Developer Study Guide: An introduction to Bluetooth Mesh Networking

Hands-on Coding Exercises - Introduction

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 Bluetooth SIG	Initial version
1.0.4	14th December 2018	Martin Woolley Bluetooth SIG	Minor errata: #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. #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 Bluetooth SIG	Release 2.0 of the Study Guide.
2.0.1	21st December 2020	Martin Woolley Bluetooth SIG	Language changes
2.1.0	29th June 2021	Martin Woolley Bluetooth SIG	**Release:** Zephyr code now based on version 2.6.0 of the Zephyr SDK **Document**: Updated to indicate the need to checkout source code version zephyr-v2.6.0 from the Zephyr github repository.

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 several types of mesh node and you'll gain experience of several of the Bluetooth mesh models. The goal is to learn though and we will not be producing complete, reference implementations which would pass all formal Bluetooth qualification tests.

Goals

Following the instructions and explanations in this set of documents, your task is to implement or part-implement three different Bluetooth mesh models using two or more physical devices. The three models will allow your nodes to act as:

1. An on/off light switch

2. A lighting colour selector

4. A light whose on/off state and colour can be controlled by other Bluetooth mesh nodes.

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

Hardware and Software

The coding exercises in this document are based upon the Zephyr RTOS running on a Nordic Semiconductor nRF52840-DK for both the on/off switch and the colour controller node and a Nordic Semiconductor Thingy acting as the light node, using the Thingy's built-in coloured LED.

Zephyr works on a significant number of different boards, some of which have Bluetooth LE support and which may therefore be suitable for use with this study guide. If you do decide to use different equipment, note that there must be sufficient memory for all functions to be available (e.g. provisioning), you may need to connect external LEDs and may need to modify the GPIO code accordingly. Bluetooth mesh related code should not be impacted by a change of developer board. 32Kb RAM should suffice whereas the 16Kb available in devices such as the BBC micro:bit will not be enough.

You'll also need a tool with which to provision and configure your nodes. The nRF Mesh application for Android was used in the preparation of this study guide. Note that this application is also available for iOS.

The Nordic Command Line tool *nrfjprog* is used to clear boards.

J-Link software is used with a debugger probe for programming the Nordic Thingy and the RTT Viewer is used to monitor console output from the Thingy.

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
Nordic Semiconductor Thingy	1 or more	To act as a light
Nordic Semiconductor nRF52840 DK	1	To act as an on/off switch and a colour conroller
A computer running Windows, Mac OS or Linux	1	Software development
micro USB cable	1	For transferring binaries to a micro:bit

J-Link debugger probe	Optional	For flashing code to the Nordic Thingy. See Development Environment Set-up for further information.

If you want to build a bigger mesh network, with more than one light to control then you can do so. The switch will be programmed to publish to a specific group address. As long as all of your light nodes are correctly provisioned to be part of your network, subscribe to that group address and bind to the right application key, they will all respond in unison to the switch node.

Development Environment Set-Up

Zephyr

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

You should clone the Zephyr source code repository and check out the zephyr-v2.6.0 release.

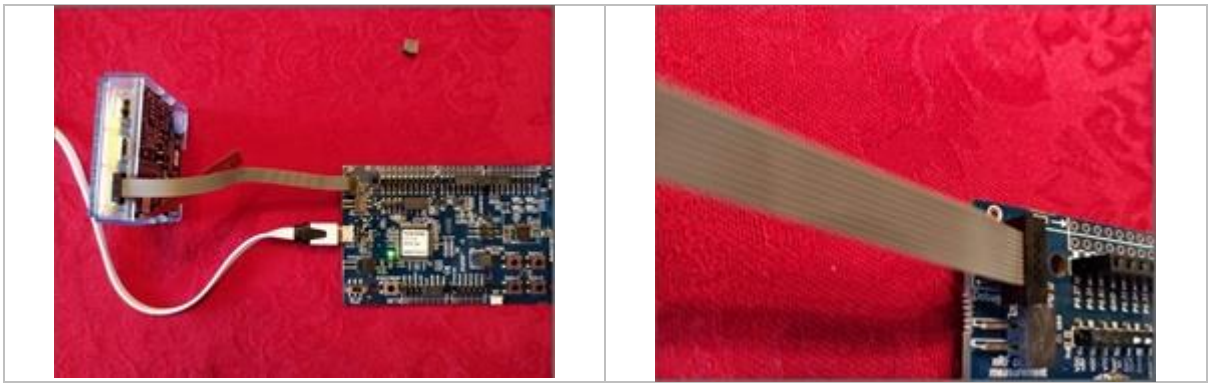
```
git clone https://github.com/zephyrproject-rtos/zephyr.git
cd zephyr
git checkout zephyr-v2.6.0
HEAD is now at 79a6c07536 release: Updates for v2.6.0
```

