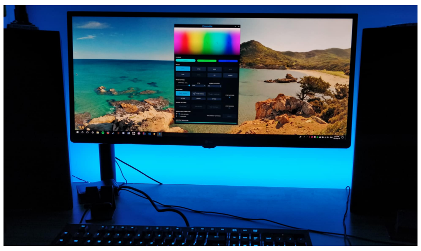
Riley Blair 300371586

Remote LED Customisation & Control

Chromolite Architecture Essay

Leyton Blackler (github.com/leytonblackler) is a third-year software engineering student at Victoria University of Wellington and is the sole developer of Chromolite, a remote RGB LED customiser and controller. While there are a number of similar open source projects, Chromolite sets itself apart by only requiring an Arduino micro-controller in order to operate. Chromolite enables RGB LED devices to become IoT devices with remote connectivity. The extensible source code make this software suitable for a wide range of users wishing to customise or control a number of LED-lit devices remotely and Chromolite provides several user interfaces in order to do so. The software and source code are licensed under a standard MIT licence and distributed in an 'as is' state acknowledging that some aspects of the software may not run or function as intended. The architecture of these types of applications generally involve a interactive user interface that communicates via wireless or serial communication channels to a controlling device, which in this case is an Arduino micro-controller. This micro-controller is then connected directly to the LED device, enabling it as an IoT device that can be controlled from the user interface. Users can generally customise colours, patterns and flash-cycles from the interface and can immediately visualise the outcome on the connected IoT device.



To execute the desktop application I simply executed the compiled JAR file containing all necessary dependencies and also initialised the Arduino application to receive serial communications from my computer. The latest build of the project appears to not contain any fundamental issues or warnings that hinder its use. The application is use with functional output can bee seen in the image on the right.

Background

Many individuals including gamers, smart-home enthusiasts and every-day computer users use RGB LED lighting to provide a mood or atmosphere to a room. Standard LED lighting is becoming more and more popular due to its low cost and every efficiency, making it suitable for those on a budget. Previously LED strips were the most common of these lighting solutions as they are cheap and easy to set up, with many supporting control and customisation from a connected device. Many manufacturers are now including similar technologies in standard home lighting, appliances and computer-related devices. This growing trend within the lighting industry is becoming more and more popular with every-day users and now many want to have the option to customise and control the settings of these LEDs rather than just simply turning them on or off. Devices such as Phillips Hue™, Razer Chroma™ peripherals and may other lighting solutions support significant levels of customisation but each require a separate interface for users to communicate with. Chromolite aim to solve this issue by being an all-in-one solution to control, customise, and synchronise popular RGB LED devices. The open source nature of the software enables developers to provide integrations to a wider set of devices in order to unify the control of LED devices for end-users.

Software Outline

Chromolite provides a user with a large amount of control over RGB LED devices that would previously not support this level of customisation or control. Devices that are primarily used with Chromolite include WS2812B LED strips and Razer Chroma™ computer peripherals. A user interacts with these devices using an abstracted user interface, making the use of the software much simpler for any first-time user. Chromolite currently supports two user interfaces including a multi-platform desktop application and a mobile-based Android application.

The multi-platform desktop application, writing in Processing 3 and Java, controls the connected IoT devices through a serial communication channel with an Arduino micro-controller. This micro-controller must be initially configured to operate Chromolite's micro-controller API which was written in C and C++ specifically for the Arduino micro-controller. Users are generally able to operate and set up the Chromolite application and necessary hardware with minimal technical knowledge, only requiring the connection of a small number of wires and possibly soldiering for a more permanent implementation of the overall system.

The Android application, written in Processing 3 and Java, provides a remote user interface that communicates with the desktop application via a Wi-Fi or Bluetooth communication channel which requires both the desktop-based server application and Android device running the Chromolite mobile application to be connected to the internet. As previously mentioned, the desktop application acts as a server on the local network that can receive incoming communications from devices connected to the correct IP address and port. The second communication method, Bluetooth, provides a different architectural view of the software, bypassing the desktop server and application and communicating directly with the Arduino micro-controller using a Bluetooth module connected to it. Using Bluetooth for controlling the Cromolite system requires the user to be within a limited range of the Arduino, and as such, Wi-Fi based control is more widely used.

