How to build a Bluetooth® Low Energy mesh application for STM32WBx5 line microcontrollers

Introduction

Bluetooth® Low Energy Technology (BLE)-mesh connects multiple low energy technology devices with mesh networking capability for internet of things (IoT) solutions. It integrates the STM35WBx5 line products with embedded Bluetooth® Low Energy technology communication in a powerful, range-extending mesh network with two-way wireless communication.

The solution contains the core functionality for secure communication and provides all the flexibility needed to build applications.

It uses STM35WBx5 line microcontroller (also referred to as STM35WBx5) with mesh stack APIs and related event callbacks. The software development kit (SDK) provides the mesh stack in library form and a sample application in source code to demonstrate how to use the library.

A BLE mesh is adapted for several application types requiring infrequent data transfer in a mesh network over Bluetooth® Low Energy technology, to create distributed control systems such as:

- Smart lighting
- Home and building automation
- Industrial automation.

The examples in this application note are to be used together with the P-NUCLEO-WB55 pack for which a demonstration example is available. The demonstration example used to change the application interface, uses the library for the required hardware and software functions. The demonstration application is available for the P-NUCLEO-WB55 pack.

The sample application implements smart light control scenarios, which can be modified to suit specific requirements.

1 General information

This document applies to STM35WBx5 line Arm®-based microcontrollers.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronym	Description
Ack	Acknowledge
API	Application programming interface
App	Bluetooth® Low Energyapplication
BLE	Bluetooth® Low Energy
BSP	Board support package
ECDH	Elliptic Curve Diffie-Hellman
GUI	Graphical user interface
HAL	Hardware abstraction layer
HW	Hardware
IDLE	Integrated development environment
LED	Light emitting diode
LPN	Low-power node
MCU	Microcontroller unit
MoBLE	Mesh over Bluetooth® Low Energy
SoC	System on chip
USB	Universal serial bus

2 BLE mesh technology concept

The Bluetooth® mesh (BLE mesh) networking specifications define requirements to enable an interoperable many-to-many (m:m) mesh networking solution for Bluetooth® Low Energy wireless technology — ideally suited for control, monitoring, and automating systems where tens, hundreds, or thousands of devices need to reliably and securely communicate with one another.

- Mesh profile: Defines fundamental requirements to enable an interoperable BLE mesh networking solution for Bluetooth® Low Energy wireless technology
- Mesh model: Introduces models used to define basic nodes functionality on a mesh network

2.1 Flooding protocol

In a mesh network, the messages are relayed to extend the communication range (multi-hop data transmission).

The mesh networks have additional benefits, like the ability to perform dynamic self-healing. For example, if one node fails, the communication can be restored using other nearby nodes.

Also, the devices communicate directly with each other. For example in the scenario where a light node responds directly to a switch, that provides the responsiveness the consumer expects.

To stay efficient, the BLE mesh takes advantage of a managed flooding technique: where the messages contain a sequence number, so that the node can understand if the received message is new or not. Old messages are not relayed in order to optimize the network usage and to protect against replay attacks.

Also the TTL (time to live) limits the number of times a message can be relayed.

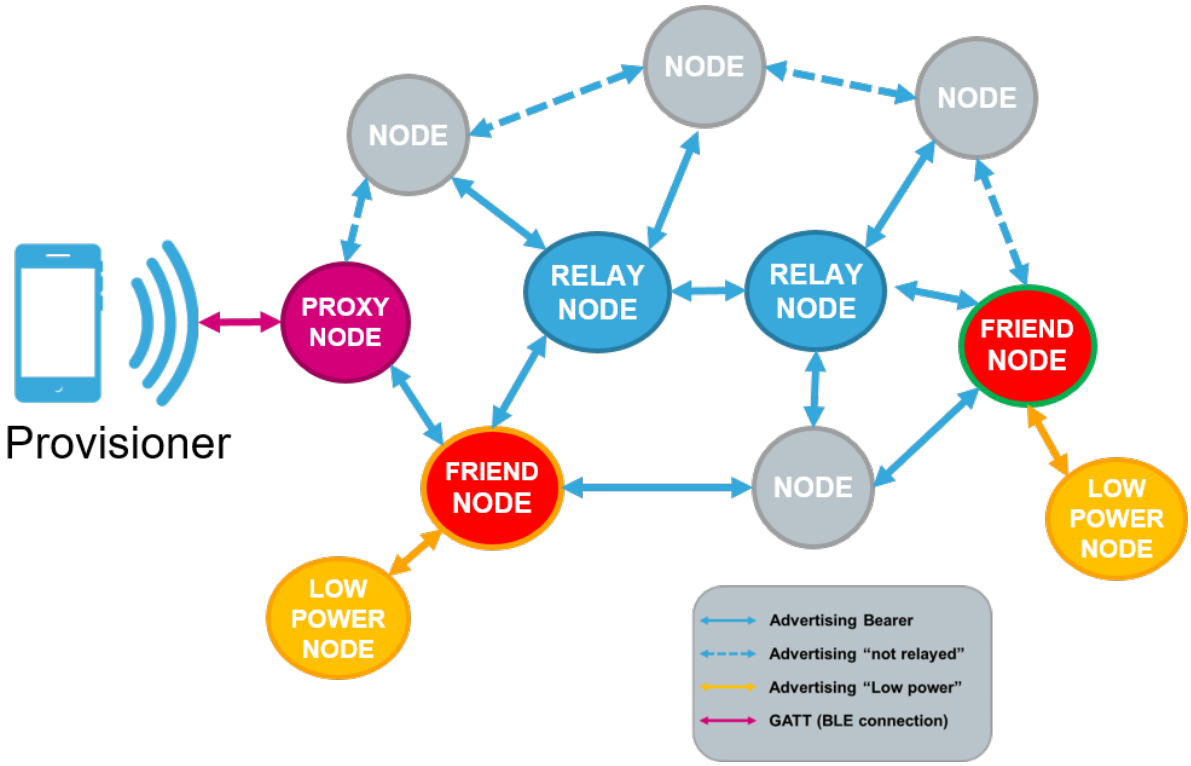
2.2 Nodes type

There are different types of nodes:

- Proxy nodes which expose the interface for a smartphone or a tablet to interact with a mesh network.
- Simple nodes are leaf nodes which cannot and do not relay messages. These are usually legacy nodes or resource constrained devices.
- Relay nodes retransmit received messages, and they enable multiple “hops” in the network.
- Low-power nodes are battery operated devices that primarily send but rarely receive messages, and so do not need a 100% duty cycle.
- Friend nodes are devices that store messages for low-power nodes (LPN) and deliver them whenever the LPNs wake up.

Low-power and Friend nodes are very important for enabling low power consumption of the mesh network. A typical BLE mesh topology is illustrated in the figure below.

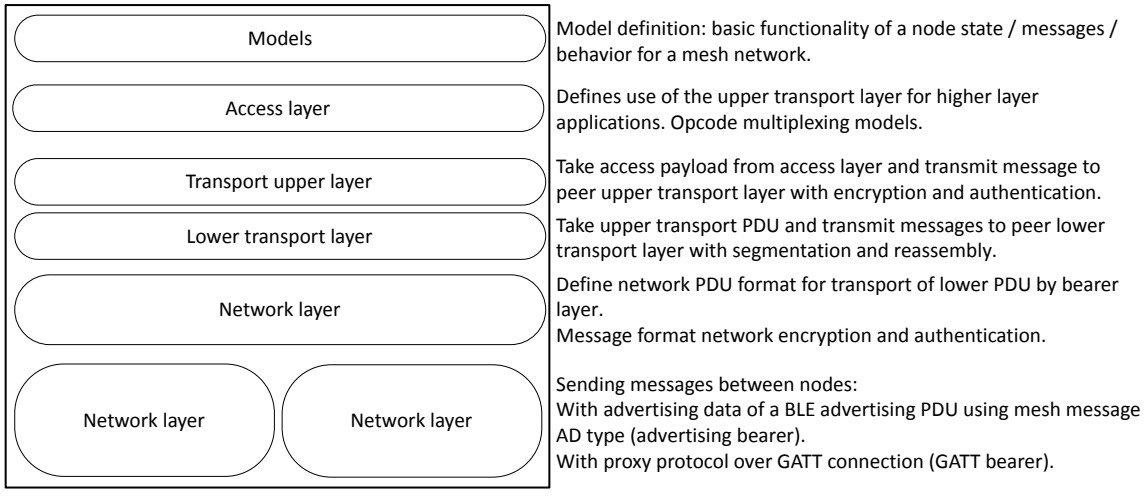
Figure 1. **BLE mesh topology**



2.3 System architecture

The STM35WBx5 BLE mesh library is built according the Bluetooth® SIG mesh profile v1.0.1 and mesh model v1.0.1 specifications as described in the following figure.

Figure 2. **BLE mesh library architecture**



2.4 Security

The security in the BLE mesh network is mandatory and cannot be disabled. All the messages are encrypted.

During the provisioning, a unique address is assigned to each node by the provisioner.

The security keys are distributed by the provisioner to the node.

To guarantee a sufficient level of security, there are three different security keys:

- Network key (NetKey) which secures the communication at the network layer and is shared across all the nodes on the network. The assignment of a given "NetKey" is what defines a given mesh network membership.
- Application key (AppKey) is used to encrypt/decrypt application messages. There may be multiple applications running on the same network: lighting, sensor monitor, and access control for example.

  To differentiate applications from each other, we use distinct AppKeys.
- Device Key is a special application key. Each device has a unique device key only known by the provisioner and the device itself, and it is used for provisioning, configuration and key management. Calculated using the shared secret derived from the ECDH key agreement between the provisioner and the device being provisioned.

By the time the message gets on the air, it has been encrypted twice. Once with an application or device key and the second time with a network key to guarantee the privacy.

The mesh profile security model uses a privacy mechanism called obfuscation through AES to encrypt the source address, sequence numbers, and other header information.

Message obfuscation makes it difficult to track messages sent within the network and, as such, provides a privacy mechanism to make it difficult to track nodes.

