Raspberry Pi Smart Mirror Project Design and Implementation

Scott English
Thomas Sanchez
Ryan Faulkenberry
Teegan Duong

TABLE OF CONTENTS

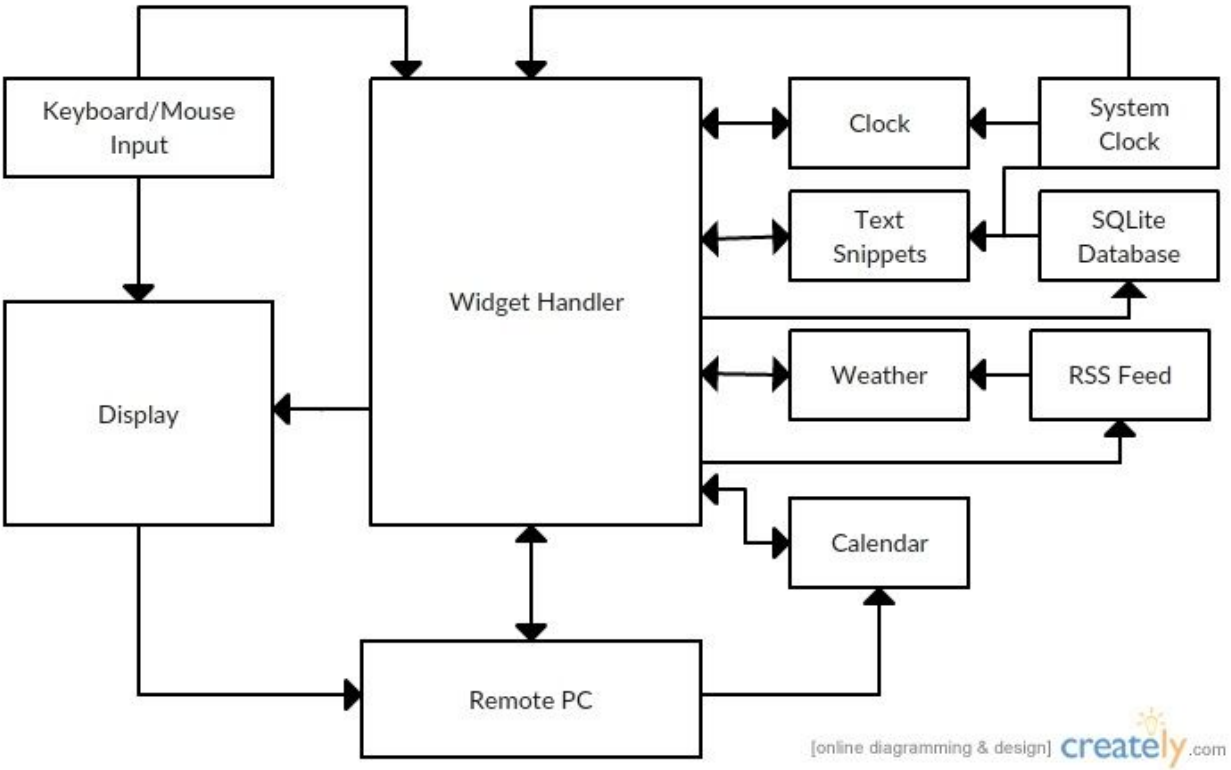
Preface

The expected readership of this document shall be the development team, as well as the product owner. This document shall be designed to describe the systems design and implementation for use during product development. This is the first version of this document. This version shall lay out the overall architecture and is intended to map out a general blueprint for use in system implementation. Here is a presentation of our systems design and implementation: Raspberry Pi Smart Mirror Display Design and Implementation Presentation

