Bright Light Systems
Code Design Document v. 1.0

by Michael Gulenko & Dallas Winger

INTRODUCTION:
Bright Light Systems is an app for Android OS and iOS that allows users to take full control over the Philips Hue light bulbs by communicating user's requests to a Hue Bridge which on its end will adjust the lights according to request.
The main set of features consist from following:

- Color/Brightness Change
- Color/Brightness Transitions Over Time
- Theme Organizer
- Scheduling (Alarm or Timer based)
- Location Based Actions (Auto On/Off)
- Message Notification (Text, Email, Facebook, etc)
- Music Mode
- Homescreen Widget Control
- Smart Suggestions

This document will describe low level system design specifications of specific features / components: including part of the UI and a gateway component helper class specification.

Other Useful Resources:
The following reading is STRONGLY suggested to fully understand the concepts and scope of this project.

Bright Light Systems Requirements Document v. 1.0

Bright Light Systems System Design Document v. 1.0

Hue SDK:
http://www.developers.meethue.com/documentation/java-multi-platform-and-android-sdk

SDK Code and Examples:
https://github.com/PhilipsHue/PhilipsHueSDK-Java-MultiPlatform-Android

Vocabulary:

To maintain understandability, some terms used to help convey the concepts involved will be defined below:

Hue Bridge - device that is directly connected to user's router, which manages communication to their Hue light bulbs.

Hue Bulb - Smart light that is controlled by a Hue bridge.

Bright Light Trait - defines a Hue light bulb configuration: including color, density(saturation), brightness.

Bright Light Trigger - defines the specific conditions that need to be met in order to cause the **Event**.

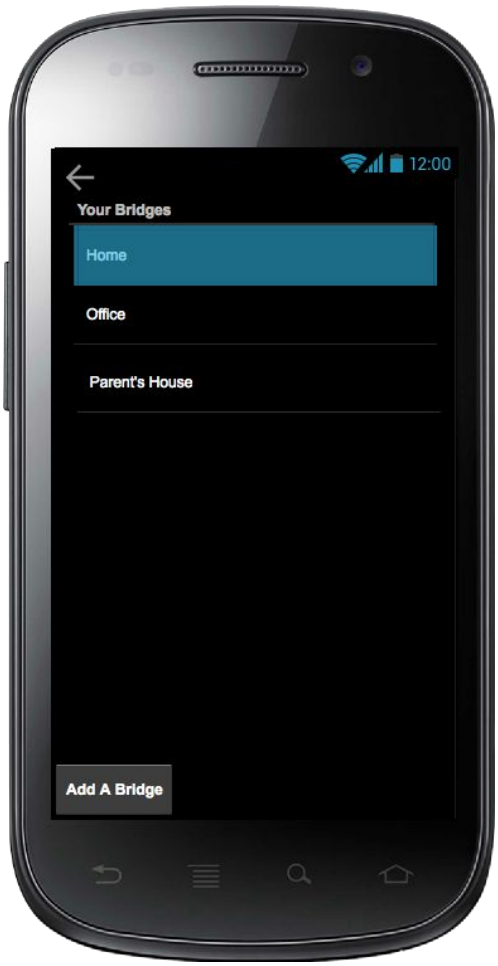
Bright Light Event - defines a change of a single light or group of lights; caused either directly by the user or by a user-defined **Trigger**

Bright Light Groups - defines a group of Hue light bulbs. This will be used to apply **Event**(s) to multiple Hue light bulbs.

UI Mockup

Below is a rough mockup for the Android UI. Note, though the images are of Android components, as if it were already coded, everything is used as a sort of placeholder. The Figures are simply for reference and should not taken literally.

Figures and their captions will be in red and also underlined, *italicized,* and **bolded**. Example: ***Figure 99 Adding a bridge to the app***

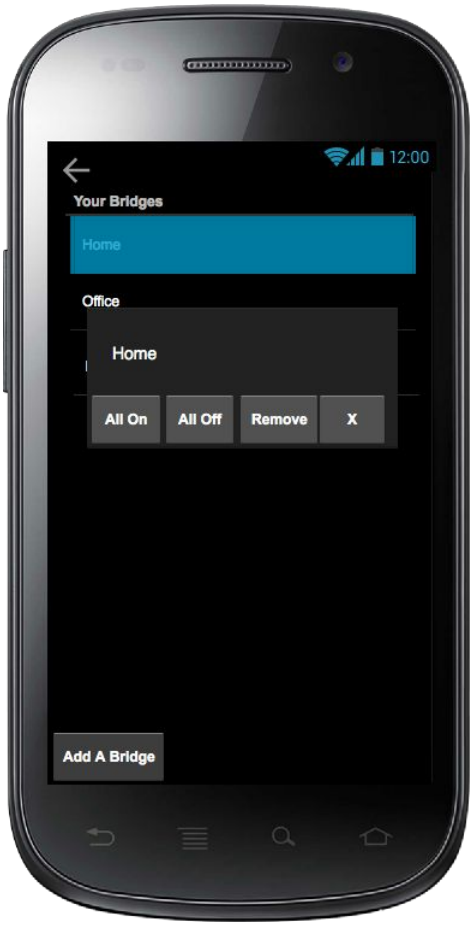


Bridge Selection Screen (Figure 1)
After selecting the "Bridges" button from the *Home Screen*, the user should be met with a list of all the bridges they have registered with the app. Bridges should be be listed top to bottom: most recent to least recently used. The top bridge should be the one associated with the connected access point.
Backpress - A back arrow should be in the top right of the screen allowing the user to return to the last screen, in this case the *Home Screen*.
Add a Bridge - A button allowing the user to register a bridge should be on the bottom of the screen. The button will bring up a wizard style setup for registering the bridge with the app and registering the app/device with the bridge.
Short Press on Bridge - Single pressing a list item should move the user to the *Bulb Control Screen* (Figure 3) - a list of bulbs and groups under control of the selected bridge.