There are choices in how you set up your Zephyr environment and we leave this to you to decide. The examples in this resource use Zephyr's multipurpose **west** tool on Windows.

Nordic Thingy

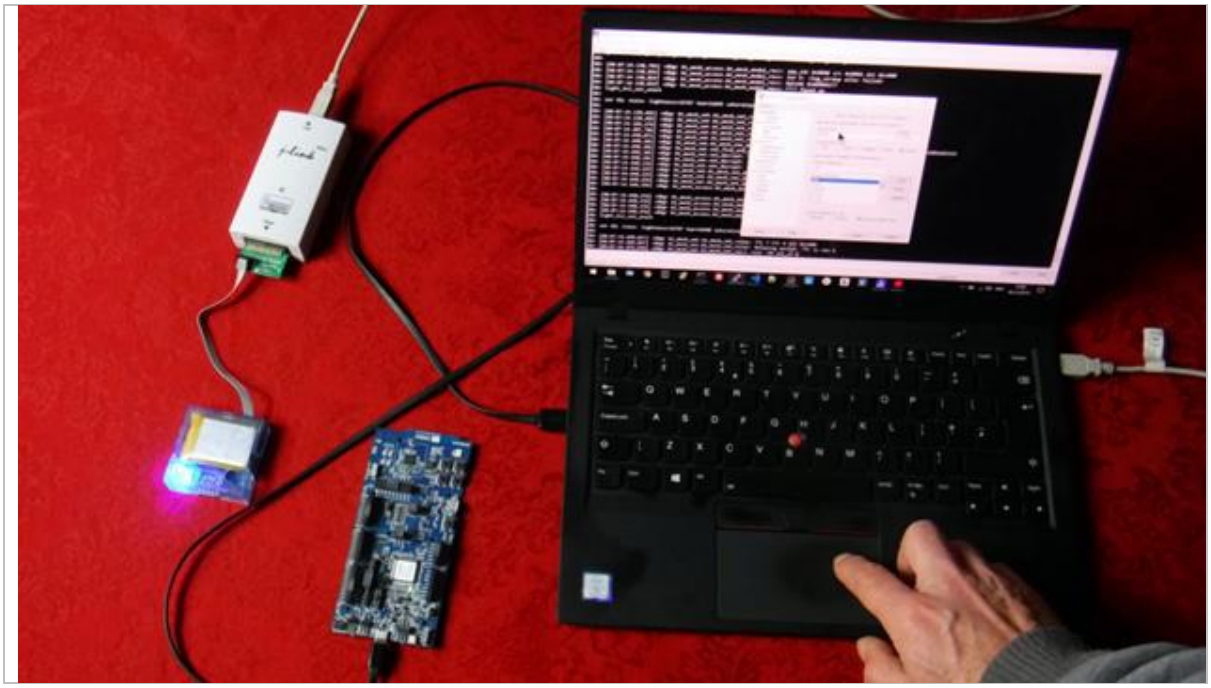
The Nordic Thingy does not contain a debugger host MCU like other Nordic developer kits. Therefore you must either use a J-Link debugger probe or a separate Nordic developer board plugged into your Thingy in order to flash it or to trace console messages. Console messages can be viewed with the J-Link RTT Viewer tool provided your Zephyr project contains the following configuration entries:

```
# J-Link Log Viewer
CONFIG_UART_CONSOLE=n
CONFIG_HAS_SEGGER_RTT=y
CONFIG_USE_SEGGER_RTT=y
CONFIG_RTT_CONSOLE=y
CONFIG_LOG_BACKEND_RTT=y
CONFIG_LOG_STRDUP_BUF_COUNT=20
CONFIG_LOG_STRDUP_MAX_STRING=80
```



