Bluetooth Mesh Developer Study Guide

Bluetooth Mesh - Hands-on Coding Lab - Introduction

Version: 1.0.0

Last updated: 15th June 2018

Contents

Revision History

Version	Date	Author	Changes
1.0.0	15th June 2018	Martin Woolley Bluetooth SIG	Initial version.

Introduction

Having reached this point, you should be feeling comfortable with the relevant concepts of Bluetooth mesh. We're going to move beyond theory in this part and put some of what we learned in the last part into practice.

Bluetooth mesh is interesting and can be a lot of fun to work with. But there's a learning curve. This set of exercises is substantial and designed to give you a lot of practice in implementing some of the most fundamental aspects of Bluetooth mesh. You'll find it repetitive at times. This is intentional and should help you recognise recurring patterns which you'll get better and better at working with as you get more practice. There are a lot of pages and there's a lot of code spanning the exercises too, so pace yourself. Don't try to do this all in one go, but progress through the exercises in increments which take however much time you find comfortable and can accommodate.

By progressing through the exercises, you'll produce code that will turn your devices into three distinct types of mesh node and you'll gain experience of a variety of the Bluetooth mesh models. The goal is to learn though and we will not be producing complete, reference implementations which would pass all qualification tests.

Goals

Following the instructions and explanations in this set of documents, your task is to create three types of mesh node which will work together in a mesh network. The three types of node will act as:

1. A light switch

2. A dimmer switch

4. A light whose on/off state and level are controlled by the switch nodes.

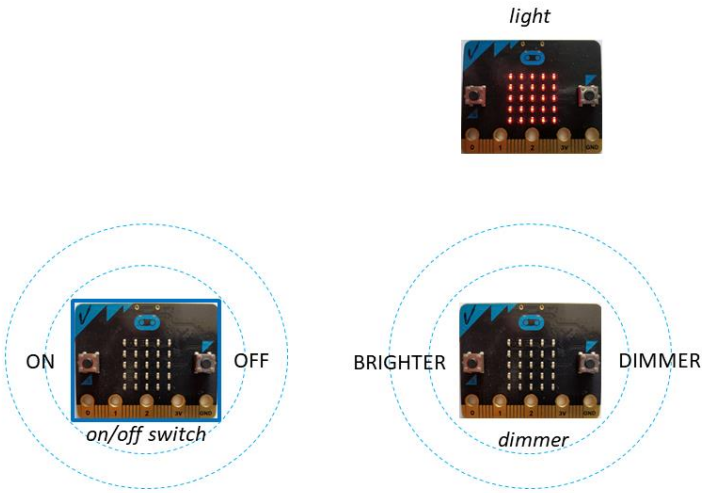

Figure 1 - hardware and primary functions

Video Demonstration

The expected behaviour relating to each of the coding exercises is demonstrated in a video you'll find in the video folder. If you're not sure what to expect from your tests, watch the video.

Pre-requisites

It is assumed that you know at least the basics of Bluetooth mesh, as covered in the *Part 2 - Basic Theory* document. You must also have basic competence in C/C++.

The Mesh Profile Specification

This is a self-study guide, which we hope will get you started with developing firmware for Bluetooth mesh products. It's not a substitute for the specifications though and you will be referred to them by this resource from time to time. Obtain the mesh profile specification and the mesh models specification from https://www.bluetooth.com/specifications/mesh-specifications and have them available for use.

Equipment Required

The coding exercises in this document are based upon the Zephyr RTOS running on a set of BBC micro:bits. Zephyr works on a significant number of different boards, some of which have Bluetooth LE support and are potential alternative devices for you to use instead of the micro:bit, with some caveats. The exercises make use of the micro:bit 5x5 LED display and so use a micro:bit-specific API for that purpose. If you decide to use a different board therefore, you'll need to change those aspects of the code which deal with anything which is specific to the micro:bit.

Why micro:bit?

The BBC micro:bit was chosen as the platform for this resource to be based upon because it has buttons, it has a display, you can connect external things to its edge connector and…. it's relatively cheap. We want this resource to be accessible to developers everywhere, so keeping the cost of the bill of materials down was important.

Why Zephyr?

Zephyr was chosen because it has support for Bluetooth mesh, to a good extent is hardware-agnostic, running on a significant number of boards and it's open source. The things you learn about Zephyr and Bluetooth mesh in this resource should be portable to other boards.

Note: it is not the intention to teach Zephyr programming as such. Zephyr APIs will be used and explained where required, but the emphasis of this and the associated resources is on learning about Bluetooth mesh so you should expect to need to consult the Zephyr documentation from time to time.

