# Developer Study Guide: An introduction to Bluetooth Mesh Networking

Hands-on Coding Lab - Light HSL Model

| | | |
|---|---|---|
| Release | : | 2.1.0 |
| Document Version | : | 2.1.0 |
| Last updated | : | 29th June 2021 |

# Contents

# Revision History

| Version | Date | Author | Changes |
|---|---|---|---|
| 1.0.0 | 15th June 2018 | Martin Woolley<br>Bluetooth SIG | Initial version. |
| 1.0.2 | 16th August 2018 | Martin Woolley<br>Bluetooth SIG | Fixed description of move set operations and added test for sending a generic move set with delta level = 0 to stop a running move on the target light node(s). |
| 1.0.3 | 21st August 2018 | Martin Woolley<br>Bluetooth SIG | Bug: min message length for generic level status was set to 1 and should have been set to 2. |
| 1.0.4 | 14th December 2018 | Martin Woolley<br>Bluetooth SIG | Minor errata:<br>#11 - Exercising the generic level state would work visibly even if the onoff state was OFF. This was not correct. Imagine a dimmer control which when pressed acts as an on/off switch. Rotating the knob will have no effect if the lights are switched off. That's how the generic onoff server and generic level server should work when incorporated together in a device. The two states are now handled completely independently. Code and documentation adjusted accordingly.<br>#12 - Light node should have subscribed to the group address once for each model. A bug in Zephyr 1.12 allowed a subscription to only one model to be sufficient for all models to have messages published to that address routed to them so there was no user discernible impact of this issue at Zephyr 1.12. Code has been adjusted. |
| 2.0.0 | 16th December 2019 | Martin Woolley<br>Bluetooth SIG | Exercises are now based on Zephyr 1.14 using the west multipurpose tool and a Nordic Thingy developer board. The Light HSL Model is now included in exercises. |
| 2.0.1 | 21st December 2020 | Martin Woolley<br>Bluetooth SIG | Language changes |
| 2.1.0 | 29th June 2021 | Martin Woolley<br>Bluetooth SIG | **Release:**<br>Zephyr code now based on version 2.6.0 of the Zephyr SDK<br><br>**Document**:<br>Code fragments revised to be based on Zephyr 2.6.0 |

# Exercise 4 – Implementing the Light HSL Model

## Introduction

Your next coding exercise will involve creating a partial implementation of the Light HSL Model. HSL stands for Hue, Saturation, Lightness and is a popular scheme for representing colours in lighting. The Bluetooth mesh Light HSL Model is a fairly ellaborate model, with dependencies on a number of other models and therefore, quite a large number of message types which the server model must support for proper compliance with the specification. People wishing to implement lighting colour control in a commercial product which must pass the Bluetooth SIG qualification tests, will need to read the models specification carefully and ensure all mandatory requirements are met. The goal in this coding exercise is to introduce the model only and so we will be implementing support for only one message type, allowing us to change the colour of the LED in our light node by sending messages from the switch.

We'll be implementing part of the Light HSL Client model in the code for our switch node and the Light HSL Server model in the code for the light node. As such, all of the common coding tasks such as that which is required to allow provisioning to take place, has already been taken care of in previous exercises.

## The Light HSL Client Model

The switch node will be enhanced so that it sends a *light HSL set unacknowledged* message when button 4 is pressed. At each press of button 4, we'll select the next colour from a circular list of supported colours. The starter code defines 8 colours. You are free to add to this list if you wish to.

```
#define NUMBER_OF_COLOURS 8

uint16_t hsl[NUMBER_OF_COLOURS][3] = {
        { 0x0000, 0x0000, 0x0000 }, // black
        { 0x0000, 0xFFFF, 0x7FFF }, // red
        { 0x5555, 0xFFFF, 0x7FFF }, // green
        { 0xAAAA, 0xFFFF, 0x7FFF }, // blue
        { 0x2AAA, 0xFFFF, 0x7FFF }, // yellow
        { 0xD555, 0xFFFF, 0x7FFF }, // magenta
        { 0x7FFF, 0xFFFF, 0x7FFF }, // cyan
        { 0x0000, 0x0000, 0xFFFF }  // white
```

Open your switch code and work through the updates in this section.

