

T A S U K E

The Developer’s guide

Table of Contents

[1. Preface 3](#_Toc382760833)

[ 1.1 Introduction to the Tasuke Developer Guide 3](#_Toc382760834)

[ 1.2 What is Tasuke? 3](#_Toc382760835)

[ 1.3 What must I know? 3](#_Toc382760836)

[ 1.4 Concept and Design Principles 4](#_Toc382760837)

[ 1.5 Organization of this document 4](#_Toc382760838)

[ 1.6 Conventions and Definitions 4](#_Toc382760839)

[ 1.8 Updates and Feedback 5](#_Toc382760840)

[2. Tasuke Build Environment 5](#_Toc382760841)

[ 2.1 Development Environment 5](#_Toc382760842)

[ 2.2 Execution Environment 5](#_Toc382760843)

[ 2.3 Application Files 5](#_Toc382760844)

[3. Software Architecture 6](#_Toc382760845)

[ 3.1 Software Architecture Overview 6](#_Toc382760846)

[ 3.2 Logic 7](#_Toc382760847)

[ 3.2 User Interface 8](#_Toc382760848)

[ 3.3 Interpreter 9](#_Toc382760849)

[ 3.4 Storage 10](#_Toc382760850)

[5. Appendices 11](#_Toc382760851)

[ 5.1 Full Software Architecture Diagram 11](#_Toc382760852)

[ 5.2 List of Functions 12](#_Toc382760853)

[ 5.3 Files within the Project 12](#_Toc382760854)

# 1. Preface

## 1.1 Introduction to the Tasuke Developer Guide

Welcome to the Tasuke Developer Guide! This guide is intended for developers and maintainers of Tasuke.

Here are the aims of the guide.

* To familiarize you with the design and implementation of Tasuke.
* To detail the organization of the software and its API.
* To provide solutions for future development.
* To assist you with troubleshooting the program.

## 1.2 What is Tasuke?

Tasuke is a desktop task manager aimed at individuals who are comfortable with keyboard-based commands for rapid data entry and retrieval. Tasuke will appeal to users who are familiar with the command-line-like style of calling and dismissing programs, and yet provides a simple but powerful GUI for clearer data organization and fine-tuning.

The basic functionality of Tasuke is as follows:

* Adding, editing and removing of tasks
* Marking tasks as done or otherwise
* Setting a starting time and deadline for tasks
* Adding tags to tasks
* Searching for tasks by date, completion and tags
* Sorting the search result.

## 1.3 What must I know?

Tasuke is written in C++ using the Qt framework for the Windows platform. Hence, you should at least be familiar with C++. The design of Tasuke’s user interface requires knowledge of the QT framework.

## 1.4 Concept and Design Principles

Tasuke was developed with speed and simplicity as its goal. Keep these principles in mind when developing Tasuke!

* **Lightweight**: Tasuke is lightweight in terms of computer resource consumption.
* **Keyboard-based:** The most important functions of Tasuke must be keyboard-accessible.
* **Intuitive**: It is easy to learn to use. The command formats should follow natural language closely.
* **Unobtrusive**: Tasuke minimizes itself to the system tray when not needed.
* **Minimal**: Tasuke’s basic command-line interface should be simple.
* **Flexible**: Even though the basic interface is minimal, give users the option to add on features.

## 1.5 Organization of this document

We will first explain the **environment** needed to develop, maintain and execute Tasuke optimally, as well as the **files** it generates.

Then, we will explain the **software architecture** that is involved from input to output.

Finally, we will provide detail on the **implementation and API** involved in the programming.

## 1.6 Conventions and Definitions

In this document, a **task** refers to an object that a user will create when he or she types the *add* command followed by a task description.

Whenever this document mentions **Windows**, this refers to the default build and execution environment, which is Microsoft Windows 7 and newer iterations of the operating system.

All code, filenames and commands are written using the Courier New font. When describing methods and functions, the parameters are written in *Calibri Italic* and the return values are written in **Calibri Bold**.

A list of technical terms may be found in the Glossary.

## 1.7 Updates and Feedback

The latest version of this document may be obtained when pulling the tip of the repository from Google Code at <http://code.google.com/p/cs2103jan2014-w15-2c/> or it may be downloaded by itself from [https://code.google.com/p/cs2103jan2014-w15-2c/source/browse/[W15-2C][V0.0]DevGuide.docx](https://code.google.com/p/cs2103jan2014-w15-2c/source/browse/%5bW15-2C%5d%5bV0.0%5dDevGuide.docx)

Feedback may be provided by raising an issue on the aforementioned Google Code website.

