report.
2. Deadline tasks: Tasks that have to be done by a specific time. Example: Write report by 2pm on 5 Sept 2012
3. Timed tasks: Tasks that have a specific start and end time. Example: Write report from 12pm 5 Sept 2012 to 2pm on 5 Sept 2012

## 5 Command Reference

Dolt's keyboard commands are easy to use! They start with a keyword for the action to take, following by some details about the action.

Note the following formatting used for the command syntax descriptions below:

- **Bold** = keyword; type exactly
- *Italics* = replace with appropriate argument
- [Item in square brackets] = optional argument
- a|b = use a or b, you cannot use them together

Syntax	Description	Example
<b><i>Adding tasks</i></b>		
<b>add</b> <i>task name</i>	Add new floating task	<code>add fix cupboard</code>
<b>add</b> <i>task name</i> [by] <i>date/time</i>	Add deadline task	<code>add fix cupboard by 2pm on 5 Sep</code>
<b>add</b> <i>task name</i> [from] <i>start date</i> [to] <i>end date</i>	Add timed task with specific start and end time and date	<code>add fix cupboard from 2pm on 5 Sept to 4pm on 6 Sept</code>
If Dolt! is getting your task names mistaken as dates, just enclose the task name in double inverted commands e.g. <code>add "fix cupboard"</code>		
<b><i>Viewing tasks</i></b>		
<b>list</b>	Lists all tasks in order of due date	<code>list</code>

<b>list</b> <i>[complete]</i> <i>[incomplete]</i> <i>[done]</i> <i>[undone]</i> <i>[floating]</i> <i>[deadline]</i> <i>[timed]</i> <i>[today]</i> <i>[tomorrow]</i> <i>[overdue]</i>	List all tasks that meet the criteria specified. Note that you can use more than one criteria, only tasks that match all the criteria will be shown.	list complete floating
<b>search</b> <i>[keyword]</i> <i>[keyword]</i> ...	List tasks with the keywords	search cupboard search cupboard shoe
<b>refresh</b>	List tasks based on previous list/search command	refresh
<p>When viewing tasks, an index number is displayed next to each task. The index number changes when different arguments are used with the list or search command, and when tasks are added or removed. When required by a command, the index number used corresponds to the index number shown with the latest list command.</p> <p>In the command-line interface, the list is not automatically refreshed after each modification. If you have made any changes to the tasks, do a refresh or list to update the index numbers.</p>		
<b>Delete</b>		
<b>delete</b> <i>index</i>	Deletes the task with number <i>index</i> .	delete 1
<b>delete</b> <b>done completed finished</b>	Deletes all tasks that has been marked as done	delete done
<b>delete all</b>	Deletes all tasks	delete all
<b>delete over</b>	Deletes tasks that ended before the current time	delete over
<b>Edit tasks</b>		
<b>edit</b> <i>index</i> <i>[-name -n new name]</i> <i>[-start -s new start time / start date]</i> <i>[-end -e new end time / end date]</i>	Changes the name/start time/end time of the task specified by <i>index</i> to the new value. To get <i>index</i> , see the note under “viewing tasks”	edit 1 -name fix cupboard edit 2 -start 1800 on 10 Sep
	You can combine multiple things to edit in the same command. Also note that shortcuts (e.g. s) can be used	edit 3 -name fix shoe rack -s 1800 10 Sept
	If no date is specified, the previous date will be kept. This also applies to the time	edit 5 -e 2100
<b>Postponing tasks</b>		
<b>postpone</b> <i>index to new start time / start date</i>	Postpone the task to <i>new start time/start date</i> (ending time and date is shifted accordingly so that the task duration remains the same). To get <i>index</i> , see the note under “viewing tasks”	postpone 1 to 3pm
<b>postpone</b> <i>index by duration</i>	Postpones the task specified by <i>index</i> by <i>duration</i> (ending time and date is shifted accordingly so that the task duration remains the same)	postpone 1 by 1 hour
<b>Marking tasks</b>		
<b>done</b> <i>index</i>	Marks the task specified by <i>index</i> as done. To get <i>index</i> , see the note under “viewing tasks”	done 1

<b>undo</b> <i>index</i>	Marks the task specified by <i>index</i> as not yet done	<b>undo</b> 1
<b><i>Undo changes</i></b>		
<b>undo</b>	Undo the last change you have made.	<b>undo</b>
<b><i>Getting command help</i></b>		
<b>help</b>	Show a list of all possible commands	<b>help</b>
<b>help</b> [ <i>command</i> ]	List the possible usage for <i>command</i>	<b>help</b> add

# Dolt! Developer Guide

## 1 Introduction

Welcome to the developer guide for Dolt!