Figure 1: Bridge Selection Screen "Home" list item short pressed.

Long Press on Bridge - Long pressing a list item will prompt the user with an alert box allowing them to turn all bulbs on, all bulbs off, remove the bridge, or cancel to bring them back to this screen. See Figure 2.

Bridge Selection Screen on Long Press (Figure 2)



Alert Dialog Title - Title of the alert dialog should display the name of the bridge long-pressed. More information may be displayed here for quick reference by the user in the future.

All On Button - All On button along the bottom of the alert dialog to allow user to turn on all bulbs under control of the selected bridge. This will change the state to on for all bulbs, meaning all bulbs will simply display the last trait applied to it (the current trait).

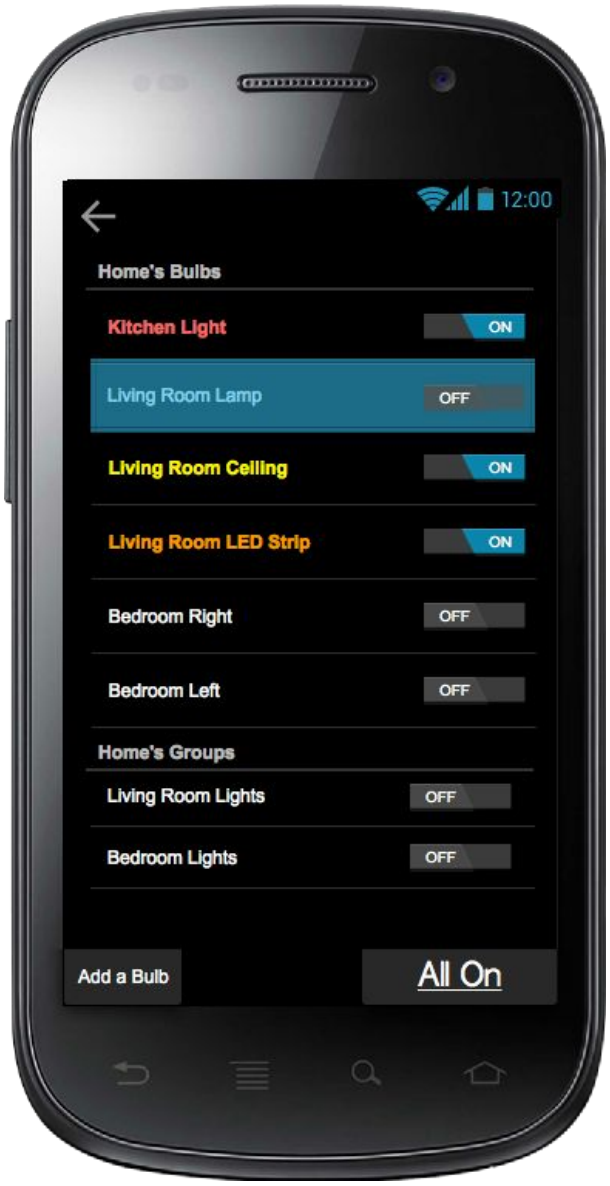
All Off Button - All Off button next to the “all off” button will allow the user to shut off all bulbs under control of selected bridge. This will simply turn all bulb states to off.

Remove Button - The remove button should allow the user to remove the bridge from the app and the app/device from the bridge. This should be confirmed with another dialog before doing so.

X Button - Alert dialog should contain an X or cancel button. Simply a way for the user to exit the alert without using the hard/soft back button on the bottom of their Android device/screen.

Figure 2: Bridge Selection Screen on Long Press Alert Dialog

Bulb Control Screen (Figure 3)



After selecting the bridge the *Bridge Selection Screen* (Figure 1), the user should be met with a list of all the bulbs, by name, controlled by the selected bridge. Bulbs should be listed top to bottom: most recently changed (trait) to least recently changed (trait).

Each bulb should have an on / off switch reflecting its current state. This toggle should change the state appropriately. Once on, the bulb name should be set to bold and reflect the color of the light based off its current trait state.

Single press on Bulb- Short pressing a bulb should bring up the *Edit Bulb Screen* (Figure 4).

Groups- Under the list of bulbs, a list of all bulb groups for this bridge should be found, with on / off toggle on each. On / off toggle will turn on all bulbs in their current trait state. Alert dialog on toggle to choose theme or leave as current traits.

Add a Bulb Button- A button to add a bulb to the bridge. Should bring up wizard to connect bulb to bridge.