# 2. Tasuke Build Environment

## 2.1 Development Environment

Tasuke is written in the C++ programming language, using the Qt 5.2.1 library. It is compiled and written in Microsoft Visual Studio 2012, on Microsoft Windows 7 and Microsoft Windows 8. We recommend you to use the same development environment to prevent incompatibility issues. You should avoid using the Express edition of Microsoft Visual Studio as it does not support the QT Framework.

It is, in theory, possible to compile the source code of Tasuke on any platform, for any platform. However, Tasuke is written with developers and users of Microsoft Windows 7 and later in mind. Thus, we will only discuss issues arising from Windows-compiled Tasuke. When we write this guide, we assume that you are using a Windows environment to develop and maintain Tasuke.

## 2.2 Execution Environment

The recommended execution environment for the default build of Tasuke is Windows 7 and newer. No additional framework or library needs to be installed, and Tasuke itself does not require installation and may be run straight from the executable binary.

During its operation, Tasuke will create an .ini file in the %APPDATA% directory, which is where Tasuke stores and retrieves user data.

## 2.3 Application Files

Tasuke.sln is the Visual Studio Solution file and should be the entry point when developing Tasuke. Open this using Microsoft Visual Studio to begin browsing the code for Tasuke.

A full list of the files in the project can be found in §8.2, at page 11.

# 3. Software Architecture

## 3.1 Software Architecture Overview

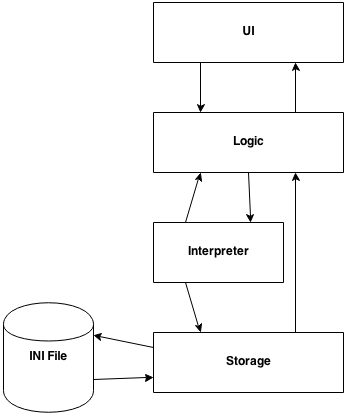


Figure 1 – High level logic of Tasuke

*For the full software architecture diagram, please go to Appendix 8.2.*

Tasuke uses four-layer architecture as shown above.

* The **UI layer** directly interacts with the user in the form of a graphical interface, and presents information to the user. It also fetches data from the user and passes it to Logic.
* The **Logic layer** handles user commands and manipulates data entered by the user.
* The **Interpreter layer** helps to organize the user’s command into a format that can be read by Logic and Storage.
* The **Storage layer** manages the file system which data is stored in. It also handles the INI File, where settings are stored.

## 3.2 Logic

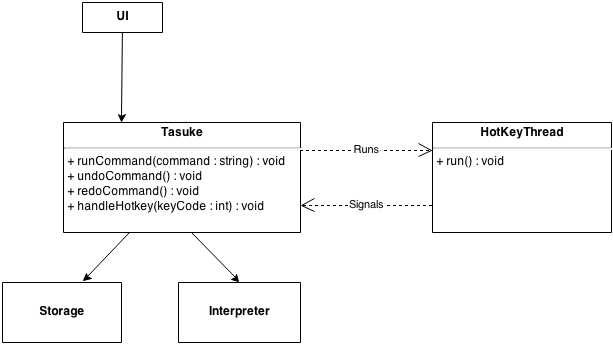


Figure 2 - logic subsystem

The **Logic** layer consists of the classes Tasuke, and HotKeyThread. Its main purpose is to maintain the state of the program, handle exceptions, and control the Interpreter and Storage.

* It calls **Interpreter** to interpret a string into a command, then executes the command.
* It calls on **Storage** to save or retrieve data. It also reacts to user actions by activating the UI classes in response to hotkey press.
* This layer also manages the HotKeyThread, which runs in the background to monitor for hotkey triggers. It reports any hotkey triggers to Tasuke, which then calls the appropriate UI window.

## 3.2 User Interface

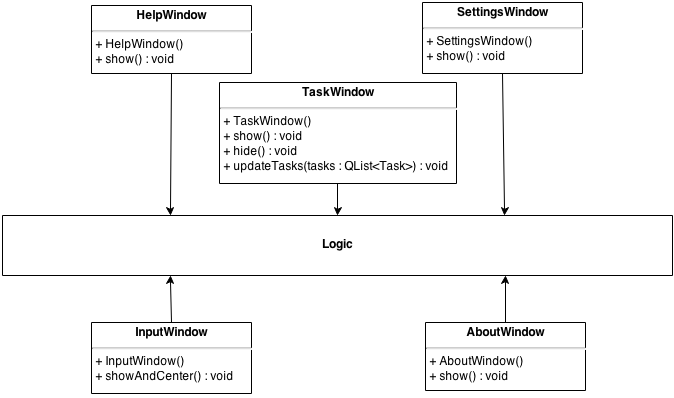


Figure 3 - User interface

The **UI** layer consists of the classes InputWindow, SettingsWindow, AboutWindow, HelpWindow, and TaskWindow. Each class represents a window in Tasuke.