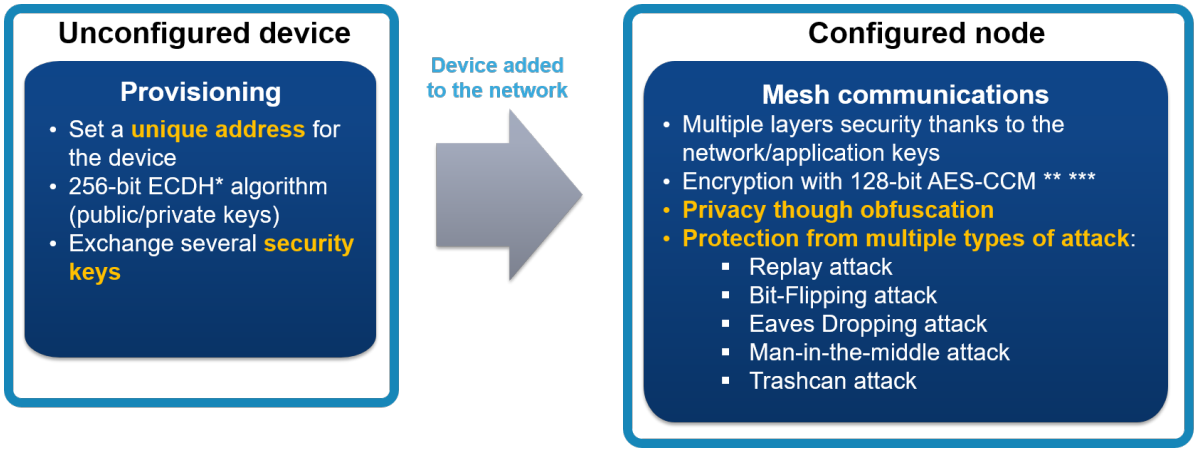
Several other security measures are used to prevent third-party interference and monitoring such as replay/eaves dropping attack protection.

Nodes can be removed from the network securely, preventing trash-can attacks, by erasing the security keys stored in the Flash memory.

The ECDH is an anonymous key agreement protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or to derive another key.

CCM is a combination of Counter mode encryption with CBC-MAC authentication, for cryptographic block ciphers. It provides both authentication and confidentiality. The CCM terminology "Message authentication code (MAC)" is called the "Message integrity check (MIC)" in Bluetooth terminology. The provisionning security model is illustrated in the figure below.

Figure 3. Security done by provisioning



2.5 Provisioning process

The process of configuring a device on a network is called provisioning. The process is started by a “provisioner”, which can be a ST BLE Mesh app running on the smartphone or the STM35WBx5 line microcontroller embedded provisioner device. The provisioning process is illustrated in Figure 101. BLE mesh node state machine.

The process by which a device joins the mesh is a secure process outlined in the five following steps:

1. Beaconing: a new GAP AD types introduced, including the “Mesh Beacon” AD type.
2. Invitation: the provisioner sends a PDU provisioning invitation to the unprovisioned device, which replies with a PDU provisioning capability.
3. Public key exchange: the provisioner and the unprovisioned device exchange their public keys (either static or ephemeral), either directly or using OOB.

4. Authentication: a cryptographic exchange takes place.
5. Distribution of the provisioning data: a session key is derived by each of the two devices from their private keys and the exchanged, peer public keys. The session key is then used to secure the subsequent data exchange required to complete the provisioning process, including the NetKey. Once the provisioning is completed, the provisioned device owns the NetKey, the IV index and a unicast address, allocated by the provisioner.

It is now known as a node.

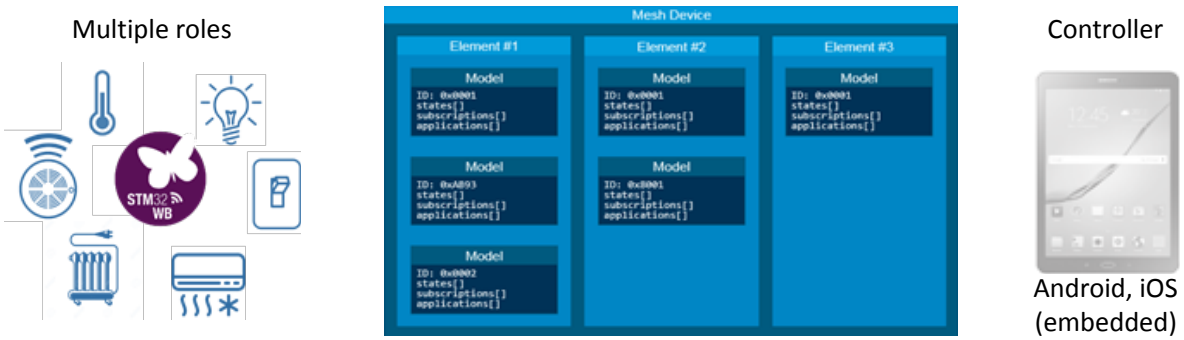
It is then configured as a device type and assigns the elements it contains to a Group

2.6 Elements and models

The monitoring and the configuration of the mesh devices are done through elements and models.

The elements and models structure is illustrated in the figure below.

Figure 4. Elements and Model – how device features are exposed



2.6.1 Elements

In the mesh network, the devices may have multiple roles. For example light switches, power strips, and so on.

The elements are what define the functionalities of a single node. The light fixture for the switch that has just been mentioned may have two lamps associated with it, and each lamp is a separate element.

This means that each element can be independently controlled. Each element must contain the required models. The number and the structure of elements of a node is static and does not change throughout the lifetime of a node. If any element needs to change, then the node must be reprovisioned first.

2.6.2 Models

A model represents a specific service and has:

- Unique sets of messages to process.
- States that may be shared with other elements.

An element within a node must support one or more models, and it is the model or models that define the functionality of a specific element. There are a number of models that are defined by the Bluetooth® SIG, and many of them are deliberately defined as “generic” models, having potential use across a wide range of device types.

Essentially, models are specifications for standard software components that defines a product behavior as a mesh device. Models are self-contained components and the products incorporate several of them. Collectively, from a network point of view, models make the device what it is.

Models defined by the Bluetooth® SIG are identified by unique 16 bits identifiers. Proprietary Vendor models use 32 bits identifiers.

2.6.3 Mandatory models

There are mandatory models to be used for the mesh network configuration.

- Configuration Server model :
  - Used to represent a mesh network configuration of a device.
  - Supported by a primary element and shall not be supported by any secondary elements.
  - It uses the device key for application-layer security.
- Health Server model :
  - Used to represent a device mesh network diagnostics.
  - Shall be supported by a primary element and may be supported by any secondary elements.
  - Uses the application key for application-layer security.
- Configuration Client model is used to represent an element that can control and monitor the configuration of a node.
- Health Server model is used to represent an element that can monitor the health messages of a node.

2.7 Addresses

Three address types are used:

- Unicast which is assigned to a device during the provisioning process. It represents a single element in a node.
- Group is a multicast address representing one or more elements in one or more nodes.
- Virtual which is similar to the group address, but it's more like a label UUID, for example it can be referred to a room name like the "kitchen".

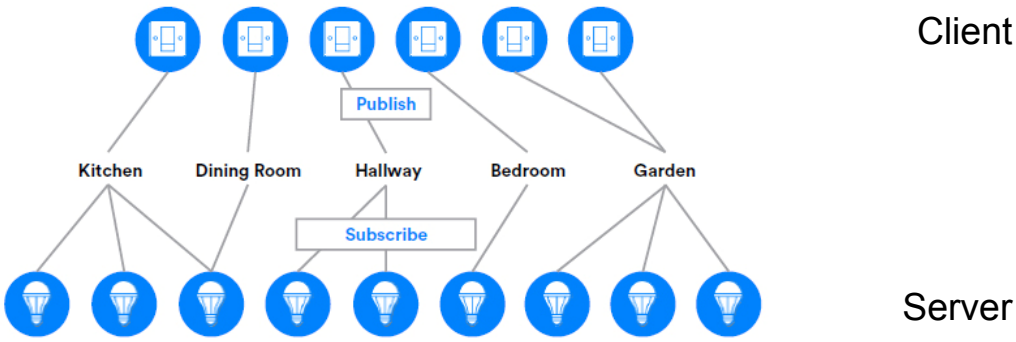
2.8 Publish and subscribe

The messaging model for the mesh is called publish and subscribe, where the nodes can publish messages using unicast, virtual, or group addresses. For example:

- A client device, such as a switch, can publish messages such as ON/OFF control and a server device, such as a the light, can be notified (if subscribed) that new message has arrived.
- Different nodes (for example the lights) can subscribe to receive messages sent to a specific group or virtual destination address. In the latter case, the virtual address can have a semantic meaning to the user, the name of a room for example.

The process is illustrated in the figure below.

Figure 5. Publish and Subscribe



2.9 Messages

The communication in the mesh network is “message-oriented”.

Messages can be reliable or unreliable.

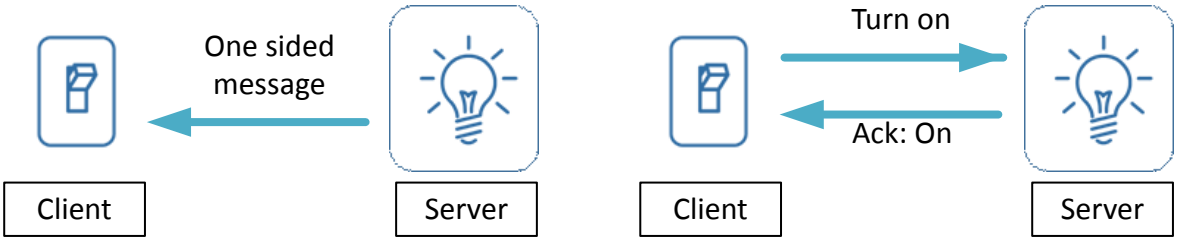
Reliable messages are acknowledged with a response on reception.

- The response message is defined by the model, and it is usually a status.
- No response implies the retransmission of the original message, or just abort the transmission.

This is illustrated in Figure 6.

For the Unreliable messages there is no acknowledge (UnAck), and they are usually sent to convey a status.