Bill of Materials

If you wish to complete all of the coding exercises in this lab and to be able to test all the related use cases at the same time, you will need the following items at least:

Item	Quantity	Purpose
BBC micro:bit	1 or more	To act as a light
BBC micro:bit	1	To act as an on/off switch
BBC micro:bit	1	To act as a dimmer
A computer running Windows, Mac OS or Linux	1	Software development
micro USB cable	1	For transferring binaries to a micro:bit

To be clear, we recommend you have at least 3 BBC micro:bits in total so that you can run three independent devices as a switch node, dimmer node and light node all at the same time.

If you do not wish to have all use cases up and running at the same time, you only require 2 x BBC micro:bits with one acting as the light and the other acting as either an on/off switch or as a dimmer.

Of course if you want to build a bigger mesh network, with more than one light to control then you can do so. The switches and sensors will be programmed to publish to a specific group address. As long as all of your lights subscribe to that group address, they will all respond in unison and as required to the switch and dimmer nodes.

Development Environment Set-Up

Zephyr can be used in a Windows, Mac OS or Linux environment. Follow the relevant instructions at http://docs.zephyrproject.org/getting_started/getting_started.html#set-up-the-development-environment to set up the Zephyr SDK on your machine.

FYI In creating this resource, the full zephyr repo was cloned and then version 1.12.0 checked out.

```
git clone https://github.com/zephyrproject-rtos/zephyr.git
cd zephyr
git checkout tags/v1.12.0
```

There are choices in how you set up your Zephyr environment and we leave this to you to decide.

Examples will be drawn from a Windows development environment.

Addresses

All Bluetooth mesh nodes must have a unique unicast address and device UUID. These are the values we'll be using. If you decide to use additional devices, make sure you allocate them theor own unique addresses.

node	unicast address	device uuid
switch	0x0001	0xcf, 0xa0, 0xea, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86, 0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0xa1
dimmer	0x0002	0xcf, 0xa0, 0xea, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86, 0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0xa2

light	0x0007	0xcf, 0xa0, 0xea, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86, 0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0xa7

Exercise 2 - System Architecture and Use Cases

The Goals section gives us our use cases. They are as follows:

1.	Switch a group of lights on by manually pressing a button
2.	Switch a group of lights off by manually pressing a button
3.	Set the brightness of a group of lights to a specified, absolute value by manually pressing a button
4.	Increase the brightness of a group of lights by a specified, relative value by manually pressing a button
5.	Decrease the brightness of a group of lights by a specified, relative value by manually pressing a button
6.	Initiate a dynamic transition of the brightness of a group of lights, from their current level to a target level and execute that transition over a specified time period and with an optional delay before starting the transition.

2A) Selecting Models

Mesh models are the fundamental building blocks of products and their behaviours. Therefore the first thing we need to do is identify standard models which we could use to meet the requirements of our use cases. Review the list of models in Section 7.3 of the Mesh Models Specification and see if you can identify the models we could use, keeping in mind that the difference between server models and client models is that servers contain state whereas clients do not. A solution has been provided on the next page.

Solution

There may be more than one way to meet these requirements but here are the models we intend to use in the exercises:

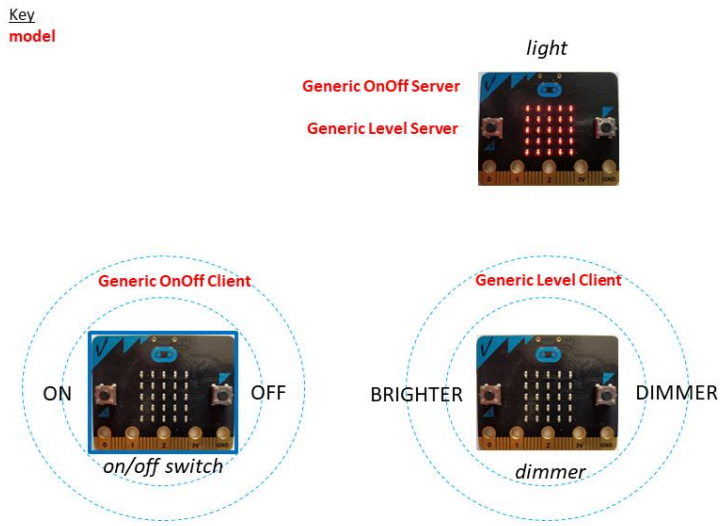

Figure 2 - models required

The Generic OnOff Client allows the on/off switch to send messages to the light, causing it to switch on or off. It includes the Generic OnOff Server to respond to those messages and has the state of being on or off and therefore is a server.