* InputWindow is where the command box resides. This component also passes user commands to **Logic**.
* TaskWindow, also our main window, displays the list of tasks to the user. The list is provided by **Logic**.
* SettingsWindow is the place users go to configure Tasuke.
* HelpWindow is opened by the user when they need help.
* AboutWindow provides brief information about Tasuke for the user.

## 3.3 Interpreter

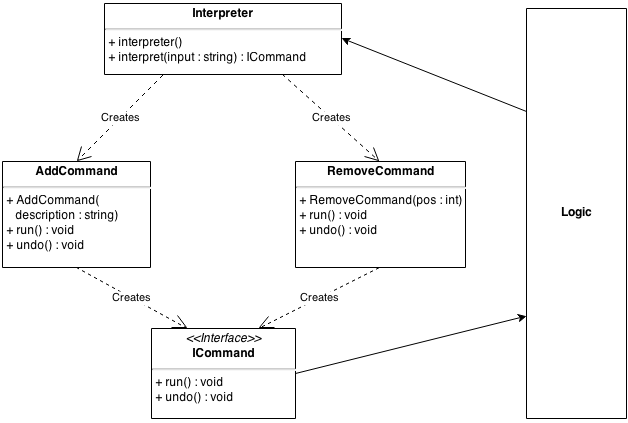


Figure 4 - Interpreter subsystem

The **Interpreter** layer does most of the heavy lifting by interpreting the text that a user enters into Tasuke, and translates them into a form that is understandable by Logic and Storage. It contains the functions that power Tasuke’s fast, powerful and flexible command structure.

New commands can easily be added by creating another command that conforms to the ICommand interface in Commands class inside Commands.cpp.

## 3.4 Storage

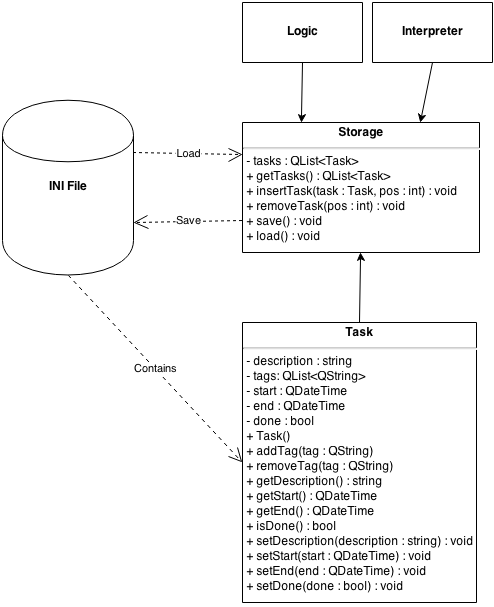


Figure 5 - Storage subsystem

The **Storage** layer handles interactions with the file system that Tasuke resides in. its main purpose is to write to the .ini file, as well as to read from it. It is responsible for transforming plaintext to task objects, which it then passes to Logic through Storage for further action, and vice versa.

If you wish to implement a storage subsystem that implements another form of storage such as JSON or databases, please note that the new subsystem must be able to wrap and unwrap Task objects, and must be able to respond to all public methods specified in Storage.cpp.

# 4. known issues and future work

For future versions of Tasuke, here is what we plan to add:

* Users should be able to append tags to tasks whether during creation or after creation. Tasks can then be configured to display differently depending on whether or not it has a particular tag. The tags can also be used to indicate urgency and importance.
* Better natural language command emulation. We plan to do this by expanding the dictionary size used to detect an appropriate English-like command, such as synonyms for existing commands.
* Highlighting command syntax in the input box. This is so that users can clearly see which words is part of which section, and demarcate them with command words.
* Sorting of the tasks in the Task Display, so that users can choose to display only what they need.
* More keyboard shortcuts, such as for undoing, redoing and scrolling.
* User reminders. We plan to implement some form of notification to the user when a task is nearing its due time. This may be anything from a tray icon message to a pop-up. It may be disabled.
* Spellcheck. Users can choose to turn on or off automatic spelling correction.