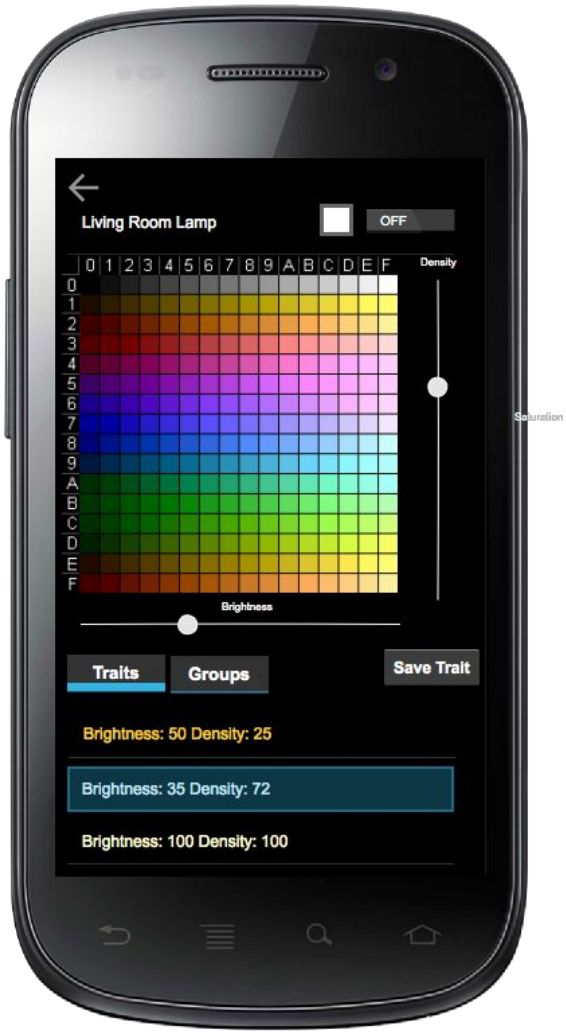
Figure 3: Bulb Control Screen "Living Room Lamp" list item short pressed.

All On- An all on button should be present allowing all bulbs to be changed to on with their current trait state applied.

(Not pictured above) **All Off** - An all off button should be present allowing all bulbs to be changed to off.

Backpress- A back arrow should be in the top right of the screen allowing the user to return to the last screen, in this case the *Bridge Selection Screen* (Figure 1).

Edit Bulb Screen (Figure 4)



After selecting a bulb from the *Bulb Control Screen* (Figure 3) list, the user is brought to this page with the following components:

Page Title - The page title should reflect the name of selected bulb. Clicking on the name should bring up an alert dialog to edit the bulb name.

Color Preview Box - A color preview box showing the current selected color for the selected bulb should be at the top of the page next to the toggle.

On/Off Toggle - The top of the page should have an on / off toggle reflecting its current state. This toggle should change the state appropriately.

Density slider - A slider reflecting the current density should be placed vertically next to the color chooser. This slider controls the density of the bulb. Both sliders should be labeled.

Brightness slider - A slider reflecting current brightness should be placed horizontally below the color chooser. This slider controls the brightness of the bulb.

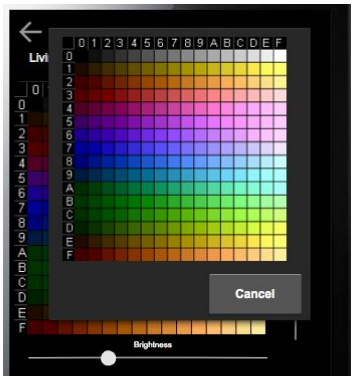
Figure 4 Edit Bulb Screen Selected trait.

Color chooser - A color chooser should occupy the majority of the screen, to be an adequate size for easy color selection. The color picked on the color chooser

should set the current color of the bulb. Color chooser in Figure 4 simply a placeholder.

Trait tab - A selectable tab listing all of the user's saved traits. The text color showing the color of the trait. If a saved trait is currently applied, it will be lightly highlighted, or marked in some way, as shown in Figure 4.

Trait Long Press - An alert dialog should appear upon a trait long press allowing the user to edit the brightness, density, or delete the trait. Upon pressing “Edit Color” the user will be brought to another alert dialog with a color chooser. Upon selecting a color the user will return to the main *Edit Bulb* screen. When the user selects delete, they will be prompted with a confirmation alert dialog asking to delete or cancel. Cancelling the delete confirmation alert dialog should return the user to the previous alert dialog (figure 5).



The X button should allow the user to leave the dialog, saving the current configuration to the trait that brought the user to this dialog.

Above Left: ***Figure 5 Trait Long Press Alert Dialog***
Above Middle: ***Figure 6 Edit Color Alert Dialog***
Below: ***Figure 7 Groups Tab Edit Bulb Screen***

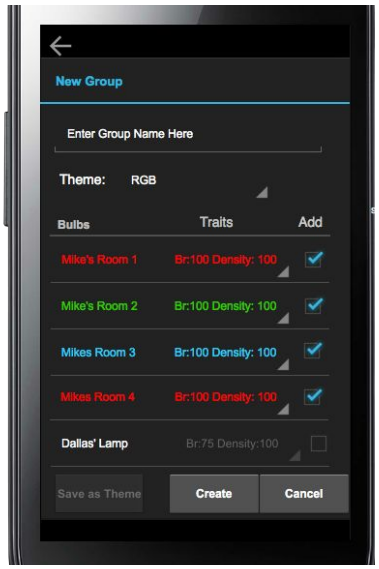
Groups tab - A selectable tab listing all of the groups for the bridge.