History

Development of Chromolite began in June of 2016 with additions to the software being progressively added over time. The Chromolite source code was originally written in Processing 3, a Java-based programming language for developing applications within the visual arts genre. Leyton is currently in the process of changing the code-base to be implemented fully in Java instead of Processing 3 due to its applicability for a greater number of end-users.

Chromolite was originally developed to support the customisation and control of WS2812B RGD LED strips such as those used used for mood-lighting and home installations due to their low cost and applicability in most situations where RGB LEDs are required. This type of device was chosen by Leyton due to its ability to support control and customisation through a micro-controller with little programming required in order to perform basic customisation and control operations. Further to this, Chromolite supports control and customisation of RGB LEDs built into Razer Chroma™ computer peripherals using the Razer Chroma™ SDK's RESTful API. This allows users to control LEDs built into computer mice, keyboards and similar devices. Support for Razer products was chosen due to their popularity within the gaming community and their large range of LED customisation features that can be accessed through the REST API.

The current version of the software (v2.0) as of the writing of this report is in a relatively stable condition and supports eight customisation modes, three colours, light adjustment (brightness and gradient) as well as saving and loading of customisation presets. The Android application can currently connect over a Wi-Fi network, but known issues currently exist with the Bluetooth communication. A simulator of the LED strip output has recently been added into Chromolite and allows users to visualise the output on their device as well as on the LEDs themselves.

Development is currently underway to support Bluetooth architecture in a similar manner to the Wi-Fi based communication channel, where communications to the Arduino would be routed through the desktop application. This architectural change is to enable a lower cost implementation of the overall system for end-users, as users would no longer need to purchase the separate Arduino Bluetooth module. This requires a desktop device running the Chromolite application and the mobile device running the Android application. Both devices must support Bluetooth in order for this network to be used. Communications sent over the Bluetooth channel are processed in a similar manner to those over the Wi-Fi communication channel previously discussed, the only difference being the API that allows the support for each network type. Future development and support for Phillips Hue™ smart-lighting devices is currently planned and being ideated by Leyton in order to find the best approach to its implementation. Alongside this, extending the applicability of the mobile application to iOS devices is planned but due to the developer's other commitments, work has not yet begun on this feature of the overall system architecture.

Domain

Due to the nature of the Chromolite system supporting a number of vastly different RGB LED devices, the software functions across a number of different domains. The technological domain involved in Chromolite's software and required hardware is primarily customisable lighting, but can be narrowed down to a number of different smaller domains that contribute to the overall system including embedded systems (Arduino micro-controller) and real-time communication to these embedded systems (device communication across Bluetooth and Wi-Fi networks).

The system domain as described in the background section earlier in this report is not a recently emerged domain, but the ability to control devices within this domain is steadily gaining traction due to increasing user needs and the number of devices that support technologies that enable customisation and control of lighting systems. The current version of Chromolite supports two out of the three major device categories within the customisable lighting domain. These being RGB LED strips and computer peripherals, with the third, general-use lighting (Phillips Hue™, Belkin WeMo™, etc.) currently in development. Having such a large coverage and applicability to the major categories and specific devices within this domain enables Chromolite to be a highly suitable solution for the entire domain, rather than just a small sub-section of the domain as many other alternatives are. Chromolite's major purpose is to solve a significant issue that currently exists with this domain, the unification of customisation and control of various domain-specific devices through a single interface.

The embedded systems domain within this system enables wireless control of devices using minimal components and resources. Providing control of all connected devices through a single embedded Arduino micro-controller allows for users to perform an initial setup once and continue to use the system without needing to directly interact with major hardware components. Chromolite's embedded systems incorporates the Arduino micro-controlled running lightweight C and C++ programs that enable the control of multiple RGB LED devices. Connected to this micro-controller is the devices themselves, either WS2812B RGB LED strips or Razer Chroma™ peripherals. The nature of having these devices running through an embedded system enables the user to achieve their desired functionality wirelessly while also providing a hardware interface that can be applied to a number of similar lighting products within the primary domain, with little changes in the overall system architecture required. Major areas of concern that Chromolite addresses within the embedded systems domain include the cost of setup and implementation, and the ability to have the system run remotely without constant user attention or input, generally just providing feedback through a user interface.