# 5. Appendices

## 5.1 Full Software Architecture Diagram

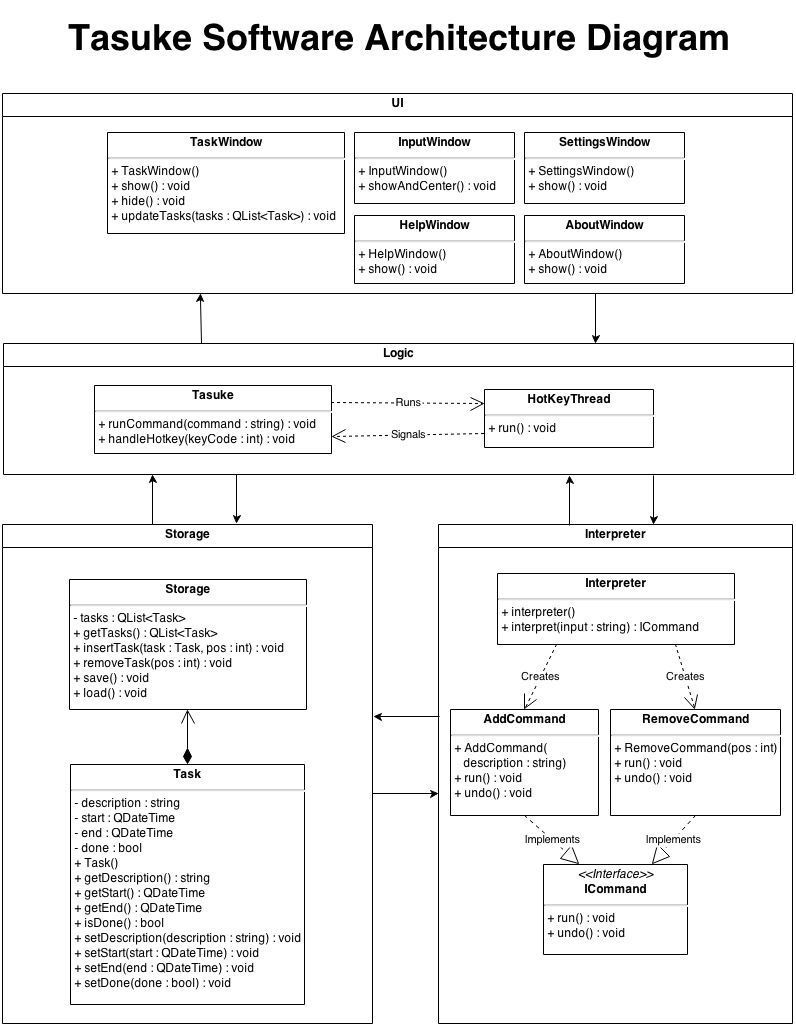


Figure 6 - Full Software architecture diagram

## 5.2 Sequence Diagrams

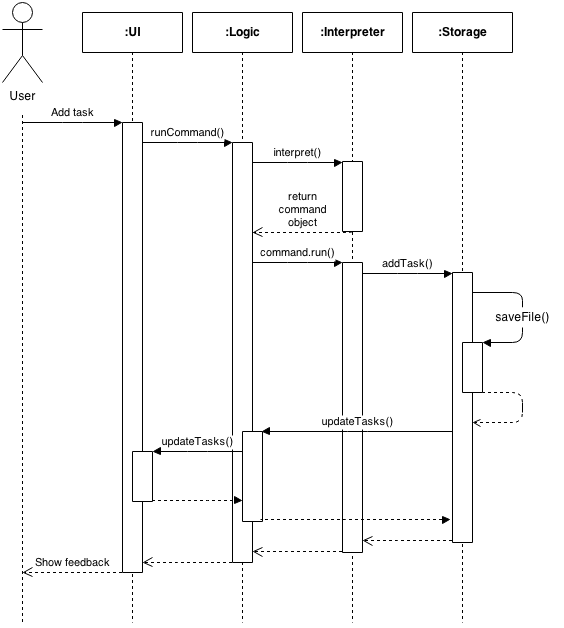


Figure 7 - Sequence diagram for add command

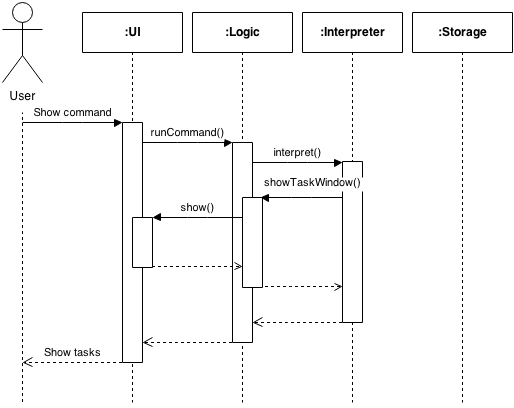


Figure 8 - Sequence diagram for show command

## 5.3 Application Programming Interface (API)

This is a list of API for Tasuke. Only exposed, essential classes are displayed for brevity.  
  