When the Groups tab is selected as shown in Figure 7, the user is presented with two lists: “Groups this bulb is in”, and a list of the remaining groups on the bridge controlling this bulb**.** Groups this bulb is in will list all of the groups the bulb is a part of along with a - button to

remove it from that group. The remaining groups will have a + button to quickly add it to that bulb group.

Each list item has a toggle to turn on/off all bulbs in the corresponding group. The “Save Trait” button as seen in Figure 5, is replaced with a “Create Group” button while this tab is selected. The “Create Group” button will open an alert dialog style wizard to create the new group as shown in Figure 8.

Below: Figure 8 New Group Dialog



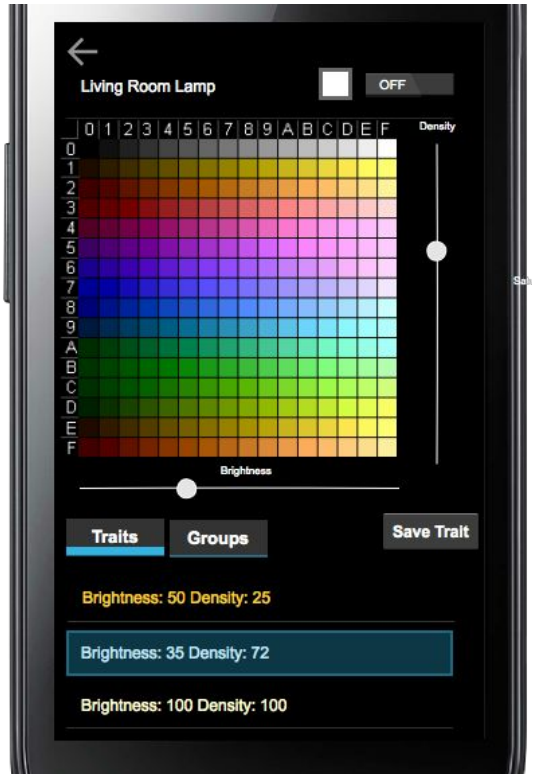
The Create Group alert dialog allows the user to easily create a new bulb group consisting of bulbs from the current bridge. The group requires a name, entered in a text field, and bulbs. Optionally, the user can select an already created Theme from the dropdown as seen in Figure 8. Each bulb has a dropdown of previously created traits. The color of the text indicates the color of the trait on both the trait dropdown, and the bulb name itself. The trait dropdown is only selectable if the bulb is added to the group via the “Add” checkbox. If a theme is selected it will automatically change the Trait spinners to the selected Traits per bulb (that make up the Theme). In the event that a previously created theme is selected, with less bulbs than there are added to the group, the Traits will repeat (in the same order for the remaining bulbs). If no Theme is selected from the spinner, a Save as Theme button will become selectable allowing the user to save the Bulbs-to-Trait configuration for later, based on the selected Trait on each bulb added to the group. The user will be prompted by alert dialog box to name the theme before being returned to the *Edit Bulb Screen*. Cancel will return the user to the *Edit Bulb Screen* also. The Create button will save the group configuration and close the dialog.

Long pressing a group list item in the Groups tab will prompt the user with a dialog allowing them to apply the set Group’s set theme to all bulbs, otherwise the on/off toggle will simply change the bulb state to ‘on’ leaving the bulbs’

color/density/brightness as it was last. This is the same behavior as the *Bulb Control Screen*.

Left: Figure 9 Group name long press

Below: Figure 4 again for reference.

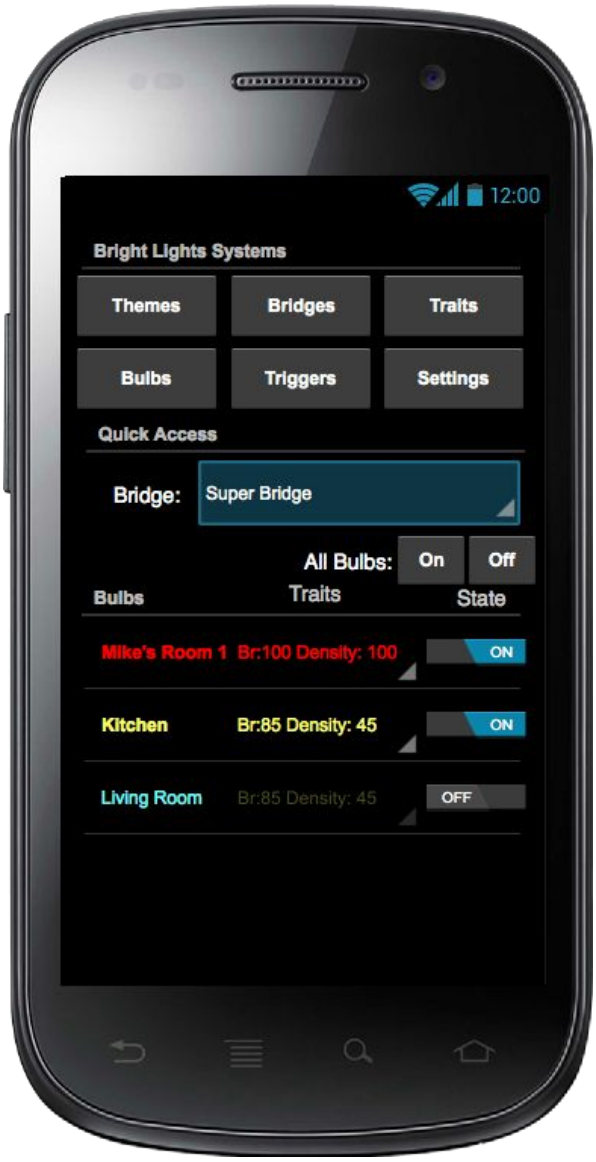


Save trait button - A “save trait” button is shown to allow the user to save the current Brightness, Density and Color of the bulb as a Trait for later use.