1.Glossary

- **Raspberry Pi**: is a tiny affordable computer. (https://www.raspberrypi.org)
- **DIY**: Do it yourself, common term when describing projects that involve doing all of the work yourself
- **One-way mirror**: A mirror that is partially reflective and partially transparent. Allows the user to see through while still seeing their reflection as if they were staring into a mirror
- **Smart Mirror Project**: a DIY project where you take a one-way mirror and put a monitor behind it and display images through the mirror as you look at yourself in the mirror.
- **SDL2**: Simple DirectMedia Library, we are using to enable our GUI and keyboard/mouse input
- **SQLite3**: Structured Query Language, we are using the SQLite3 version of the SQL software to implement a self-contained, serverless, zero-configuration,transactional database that is local to the Raspberry Pi. (https://www.sqlite.org/about.html)
- **Raspbian**: The operating system most commonly used on Raspberry Pi's
- **Widget**: an application, or a component of an interface, that enables a user to perform a function or access a service.
- **ARM:** a processor type that uses reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. (https://en.wikipedia.org/wiki/ARM_architecture)
- **IDE:** An **integrated development environment** (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. (https://en.wikipedia.org/wiki/Integrated_development_environment)

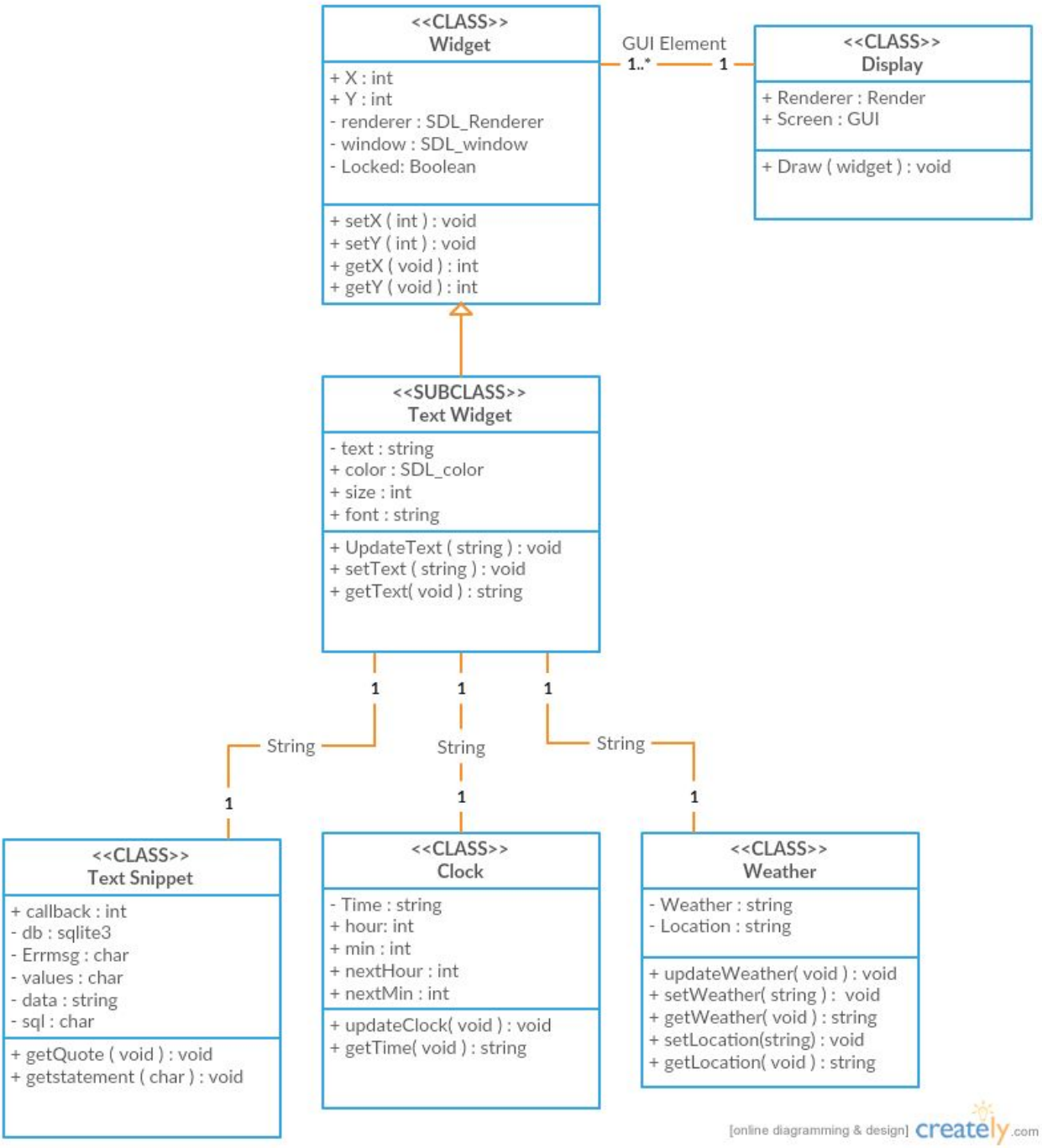
2. Interface Specifications



The interface interactions in this system can be broken into four major implemented groups of interactions:

- **Widget - Handler**: Each of the widgets (The clock, text snippets, weather, calendar) interacts with the widget handler by serving information about the widget to be fed to the user output.
- **Handler - Widget:** The widget handler sets sets the information about widgets (position on screen, color, size, text to display) based on the information passed to it by the user input.
- **Input - Handler:** The user input changes the state of the widget handler, which will in turn alter the widgets within the system.
- **Handler - Display:** The widget handler captures the needed drawing information from each widget in the system, and sends the information to the screen to be displayed.

3. Objects of the System



- For our system we have 4 objects, Widget objects which are drawn and shown on the Display, Text Widgets which are a subclass of Widget, Display which handle drawing and updating the Widgets it is currently showing. Three of the objects we have are text widgets labeled Clock, Text Snippet, and Weather which are provided string data from their respective class and the last one being the Display

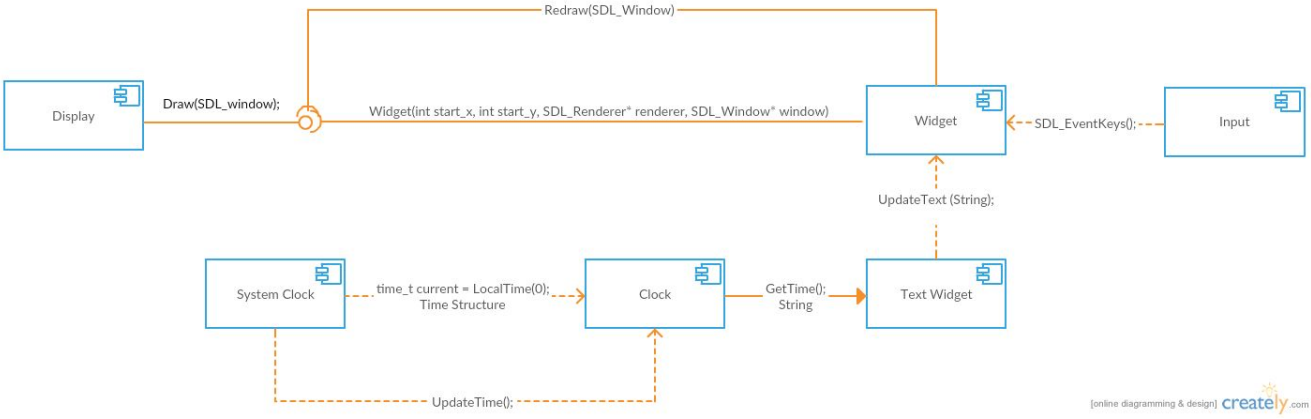
- The Display object in our system will be directly responsible for maintaining the renderer. This renderer will hold all the currently created widgets and will display them in the interface. Upon getting certain event keys from the I/O devices the Display object will run a check to determine which action to take in order to either resize the dimensions of the specific widget, move the location on the interface, or hide/ show on the interface. After every check, the Display will auto redraw the Widget object at it new location, dimensions, or not draw it at all if hidden.