In case you do not know yet, Dolt! is a simple and easy-to-use task list management software. Dolt! is distinct from other task management software as it allows for complete control of it via the keyboard.

This document aims to help developers like you understand how Dolt! is designed and how you can build and extend Dolt.

So let's get started!

## 2 Architecture

Dolt! is separated into various components: UI, Logic, Storage and some shared components, as illustrated in the overview diagram below. In the code, this is made distinct by having *one package* for each major component.

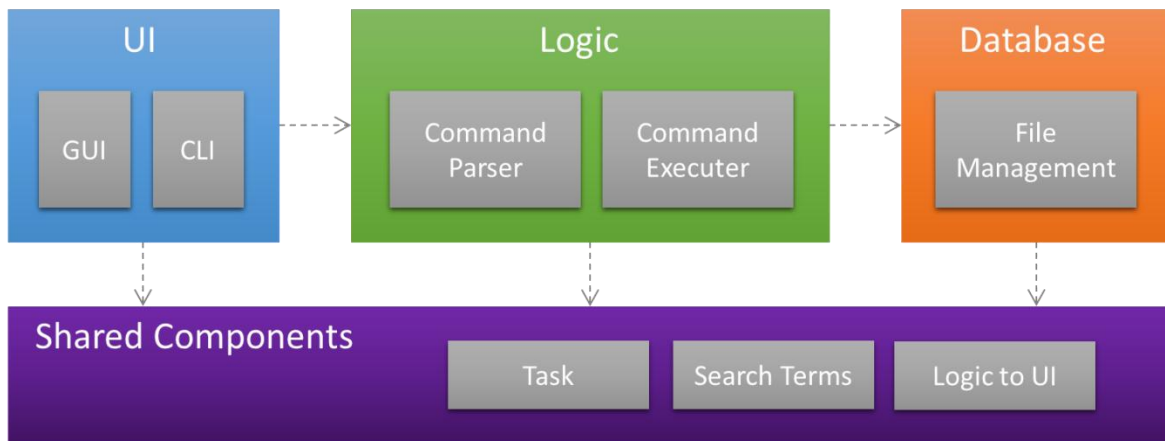


Figure 1: Architecture of Dolt!



### 3 Understanding the Command Flow

In Dolt!, all actions start from the user (and correspondingly, the user interface), that is, all actions start with the user doing something. The UI sends the command to the Logic component, which parses the command and then takes the appropriate action, usually invoking the Storage component. The response then goes back to the user the same way. This is reflected in the sequence diagrams:

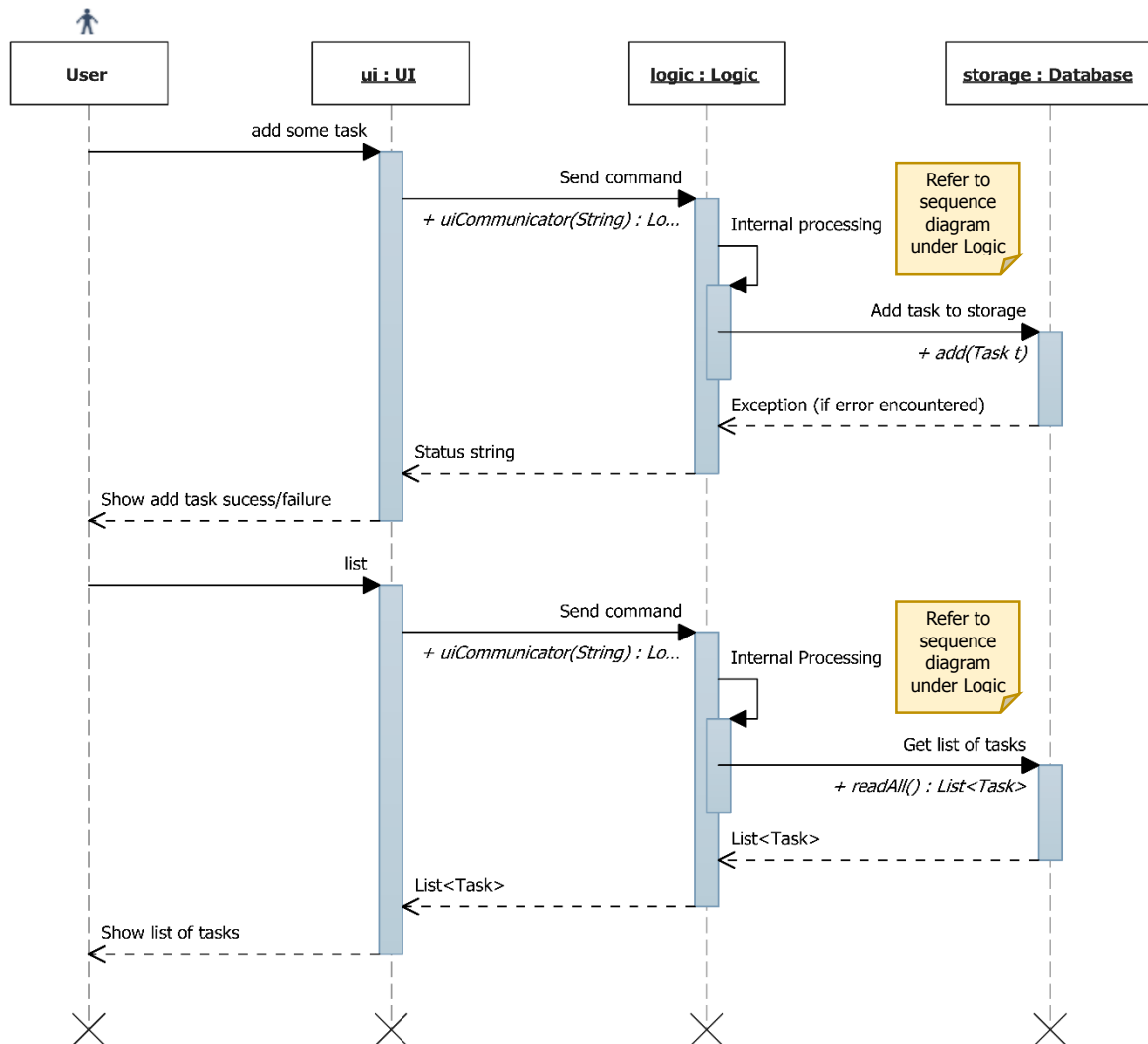


Figure 2: Sequence diagram for adding a new task and viewing tasks.

## 4 Components

Now that you have seen an overview of the various components in Dolt!, as well as how they interact, you are probably interested in some more details of a component or two... Therefore, you can get more details about your component of interest in this section.

### 4.1 UI

This is the place to look for if you want to modify the user interface or develop an alternative user interface.

#### 4.1.1 Classes

The UI in Dolt! is designed as a UI abstract class from which actual concrete user interfaces can be extended from. It is thus easy to add an alternative user interface, for example one that works with mobile devices.

There are currently four concrete UIs: the GuiMain subclass, the GuiQuick subclass, the Cli subclass and the CliWithJline subclass.

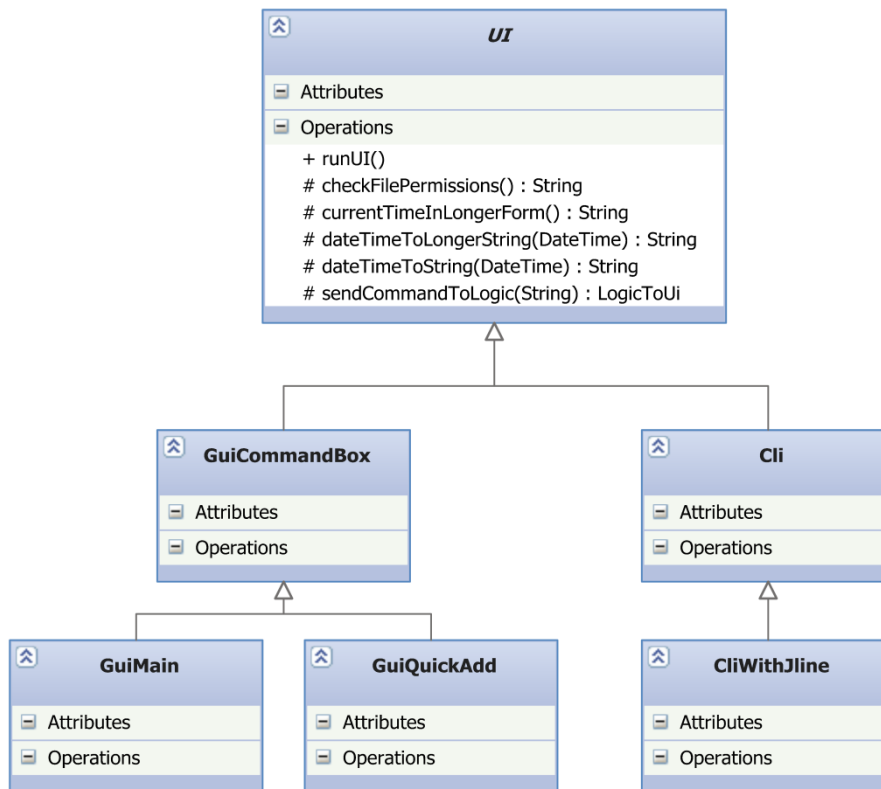


Figure 3: Class diagram for UI Component

#### 4.1.2 Important Attributes and Methods of the UI Abstract Class

<code>List&lt;String&gt;</code>	<b>commandList</b> List of commands supported by Dolt as given by the Hint class. For command help.
<code>abstract void</code>	<b>runUI()</b> This method is the starting point of every UI; it is called when the UI is to be run.
<code>LogicToUi</code>	<b>sendCommandToLogic(String command)</b> Passes command to logic to be processed. Returns a <code>LogicToUi</code> object comprising a String message, or a String and a <code>List&lt;Task&gt;</code> depending on command.