Trait Short Press - Selecting a Trait from the list under the Traits tab will apply it to the bulb.

Backpress - A back arrow should be in the top right of the screen allowing the user to return to the last screen, in this case the *Bulb Control Screen* (Figure 3).

Home Screen (Figure 10)



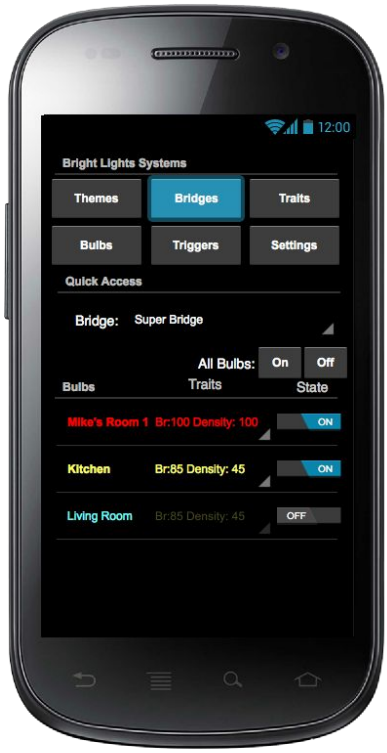
The home screen will consist of a top menu and a lower quick access portion.(See Figure 10) The top menu will have buttons for the user to get to Themes, Bridges, Traits, Bulbs, Triggers, and Settings. The Bridges button should bring the user to the Bridge Selection Screen. All others (For the scope of this document) have not been defined, but are placeholders for the remaining necessary functionality that the app requires. The quick access controls should allow the user to select a bridge from the dropdown, and populate the remainder of the screen with a list of bulbs from that bridge. Each bulb list item should follow the same style as the *bulb control screen*: 'On' bulbs should be bolded, the text color should reflect the color of the current Trait (selected from the dropdown on each bulb), and the on/off toggle should reflect the bulbs state. Two buttons, on/off, should allow the user to turn all bulbs for the selected bridge to the On state.

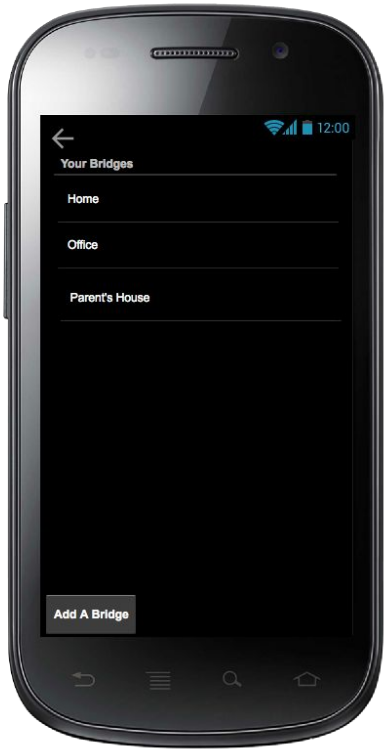
Figure 10 Bright Lights Home Screen

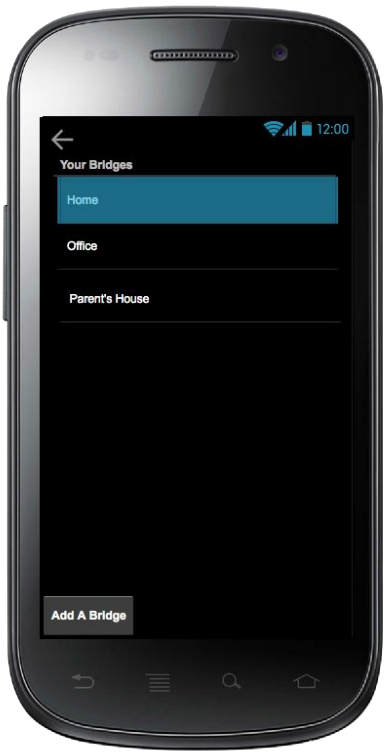
Demo Use Case:

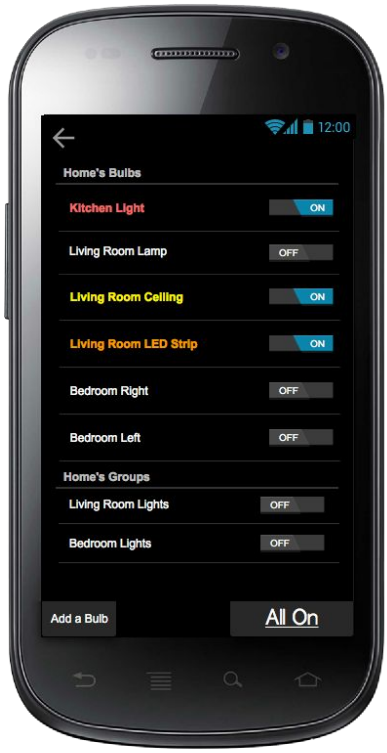
Below is a series of images to outline the UI flow and user experience while using the bridges portion of the app. The images should be read left to right, top to bottom to follow the correct order of events. The example shown consists of the following actions by the user, all of which are marked on each image in some way:

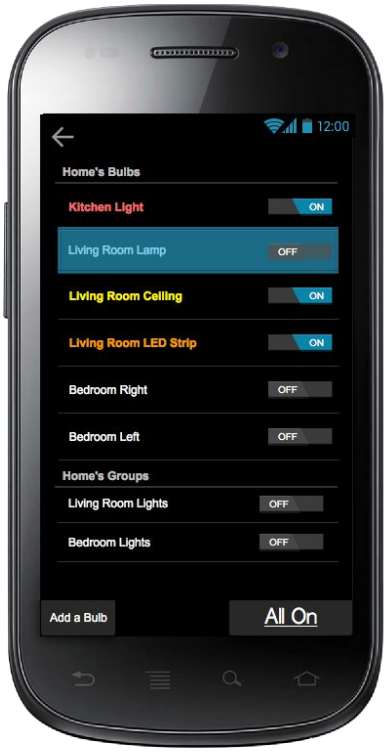
1. Selecting Bridges from the main menu
2. Selecting the bridge “Home”
3. Selecting the bulb “Living Room Lamp”
4. Selecting the top trait in the Traits list on the Edit Bulb screen
5. Long pressing the trait to bring up edit alert dialog
6. Selecting the Edit color button on the alert dialog
7. Selecting a color with the color chooser dialog that follows
8. Pressing the ‘x’ to save and exit out of the edit alert dialog
9. Following the <- (back arrow) back to the Bulb Control screen
10. Turning the Living Room Lamp light on via the on/off toggle
11. Following the <- (back arrow) back to the bridges page, and then again to the home page


12:00
Bright Lights Systems
Themes
Bridges
Traits
Bulbs
Triggers
Settings
Quick Access
Bridge: Super Bridge
All Bulbs: On Off
Bulbs
Traits
State
Mike's Room 1 Br:100 Density: 100 ON
Kitchen Br:85 Density: 45 ON
Living Room Br:85 Density: 45 OFF