- The Widget object in our system is an individual object with a 1..* to 1 relation to the Display object. We have three of these labeled Clock, Text Snippet and Weather. These Widgets have an initial (x,y) coordinate on the Display but that can changes using the getX and the getY to add or subtract a predefined incremented value to its original coordinate then setX and setY method from within the Display object in order to update its location on the interface that is defined in the render and window variable. If the object is locked however no change will be made to its coordinates, this is done by the check within the Display object using the Widgets setLocked method to either set it to locked or unlocked after the certain event keys. The next time the Display draws the Object it will be in the new location and will either hidden or shown. The three widgets we have all use this same methods and functionity described above.

- The Text Widget Object is a subclass of Widget so the coordinate for its location, locked, and hidden variables are inherited from the parent class.The 3 objects we have drawn to our display are of this class. The new variables added are related to text and fonts which include its Text, what the widget will display, its Font, what the widget will display in, the size and color of the font as well. Of these variables the Text will be changed from within the Display object whenever the method ChangeText is called, there are different times for this to happen. The clock text widget will change every second to provide a real time graphical interface representing the current time whereas the Text Widget object called Text Snippet will change once every predetermined time interval, this distinction is also made for weather text widget which will update every minute. Allowing the individual text widgets objects to be updated separate of each other reduces the consumption of system resources by prioritizing the number of times an object is redrawn. In example, a clock needs to be updated every second in order to be real time whereas the weather doesn’t change every second