Similarly, the Generic Level Client allows the dimmer to send messages which will cause the light to get brighter or dimmer. To respond to those messages and have state which represents brightness, the light has the Generic Level Server.

All nodes must implement the configuration server and health server models too. We haven't shown them in the graphic but we'll need them.

2B) Identifying Messages and States

Review sections 3.2.1 (Generic OnOff Messages) and 3.2.2 (Generic Level Messages) and identify the messages that will be involved in our use cases. Going a step further, identify the states that will be updated or reported on by these messages. A solution is presented on the next page.

Solution

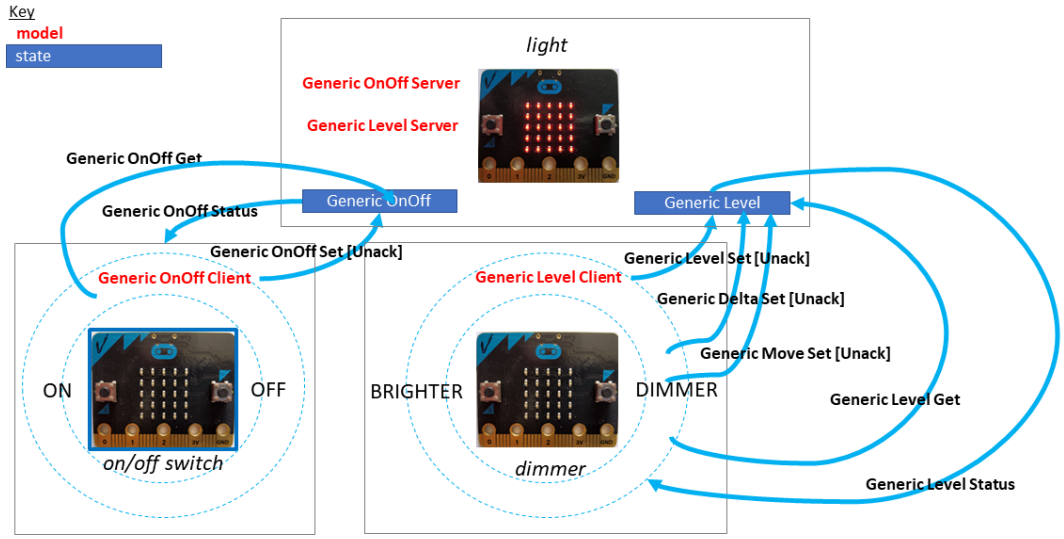

Figure 3 - models, messages and states required

You might not have included all of the messages shown in Figure 3. Ultimately, which of the client messages we send and whether we choose to use acknowledged or unacknowledged messages will depend on more detailed product requirements. It's rare to use acknowledged messages but we'll cover them for completeness and because server models must support them.

Generic Level servers are quite sophisticated and support changing level to an absolute value, by a relative delta and in a dynamic way known as a "move". We'll work with all three of these types.

The mesh models specification clearly illustrates which message types must be supported by each model and in some cases, under what conditions. As an example, Table 3.86 is repeated from the mesh models specification here:

Element	SIG Model ID	States	Messages	Rx	Tx
Main	0x1000	Generic OnOff (see Section 3.1.1)	Generic OnOff Get	M	
			Generic OnOff Set	M	
			Generic OnOff Set Unacknowledged	M	
			Generic OnOff Status		M

Table 3.86: Generic OnOff Server elements, states, and messages

This tells us that the light node must implement support for receiving Generic OnOff Get, Generic OnOff Set and Generic OnOff Set Unacknowledged. It must also be able to send Generic OnOff Status messages, which are sent in response to Generic OnOff Get and Generic OnOff Set messages.

The Generic OnOff Client model, on the other hand, defines all of its message types as optional:

Element	SIG Model ID	Procedure	Messages	Rx	Tx
Main	0x1001	Generic OnOff	Generic OnOff Get		O
			Generic OnOff Set		O
			Generic OnOff Set Unacknowledged		O
			Generic OnOff Status	C.1	

C.1: If any of the messages: Generic OnOff Get Generic OnOff Set are supported, the Generic OnOff Status message shall also be supported; otherwise, support for the Generic OnOff Status message is optional.

Table 3.116: Generic OnOff Client elements and messages

So we can pick and choose the types of messages which clients will send but servers must generally be able to respond to anything they *might* receive from an associated client model.

Next

The coding exercises are described in three documents, one for each of the three types of node we'll be working with. You should work through them in order, starting with the switch node. Good luck!