Using a Nordic development kit to flash a Nordic Thingy

See https://docs.zephyrproject.org/latest/guides/tools/nordic_segger.html



Using a J-Link debug probe with Nordic Thingy and a nRF52840-DK connected directly over USB

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.	Change the colour of a group of lights to a new colour, selected in some predefined sequence by the colour controller node.

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

Here are the models we intend to use in the exercises:

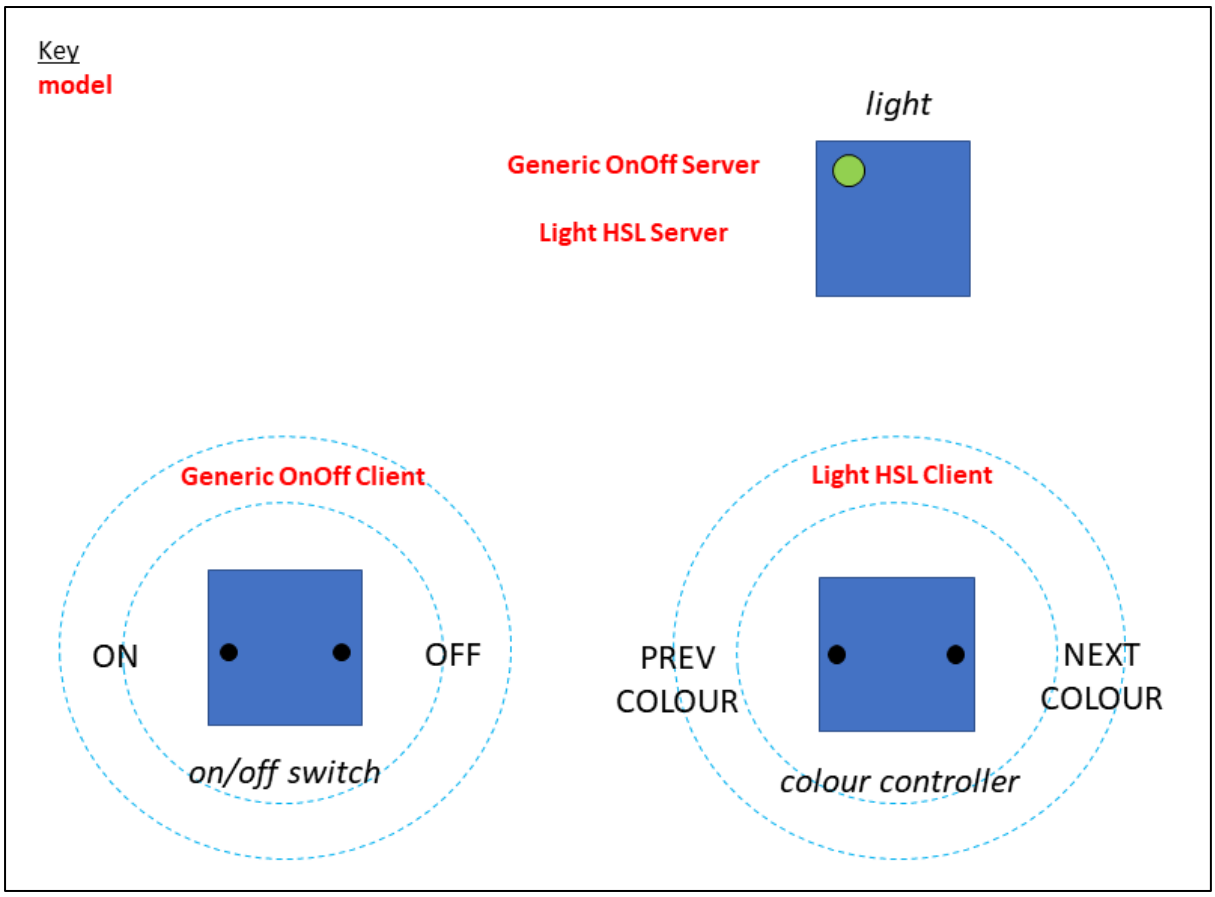


Figure 1 - models required

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

The Light HSL Client allows the node to send messages which will cause the light to change to a selected colour which is expressed in terms of Hue, Saturation and Lightness values. To respond to those messages, the light node has the Light HSL Server model.