String	<b>checkFilePermissions()</b> Sends a command to logic to check for the database file permissions. Returns a string containing the status of the file. <b>It is important to call this method before allowing any user interaction with tasks so that the user will be warned of a read-only situation.</b>
String	<b>getHTMLHelp(String command), getNoHTMLHelp(String command)</b> To get command specific help for display to the user. See Hint Class for more details.

#### 4.1.3 GuiMain Subclass

The GuiMain subclass holds the code for the main window of Dolt's graphical user interface. It is developed in Java Swing, and uses (mainly) a JTable and a JTextField. Pop-up boxes are implemented using a JEditorPane in a JPopupMenu, a technique used in <sup>1</sup>.

#### 4.1.4 GuiQuick Subclass

This subclass implements the quick view/quick add window of Dolt!

#### 4.1.5 Cli Subclass

This subclass implements a plain-Jane command-line interface that works using standard Java I/O.

#### 4.1.6 CliWithJline Subclass

This subclass extends the Cli subclass and implements an enhanced command-line interface that uses the Jline console library to implement features such as Tab Completion and support for a resizable terminal window. Note that the Jline library interfaces natively with the terminal of the underlying operating system, and as such does not work when a full terminal is not available, for instance in the Eclipse console.

The hint class will accept only one name and summary field. The name field should be enclosed in <h1> html tags to allow for custom styling by the GUI. Use of <b> tags are encouraged to emphasise a specific aspect of the command. Multiple usage and extra tags are allowed.

#### 4.1.7 Hint Class

This class provides some help text for each of the commands. Its methods are not accessible directly; to get the help text, use getHTMLHelp() or getNoHTMLHelp() in the UI class.

Help Text File: help.xml

The help text is stored in this XML file in src/main/resource. The following snippet shows the XML for the postpone command.

```
<hint>
<command>
<name><![CDATA[<h1>Postpone</h1>]]></name>
<summary>Postpones a task</summary>
<usage><![CDATA[<b>postpone</b> [index] [time parameter] ]]></usage>
<usage><![CDATA[<b>postpone</b> 1 3 hours (postpones task at index 1 by 3
hours) ]]></usage>
<extra>This command has no effect on floating tasks. Start and End times for timed
tasks will shift together.</extra>
```

<sup>1</sup> [http://www.jroller.com/santhosh/date/20050620#file\\_path\\_autocompletion](http://www.jroller.com/santhosh/date/20050620#file_path_autocompletion)

</command>  
</hint>

## 4.2 Logic

This is where the commands from the user interface are processed and the relevant action taken.

