

T A S U K E

The Developer’s guide

# Contents

1. **PREFACE**
   1. Introduction
   2. Who should read this document?
   3. Concept and Design
   4. Organization of this document
   5. Conventions and Definitions
   6. Related Documents
   7. Updates and Feedback
2. **TASUKE BUILD ENVIRONMENT**
   1. Development Environment
   2. Execution Environment
   3. Application Files
3. **Software Architecture**
   1. Software Architecture Overview
   2. Logic
   3. User Interface
   4. Interpreter
   5. Storage
4. **Known Issues and Future Work**
5. **AppendiCES**
   1. Full Software Architecture Diagram
   2. Application Programming Interface
   3. Files within the Project
6. **Glossary**

# 1. PREFACE

* 1.1 Introduction

Welcome to the Tasuke Developer Guide!

Tasuke is a desktop task and event 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 and events
* Marking tasks as done or otherwise
* Setting a starting time and deadline for tasks and events
* Adding tags to tasks and events
* Searching for tasks by date, completion and tags
* Sorting the search result.

This guide aims to provide the reader with practical information of the design and implementation of Tasuke. This guide will detail the organization of the software, its API, and provide solutions for future development. This will also provide information in assisting with troubleshooting the program should the need arise.

* 1.2 Who should read this document?

This document is a guide intended for developers and maintainers of Tasuke.

Tasuke is written in C++ using the Qt framework for the Windows platform, and this document thus assumes that the reader is familiar with C++ at least. It is optimal if the reader is also familiar with Qt, but that is not a necessity.

* 1.3 Concept and Design

Tasuke was developed with speed and simplicity as its goal. Tasuke is lightweight in terms of computer resource consumption; it is easy to learn to use and neatly folds itself away when the information entry is done.

Tasuke takes input using an interface not unlike that of a command-line, and produces output in a visually minimal graphical window on screen.

In order for the program to be intuitive, the user commands for Tasuke follows normal language loosely, and is similar to some SQL statements. For more details on commands available to the user, refer to the user guide.

This principle should be followed as close as possible when developing and maintaining Tasuke.

* 1.4 Organization of this document

This document is written in such a way that debugging and troubleshooting Tasuke is done in a top-down manner. Thus, it can generally be divided into 3 parts: External, Intermediate and Internal. External refers to the external factors that can affect the programming and operation of Tasuke, intermediate refers to the expected flow and execution of Tasuke, and Internal refers to the actual code that Tasuke runs on.

This document will first explain the environment needed to develop and maintain Tasuke optimally, and provide detail on optimal execution environment of Tasuke as well as the files it generates. Then, this will explain the software architecture that is involved from input to output. Finally, this will provide detail on the implementation and API involved in the programming.

* 1.5 Conventions and Definitions

In this document, a **task** or an **event** refer to an object that a user will create when he or she types the *add* command followed by a task or event description. The terms are used interchangeably.

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. This guide is written assuming the reader is using a Windows environment to develop and maintain Tasuke.

* 1.6 Related Documents

As Tasuke is written with Qt 5.2.1, please refer to <http://qt-project.org/doc/> for the latest documentation relating to the methods and functions written with Qt. However, this document will attempt to annotate functions and methods that are from Qt and are used in Tasuke.

* 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. Therefore, the recommended development environment is as such. Note that Qt is not compatible with the Express edition of Microsoft Visual Studio, and therefore should be avoided.

As Qt is a cross-platform application framework, all code written within Tasuke is platform-agnostic. It is, in theory, possible to compile the source code of Tasuke on any platform, for any platform. However, this guide is written with developers and users of Microsoft Windows 7 and later in mind. Thus, this guide will not discuss issues arising when Tasuke is compiled in any platform besides Windows.

* 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

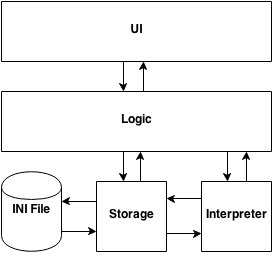


Figure – High level logic of Tasuke

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

Tasuke uses three-layer architecture. The components of Tasuke can be identified as three main groups, as shown above.

* 3.2 Logic

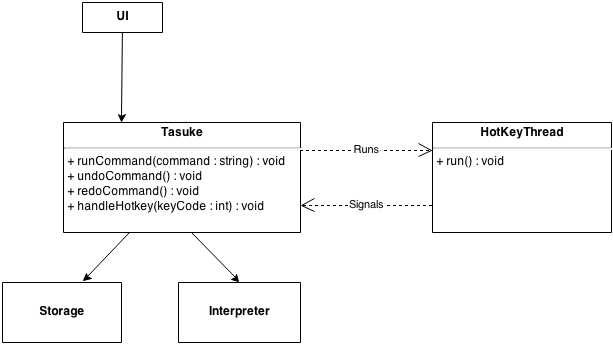


Figure - logic subsystem

The **Logic** layer consists of the classes Tasuke, HotKeyThread, and ICommand. Its main purpose is to maintain the state of the program, handling exceptions, and controlling the Interpreter and Storage. It calls Interpreter to interpret a string into a command, then executes the command, and 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

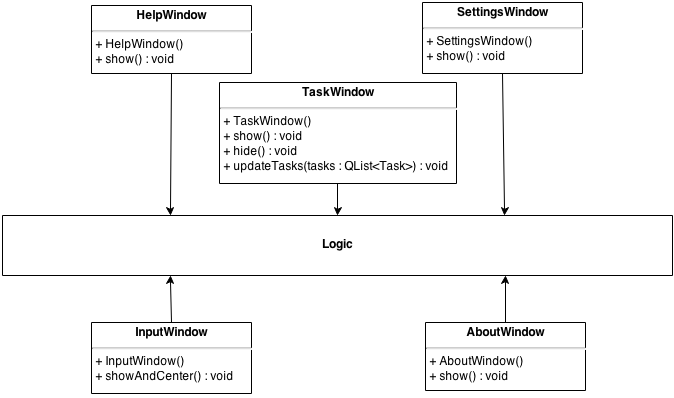


Figure - User interface

The **UI** layer consists of the classes InputWindow, SettingsWindow, AboutWindow, HelpWindow, and TaskWindow. This layer directly interacts with the user in the form of a graphical interface, and presents information to the user as well as fetching data from the user and passing it to Logic.