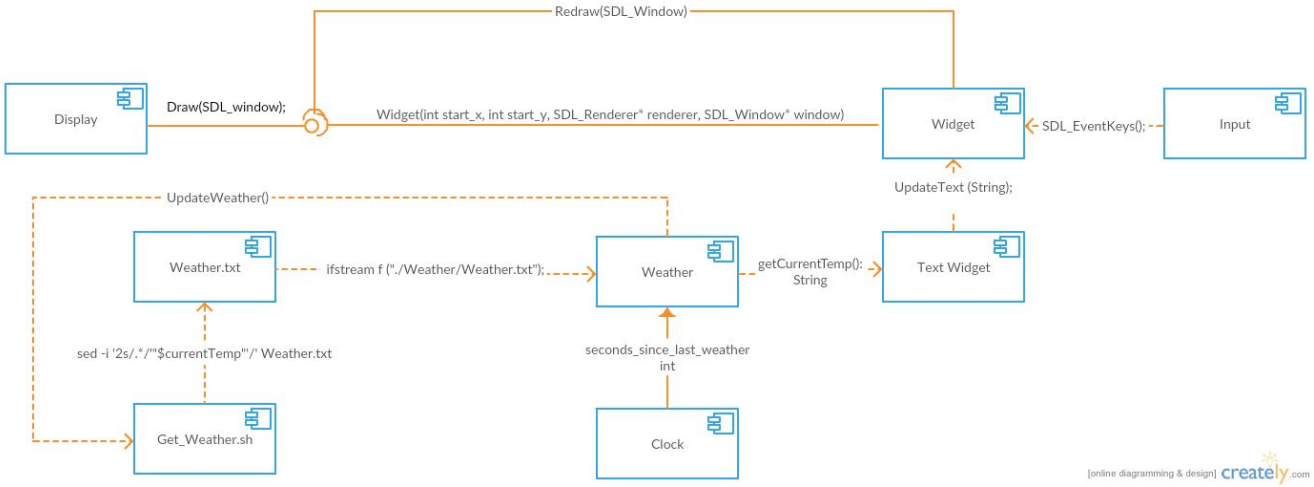
4. Low Level Design

- Clock Object Low Level Design



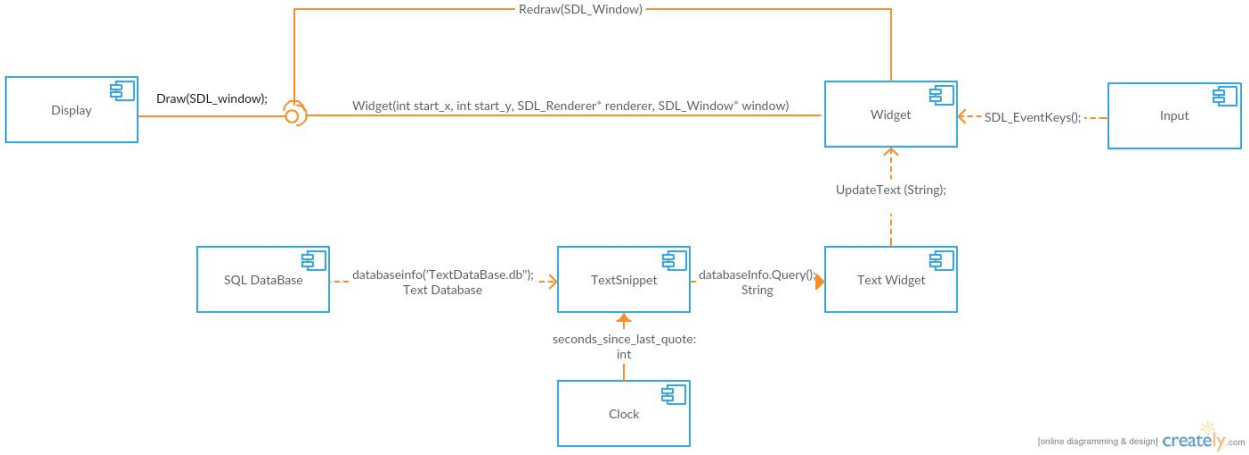
. The Clock class will create a structure holding the current time upon its creation, this structure will be formatted into a string variable, which will be passed from the clock class into the constructor for the Clock object using the getTIme method. This object will then be displayed on the interface by having the Display object draw it. Since the clock is real time the string variable will need to be update every second, this is handled within the Clock class and sent to the Clock text widget object through the updateText method within the widget and redrawn onto the Display object.

- Weather Object Low Level Design



At the system's initialization a script labeled Get_weather.sh is ran which will write the current temperature into a text file, then the Weather class will read the file and format its information into a string variable, this string variable is passed out of the Weather class into the constructor of the text widget and then drawn to the display. A integer variable tracking seconds since the last update is started and upon reaching its preset elapsed time the method update weather is called which will run the script again, updating the text file and subsequently the string that is passed into the updateText method called from within the Weather text widget and updated on the interface visually by redraw method called from Display object.

- Text Snippet Low Level Design



The TextSnippet class initializes a text database at the system's start. There is an initial call to the DatabaseQuery method which accesses the database and pulls a random quote and assigns it to a string variable, this string is passed to the Text Snippet text widget's constructor. This allows for the Object to be created and shown on the interface by having the display object draw it. It will also initialize a counter to see how many seconds have passed since the last quote, when it allotted time has passed the Text Snippet text widget object calls the updateText method with a new call to the databaseQuery method as its string parameter, this pulls a new random quote and allows for the Display object to call redraw method and show a new quote on the interface.