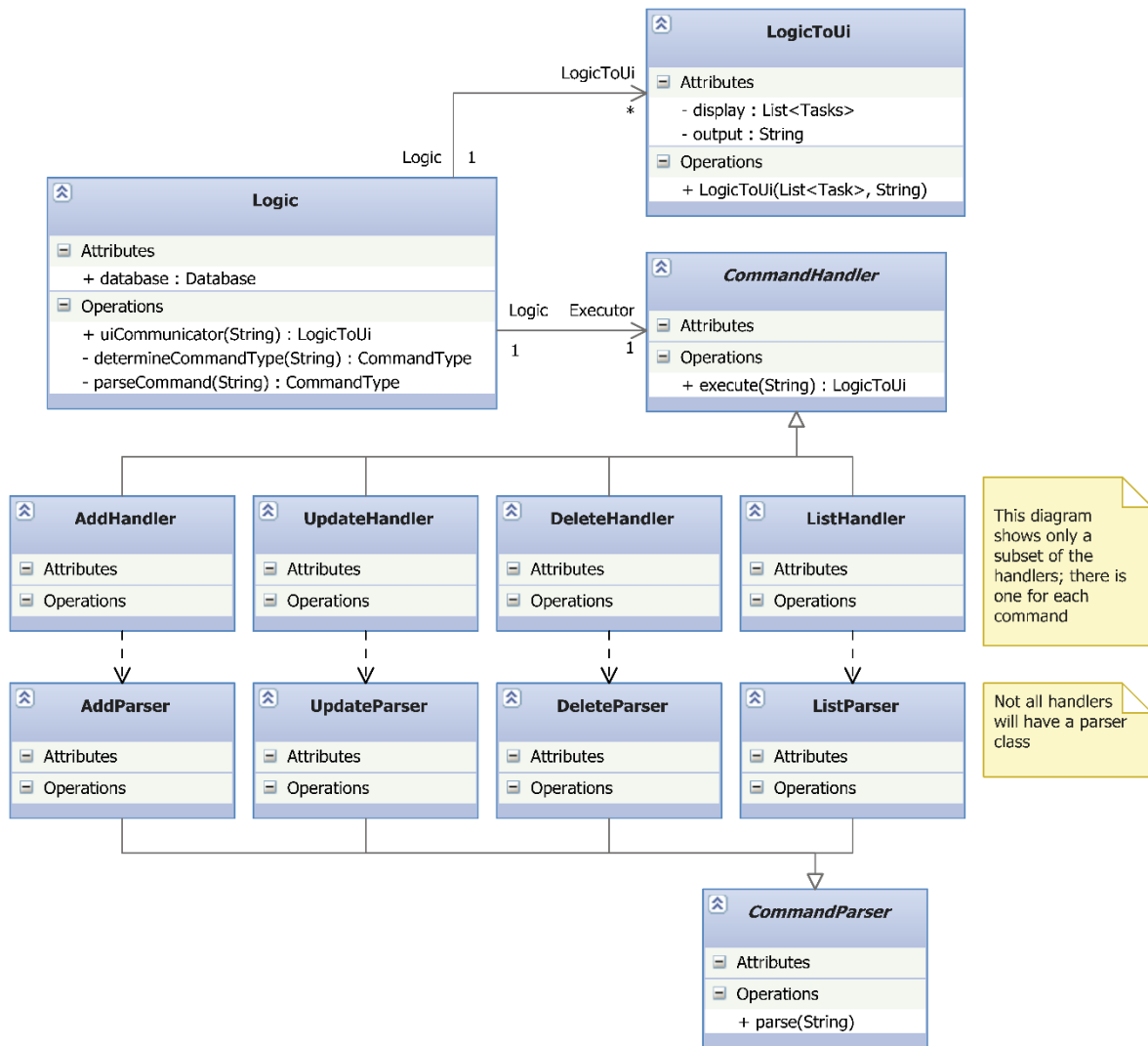


Figure 4: Class diagram for Logic Component

### 4.2.1 Important APIs

The most important interface to Logic is the `uiCommunicator(String)` method, which serves as the entry point to the entire Logic component.



#### Logic and serial numbers in tasks

The logic component holds a complete representation of the tasks currently shown to the user. This is needed because the index number shown in the UI is different from the serial number in the tasks, and the logic is responsible for mapping between them.

### 4.2.2 Command Handlers

Command handlers act on a particular command they are designed to handle.

Every command handler must extend the `CommandHandler` abstract class, which specifies that the handler must implement an `execute()` method. Note that each command handler is expected to repeat the same action if the `execute()` method of a handler object is called again.

#### Supporting undo operations

Support of the undo operation is only required for handlers which modify the database like add and delete. List and Search should not be undoable. In order to provide the undo functionality, the `CommandHandler` will maintain a Stack (implemented as `LinkedList`) of database clones and their associated return messages. They are stored in the private variables.

```
private static LinkedList<String> undoMsgHistory;
private static LinkedList<List<Task>> undoClones;
```

They are only accessible through the protected methods in the `CommandHandler`.

To ensure a consistent saving of the undo step, the following steps have to be followed strictly for all handlers which modify the database.

1. `List<task> currentTaskList = super.getCurrentTaskList();` Put this as the first line inside the try block.
2. Conduct necessary modification operation
3. Place this statement as the second-last line inside the try block, just before you return `LogicToUi`.  
`super.pushUndoStatusMessageAndTaskList(undoMessage, currentTaskList);`

“undoMessage” refers to what you want the user to see if the step is undone. The Undo Handler will display the message in this manner.

"The " + undoMessage + " has been undone"

So phrase your message as if it is inside there.

If any exceptions are thrown, the statement in (3) will not be executed and thus will not be pushed to undo stack.

There is no need to write to the database. If there is an error, the database class will retain the old version of the list.

#### Supporting refresh operations

The `CommandHandler` will store what was the last view operation in this variable.

```
protected static CommandHandler latestRefreshHandlerForUI
```

All viewing/filter operations are required to store a reference of themselves into this variable should they require the GUI to maintain that view after a refresh.

The refresh command will execute the handler when it is called after every operation in the GUI.