## Node Composition Update

We need to add the Light HSL Model to the node composition definitions. Start by adding a model publication context definition underneath the one for the generic on off client, called *gen_onoff_cli*.

```
BT_MESH_MODEL_PUB_DEFINE(light_hsl_cli, NULL, 7);
```

Next, update the sig_models array definition so that it includes an entry for the Light HSL Client model, as shown here:

```
static struct bt_mesh_model sig_models[] = {
  BT_MESH_MODEL_CFG_SRV,
  BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_CLI, gen_onoff_cli_op, &gen_onoff_cli,
&onoff[0]),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_LIGHT_HSL_CLI, NULL, &light_hsl_cli, &hsl[0]),
};
```

## Light HSL set messages

Underneath the comment *Light HSL Client - TX message producer functions*, add the following function:

```
int sendLightHslSet(uint16_t message_type)
{
    int err;
    struct bt_mesh_model *model = &sig_models[3];
    if (model->pub->addr == BT_MESH_ADDR_UNASSIGNED) {
        printk("No publish address associated with the light HSL client model - add one with
a configuration app like nRF Mesh\n");
        return -1;
    }
    struct net_buf_simple *msg = model->pub->msg;

    bt_mesh_model_msg_init(msg,       message_type);
    net_buf_simple_add_le16(msg, hsl[current_hsl_inx][2]); // Yes, index value 2 is correct
- ordering of params is Lightness, Hue, Saturation
    net_buf_simple_add_le16(msg, hsl[current_hsl_inx][0]);
    net_buf_simple_add_le16(msg, hsl[current_hsl_inx][1]);
    net_buf_simple_add_u8(msg, hsl_tid);
    hsl_tid++;
    printk("publishing light HSL set message\n");
    err = bt_mesh_model_publish(model);
    if (err) {
        printk("bt_mesh_model_publish err %d\n", err);
    } else {
        current_hsl_inx++;
        if (current_hsl_inx == NUMBER_OF_COLOURS) {
            current_hsl_inx = 0;
        }
    }

    return err;
}
```

## Explanation

The sendLightHslSet function publishes either a *light HSL set* or a *light HSL set unacknowledged* message, depending on the opcode in the message_type parameter. It publishes the message and therefore requires the model to have been configured with a Publish Address. The buffer associated with the model's publication message is initialised with the specified opcode and then populated with the lightness, hue and saturation parameters, in that order. The transaction identifier is then added and incremented.

*bt_mesh_model_publish* is called to publish the message and an index into our circular list of colours is incremented, ready for the next time button 4 is pressed.

## Light HSL set unacknowledged

Directly under the *sendLightHslSet* function, add this function:

```
void lightHslSetUnAck()
{
        if (sendLightHslSet(BT_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK))
        {
                printk("Unable to send light HSL set unack message\n");
        }
}
```

### Explanation

This simple function, calls *sendLightHslSet* with the opcode for the *light HSL set unacknowledged* message type.

### Button 4

Update the button4_work_handler function so that it calls lightHslSetUnack when button 4 is pressed.

```
void button4_work_handler(struct k_work *work)
{
        lightHslSetUnAck();
}
```

## Light HSL Server Model

Open your light code and work through the updates in this section.

### Node Composition Update

Update the sig_models array definition so that it includes an entry for the Light HSL Server model, as shown here:

```
static struct bt_mesh_model sig_models[] = {
  BT_MESH_MODEL_CFG_SRV(&cfg_srv),
  BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_GEN_ONOFF_SRV, generic_onoff_op,&generic_onoff_pub, NULL),
  BT_MESH_MODEL(BT_MESH_MODEL_ID_LIGHT_HSL_SRV, light_hsl_op,&light_hsl_pub, NULL),
};
```

Under the comment *light HSL server message opcodes*, add the following:

```
#define BT_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK BT_MESH_MODEL_OP_2(0x82, 0x77)
```

## Light HSL set unacknowledged message handler

Underneath the comment *Light HSL Server functions*, add the following functions:

```
static void set_hsl_state(struct bt_mesh_model *model, struct bt_mesh_msg_ctx *ctx, struct
net_buf_simple *buf)
{
        uint16_t msg_hsl_lightness = net_buf_simple_pull_le16(buf);
        uint16_t msg_hsl_hue = net_buf_simple_pull_le16(buf);
        uint16_t msg_hsl_saturation = net_buf_simple_pull_le16(buf);

        if (msg_hsl_lightness == hsl_lightness && msg_hsl_hue == hsl_hue &&
msg_hsl_saturation == hsl_saturation) {
                // no state change so nothing to do
                return;
        }

        hsl_lightness = msg_hsl_lightness;
        hsl_hue = msg_hsl_hue;
        hsl_saturation = msg_hsl_saturation;

        printk("set HSL state: lightness=%u hue=%u saturation=%u\n", hsl_lightness, hsl_hue,
hsl_saturation);
    convert_hsl_to_rgb(hsl_hue,hsl_saturation,hsl_lightness);
    if (onoff_state == 1) {
                thingy_led_on(rgb_r, rgb_g, rgb_b);
        }

        /*
         * If a server has a publish address, it is required to publish status on a state
change
         * See Mesh Profile Specification 3.7.6.1.2
         */

        if (model->pub->addr != BT_MESH_ADDR_UNASSIGNED) {
                // if we had implemented light HSL status messages, we'd send one here
                printk("A status message should be sent here - not implemented\n");
        }
}

static void light_hsl_set_unack(struct bt_mesh_model *model,struct bt_mesh_msg_ctx *ctx,
        struct net_buf_simple *buf)
{
        printk("light_hsl_set_unack\n");
        set_hsl_state(model, ctx, buf);
}
```