Figure 6. Messages

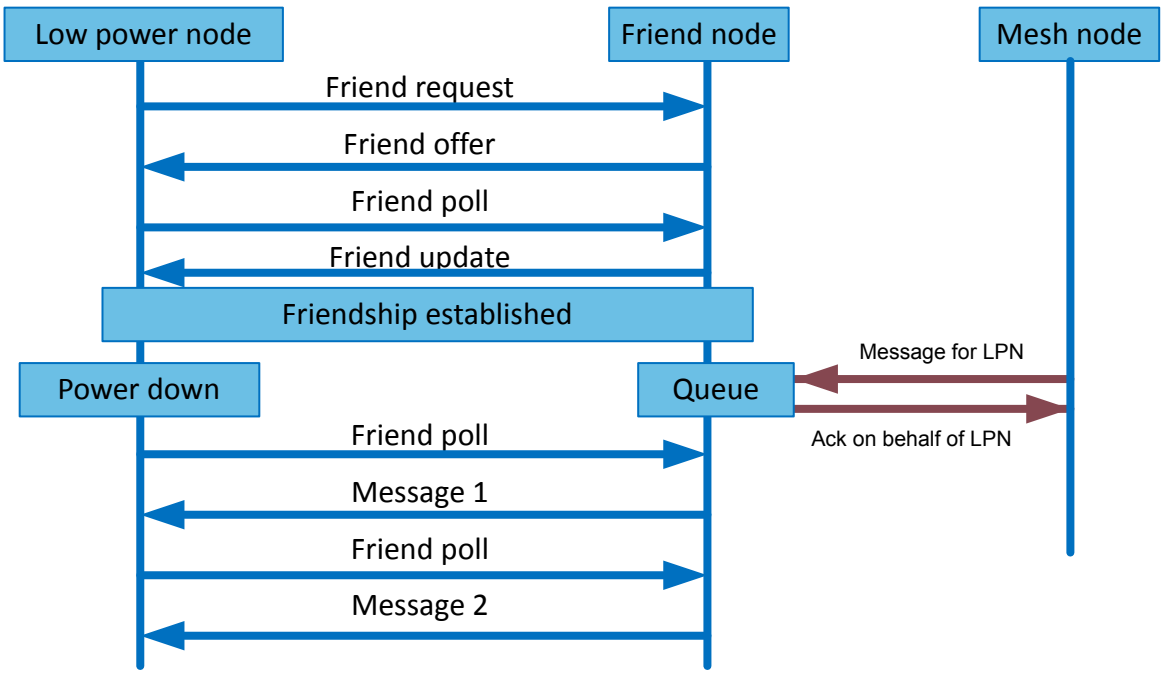


2.10 Friendship and low-power nodes

The active nodes must listen all the time to avoid missing messages which is power consuming. The duty cycles must be near 100%. For some application, devices may use coin cell batteries or energy harvesting, but still need to stay synchronized to be part of the network.

The low-power node (LPN) is used to reduce the duty cycles and save power. The LPN is associated to a friend node with a friendship implementation procedure. At this stage the friend node caches all incoming message and waits for the polling message from the LPN to provide it with any message. The friendship messaging process is illustrated in the figure below.

Figure 7. Friendship messaging



2.11 Mesh bearers

In the BLE mesh principle, two different bearers are used for sending messages through the networks:

- Advertising bearer

  The nodes use the Bluetooth® Low Energy GAP advertising and scanning to send and receive the messages. Each node uses non-connectable advertising but this method is not supported by all available smartphones.

- GATT bearer

  The smartphone and the Proxy node use a Bluetooth® Low Energy GATT connection to send/receive messages in a point to point topology. The GATT bearer uses a standard GATT service and provides two characteristics:

  – One for handling the value notifications
  – One for writing commands

This method is supported by all BLE-supporting smartphones.

3 Getting started

The sample application in this document implements a smart lighting control scenario. To modify the application, follow the sequence below:

- Step 1. Connect the board to the PC
- Step 2. Compile the firmware in IDE
- Step 3. Flash the firmware to the board
- Step 4. Provision the board in the ST BLE Mesh App
- Step 5. Toggle LED on board using the app.

3.1 Software and system requirements

The minimum system requirements to set up and run the BLE mesh smart lighting application are:

- PC with intel® or AMD processor running one of the following Microsoft® operating systems:
  - Windows® XP®
  - VISTA®Vista
  - Windows 7®
  - Windows 10®.
- At least 128 MB of RAM
- Two USB ports
- 40 MB of hard disk space.

Software required are the following:

- IDE:
  - IAR Systems® v8.20.2
  - Keil® µVision® v5.23
  - STM32CubeIDE.
- Programmer:
  - STM32CubeProgrammer: STM32CubeProg To flash the board with an already generated binary
- Mobile application:
  - Android™ available from the Google® playstore.
  - iOS™ available from the Apstore.

3.2 Development kit description

3.2.1 P-NUCLEO-WB55 pack board presentation and setup

The Nucleo board (illustrated in the figure below) uses STM32WB 32-bit microcontrollers, based on Arm® Cortex® processor(s).

It contains a 2.4 GHz RF transceiver supporting Bluetooth® specification v5.0 and IEEE 802.15.4-2011 PHY and MAC.

A dedicated 32-bit Arm® Cortex® M0+ CPU for real-time radio layer.

Three user LEDs shared with the ARDUINO®, and four push-buttons.

A 32.768 KHz LSE crystal oscillator and a 32 MHz crystal oscillator with integrated trimming capacitors.

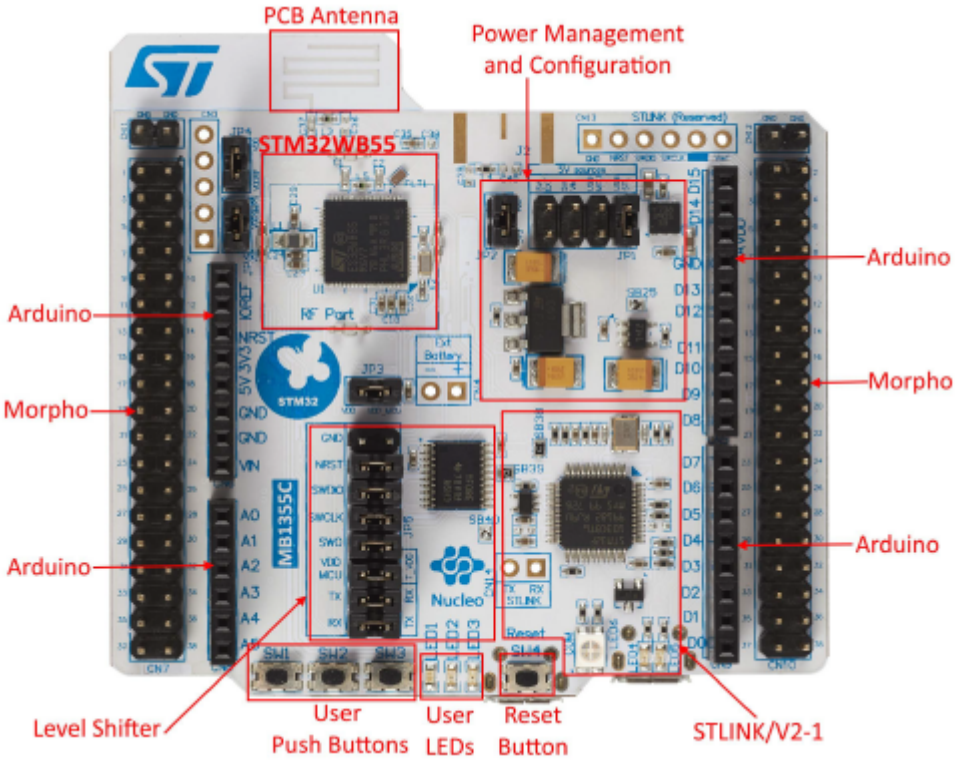
Connectors for board expantion:

- ARDUINO® Uno V3
- ST Morpho

Flexible board power supply: ST-LINK/V2-1 USB VBUS and external sources.

It support of a wide choice of integrated development environments (IDEs) including IAR Systems®, Keil®, GCC-based IDEs, Arm® Mbed™.

Figure 8. STM32WBx5 board connection with PC



Connect the main P-NUCLEO-WB55 pack board to the host PC with a USB cable. This cable is used for two purposes: to provide power to the P-NUCLEO-WB55 pack board and a serial data link with the PC.

The P-NUCLEO-WB55 pack board integrates the ST-LINK/V2-1 debugger/ programmer. In order to connect to the ST-LINK, the user must use STM32CubeProgrammer, available at STM32CubeProg.

Comprehensive free software libraries and examples are also available with a variety of examples, as part of the STM32Cube package.

Note: Ensure that STM35WBx5 mirocontroller is updated with Bluetooth® Low Energy stack version 1.6.0 or higher. Use STM32 ST-LINK Utility GUI to manage the update.

3.2.2 P-NUCLEO-WB55 board button usage

Table 2. Button usage and LEDs management

Features	Buttons	LEDs
Mesh Lib error	-	LED2 blinks continuously
Flash error access	-	LED3 lights up
Mesh Lib started successfully	-	No blink
Provisioning	-	No blink
Unprovisioning	1. RESET + SW1 buttons pressed 2. Release RESET button 3. Long press SW1 button causes unprovisioning 4. Reset the board	LED1 lights up and then keeps blinking during the unprovisioning process
Running the from board	Press SW1 button on any board. It sends the command on the Publication Address	1. LED1 on the board toggles (if subscribed to the published address) 2. LED1 on other boards also toggle (if subscribed to the publish address)
Proxy connection	-	LED2 lights up

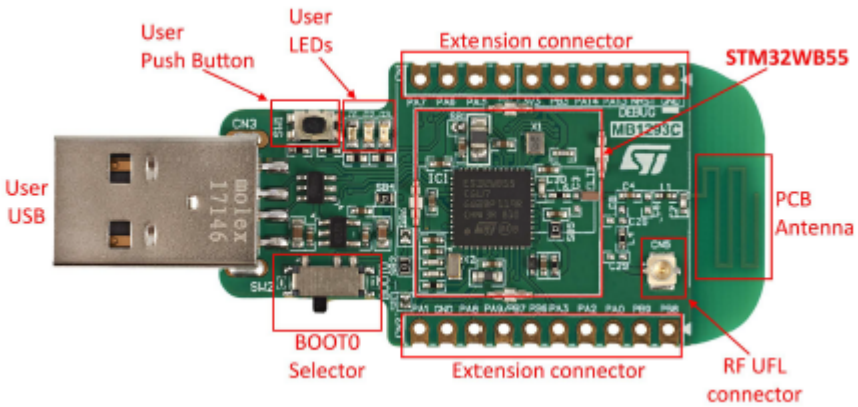
3.2.3 Dongle presentation and setup

The USB dongle is designed around the STM32WB55CG microcontroller in a 48-pin UFQFPN48 package. The dongle is illustrated in the figure below.

It contains:

- A 2.4 GHz RF transceiver supporting the Bluetooth® specification v5.0 and IEEE 802.15.4-2011 PHY and MAC.
- A dedicated 32-bit Arm® Cortex® M0+ CPU for real-time radio layer.
- User push button and three user LEDs for application purpose.