5. Development Details

5.1 Size of Project Details

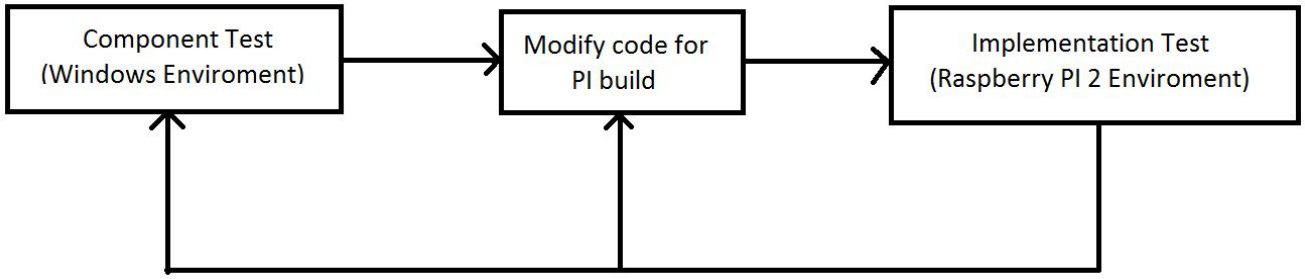
- The total completed project totalled 932 lines of C++ code, with a projected total of 1100 lines after implementing remote PC connectivity and a calendar widget. Most of the code was built with modularity in mind, thus we had very little code reuse.
- The total completed project totalled 56 lines of bash script code, and we do not project there to be any need for further bash script code in the evolution of this software.
- Because we spent time initially planning for modularity of code, every additional functional component is projected to be only 100-200 additional lines, so the size of the project even into the future is expected to be very manageable.

5.2 Development Process and IDE Details

- Github was our primary way of source control. We followed an agile development schedule known as SCRUM and every 2 weeks or so, we would push our work onto github and the SCRUM master would look it over and push it to main after testing. We did started this development style on the 20th of February, 2017 and finished it the 4th of May, 2017.
- Through this development process we used discord, group text, and class time to discuss our projected timeline of events and work for the project as we progressed through the development period.
- Our primary IDE for the development of the software was Visual Studio and Xcode. We used these IDE's primarily because Visual Studio worked well with everyone's computers and since Scott's computer was a 2015 MacBook Pro, he used Xcode to test everyones code and also write his because it was the closest we could get to a linux environment without using the pi.
    - Both of these IDE's have easy integration with Github.
    - They both also work extremely well with our included libraries SDL and SQLite3.

6. Build Process

Figure 6.1:

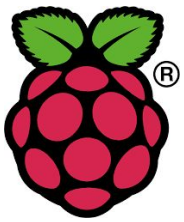


Our build process follows a general structure given by figure 6.1 above.

The detailed process for each component build closely follows:

1. Each initial component test build is built and tested on a Windows 7 or Mac OS machine (A linux virtual environment is used for some UNIX-specific component testing).
2. Once the component test build is working as specified by the requirements, it is integrated into the project and interface tests are conducted.
3. Once the interface tests are each passed, the project is modified to run on the Raspbian system. This process involves altering the library headers to conform with the Raspbian structure.
4. The modified project with the new component is then built and tested on the Raspbian system.
5. If the new components operate as specified by the functional requirements document, we then test for performance in order to comply with non-functional requirements.
6. This process is repeated until each component is implemented.

7. Targeted Systems



7.1 System Performance and Targeted Systems

- The expected targeted systems of our software shall be Raspberry Pi's. Most Raspberry Pi's running the basic operating system (Raspbian) imaged onto the sd card straight out of the box, shall be able to run our software. Our software shall be capable of being run on ARM type architectured processors as well.
- The minimum system requirements for our software shall be a Raspberry Pi that is running Raspbian and connection to the internet. The Raspberry Pi 3 has built in wifi and bluetooth connectivity and is recommended for this project. If A Pi 2 or anything below is used, it's recommended that a network adapter be purchased so that the software can function properly.
- For Optimal Performance a Raspberry Pi 3 is recommended.
    - The Pi 3 has a quad core processor that is four times as fast as earlier Pi models, wifi and bluetooth capabilities built in, and 1 Gb of Ram.
    - System performance is better on a Pi 3 because the processor speed is higher and since wifi and bluetooth are builtin, there is less latency from communicating with them then using a wifi adapter plugged into a usb port on the Pi.
- For Recommended Performance a Raspberry Pi 2 is suggested.
    - The Pi 2 is a dual core system that shall perform the functionality of the software in a reasonable amount of time.
    - The processor is not as fast as a Pi 3 so the computation time for the software will take a little longer then a Pi 3. A network adapter must be plugged in so network connectivity and speeds might be slower then a Pi 3 as well.
- Anything below a Pi 2 is not recommended because the time complexity is not optimized for anything below the model Pi 2.

