SSRGBB Scalable Smart RGB Bulb Protocol

How the light bulbs work.

These lights bulbs have two fields and various colors stored internally. They have a **Status Field** and a **Mode Field**. The Status Field will hold a Boolean indicating whether it is **ON** or **Off**. The Mode field can hold a range of values, but for this project the light bulb you are implementing will only have two modes supported **Static** and **Gradient**. These light bulbs also have to capacity to handle a **Device Information Request** which essentially allow the client to know the light bulb name, light bulb model number and light bulb version number. It will also include information on the current **Status**, **Mode**, **Current Colors** of the light bulb. When the server is first initialized it will be in **Status OFF, Mode Static,** and the **Color White at full luminosity.**

Note: In the actual packets there is a status field, but it can **also** be used to indicate whether that message is a **Device Information Request** (See the **Status** paragraph; page 2). The status field within the light bulb itself is still going to be a Boolean. You are responsible for translating the integer being sent to a Boolean when it is not a **Device Information Request**.

Message Type (Unsigned Char 1 byte)

- **Request (0) (Client)**
- **Response (1) (Server)**

Respond with a UNRECOGNIZED_MESSAGE_FORMAT if any other values are sent by the client.

Error Number (Unsigned Char 1 bytes) – Both the client and server must agree and know what the error codes are and what they represent. For this specific light bulb with model number 1, version number 1, these are the error numbers that you must support on both the client AND server, meaning the server will generate error codes in the response message, and the client will need to understand what they mean:

- 0 - NO_ERROR
- 1 - LIGHT_BULB_FAILURE
    - On the server end you must add support to simulate light bulb failure. This will be in the form of a true or false Boolean as the third argument the server program receives.
- 2 - COLOR_NOT_SUPPORTED
    - This error is thrown when ALL the **RGB** values are below 40 **OR** the **L** value is below 40.
- 3 - STATUS_UNSUPPORTED:
    - Throw when not 0, 1, or 2
- 4 - MODE_UNSUPPORTED:
    - Throw when not 0 or 1
- 5 – INCORRECT_NUMBER_OF_COLORS:
    - Static requires 1 color specified; Gradient requires 2 color specified. This will rely on the **Message Type** field and **Number of Colors** field. If the client request claims it sent a

        - certain number of colors, but the byte array does not contain that many numbers, the server will also respond with this error.
- 6 - UNRECOGNIZED_MESSAGE_FORMAT:
    - Throw this if the byte array does not match up with what is expected, and no other error code could be attributed.
    - For example, if a byte array of size is 4 bytes, even though that is impossible for any proper message.
    - **See the next paragraph**

The client will always send 0 for the error code. Future lightbulbs might have bi-directional error code support, but for this project, the errors only occur on the server. For this project, UNRECOGNIZED_MESSAGE_FORMAT will also be used if any other values are sent by the client.

Message Identifier (Unsigned Short 2 bytes)

The request from the client will follow incremental pattern but does not have to be enforced. We will not be using a random IDs on the client end to ensure that an ID collision is reduced substantially. The client will have an ID counter starting at 0 and for each new request it makes (assuming the previous one was successful) it will use the next number on the counter. You are responsible for making sure that the number stays within unsigned short ranges [0, 2^16 - 1].

Numbers of Colors (Unsigned Char 1 bytes)

A number indicating how many RGBL values that were sent. This number needs to be checked depending on the mode selected.

Respond with INCORRECT_NUMBER_OF_COLORS if bad values are sent by the client or if the number of colors sent do not match up with the number of colors field.

Pad Byte/Company Specific Info (Unsigned Char 1 byte)

This byte can be used by future products, but for now this will simply be a pad byte, or an ignored unsigned char. Future lightbulb will use this for more complex light modes that need speed information. For example, a flashing mode, rainbow mode, etc.…

Status (Unsigned Char 1 bytes)

In the actual packets there is a status field, but it can **also** be used to indicate whether that message is a **Device Information Request**. The status field within the light bulb is still going to be a Boolean. You are responsible for translating the integer being sent to a Boolean when it is not a **Device Information Request**.

- **Device Information Request (0)** - This will follow a separate procedure, which will be specified in the Device Information Request section.

- **Off (1)** - No color is needed and if a color is sent it is ignored. You will ignore Mode fields and Color Specified fields at this point. This turns the light bulb off. If it is already off leave it off.
- **On (2)** - This will turn on the light. If it is already on leave it on.

Respond with STATUS_UNSUPPORTED if bad values are sent by the client.

Mode (Unsigned Char 1 bytes) (Ignore this is the light mode is off)

- **Static (0)** - Specify 1 color in the format of R G B L in the **Colors Specified** section.
- **Gradient (1)** - Specify 2 colors in the format R G B L in the **Colors Specified** section.

Respond with MODE_UNSUPPORTED if bad values are sent by the client.

Colors Specified (Variable Size. Multiple of 4 bytes) $(\boldsymbol{4 * Number\ of\ Colors})$

One Unsigned Char for R (Red), One Unsigned Char for G (Green), One Unsigned Char for B (Blue), and One Unsigned Char for L (Luminosity/Brightness/Opacity).