### 4.2.3 Command Parsers

### 4.2.4 Sequence Diagram

The following is a sequence diagram of the process in the Logic class when **adding a task**. Note that the diagram is largely similar for other commands, except **(1)** the `AddExecutor` class is replaced by

the appropriate executor, **(2)** *Add created task to database* is replaced by the corresponding call to Database, and **(3)** calls to *AddParser* are replaced by calls to the corresponding parser class. For some commands with simpler parsing, you will not find an associated parser class as the parsing is done in the method itself.

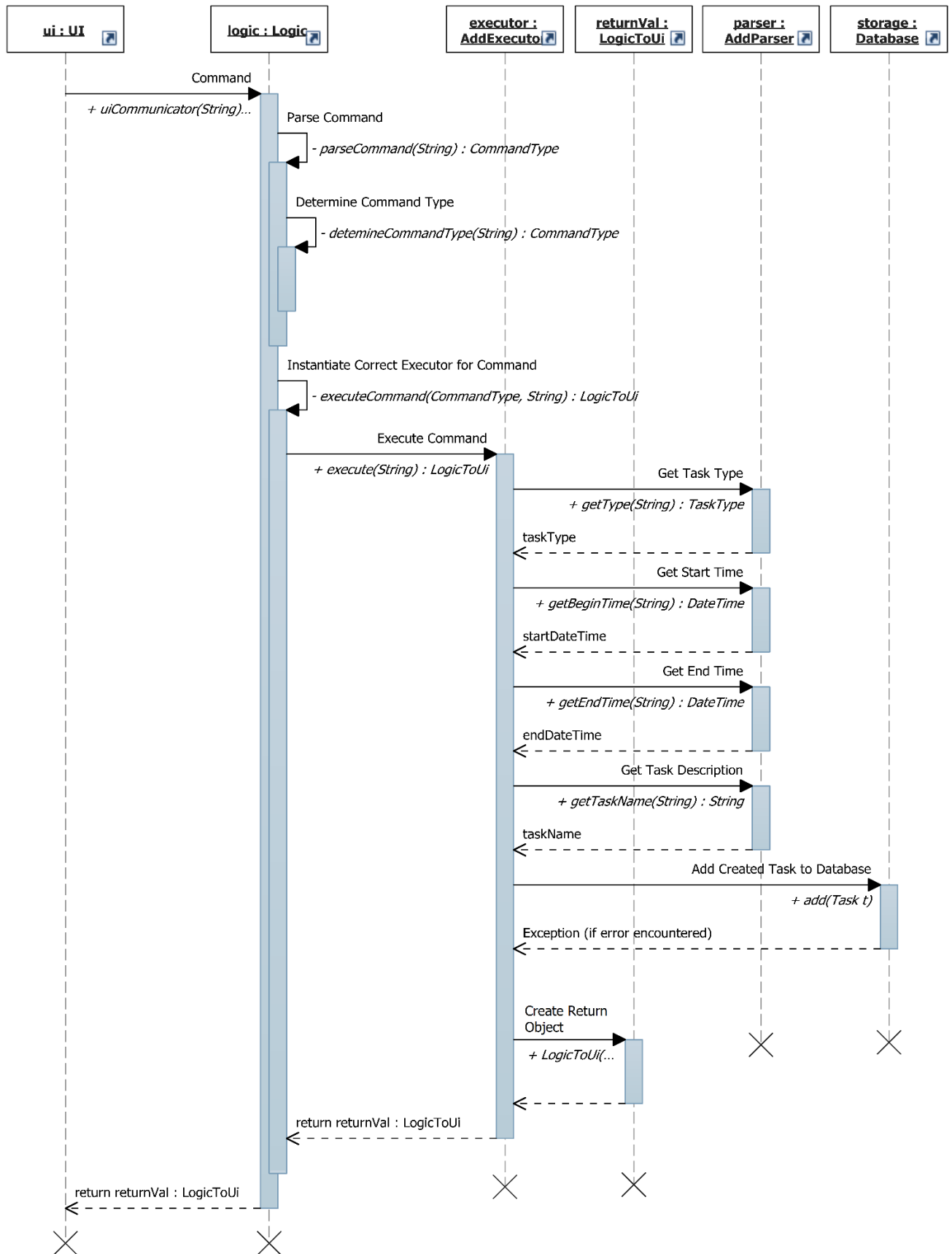


Figure 5: Sequence Diagram showing the process of adding a task

## 4.3 Storage

The storage component handles the writing and reading of tasks to the text file storage component on the disk.

This component is implemented with two classes: Database and FileManagement.