12:00
Your Bridges
Home
Office
Parent's House
Add A Bridge

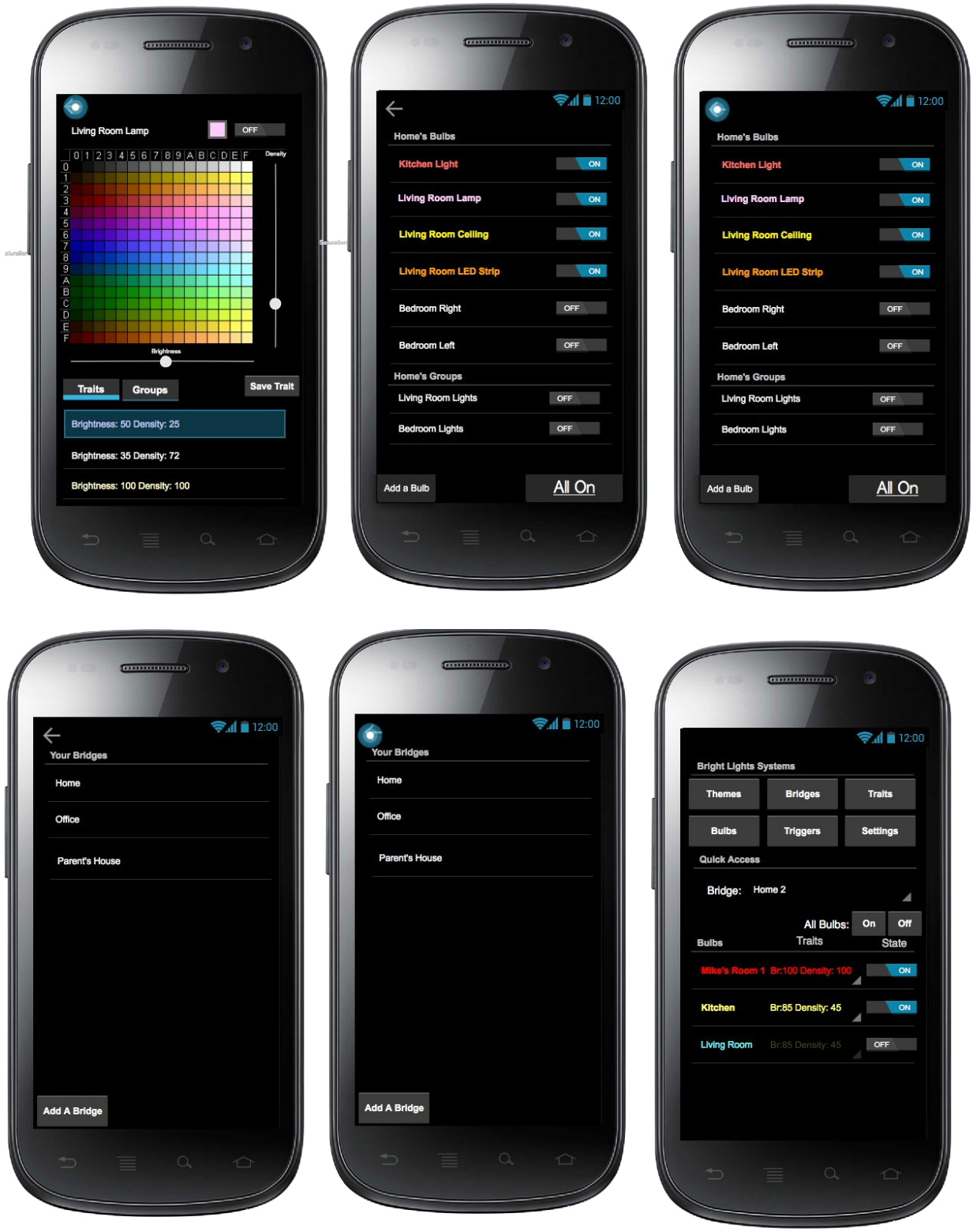

12:00
Your Bridges
Home
Office
Parent's House
Add A Bridge


12:00
Home's Bulbs
Kitchen Light ON
Living Room Lamp OFF
Living Room Ceiling ON
Living Room LED Strip ON
Bedroom Right OFF
Bedroom Left OFF
Home's Groups
Living Room Lights OFF
Bedroom Lights OFF
Add a Bulb
All On


12:00
Home's Bulbs
Kitchen Light ON
Living Room Lamp OFF
Living Room Ceiling ON
Living Room LED Strip ON
Bedroom Right OFF
Bedroom Left OFF
Home's Groups
Living Room Lights OFF
Bedroom Lights OFF
Add a Bulb
All On


Living Room Lamp
OFF
Density
Brightness
Traits
Groups
Save Trait
Brightness: 50 Density: 25
Brightness: 35 Density: 72
Brightness: 100 Density: 100


Living Room Lamp
OFF
Density
Brightness
Traits
Groups
Save Trait
Brightness: 50 Density: 25
Brightness: 35 Density: 72
Brightness: 100 Density: 100
Brightness: 50 Density: 25
Edit Color
Delete
x
Cancel


Living Room Lamp
OFF
Density
Brightness
Traits
Groups
Save Trait
Brightness: 50 Density: 25
Brightness: 35 Density: 72
Brightness: 100 Density: 100
12:00
Home's Bulbs
Kitchen Light
Living Room Lamp
Living Room Ceiling
Living Room LED Strip
Bedroom Right
Bedroom Left
ON
OFF
Home's Groups
Living Room Lights
Bedroom Lights
Add a Bulb
All On
Your Bridges
Home
Office
Parent's House
Add A Bridge
Bright Lights Systems
Themes
Bridges
Traits
Bulbs
Triggers
Settings
Quick Access
Bridge:
Home 2
All Bulbs:
On
Off
Bulbs
Traits
State
Mike's Room 1
Br:100 Density: 100
Kitchen
Br:85 Density: 45
Living Room
Br:85 Density: 45

Gateway

Because Philips Hue has handed the community a Java Multi-Platform and Android SDK, the communication with the bridge and control of the lights is fairly straight forward. To abstract from the details of the Philips API and to allow for versatility on our end in future development when releasing new Triggers or features, a helper class will be used. Returning null on error will be replaced with appropriate exceptions and the proper handling once development has started and common issue causes are identified.

Constructor:

```
public class gatewayHelper(phHueSDK, _username, _pass);
```

Representation:

```
private PHHueSDK _SDK;
```

_username will store the username to be used in Hue SDK calls on this instance of gatewayHelper

```
private String _username;
```

_pass will store the password to be used in Hue SDK calls on this instance of gatewayHelper. **Note: there should be NO getter method. Only a setter.** (obvious security reasons)

```
private String _pass;
```

Methods:

```
public List<PHAccessPoint> getAccessPoints();
```

The getAccessPoints() method will be tasked with returning a list of PHAccessPoint objects to the caller. A single PHAccessPoint object will be used to retrieve all bridges belonging to that access point, this will be done with the getBridges() method. If no Access Points are found it should return a list containing a only a single null.

```
public List<PHBridge> getBridges(PHAccessPoint ap);
```

getBridges() will be responsible for returning a list of PHBridge objects to the caller, the only parameter being “ap”, a PHAccessPoint reference. This access point will be used to find all bridges on it. If no bridges are found it should return a list containing only a single null.

```
public Map<String, PHBridge> getBridgeNames(List<PHBridge> bridges);
```

The getBridgeNames will iterate over a List of PHBridge objects, and retrieve the name of each bridge. The method will return a Map of Strings to PHBridge objects, or in the

case of an error a list containing a single null. The strings will be the names of each bridge paired to the corresponding PHBridge object. This is mainly done for convenience of displaying on the android front end.

```
public List<PHLight> getBulbs(PHBridge bridge);
```

getBulbs will accept a PHBridge and return a list of PHLight objects. The list will represent all bulbs on the given bridge. If no bulbs are found it will return a list with a single null.

```
public Map<String, PHLight> getBulbMap(List<PHLight> bulbs);
```

getBulbNames will take a List of PHLight objects and return a Map of Strings to PHLights. The strings will be the names of each light bulb paired to the corresponding PHLight object. This is mainly done for convenience of displaying on the android front end.

```
public boolean applyBulbTrait(PHLight light, int color, int brightness, int density);
```

applyBulbTrait will take a PHLight object, int for color, brightness and density and will return a boolean depending on weather or not the change was successful. (true - success, false - error)

```
public PHLightState getBulbState(PHLight light);
```

getBulbState will take a PHLight object and return a PHLightState object. This object being returned can be used to gather all state information from the specified bulb. If an error occurs return null.

```
public Map<PHLightState, PHLight> getAllBulbStates(PHBridge bridge);
```