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.

Three distinct nodes are shown in our idealised diagram, but as you'll see, we shall implement both the *generic onoff client model* and the *light HSL client model* in the same physical node.

2B) Identifying Messages and States

Review sections 3.2.1 (Generic OnOff Messages) and 6.3.3 (Light HSL 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

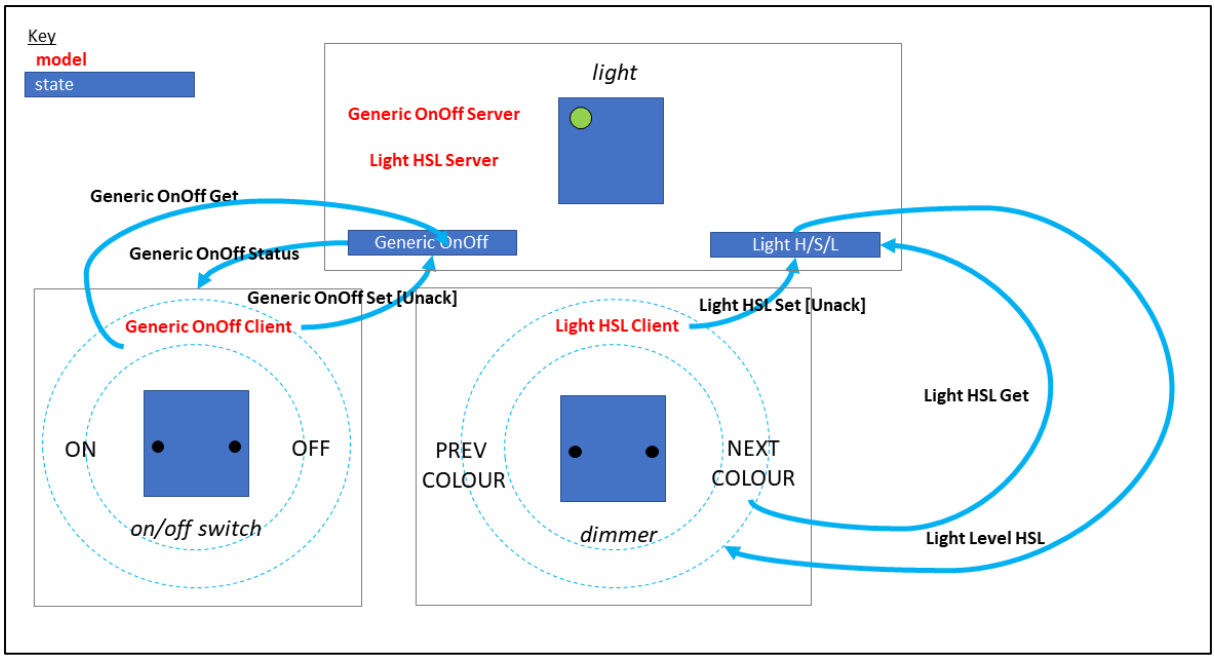


Figure 2 - models, messages and states required

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.

The Light HSL server model is quite sophisticated and has dependencies on various other models, each with its own message types. The mesh models specification and the Bluetooth Mesh Models technical overview paper explain this. We shall only implement a small subset of the messages associated with the Light HSL states in this study guide. Note that each of the H(ue), S(aturation) and L(ightness) component of a colour, exist in independent Bluetooth mesh states within the Light HSL server model, namely Light HSL Hue, Light HSL Saturation and Light HSL Lightness.

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 dealing with the generic on/off client model, the generic on/off server colour and the light client and server HSL models. You should work through them in order, starting with the generic on/off client. Good luck!