Standard Message Format for Standard a Normal Request

CLIENT Request Message (N = Numbers of Colors)

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [0]	Error Number (Unsigned Char 1 byte) [0]	Message ID (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	
Numbers of Colors (Unsigned Char 1 byte) [0, 1, 2]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [1, 2]	Mode (Unsigned Char 1 byte) [0, 1]
Red of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Green of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Blue of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color 1 (Unsigned Char 1 byte) [0, …, 255]
…	…	…	…
Red of Color N (Unsigned Char 1 byte) [0, …, 255]	Green of Color N (Unsigned Char 1 byte) [0, …, 255]	Blue of Color N (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color N (Unsigned Char 1 byte) [0, …, 255]

SERVER Response Message (If an error occurred)

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [1]	Error Number (Unsigned Char 1 byte) [1, 2, 3, 4, 5, 6]	Message ID (Same as client) (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	
Numbers of Colors (Unsigned Char 1 byte) [0]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [0]	Mode (Unsigned Char 1 byte) [0]

SERVER Response Message (If no error occurred) (N = Numbers of Colors)

Redundantly echo information of the lightbulb back to the client.

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [1]	Error Number (Unsigned Char 1 byte) [0]	Message ID (Same as client) (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	
Numbers of Colors (Unsigned Char 1 byte) [0, 1, 2]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [1, 2]	Mode (Unsigned Char 1 byte) [0, 1]
Red of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Green of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Blue of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color 1 (Unsigned Char 1 byte) [0, …, 255]
…	…	…	…
Red of Color N (Unsigned Char 1 byte) [0, …, 255]	Green of Color N (Unsigned Char 1 byte) [0, …, 255]	Blue of Color N (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color N (Unsigned Char 1 byte) [0, …, 255]

Standard Message Format for a Device Information Request

Since we do not always know the exact light bulb we are working with, we need to ask the server what light bulb it is to see if the client can support that lightbulb. This is also used in the case we just genuinely want to know what status, mode and color the device is currently in, while not requesting a change to the device. This will be indicated by the status number 0. If the server sees status number 0, after reading message type, error number and message ID, it will ignore the number of colors and mode fields.

CLIENT Request Message

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [0]	Error Number (Unsigned Char 1 byte) [0]	Message ID (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	
Numbers of Colors (Unsigned Char 1 byte) [0]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [0]	Mode (Unsigned Char 1 byte) [0]

SERVER Response Message

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [1]	Error Number (Unsigned Char 1 byte) [0, 1, 2, 3, 4, 5, 6]	Message ID (Same as client) (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	

Numbers of Colors (Unsigned Char 1 byte) [0, 1, 2]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [1, 2]	Mode (Unsigned Char 1 byte) [0, 1]
Red of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Green of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Blue of Color 1 (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color 1 (Unsigned Char 1 byte) [0, …, 255]
…	…	…	…
Red of Color N (Unsigned Char 1 byte) [0, …, 255]	Green of Color N (Unsigned Char 1 byte) [0, …, 255]	Blue of Color N (Unsigned Char 1 byte) [0, …, 255]	Luminosity of Color N (Unsigned Char 1 byte) [0, …, 255]
Model Number (Unsigned Short 2 bytes) [1] *		Device Version (Unsigned Short 2 bytes) [1] *	
String Length (Unsigned Integer 4 bytes) Example - 40			
String with Device Information (Variable Size determined by String Length field) Example - “Samsung - SmartThings A19 Smart LED Bulb”			

SERVER Response Message (If there is a long-standing error like light bulb failure)

0 1 2 3 4

Message Type (Unsigned Char 1 byte) [1]	Error Number (Unsigned Char 1 byte) [1, 2, 3, 4, 5, 6]	Message ID (Same as client) (Unsigned Short 2 bytes) [0, …, 2^16 - 1]	
Numbers of Colors (Unsigned Char 1 byte) [0]	Pad Byte/Company Specific (1 byte)	Status (Unsigned Char 1 byte) [0]	Mode (Unsigned Char 1 byte) [0]
Model Number (Unsigned Short 2 bytes) [1] *		Device Version (Unsigned Short 2 bytes) [1] *	
String Length (Unsigned Integer 4 bytes) Example - 40			
String with Device Information (Variable Size determined by String Length field) Example - “Samsung - SmartThings A19 Smart LED Bulb”			

* For this project we are going to be assuming the client can only support Model Number 1 and Device Version 1 and that the server is running a light bulb that is of model number 1 and device version 1. That how we “know” what error codes it supports and has.

* The string with Device Information can be anything, but it is meant to be printed to the user to show them what device they are trying to work with.