The getAllBulbStates method will take a PHBridge object and return a Map of PHLightState objects paired to their PHLight object.

```
public boolean connectToBridge(PHBridge bridge);
```

The connectToBridge method will be responsible for establishing the connection to the bridge, including for the first time. It should handle setting up heartbeat and the handshake required to authenticate the device/app with the bridge. The method will return true if successful, false if an error occured.

```
public boolean disconnectFromBridge(PHBridge bridge);
```

The disconnectFromBridge should be used when “disconnecting” from a bridge; this method will handle cleaning up after being connected to a bridge. This entails resetting the heartbeat (caching service) on the bridge, etc. The method will return true if successful or false if an error occured.

```
public void setUsername(String uname);
```
The setUsername method will take a String parameter “uname”, this will be used to change the _username representation of this class.

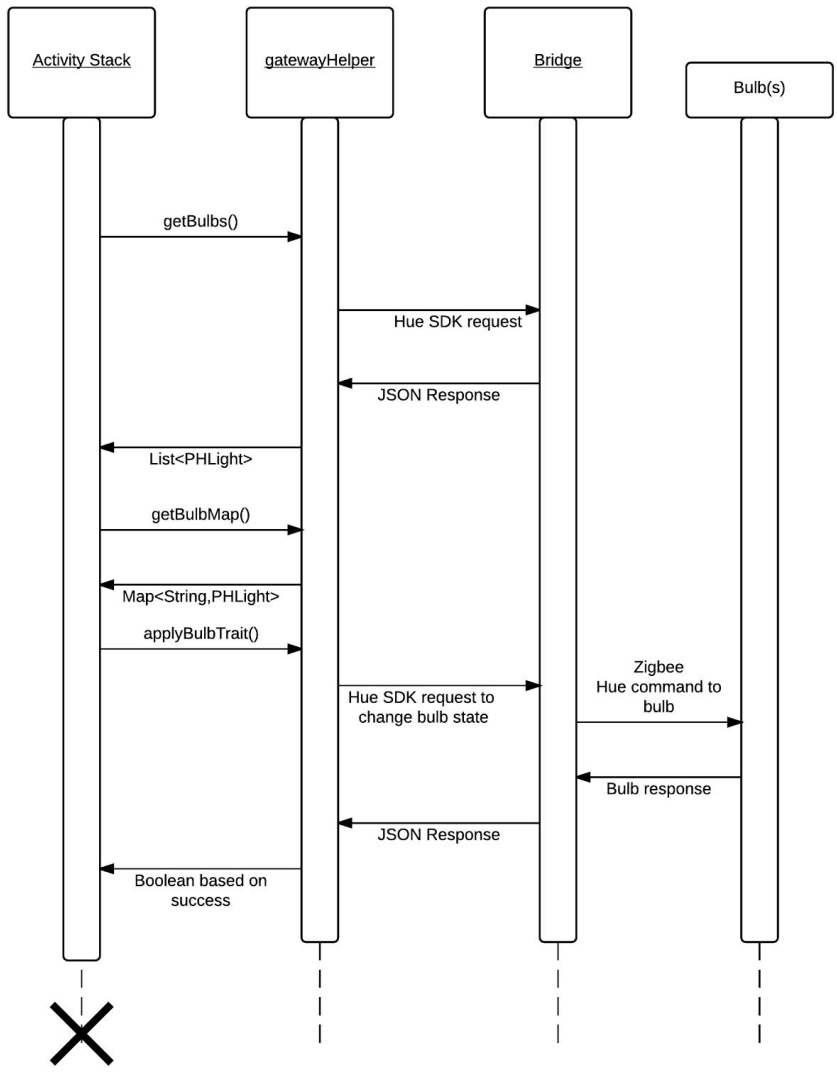
```
public String getUsername();
```
The getUsername method will return the _username representation to the caller as a String, will return null if not already set.

```
public void setPass(String pass);
```
The setPass method will be used to change the _password representation of this class. It takes a String “pass” parameter which will be used to modify the representation.

```
public PHHueSDK getSDK();
```
This method, getSDK, will return a reference to the _SDK object on this class (PHHueSDK). This method should not return null for any reason, as _SDK is a required parameter on the constructor of this class, it will exist.

```
public void setSDK(PHHueSDK sdk);
```
The setSDK method will take a PHHueSDK “sdk” parameter that will be used to change the _SDK field on this class. The method does not return anything.

Below is an example of how the gatewayHelper class will be used in the app’s communication with the Philips Hue system. Mainly, abstracting from Philips API.


Activity Stack
gatewayHelper
Bridge
Bulb(s)
getBulbs()
Hue SDK request
JSON Response
List<PHLight>
getBulbMap()
Map<String,PHLight>
applyBulbTrait()
Zigbee
Hue command to
bulb
Hue SDK request to
change bulb state
Bulb response
JSON Response
Boolean based on
success