Real-time communications is another sub-domain that forms a critical component of the primary domain. The Android application and desktop application act as a client and never respectively, with the desktop application functioning as both client and server in order to connect to the embedded system. These components must connect in real-time to the embedded system in order to provide feedback to the user based upon their actions within the user interface. The efficiency and latency of everything happening within the real-time communications domain enables the embedded system domain to also form a further user interface that is a visual output for the user's actions. Critical areas of concern for all systems functioning in a real-time communications domain include the latency of communication and the implementation of peer-to-peer system architecture that enables this.

Component Architecture

The Cromolite system involves a number of different hardware and software components which are constructed using the well-known client-server system architecture. The first major component within the system is the distributed client-server architecture of the Arduino micro-controller and connected RGB LED device, whether that be a strip of RGB LEDs or a Razer Chroma™ peripheral. The second significant component architecture is the client-server architecture of the wireless Bluetooth connection between the Android application and embedded Chromolite system enabling Chromolite to be installed in situations where a connection through the Chromolite desktop application may not be appropriate or viable. This component architecture differs only slightly from that of the Wi-Fi and serial communication based client-server architecture that utilises the mobile or desktop application as the client and the desktop application as the server, routing information through a serial communication channel to the Arduino as the client. By using this system architecture for different use-cases, Cromolite is more applicable in a variety of circumstances and is more applicable to users needing a variety of different control and customisation options.

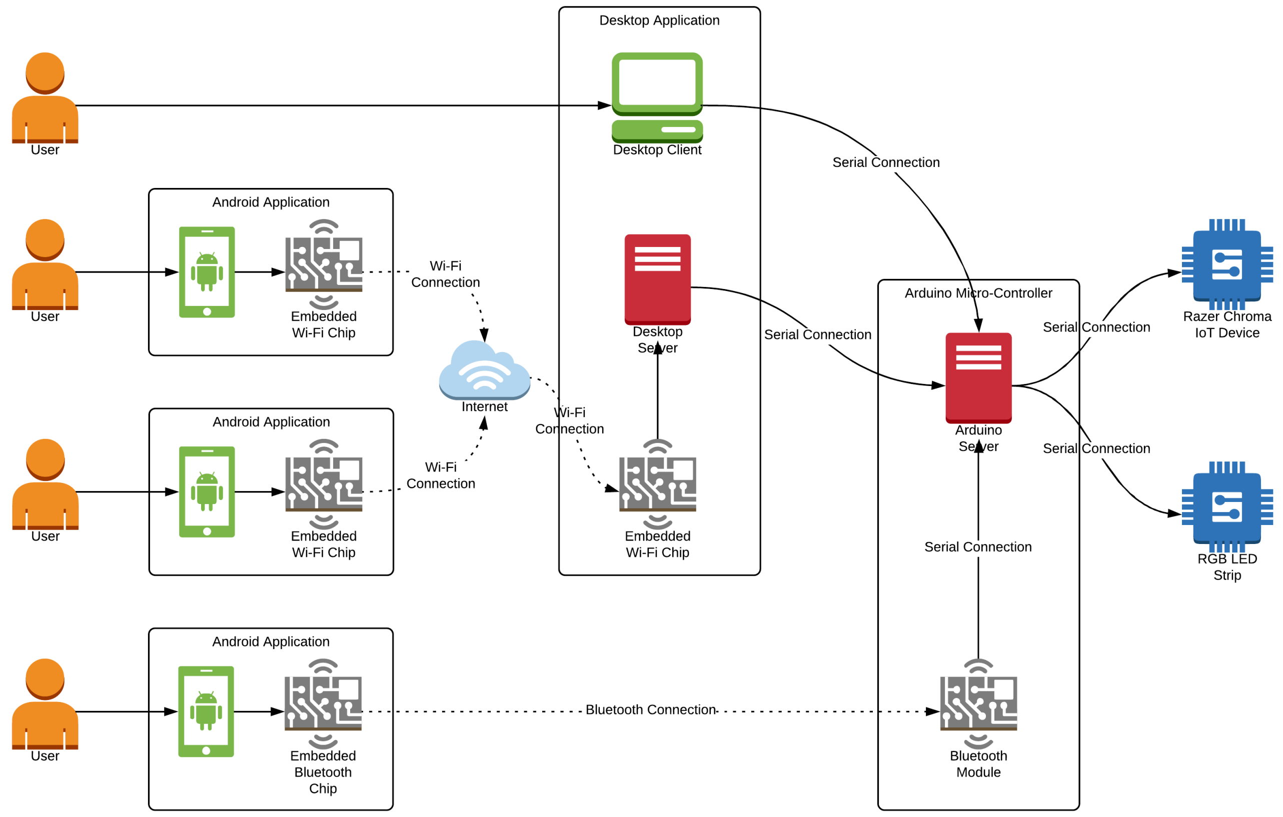
The connection of the Arduino to the connected IoT devices is best modelled by a distributed client-server architecture model, where each component is a separate system block and these components are connected together to achieve the desired system output. The distributed system architecture used here enables blocks of the system (in this case, the connected IoT devices) to be swapped out or replaced within the embedded system, thus achieving system growth. The use of this architectural model in Chromolite's design enables the system to grow and scale with multiple workloads, with this primarily being the operation of a variety of different connected devices within the customisable lighting domain identified above. The client-server architecture of this section of the overall system enables the Arduino to act as a server, distributing commands and information to connected clients. Making this a distributed client-server architecture enables greater growth of the overall system as components can be swapped in and out as required by the end-user with little technical knowledge required.