Under the functions that you just added, include the following code to define the function which should be called by the stack to handle *light HSL set unacknowledged* messages.

```
static const struct bt_mesh_model_op light_hsl_op[] = {
                {BT_MESH_MODEL_OP_LIGHT_HSL_SET_UNACK, 7, light_hsl_set_unack},
                BT_MESH_MODEL_OP_END,
};
```

And under the comment light *HSL server model publication context*, add

```
BT_MESH_MODEL_PUB_DEFINE(light_hsl_pub, NULL, 2 + 6);
```

## Explanation

The changes made here should be familiar to you from the work you did with the *generic on off server* model. We've included the *light HSL server model* in the node composition hierarchy, defined the opcode for the one and only light HSL message type that we are aiming to support and mapped this to a function called *light_hsl_set_unack* which will be called whenever the stack receives a message of that type.

In the function lset_hsl_state, we check for a state change and if one is being requested, we then unpack the lightness, hue and saturation fields from the message and convert them to RGB values (since this is the format the GPIO interface to the LED understands). If the LED's *generic on off state* indicates that it is on, we change the colour of the LED. As a placeholder, we also check for the presence of a *model publish address*. This is where we'd publish a *light HSL status* message if we were implementing that message type in this exercise.

## Checkpoint

Clear persisted data by running *nrfjprog -e* against each device. Using your provisioning and configuration application (e.g. nRF Mesh), reset your provisioning database so we can start again.

Flash your updated code to each of the switch and light nodes.

Provision and configure each node so that the generic on off client, generic on off server, light HSL client and light HSL server models both publish to group address 0xC000 and also subscribe to it. Subscribing the clients to the group address means they will receive status messages pubished by the servers. Bind all models to the same application key.

*These steps are critical. If when testing you find that your light is not responding to messages sent by the switch and you are sure that your code is correct, check that each of the two models has been bound to the same application key on both devices and check that each model both publishes and subscribes to group address 0xC000.*

## Testing

Switch on your light node's LED by pressing button 1 on the switch to send a *generic onoff set unacknowledged* message with a state value of 1.

Test colour control by pressing button 4 repeatedly, to cycle through the supported colours. The light node console should look something like this:

```
[00:11:46.242,218] <dbg> bt_mesh_net.bt_mesh_net_decode: 27 bytes:
637e753c53567c0fdec604f2e7e0302b18ff643738eccd385f3473
[00:11:46.242,218] <dbg> bt_mesh_net.net_find_and_decrypt:
[00:11:46.242,218] <dbg> bt_mesh_net.net_decrypt: NID 0x63 net_idx 0x0000
[00:11:46.242,218] <dbg> bt_mesh_net.net_decrypt: IVI 0 net->iv_index 0x00000000
[00:11:46.242,248] <dbg> bt_mesh_net.net_decrypt: src 0x0002
[00:11:46.242,340] <dbg> bt_mesh_net.bt_mesh_net_decode: Decryption successful. Payload len
23
[00:11:46.242,340] <dbg> bt_mesh_net.bt_mesh_net_decode: src 0x0002 dst 0xc000 ttl 7
[00:11:46.242,370] <dbg> bt_mesh_net.bt_mesh_net_decode: PDU: <log_strdup alloc failed>
[00:11:46.242,462] <dbg> bt_mesh_access.bt_mesh_model_recv: **** bt_mesh_model_recv

[00:11:46.242,492] <dbg> bt_mesh_access.bt_mesh_model_recv: app_idx 0x0000 src 0x0002 dst
0xc000
[00:11:46.242,492] <dbg> bt_mesh_access.bt_mesh_model_recv: len 9: <log_strdup alloc
failed>
[00:11:46.242,523] <dbg> bt_mesh_access.bt_mesh_model_recv: OpCode 0x00008277
[00:11:46.242,523] <dbg> bt_mesh_access.bt_mesh_model_recv: **** found op
light_hsl_set_unack

set HSL state: lightness=32767 hue=0 saturation=65535
```

## Next

You have completed the exercises in this study guide. Congratulations!

Your next steps, should be to read the Mesh Profile specification and the Mesh Models specification. See https://www.bluetooth.com/specifications/mesh-specifications/. You may find the Mesh Models technical overview paper a useful companion to this activity. See https://www.bluetooth.com/bluetooth-resources/bluetooth-mesh-models/