Figure 9. USB dongle board



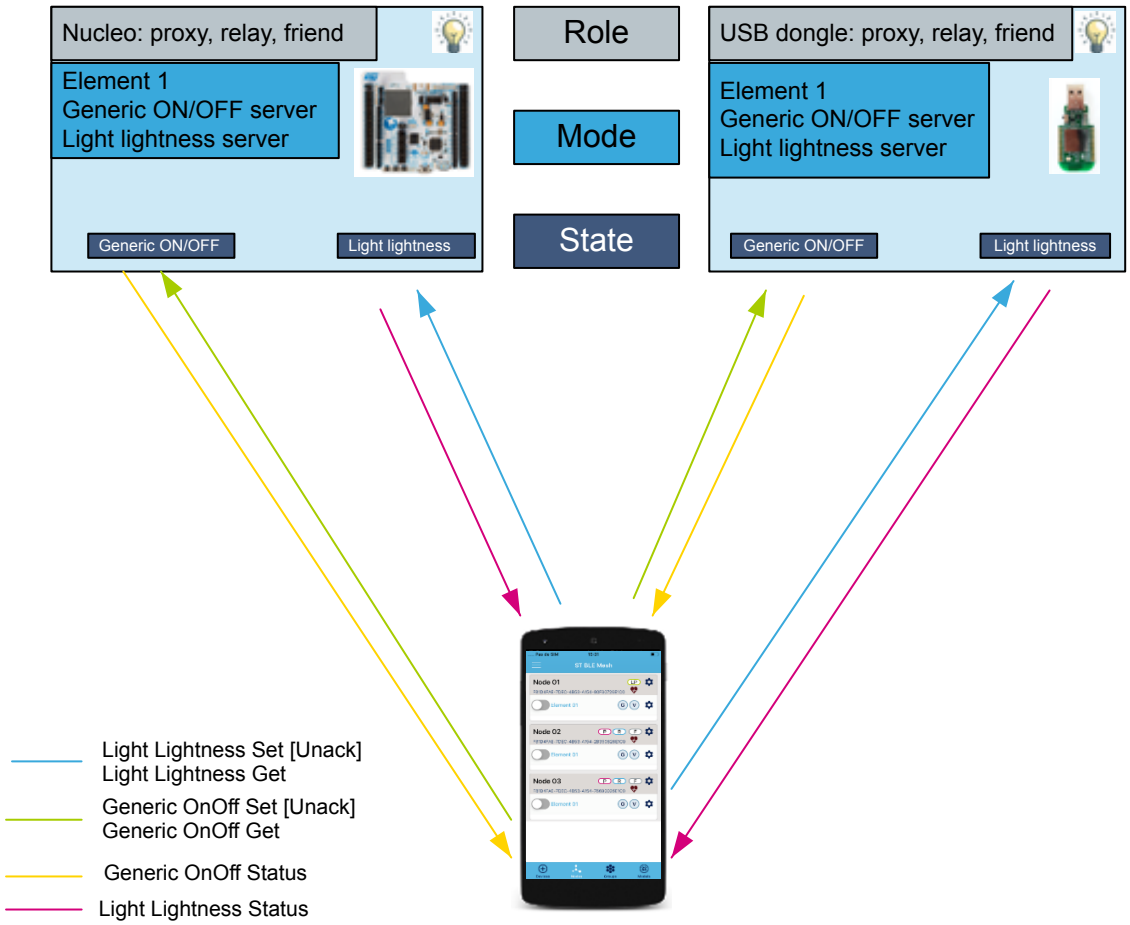
Plug the USB dongle to a USB port of the computer.

In order to connect to the USB dongle, the STM32CudeProgrammer must be set up, available at STM32CubeProg.

3.2.4 Simple light system control demonstration

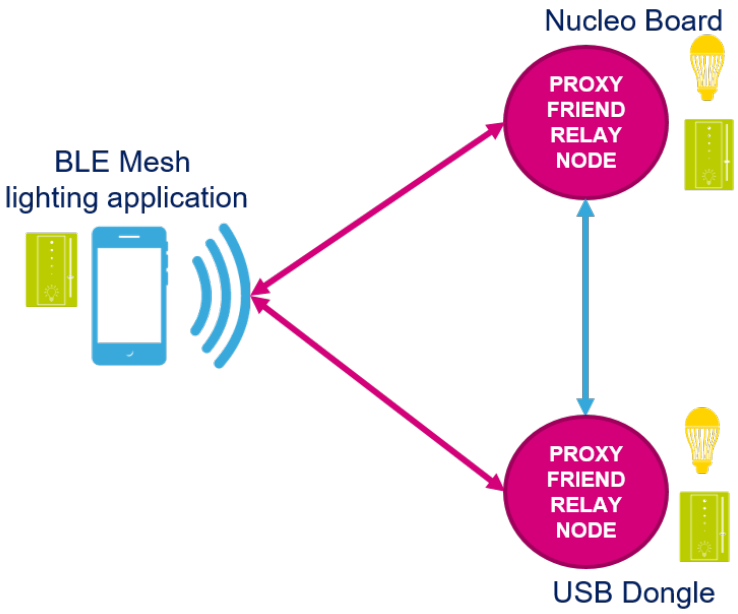
This section describes how to setup a Bluetooth® Low Energy mesh network demonstration quickly and easily, based on a lighting application using the code present in the BLE mesh package. The demonstration structure is illustrated in Figure 10.

Figure 10. Demonstration details



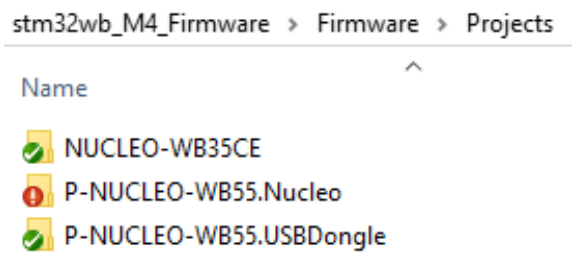
The ST BLE Mesh package contains Bluetooth® Low Energy application examples that can be used to setup a BLE mesh network demonstration using a smartphone and a P-NUCLEO-WB55 pack. The physical implementation of the demonstration is illustrated in Figure 11.

Figure 11. **Demonstration illustration**



All the available platform code is in the project directory illustrated in Figure 12.

Figure 12. **Sample project structure**

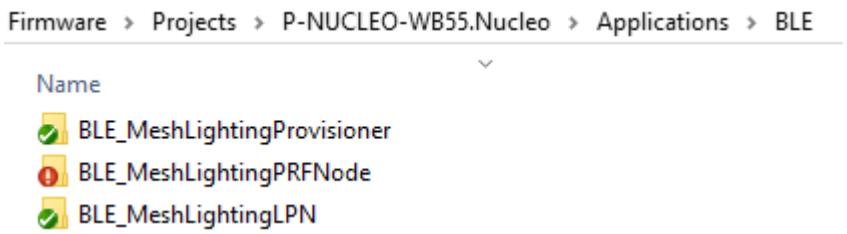


Refer to Section 3.4 to build the application code.

Nucleo Application

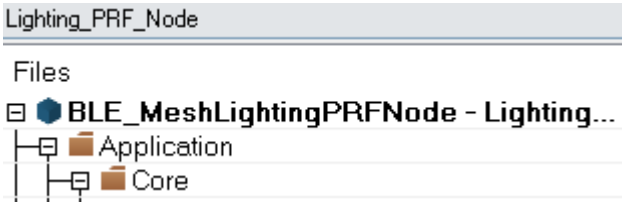
Select the `BLE_MeshLightingPRFNode` project for Nucleo target and open it in a suitable IDE, IAR Systems® for example. See the Figure 13 for an example of the Nucleo project.

Figure 13. **Nucleo BLE_MeshLightingPRFNode project**



The following information is now available as illustrated in Figure 14.

Figure 14. **Nucleo BLE_MeshLightingPRFNode file structure**



Compile the project a first time if it is not already done.

In `mesh_cfg_usr.h` file, the following models must enabled:

- `ENABLE_GENERIC_MODEL_SERVER_ONOFF`
- `ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS`

Build the project and flash it to the P-NUCLEO-WB55_pack board using the IDE Debugger.

Open a terminal program and connect it over the appropriate serial port. Reset the board and following information information is visible.

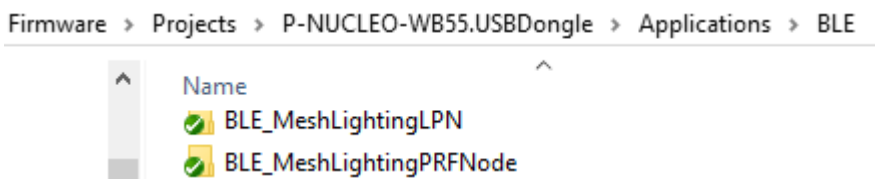
```
Unprovisioned device                    /* The device is not yet provisioned
********************
PB-ADV Enabled
PB-GATT Enabled
Feature: Relay Enabled
Feature: Proxy Enabled
Feature: Friend Enabled
To Unprovision: Do Power On-Off/Reset 5 time
models data are saved in Flash
models data will be saved on Power failure
Number of Elements enabled: 1
Neighbour Table is enabled
Generic On Off Server model enabled        /* List of the models available in the device
Light Lightness Server model enabled
********************

BLE-mesh Lighting Demo v1.12.008
BLE-mesh Library v01.12.008
BLE Stack v1.6.0 Branch=0 Type=3
FUS v1.0.2
BD_MAC Address = [c0]:[e1]:[26]:[08]:[35]:[2b]         /* BD Address
UUID Address = [f8] [1d] [4f] [ae] [7d] [ec] [4b] [53] [a1] [54] [2b] [35] [08] [26] [e1]
[c0]
485 Generic_PowerOnOff_Set - Generic_PowerOnOff_Set callback received
```

Dongle Application:

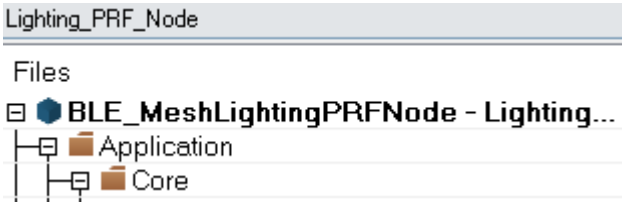
Select the `BLE_MeshLightingPRFNode` project for USB Dongle target and open it into your favorite IDE as illustrated in Figure 15.