Storage

|  |  |
| --- | --- |
| **Modifier and Type** | **Function and Summary Description** |
| void | loadFile()  Opens a stored .ini file at %appdata% and loads it to memory. |
| void | saveFile()  Writes the contents in memory to an .ini file in %appdata%. Append only. |
| void | addTask(Task& task)  Adds a single task to the list of tasks in memory. |
| void | addTask(Task& task, int pos)  Adds a single task to the list of tasks in memory at position pos. |
| Task& | getTask(int pos)  Returns a single Task at position pos in the list of tasks in memory. |
| void | removeTask(int pos)  Removes the task located at position pos in the list of tasks in memory. |
| Task& | popTask()  Returns the next task in the list of tasks in memory. |
| QList<Task> | getTasks()  Returns the whole list of tasks |
| int | totalTasks()  Returns the total number of tasks in the list of tasks in memory. |

Task

|  |  |
| --- | --- |
| **Modifier and Type** | **Function and Description** |
| void | setDescription(QString& \_description)  Sets the description of a task to \_description. |
| QString const | getDescription()  Retrieves the description of the task. |
| void | addTag(QString& tag)  Appends a tag tag to the task. |
| void | removeTag(QString& tag)  Searches for tags that matches tag and removes it from the task. |
| QList<String> | getTags()  Retrieves all the tags of the task. |
| void | setBegin(QDateTime& \_begin) Sets or changes the starting time of a task to \_begin. |
| QDateTime const | getBegin()  Retrieves the starting time of the task. |
| void | setEnd(QDateTime& \_end)  Sets or changes the due time of a task to \_end. |
| QDateTime const | getEnd()  Retrieves the due time of the task. |
| void | setDone(bool \_done)  Sets the “done” status of the task to \_done. |
| void | markDone()  Sets the “done” status of the task to TRUE. |
| bool const | isDone()  Retrieves the “done” status of the task. |

Commands

|  |  |
| --- | --- |
| **Modifier and Type** | **Function and Description** |
| Constructor | AddCommand(Task& task) Constructs a command that has the ICommand interface. |
| Destructor | ~AddCommand() |
| void | run()  Executes the current command. |
| void | undo()  Undoes the command. |

## 5.4 Files within the Project

This is what the working directory should contain when it is freshly cloned from the repository, sorted by alphabetical order.

|  |  |  |
| --- | --- | --- |
| **Directory Root** | | |
| .hgIgnore  AssemblyDependencies.dgml  Tasuke.sln  [W15-2C][V0.1]DevGuide.docx | | |
|  | **Folder: Tasuke** | |
|  | **Folder: fonts** |
| PrintBold.otf  PrintClearly.otf  Quicksand\_Bold.otf  Quicksand\_Bold\_Oblique.otf  Quicksand\_Book.otf  Quicksand\_Book\_Oblique.otf  Quicksand\_Light.otf  Quicksand\_Light\_Oblique.otf |
| **.h Header Files** | |
| AboutWindow.h  CommandFactorry.h  Commands.h  Constants.h  Exceptions.h  HotKeyThread.h  InputWindow.h  Storage.h  Task.h  TaskEntry.  TaskWindow.h  Tasuke.h | |
| **.cpp Source Files** | |
| AboutWindow.cpp  CommandFactorry.cpp  Commands.cpp  Exceptions.cpp  HotKeyThread.cpp  InputWindow.cpp  main.cpp  Storage.cpp  Task.cpp  TaskEntry.cpp  TaskWindow.cpp  Tasuke.cpp | |
| **.ui UI Files** | |
| AboutWindow.ui  InputWindow.ui  SHAWindow.ui  TaskWindow.ui  TutorialWindow.ui | |
| **PNG Image Files** | |
| about.png  closebButton.png  closeButtoHover.png  InputWindowMask.png  minButton.png  minButtonHover.png  roundedEntryMask.png  roundedMask.png  TaskWindowBorderGrey.png  TaskWindowMaskWhite.png  Tasuke.png  Tasuke\_OLD.png  Traysuke.png | |
| **Misc** | |
| Resources.qrc  Tasuke.vcxproj  Tasuke.vcxproj.filters | |
|  | **Folder: UnitTests** | |
| stdafx.cpp  stdafx.h targetver.h TasukeTests.cpp UnitTests.vcxproj  UnitTests.vcxproj.filters | |