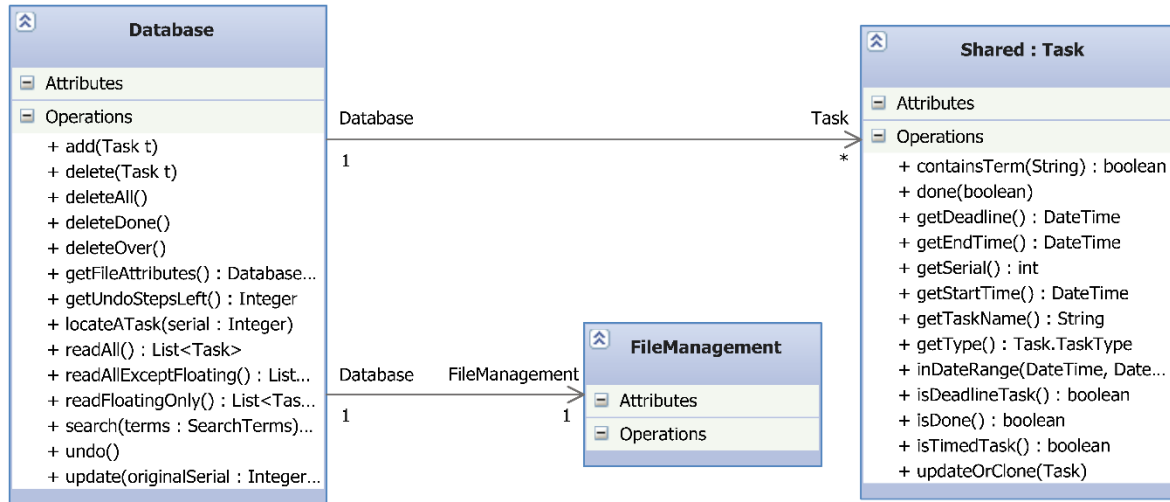


Figure 6: Class Diagram for Storage Component

### 4.3.1 Database class

Upon instantiation, the Database class reads in the text file of the tasks through the FileManagement class and stores the tasks in a `List<Task>`. The tasks in the list are stored in order of the start date of the tasks.

#### Important APIs

void	<b>add(Task newTask)</b> Adds a new task to database.
void	<b>delete(int serial)</b> <b>delete(int[] serial)</b> Deletes an existing task(s) in database. The serial number can be obtained from an existing Task object you want to delete.
void	<b>deleteAll()</b> Deletes all tasks in database.
Database.DB_File_Status	<b>getFileAttributes()</b> Gets the file permissions of the database file. One of the following values will be returned: FILE_ALL_OK, FILE_READ_ONLY, FILE_UNUSABLE, FILE_PERMISSIONS_UNKNOWN, FILE_IS_CORRUPT <b>You are strongly suggested to call this method at the earliest opportunity to verify read and write permissions.</b>
Task	<b>locateATask(int serial)</b> Locates an existing task in the database that has serial.
java.util.List<Task>	<b>readAll()</b> Returns a List<Task> of all the tasks in the database.
java.util.List<Task>	<b>search(shared.SearchTerms terms)</b> Returns a List<Task> of all the tasks that match the search term.



void	<b>update</b> (int originalSerial, Task updated) Updates an existing task in the database.
------	---

### Exceptions

Any methods above that involve a change in the database may throw the following exceptions:

1. `java.io.IOException` is thrown if the Database class is unable to write to the file due to some permission issues.
2. `storage.WillNotWriteToCorruptFileException` is thrown if the file can be accessed but is corrupt. As the name implies, the Database class will not write to the file to prevent corrupting the text file further. The user should be shown the list of tasks that can still be read.
3. `java.util.NoSuchElementException` is thrown if the database cannot locate one or more tasks it is required to modify.

The `List<Task>` in the Database class will not be modified if any of the above exceptions occur.

## 4.4 Shared Components

### 4.4.1 Task

The task object is the main data type used by Dolt! Each task object holds one task.

#### Constructors

The class has 6 constructors, allowing you to create the 3 different types of tasks (floating, deadline and timed) and 3 more that allow you to set the (boolean) done status of the task when creating the object.

To instantiate an undone floating task:

```
Task(java.lang.String name)
Example: new Task("This is an undone floating task");
```

To instantiate an undone deadline task:

```
Task(java.lang.String name, org.joda.time.DateTime deadline)
Example: new Task("This is an undone deadline task", new DateTime(2011, 9, 5, 23,59))
```

To instantiate an undone timed task:

```
Task(java.lang.String name, org.joda.time.DateTime startTime,
org.joda.time.DateTime endTime)
Example: new Task("This is a timed task", new DateTime(2011, 9, 5, 23,59), new
DateTime(2013, 12, 31, 00, 00))
```

### Important APIs

void	<b>done</b> (boolean newDoneStatus) Marks the task as done or undone.
DateTime	<b>getDeadline</b> () Gets the deadline of a deadline task. <b>If this field is not applicable, the Task class will return a public constant</b>