Figure 15. **Dongle BLE_MeshLightingPRFNode project**



The following information is now available as illustrated in Figure 16.

Figure 16. **Dongle BLE_MeshLightingPRFNode file structure**



Compile the project a first time if it is not already done.

In `mesh_cfg_usr.h` file, the following models must be enabled:

- `ENABLE_GENERIC_MODEL_SERVER_ONOFF`
- `ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS`

Build the project using a suitable IDE. Once this is done complete the steps below:

1. Move the SW2 switch of the USB dongle to Boot0 and connect it to the computer.
2. Open the STM32CubeProgrammer GUI and select the USB mode. In USB configuration using Port: USB1, select “Connect”.
3. Go to Address 0x08000000, and click on “Erasing & programming” icon.
4. Select the path to your binary, check “Verify programming” and uncheck “Run after programming” checkboxes.
5. Click on “Start programming”.
6. Once completed, disconnect STM32CubeProgrammer and move the SW2 switch of the USB dongle to the original position.

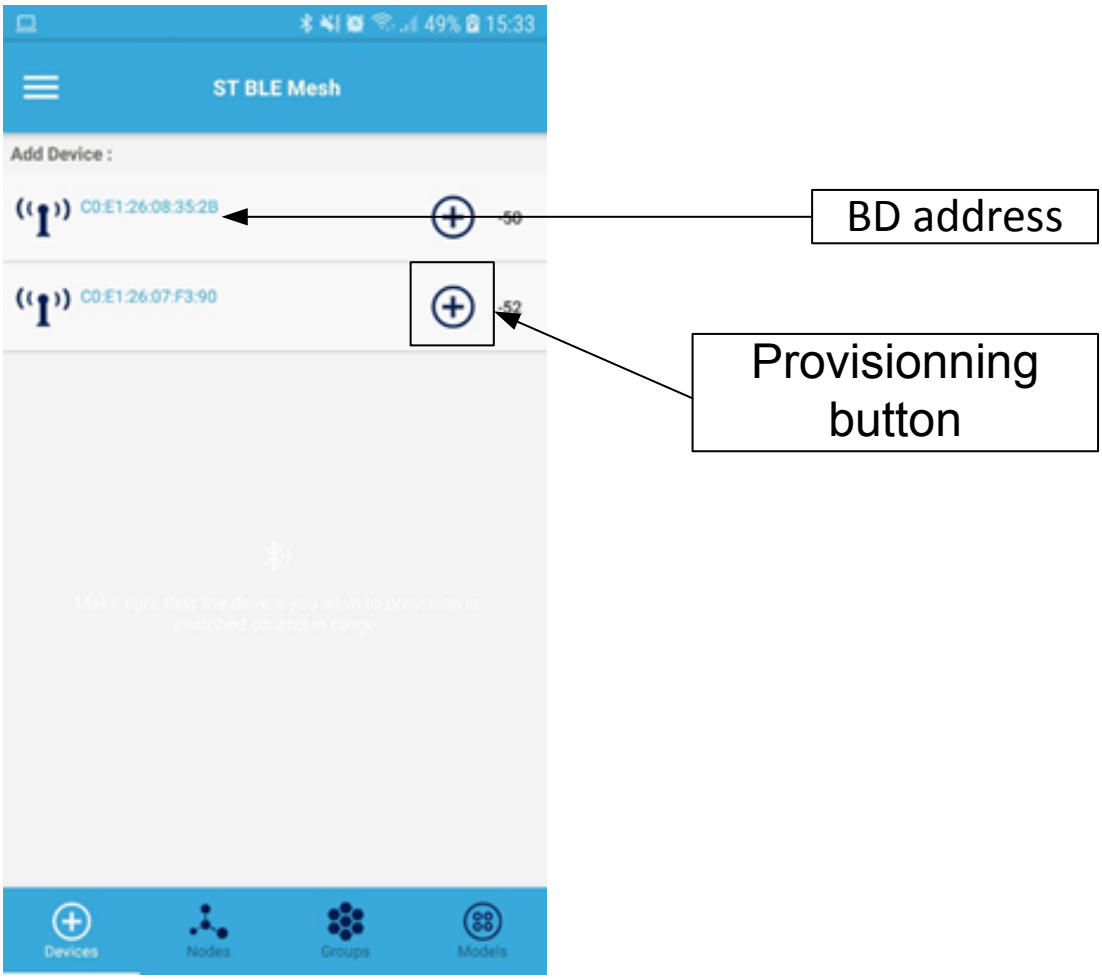
Now the dongle is detectable by the ST BLE Mesh Android™/iOS™ application and available to the network.

See the following section to know how to use the ST BLE Mesh application.

ST BLE Mesh Android™/iOS™ application

Launch the ST BLE Mesh application on a smartphone and check it detects both devices in the unprovisioned device list as illustrated in Figure 17.

Figure 17. **ST BLE device list**

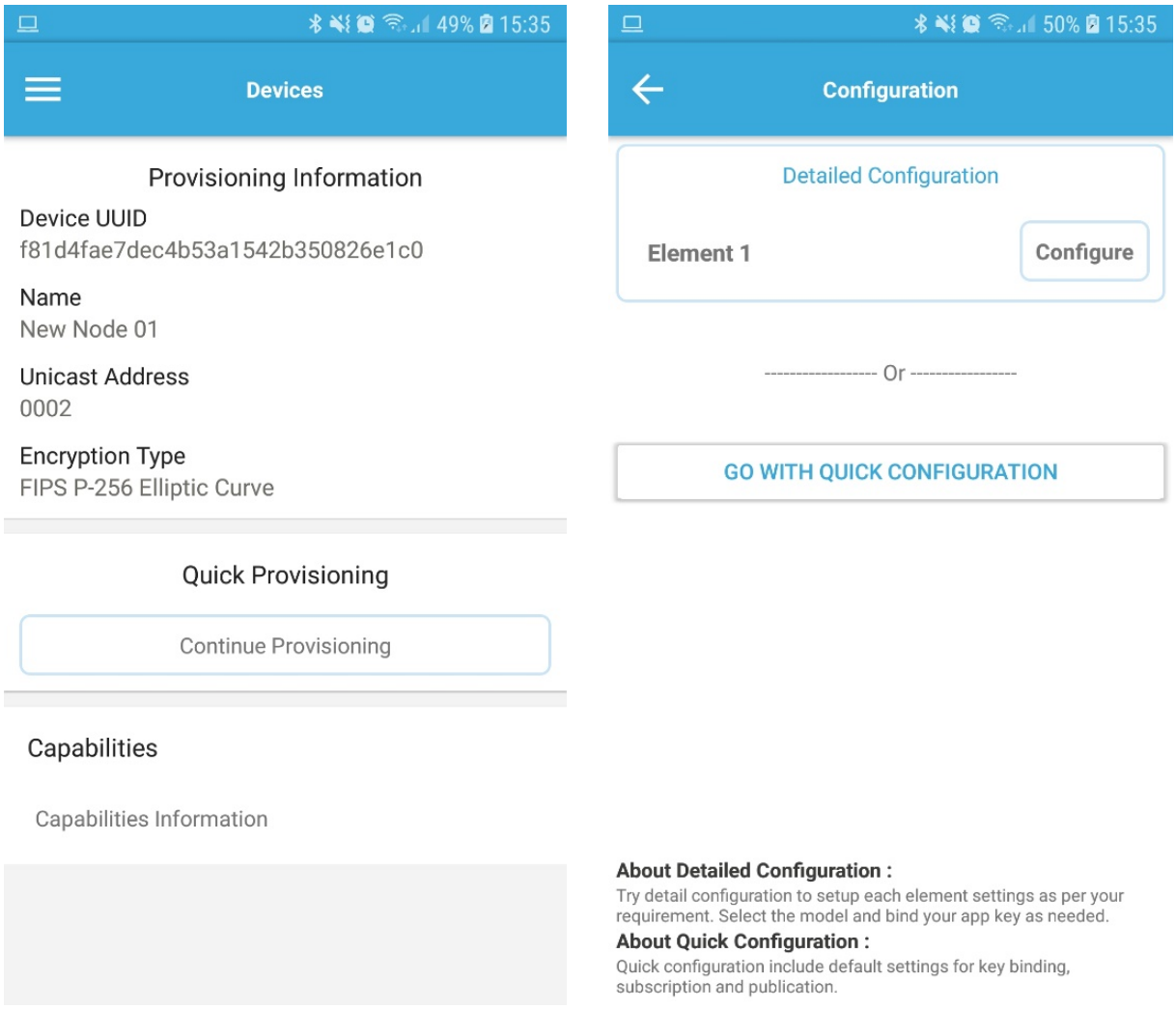


Te device provisionning can now be carried out on the network. Provision the devices by clicking on the "Provisioning button" illustrated in Figure 17.

The unprovisionning process is described in Device unprovisioning section.

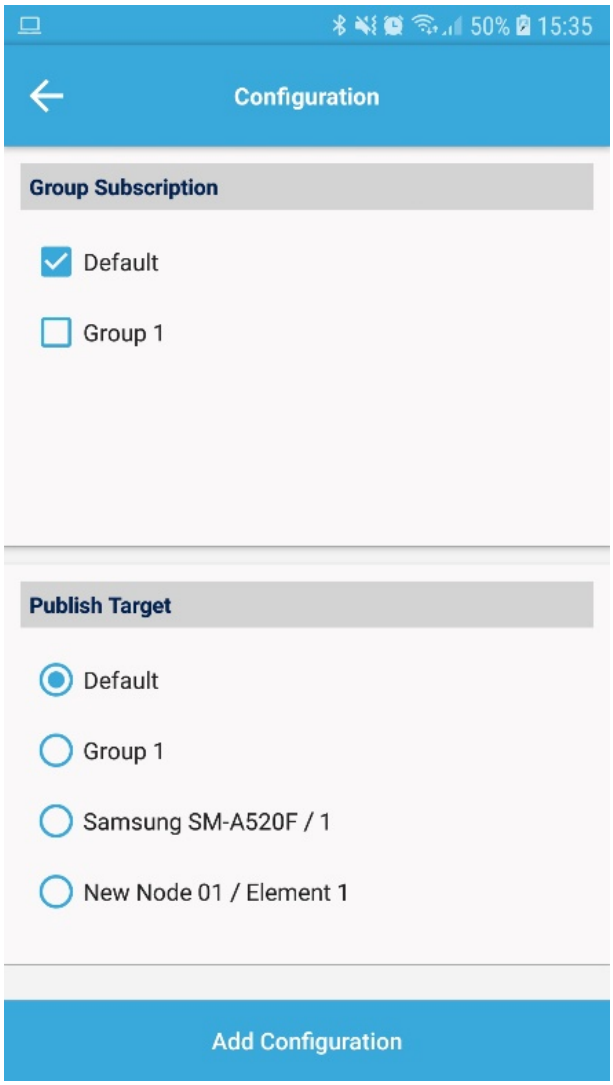
After the first provisioning steps are completed, click on “Continue Provisioning”, and proceed with the configuration, click on “GO WITH QUICK CONFIGURATION” and select the proposed AppKey. The "QUICK CONFIGURATION" screens are illustrated in Figure 18.