All changes to the UI while the program is in operation are determined by Logic, and UI by itself does not handle any changes to the information supplied to it.

If you wish to add more windows, please note that the Logic subsystem must be able to compensate for the addition.

* 3.3 Interpreter

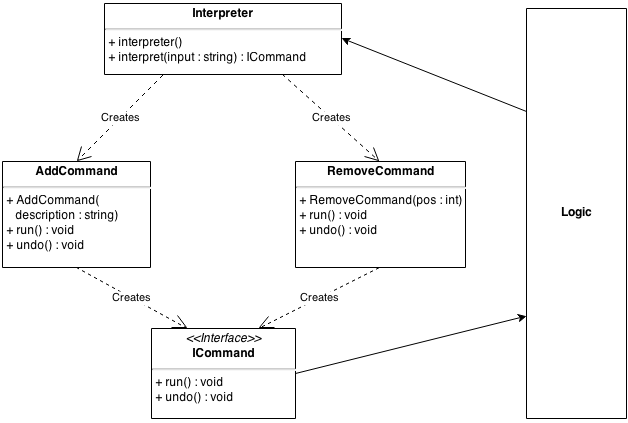


Figure - Interpreter subsystem

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

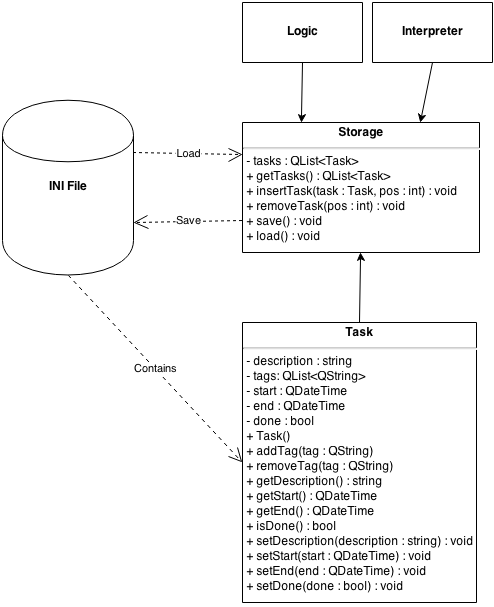


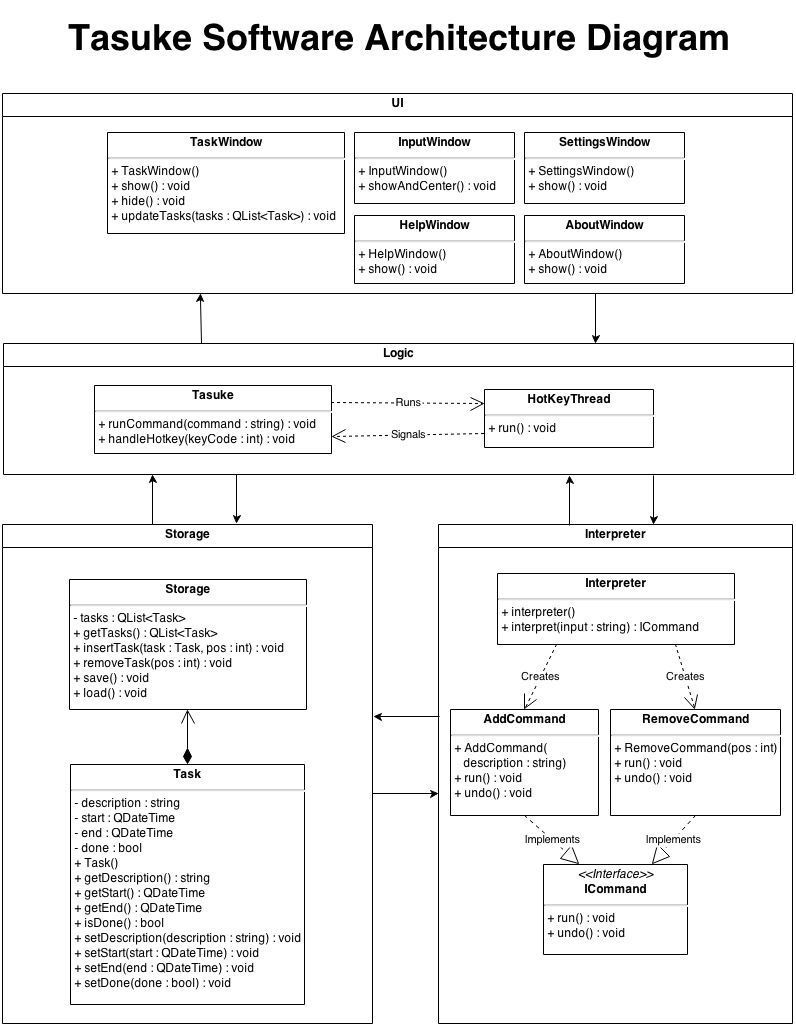
Figure - Storage subsystem

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.

# 5. APPENDICES

* 8.1 Full Software Architecture Diagram



* 8.2 List of Functions

This is a list of all functions and methods that exist in the Tasuke source code, sorted according to the .cpp file they reside in.

* 8.3 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][V0.0]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 | |