Chromolite's client-server system architecture enables the abstraction of system components while also providing low-latency communication between the user's interface (the Android application or desktop application) and the embedded Arduino micro-controller. By utilising this architectural style for the communication channels, Chromolite enables the output from the connected IoT devices to become another user interface, where the user can immediately see the results of their actions from elsewhere in the system. Cromolite's use of this system architecture allows for multiple different user interfaces to connect with the Arduino micro-controller which operates as a distributed embedded server.

This architecture is used within the forward-facing section of the system in three different use-cases, the first two being direct communication between the client (desktop application or Android application) and the server (Arduino micro-controller). In the first instance, the client is the desktop application (client) that provides a user interface through which the user can perform a number of useful actions. This information is sent to an API that converts the instructions into a useful format for the physical serial connection between the computer running the desktop application (client) and the micro-controller (server) which receives instructions and distributes them to the connected devices through established APIs.

The second of these three use cases is a user interaction with the system through a wireless mobile device (client) directly to the server through the Arduino (server) through a wireless Bluetooth communication channel to the Arduino's Bluetooth module. Again, once information reaches the micro-controller distribute commands to the connected devices through established APIs. The key difference between the first and second use-cases explain here is the method of communication between the client and server. In the first case a physical connection is used with an API, and in the second, a wireless communication is used with the between client and server using an API. The majority of the communications in both cases are from the client directly to the server which sends one-way communications to the other clients (connected IoT devices).

The third use-case expands on this client-server architecture by using server-server (inter-server) communication when the user connects their phone to the desktop application using a Wi-Fi connection. The desktop application then forwards this information on to the micro-controller. In this case the mobile phone acts as a client with a connection to the desktop application acting as a server over a Wi-Fi connection. This server then connects to the Arduino through the same physical serial communication channel as described in the first use case. By utilising server-server architecture for this use-case, Cromolite enables multiple phones (clients) to be connected and control the IoT LED devices simultaneously which is not possible when connected to the Arduino directly as it only allows for a single input, but multiple outputs.