Figure 18. **Quick configuration screens**



The publication and subscription groups are available to be selected as illustrated in Figure 19.

Figure 19. **Publication and subscription choices**



The node is now be able to:

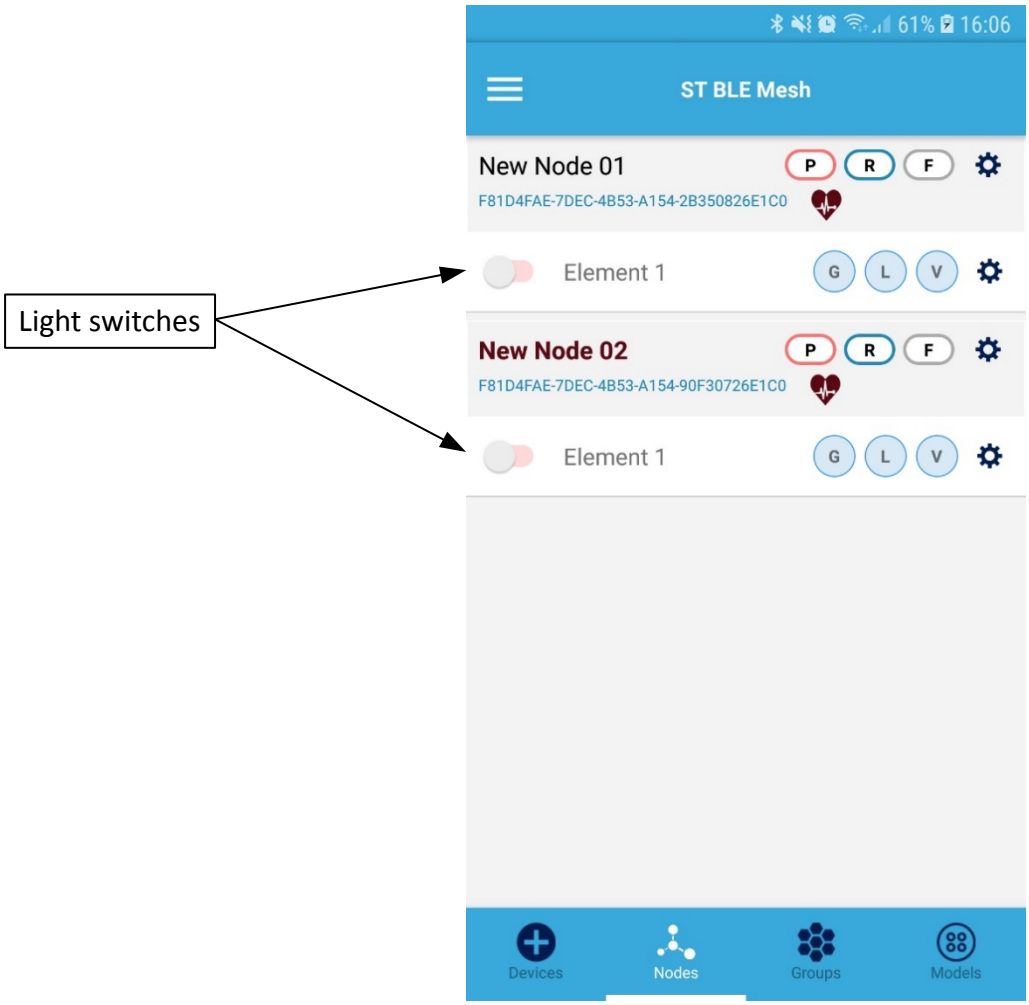
- Receive messages destined to the selected subscription group.
- Send messages to the selected publication group.

Select the groups , click on the “Add Configuration” button.
When the provisioning and model binding is complete, the terminal displays the following logs:

```
3182936 Appli_BleUnprovisionedIdentifyCb - Unprovisioned Node Identifier received: 0a
3183936 Appli_BleAttentionTimerCb -
[…]
Device is provisioned by provisioner
```

Then, go to the node interface screen illustrated in Figure 20.

Figure 20. **BLE mesh interface screen**



The lights on each of the corresponding nodes are turned on and off by clicking on the corresponding light switch button.

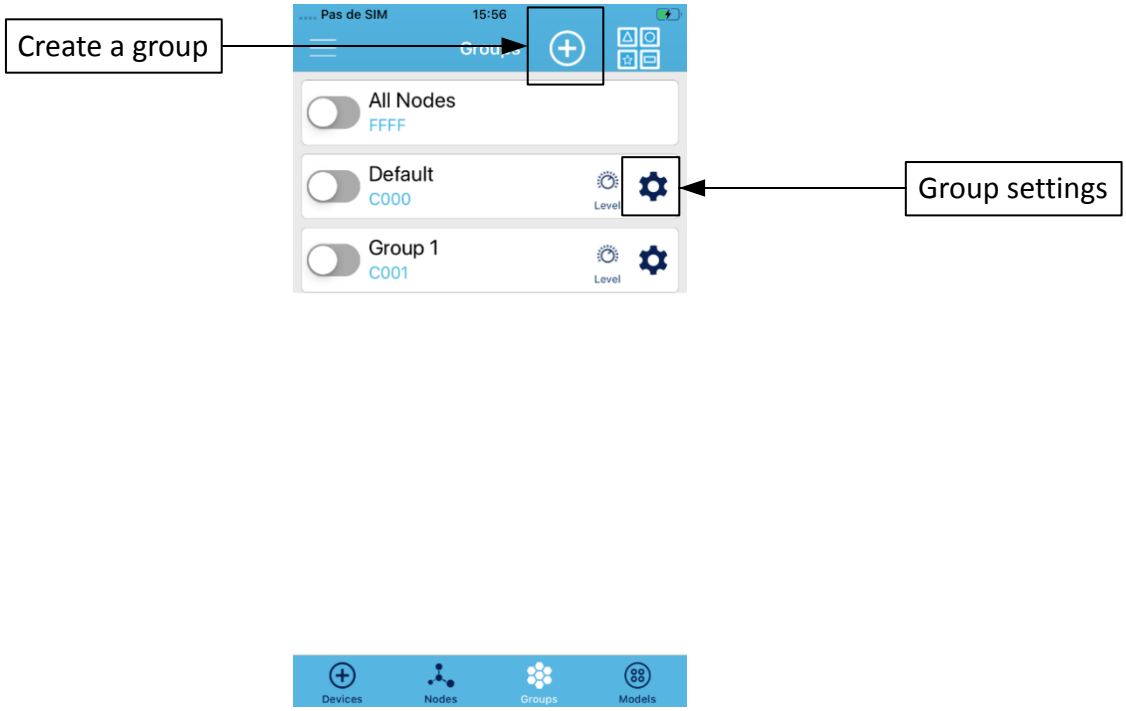
All of the LEDs on the network can also be turned on and off by clicking on the SW1 button directly in one of the connected nodes.

Note: The smartphone application switches are not be updated when SW1 button is pressed.

Group creation and management

To create a group, go to group interface as illustrated in Figure 21.

Figure 21. **Group interface**



Select the group settings button. Open the group setting and add the required nodes to the group. A group member is also known as a subscriber.

The LEDs in the group now all operate simultaneously when operated from this interface.

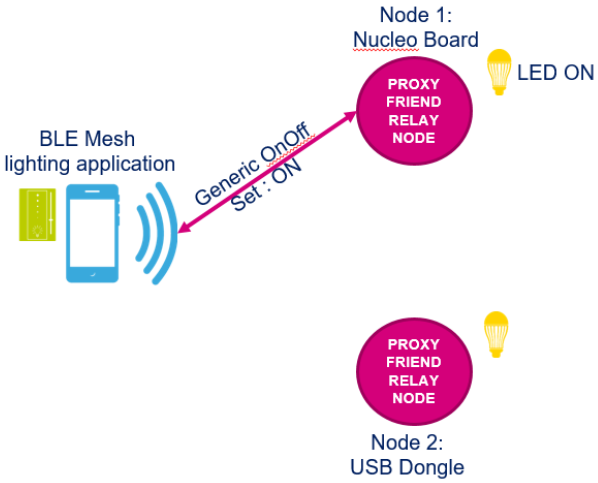
The first BLE mesh light in the network is now set up and operational.

For more information about the application interface, refer to the Android™/iOS™ Application How To document.

Demonstration

From the node interface on the smartphone, click on the light switch of one of the nodes, for example node 1. The LED of the corresponding node switches on. The process is illustrated in Figure 22.

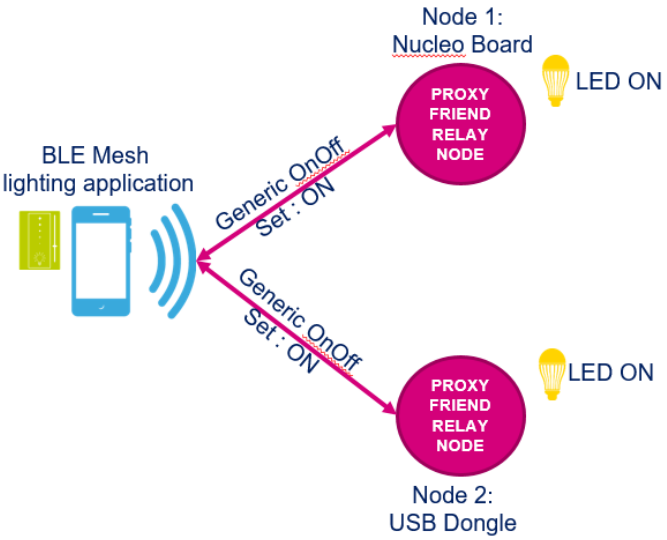
Figure 22. **Single node demonstration illustration**



Click on the switch again to turn the LED Off.