The Client

The client will perform the following function:

1. Read in the following arguments from the command line:
   a. IP address of the server (127.0.0.1)
   b. Port of the server (e.g. 9999)
2. Ask the user for the following information
   a. Status of request – [0, 1, 2]
   b. (REQUIRED IF ON Request) Mode – [0, 1]
   c. (REQUIRED IF ON Request) Color 1 – Format like this “255,255,255,255”
   d. (REQUIRED IF ON Request) Color 2 – Format like this “255,255,255,255”
   e. …
   f. (REQUIRED IF ON Request) Color N – Format like this “255,255,255,255”
   g. The amount of colors you ask for depends on the mode
3. Construct the packet that corelates with the arguments sent in.
4. Print out an information of that packet.
5. Send the packet.
6. Wait for a response from the server
   a. 1 second timeout, resend the same message again, with the same message ID, a maximum of 3 times. If that limit is reached, kill the program.
   b. If the client receives an error, make sure to also print out an error message.
7. Print the response from the server.
8. Repeats steps 2-8
9.

The Server

The server will perform the following function:

1. Read in 3 arguments from the command line:
   a. IP address of the server (127.0.0.1)
   b. Port of the server (e.g. 9999)
   c. Light Bulb Failure Simulator
      i. True = All responses from the light bulb must have error code 1.
      ii. False = Work normally
2. Setup an internal representation of the light bulb. Current Status, Mode, Colors (Defaults: OFF, Static, White). Also initialize the model number, device number, and device information.
3. Wait for either normal request or Device Information Request.
4. Process the request, give a printout of it and respond to the client.
5. Repeat steps 3-5

Test Cases

Have both the client AND server running for the entirety of the next 5 test cases:

*** Positive Test Cases**

Test Case 1: Input into the client to send a static on request with the color 255,0,0,255.

Test Case 2: Input into the client to send a gradient on request with the colors 255,0,0,255 and 0,255,0,255.

Test Case 3: Input into the client to send a **Device Information Request.**

Test Case 4: Input into the client to send an Off Request.

*** Negative Test Cases**

Test Case 5: Input into the client to send a static on request with the color 10,20,30,255.

**** Stop the server and turn it back on again with the light bulb failure argument set to true before doing test case 6.**

Test Case 6: Input into the client to send a **Device Information Request.**

**** Stop the server completely before doing test case 7**

Test Case 7: Input into the client to send a **Device Information Request.**

Test Output

```
$ python3 iot-client.py 127.0.0.1 9999
# Test Case 1
Status? 2
Mode? 0
Color 1? 255,0,0,255

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 0
Number of colors: 1
Status: 2
Mode: 0
Color 1 RGBL: (255,0,0,255)

Received Response from 127.0.0.1, 9999:
Error Number: 0 (No Error)
```

```
Message Type: 1
Message ID: 0
Number of colors: 1
Status: 2
Mode: 0
Color 1 RGBL: (255,0,0,255)


# Test Case 2
Status? 2
Mode? 1
Color 1? 255,0,0,255
Color 2? 0,255,0,255

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 1
Number of colors: 2
Status: 2
Mode: 1
Color 1 RGBL: (255,0,0,255)
Color 2 RGBL: (0,255,0,255)

Received Response from 127.0.0.1, 9999:
Error Number: 0 (No Error)
Message Type: 1
Message ID: 1
Number of colors: 2
Status: 2
Mode: 1
Color 1 RGBL: (255,0,0,255)
Color 2 RGBL: (0,255,0,255)


# Test Case 3
Status? 0

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 2
Number of colors: 0
Status: 0
Mode: 0

Received Response from 127.0.0.1, 9999:
```

```
Error Number: 0 (No Error)
Message Type: 1
Message ID: 2
Number of colors: 2
Status: 2
Mode: 1
Color 1 RGBL: (255,0,0,255)
Color 2 RGBL: (0,255,0,255)
Device Information:
Model Number: 1
Device Version: 1
Device Information: Samsung - SmartThings A19 Smart LED Bulb


#Test Case 4
Status? 1

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 3
Number of colors: 0
Status: 1
Mode: 0

Received Response from 127.0.0.1, 9999:
Error Number: 0 (No Error)
Message Type: 1
Message ID: 3
Number of colors: 2
Status: 1
Mode: 1
Color 1 RGBL: (255,0,0,255)
Color 2 RGBL: (0,255,0,255)


#Test Case 5
Status? 2
Mode? 0
Color 1? 10,20,30,255

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 4
Number of colors: 1
Status: 2
```

```
Mode: 0
Color 1 RGBL: (10,20,30,255)

Received Response from 127.0.0.1, 9999:
Error Number: 2 (Color Not Supported)
Message ID: 4



# Test Case 6
Status? 0

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 5
Number of colors: 0
Status: 0
Mode: 0

Received Response from 127.0.0.1, 9999:
Error Number: 1 (Light Bulb Failure)
Message ID: 5
Device Information:
Model Number: 1
Device Version: 1
Device Information: Samsung - SmartThings A19 Smart LED Bulb


#Test Case 7
Status? 0

Sending Request to 127.0.0.1, 9999:
Message Type: 0
Message ID: 6
Number of colors: 0
Status: 0
Mode: 0

Request timed out ...
Sending Request to 127.0.0.1, 9999:
Request timed out...
Sending Request to 127.0.0.1, 9999:
Request timed out ... Exiting Program.
```