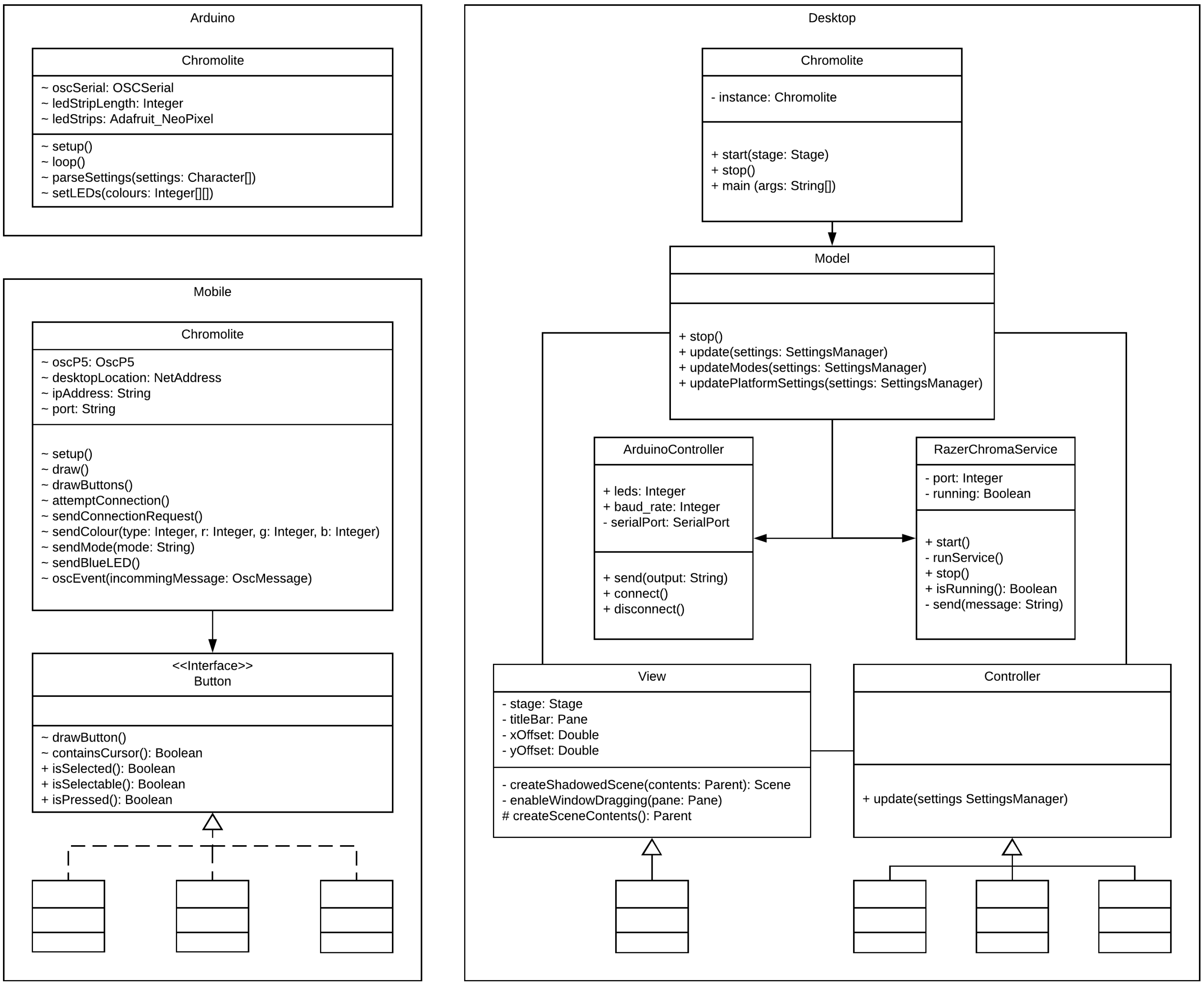
Data Structures

Chromolite's software structure is comprised of three major sections outlined in the architectural diagram above (Arduino micro-controller, Android application, and desktop application). The Arduino program is written in Arduino's C-based language. Because of this language not being object-oriented, its functionality and major data structures are all based within one file and simply defined as a series of variables and methods. The major components of the Arduino program as outlined in the UML class diagram below are the data structures from the OSCSerial and Adafruit NeoPixel libraries that provide the required input and output support respectively. The Arduino's responsibility within the system is simply to operate as a server to translate and distribute input commands to the connected IoT devices. The methods within this program initialise the input and output, then loop until the program is terminated. The loop updates the RGB LEDs based upon the settings information that it parses from the serial or Bluetooth input.