From the smartphone group interface, click on the light switch of “All Nodes” group, all the LEDs are switched on as illustrated in Figure 23.

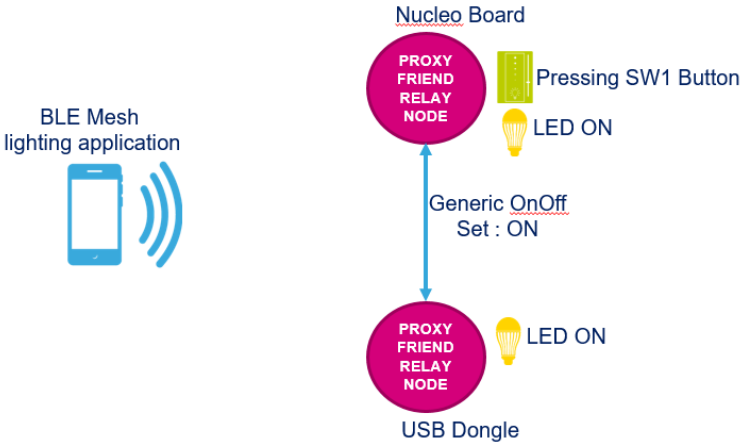
Figure 23. All node demonstration illustration



Click on the switch again to turn the LED Off.

Pressing on the SW1 button also switch the LEDs of the connected nodes to the network on or off. See Figure 24.

Figure 24. SW1 LED actuation model



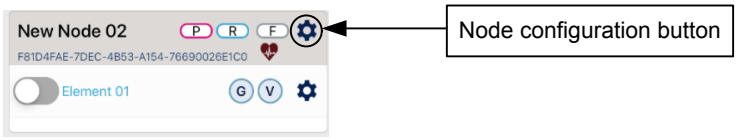
Note: The LED status on the smartphone application is not be updated with this action.

Device unprovisioning

To unprovision a node, use one of the methods below.

- From the smartphone application, go to the nodes interface and click on the node configuration button as illustrated in Figure 25:

Figure 25. Node unprovisioning



To finish the process, click on "Remove Node". The node is removed from the network.

- From the board itself, maintain on SW1 button pressed, click on the "RESET" button, release "RESET" button but keep SW1 button pressed until LED1 starts blinking. When LED1 is blinking, your board is unprovisioned.

3.3 UART interface on the firmware

The boards are connected to a PC via a USB connection. Use any convenient software terminal such as HyperTerminal, Hercules, Putty, and so on to open the serial communication port on the PC to check the messages from the board.

The UART of the controller on the board is connected to the PC via a virtual communication (VCOM) port. The settings to open the communication port are given in the table below.

Table 3. Virtual communication port settings

Setting	Value
Baud	115200
Data size	8
Parity	none
Stop bits	1
No hardware control	

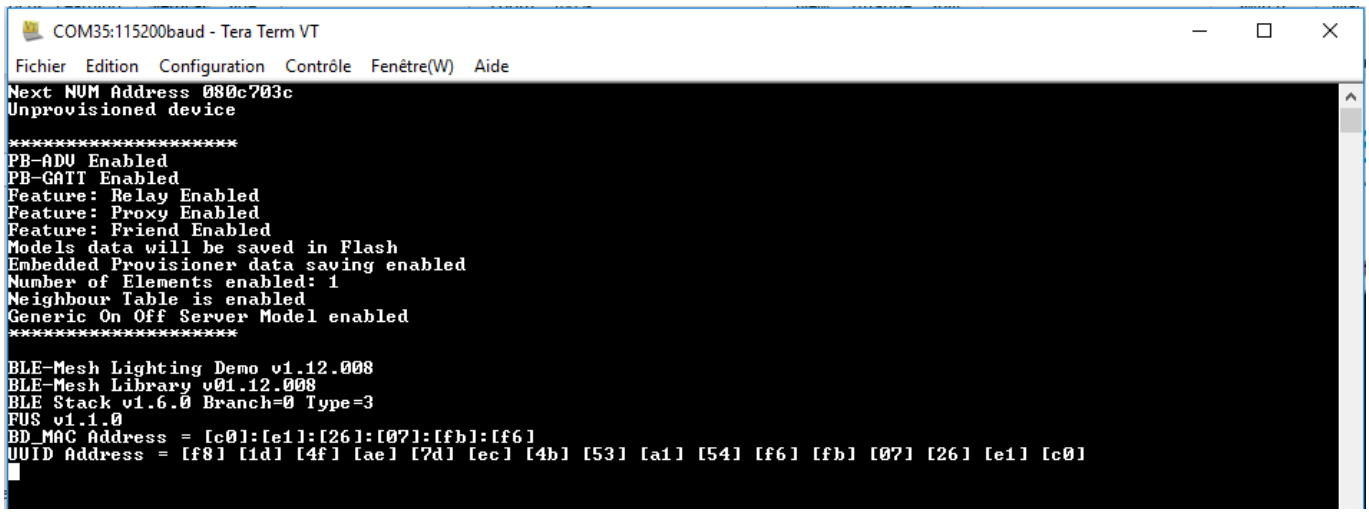
Using the firmware `mesh.c` file, certain messages are be printed to the VCOM using the following code:

```
/* Prints the MAC Address of the board */
TRACE_I(TF_INIT,"BLE-mesh Lighting Demo v%s\n\r",
        BLE_MESH_APPLICATION_VERSION);
TRACE_I(TF_INIT,"BLE-mesh Library v%s\n\r",
        BLEMesh_GetLibraryVersion());
if (SHCI_GetWirelessFwInfo(p_wireless_info) != SHCI_Success)
{
        // Error
}
else
{
  TRACE_I(TF_INIT,"BLE Stack v%d.%d.%d Branch=%d
        Type=%d\n\r",
        p_wireless_info->VersionMajor,
        p_wireless_info->VersionMinor,
        p_wireless_info->VersionSub,
        p_wireless_info->VersionBranch,
        p_wireless_info->VersionReleaseType);
  TRACE_I(TF_INIT,"FUS v%d.%d.%d\n\r",
        p_wireless_info->FusVersionMajor,
        p_wireless_info->FusVersionMinor,
        p_wireless_info->FusVersionSub);
}
TRACE_I(TF_INIT,"BD_MAC Address =[%02x]:[%02x]:[%02x]:[%02x]:[%02x]:[%02x] \n\r",
        bdaddr[5],bdaddr[4],bdaddr[3],bdaddr[2],bdaddr[1],bdaddr[0]);
TRACE_I(TF_INIT,"UUID Address = ");
for(MOBLEUINT8 i=0;i<16;i++)
{
  TRACE_I(TF_INIT,"[%02x] ",uuid[i]);
}
TRACE_I(TF_INIT,"\r\n");
```

Once the board is connected and the terminal window is opened, press the reset button. The following messages are printed to the virtual com window when the firmware starts successfully:

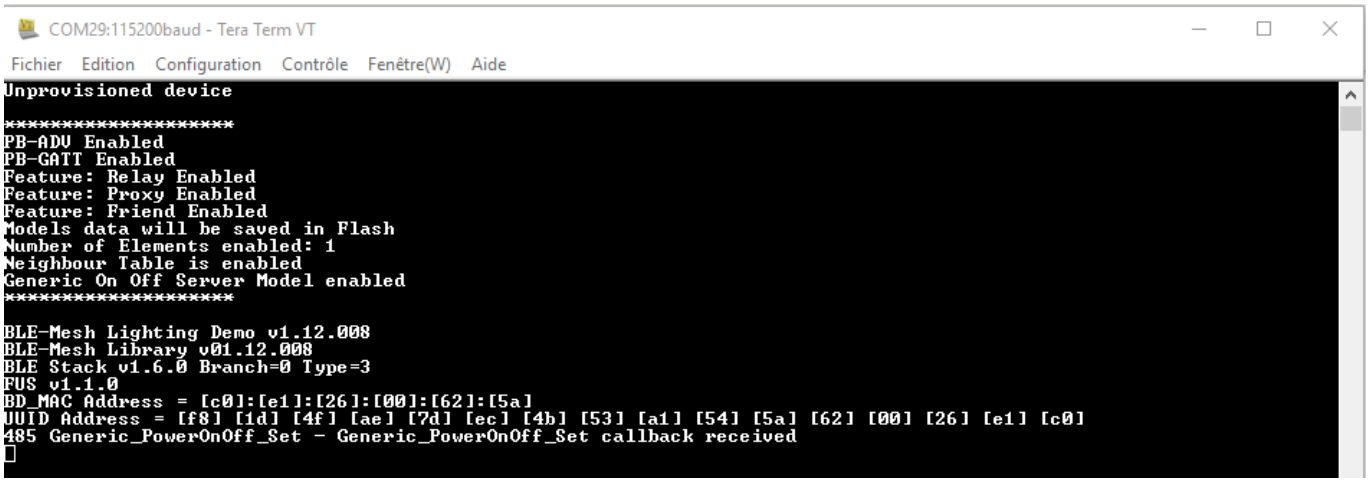
- For a provisioner node see Figure 26

Figure 26. BLE_MeshLightingProvisioner VCOM window



- For a proxy, relay, and friend nodes see Figure 27

Figure 27. BLE_MeshLightingPRFNode VCOM window



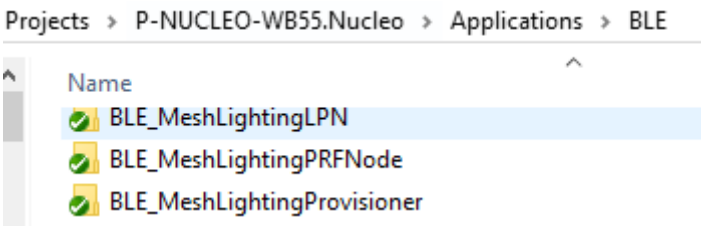
- For a low-power node, the VCOM traces are disabled for to reduce power consumption.