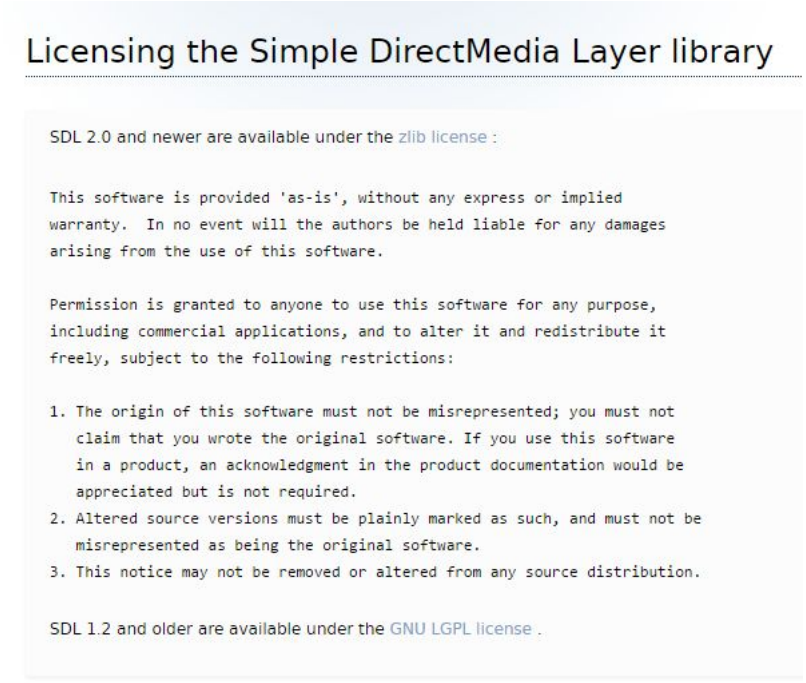
7.2 Libraries and Supporting Databases

- Our supporting libraries are SDL2, SQLite3, and Weather Utility.
    - SDL2 shall be our main renderer for our program.
    - SQLite3 shall be our main database used for storing information such as text quotes.

- Weather Utility shall be our main way of obtaining the weather info for the city we are currently in.

8. Open Sourcing and Licensing

- Our software is completely open sourced. We have no intention of selling this product and have made it free to the public via github following this link: https://github.com/Dungeongate/Smart-Mirror-Display
- Included in our software is some free libraries such as SDL2 and SQLite3. Both of these libraries are open source and as stated by their licensing can be used freely so long as credit is attributed to their software where needed.