The mobile application, based in Java, is the latest of the three major components to be developed, and as such does not exhibit clear traits of any specific design pattern. The Chromolite class executes the entire program from start to finish and performs all the necessary functions for the software to run. The main variables within this major data structure are used for supporting wireless communication over Wi-Fi and Bluetooth, and many of this class' methods are also designed to support this functionality such as initiating a communication channel and sending data over this channel. The structure of this program exhibits some traits of the MVC (Model-View-Controller) but the view and controller components of the pattern are combined into one and represented by the button classes. The user interacts with the buttons with the view and controller functionality combined into a small set of drawing and interaction methods. The button interface provides a blueprint for each button within the interface, with each button having different implementations of the blueprint methods depending on the desired functionality within program such as changing colour or changing the colour pattern.

The desktop application is the most complex data structure within the whole system and is based upon an MVC design pattern. The Chromolite class is responsible for initialising and terminating the program when necessary. This initialises the MVC and passes off the execution of the entire program to the appropriate model, view and controller classes and data structures. The MVC handles most of the user input, serial output and execution of key functionality. Firstly, the model processes input from the controller and passes off processing to specific service handlers for each type of connected IoT device. These handlers initialise connections to the connected devices through serial ports and send messages in the appropriate format to them. The connection here is one-way as the desktop program simply sends information to the Arduino program in order to be executed. The view component and associated GUI are updated as appropriate with the changes that occur to the model by querying the global SettingsManager data structure. The view component is based upon the JavaFX graphics library and consists of a number of settings panes and buttons that are visible to the user and which the user can interact with. The controller observes the view and has a number of different controller types that extend the functionality of the controller class. Each of these extensions contains action listeners that observe changes to the view and respond by updating the SettingsManager object using the update method that triggers the model to perform the appropriate functional logic based upon the changes.

While none of these three programs interact with each other on the same device, they do connect in a number of different ways as outlined in the component architecture on the previous page. The Arduino program and mobile application communicate across a Bluetooth communication channel between the connectionRequest function (mobile) and parseSettings function (Arduino) that gets information from the connected Bluetooth module. The desktop application communicates with the Arduino through the send(message) functions within the appropriate communication service (desktop) to the parseSettings function (Arduino) via a direct serial communication through the specified serial posts on each device. By providing these three different major data structures, Chromolite is able to achieve its goal of unifying customisation and control of multiple categories of RGB LED lighting.


Arduino
Chromolite
~ oscSerial: OSCSerial
~ ledStripLength: Integer
~ ledStrips: Adafruit_NeoPixel
~ setup()
~ loop()
~ parseSettings(settings: Character[])
~ setLEDs(colours: Integer[][])
Mobile
Chromolite
~ oscP5: OscP5
~ desktopLocation: NetAddress
~ ipAddress: String
~ port: String
~ setup()
~ draw()
~ drawButtons()
~ attemptConnection()
~ sendConnectionRequest()
~ sendColour(type: Integer, r: Integer, g: Integer, b: Integer)
~ sendMode(mode: String)
~ sendBlueLED()
~ oscEvent(incommingMessage: OscMessage)
<<Interface>>
Button
~ drawButton()
~ containsCursor(): Boolean
+ isSelected(): Boolean
+ isSelectable(): Boolean
+ isPressed(): Boolean
Desktop
Chromolite
- instance: Chromolite
+ start(stage: Stage)
+ stop()
+ main (args: String[])
Model
+ stop()
+ update(settings: SettingsManager)
+ updateModes(settings: SettingsManager)
+ updatePlatformSettings(settings: SettingsManager)
ArduinoController
+ leds: Integer
+ baud_rate: Integer
- serialPort: SerialPort
+ send(output: String)
+ connect()
+ disconnect()
RazerChromaService
- port: Integer
- running: Boolean
+ start()
- runService()
+ stop()
+ isRunning(): Boolean
- send(message: String)
View
- stage: Stage
- titleBar: Pane
- xOffset: Double
- yOffset: Double
- createShadowedScene(contents: Parent): Scene
- enableWindowDragging(pane: Pane)
# createSceneContents(): Parent
Controller
+ update(settings SettingsManager)