	<b>INVALID_DATE_FIELD DateTime object.</b>
DateTime	<b>getEndTime()</b> Get the ending time of a timed task. <b>If this field is not applicable, the Task class will return a public constant INVALID_DATE_FIELD DateTime object.</b>
int	<b>getSerial()</b> Gets the serial number of the task, for the purposes of deletion and updating. This is so as the index number the user sees varies according the UI filters. ???
DateTime	<b>getStartTime()</b> Get the start time of a timed task. <b>If this field is not applicable, the Task class will return a public constant INVALID_DATE_FIELD DateTime object.</b>
String	<b>getTaskName()</b>
Task.TaskType	<b>getType()</b> Returns the type of task.
boolean	<b>isDeadlineTask()</b>
boolean	<b>isDone()</b> Returns true if the task has been marked as done.
boolean	<b>isFloatingTask()</b>
boolean	<b>isTimedTask()</b>
boolean	<b>inWithinDateRange(DateTime startRange, DateTime endRange)</b> Returns true if the time of the task overlaps the date range provided.
boolean	<b>containsTerm(String term)</b> Returns true if the task contains the term provided.
String	<b>showInfo()</b> Returns the entire contents of the Task object in the same format as database.txt. <b>For debugging purposes only.</b>
void	<b>updateOrClone(Task updated)</b> Copies all the fields from the updated Task object to this Task object. All fields including the serial number will be copied.

#### 4.4.2 SearchTerms

The SearchTerms object is a container for passing search terms. It is used by the list and search commands to pass search terms from Logic to Storage. It is also used to pass search terms from Logic to UI (through the LogicToUi class) for display purposes – for example, to notify the user on the search terms used in the current view.

It holds the following attributes that can be set in the constructor:

Attribute	Description
completedTasks : boolean	If true, matches only completed tasks
incompleteTasks : boolean	If true, matches only incomplete tasks
timedTasks : boolean	If true, matches only timed tasks
deadlineTasks : boolean	If true, matches only deadline tasks
floatingTasks : boolean	If true, matches only floating tasks
startRange : DateTime and endRange : DateTime	If specified, matches only between startRange and endRange inclusive
keywords : String[]	If specified, matches only tasks that contain ALL the strings in the array.

	Keywords are case-insensitive.
--	--------------------------------

Attributes can be combined. For example:

- Get incomplete timed tasks: incomplete = true, timed = true
- Get deadline tasks with "market" and "home" keywords: deadlineTask = true, String[] keywords = { "home", "market" }
- Get tasks that happen on 27 Dec 2012: startRange = new DateTime(2012, 12, 27, 00,00), endRange = new DateTime(2012, 12, 27, 23, 59)

#### 4.4.3 LogicToUi

The LogicToUi object is a container for passing results from the executors to the UI. It is required because executors will return a multitude of different things.

It holds the following attributes that can be set in the constructors:

Attribute	Description
output : String	A message for the UI to display to the user
display : List<Task>	A list of tasks for the UI to display to the user
filters : SearchTerms	The search terms used to obtain the list of tasks
currentSorting : SortStatus	The parameter used for sorting the list of tasks
sortReverse: boolean	Whether the sort used is in ascending or descending order

#### 4.4.4 NattyParserWrapper

Blah blah

## 5 Testing

Testing is an integral part of the development of Dolt. You are encouraged to employ test-driven development (TDD) whenever feasible.

Automated testing is carried out using test cases based on the JUnit4 framework. These are the conventions set out for the development of Dolt. The tests are stored in the directory /src/test/"package name", where the package name has to be identical to the class under test.

The test classes have to be named "Class Name"Test. For example, to test the Logic class, name the test class LogicTest.

The text file database on disk may be irreversibly modified during the automated testing procedure. Therefore if you have important information, you should do a backup of the text file before the commencement of any test.

GUI testing is not required since we do not have the resources at this time.

## 6 Walkthrough

## 7 Known Issues

Pure CLI interface supports only a minimum of 80 characters width for the terminal console. CliWithJline will support as low as 67 characters as it can work in resizable console windows. Display on consoles smaller than their respective minimums is not defined.

If the user has  $\geq 1000$  tasks, the CLI index column will shift. A 3 digit limit for the index number is hardcoded on the assumption that few users have 1000 tasks and to balance against the limited console width.

## 8 Future Work

### 8.1 Google Calendar and Tasks integration

Dolt currently does not support Google integration. However, there exist some users that would prefer this functionality so they can use Google features like SMS and email reminders.

### 8.2 Password-protected database

The database file is stored in plaintext format. A password-lock with a suitable encryption scheme is envisioned to cater to more security conscious users.

### 8.3 Internationalisation

The interface of Dolt and the commands it accepts is currently only limited to the English language. To cater to a greater group of users, support for more languages is intended.

## 9 Appendix