Licensing the Simple DirectMedia Layer library

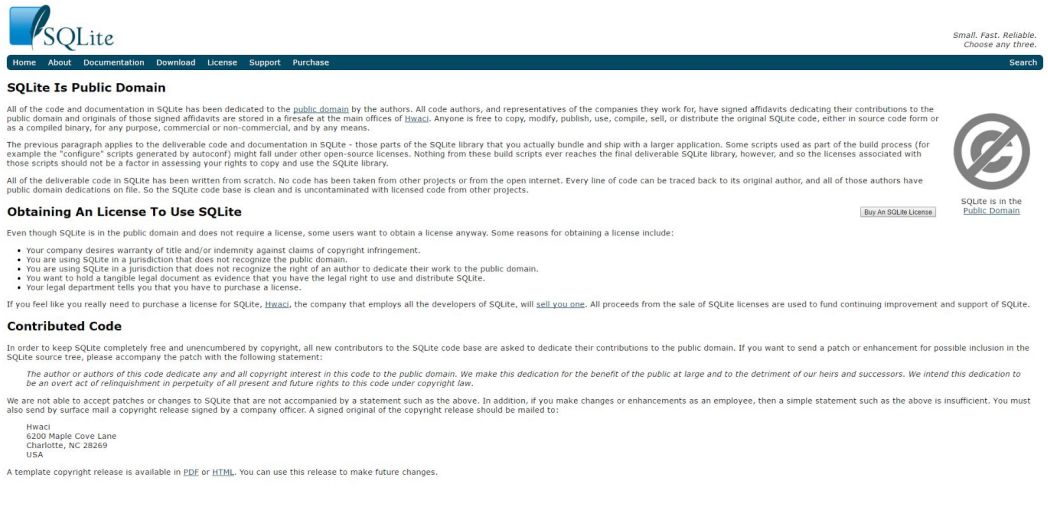
SDL 2.0 and newer are available under the zlib license :

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

SDL 1.2 and older are available under the GNU LGPL license .



SQLite

Small. Fast. Reliable. Choose any three.

Home About Documentation Download License Support Purchase Search

SQLite Is Public Domain

All of the code and documentation in SQLite has been dedicated to the public domain by the authors. All code authors, and representatives of the companies they work for, have signed affidavits dedicating their contributions to the public domain and originals of those signed affidavits are stored in a firesafe at the main offices of Hwaci. Anyone is free to copy, modify, publish, use, compile, sell, or distribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

The previous paragraph applies to the deliverable code and documentation in SQLite - those parts of the SQLite library that you actually bundle and ship with a larger application. Some scripts used as part of the build process (for example the "configure" scripts generated by autoconf) might fall under other open-source licenses. Nothing from these build scripts ever reaches the final deliverable SQLite library, however, and so the licenses associated with those scripts should not be a factor in assessing your rights to copy and use the SQLite library.

All of the deliverable code in SQLite has been written from scratch. No code has been taken from other projects or from the open internet. Every line of code can be traced back to its original author, and all of those authors have public domain dedications on file. So the SQLite code base is clean and is uncontaminated with licensed code from other projects.

SQLite is in the Public Domain

Obtaining An License To Use SQLite

Buy An SQLite License

Even though SQLite is in the public domain and does not require a license, some users want to obtain a license anyway. Some reasons for obtaining a license include:

- Your company desires warranty of title and/or indemnity against claims of copyright infringement.
- You are using SQLite in a jurisdiction that does not recognize the public domain.
- You are using SQLite in a jurisdiction that does not recognize the right of an author to dedicate their work to the public domain.
- You want to hold a tangible legal document as evidence that you have the legal right to use and distribute SQLite.
- Your legal department tells you that you have to purchase a license.

If you feel like you really need to purchase a license for SQLite, Hwaci, the company that employs all the developers of SQLite, will sell you one. All proceeds from the sale of SQLite licenses are used to fund continuing improvement and support of SQLite.

Contributed Code

In order to keep SQLite completely free and unencumbered by copyright, all new contributors to the SQLite code base are asked to dedicate their contributions to the public domain. If you want to send a patch or enhancement for possible inclusion in the SQLite source tree, please accompany the patch with the following statement:

The author or authors of this code dedicate any and all copyright interest in this code to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this code under copyright law.

We are not able to accept patches or changes to SQLite that are not accompanied by a statement such as the above. In addition, if you make changes or enhancements as an employee, then a simple statement such as the above is insufficient. You must also send by surface mail a copyright release signed by a company officer. A signed original of the copyright release should be mailed to:

Hwaci
6200 Maple Cove Lane
Charlotte, NC 28269
USA

A template copyright release is available in PDF or HTML. You can use this release to make future changes.