3.4 Build and install the BLE mesh application

To build and install a BLE mesh application, follow the process detailed below:

1. Download and unpack the release package
2. Select the desired platform, here: NUCLEO-WB55
3. Select the application project type illustrated in the figure below

Figure 28. Project type list



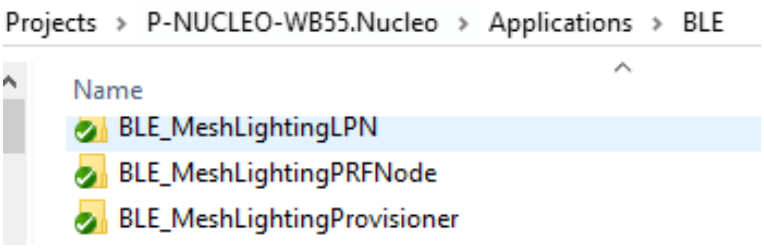
4. Open the selected project in IDE, for example Project.eww for IAR Systems®
5. Build the application by selecting Project → Make option in the menu. If the project needs to be changed, modify as per requirements and re-build
6. Flash the board using the download and debug button, and run the code
7. Install the Android™/iOS™ ST BLE Mesh application and launch it to play with the demo.

3.5 Installing BLE mesh application from Cube Programmer

To install a project, follow the steps below:

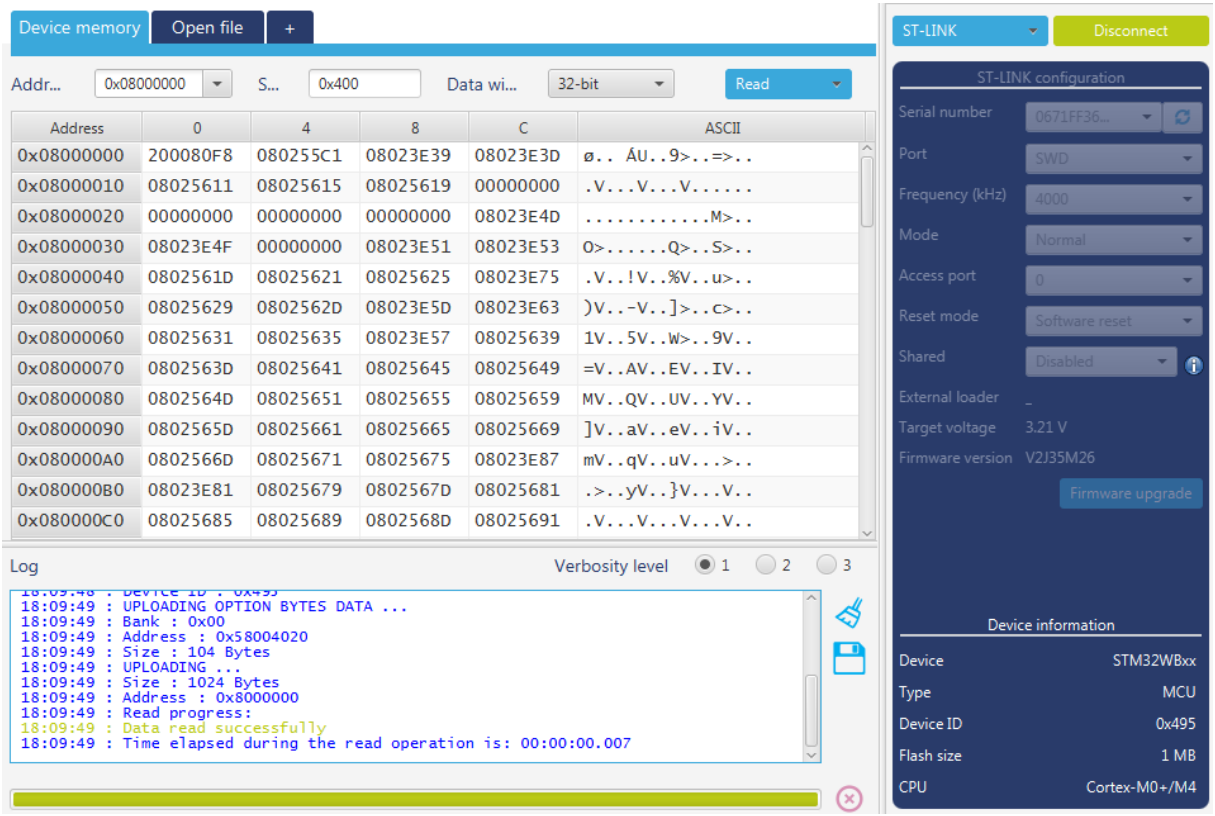
1. Select the application project type as illustrated in Figure 29

Figure 29. Sample applications



2. Find the corresponding project binary, built with instructions from Section 3.4 , with the following path name: `[BLE_projectName]\EWARM\[projectType]\Exe`
3. Launch STM32 Cube Programmer and the activity can be monitored as illustrated in Figure 30.

Figure 30. STM32CubeProgrammer interface

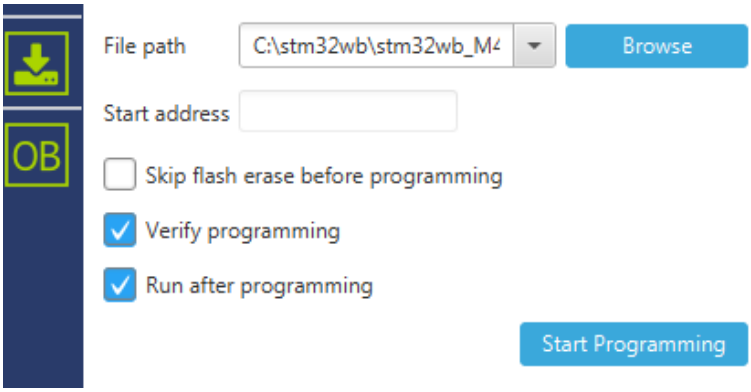


   – Select "STLINK" connectivity
   – Click on connect button

4. Click on the "Erasing and programming" button

5. Select the appropriate binary and check the options illustrated in Figure 31.

Figure 31. Binary programming options



6. Start programming.
7. Install either the Android™ or iOS™ ST BLE Mesh application and launch the application to play with the demo.

4 STM32WB BLE mesh architecture

The STM35WBx5 Bluetooth® Low Energy architecture separates, the Bluetooth® Low Energy profiles and the application run on the Cortex®-M4, from the Bluetooth® Low Energy real-time aspects residing in the Bluetooth® Low Energy peripheral.

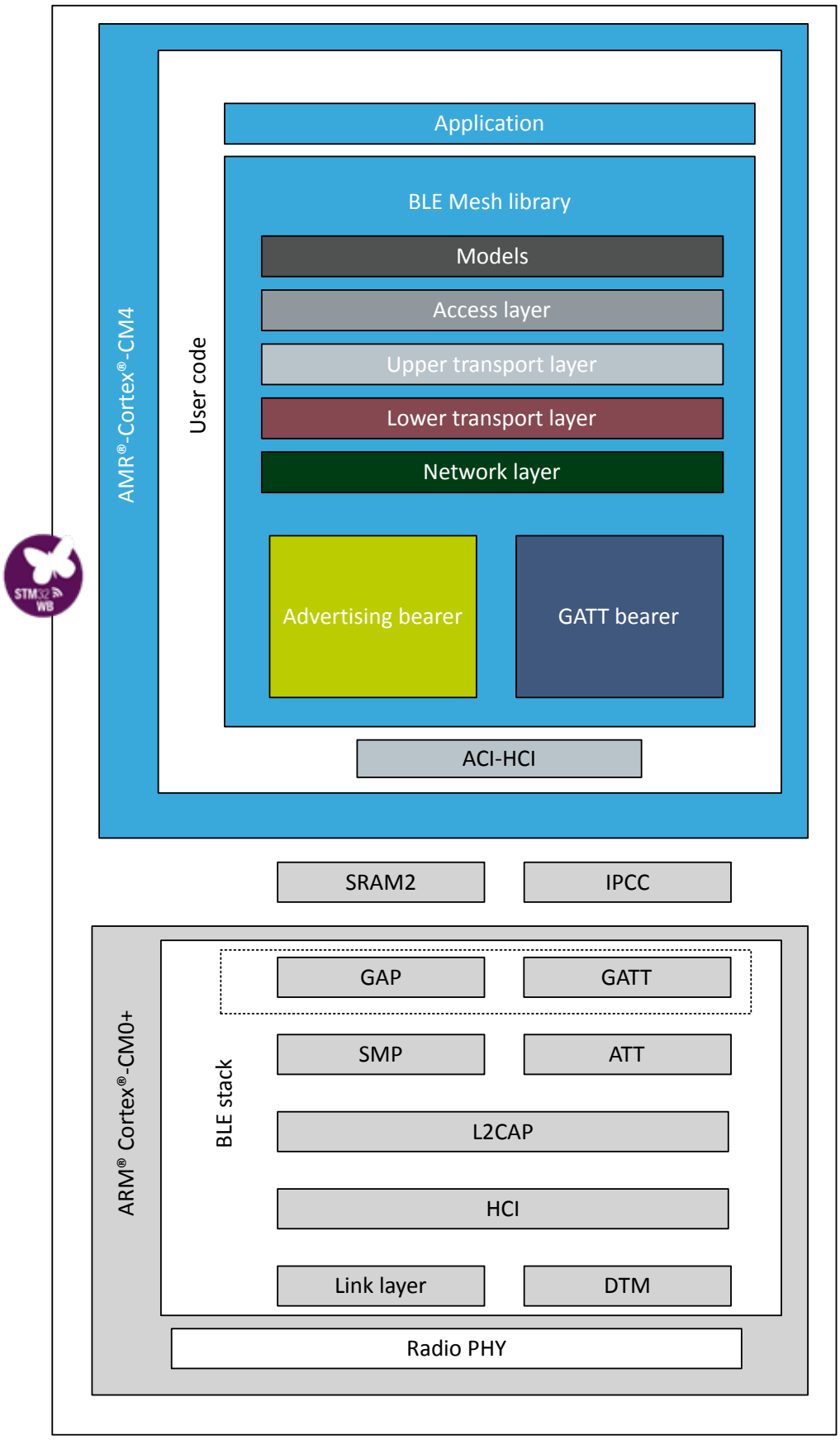
The Bluetooth® Low Energy peripheral incorporates a Cortex®-M0+ processor containing the Bluetooth® Low Energy stack handling the link layer up to the generic attribute profile and generic access profile layers. It also incorporates the physical 2.4 GHz radio.

The Cortex®-M4 application processor collects and computes the data to be transferred over BLE.

The Cortex®-M0 contains the LE controller and LE host required to manage all real time link layer and radio PHY interaction.

The overall architecture is illustrated in Figure 32.

Figure 32. **STM32WB Bluetooth® Low Energy Architecture**



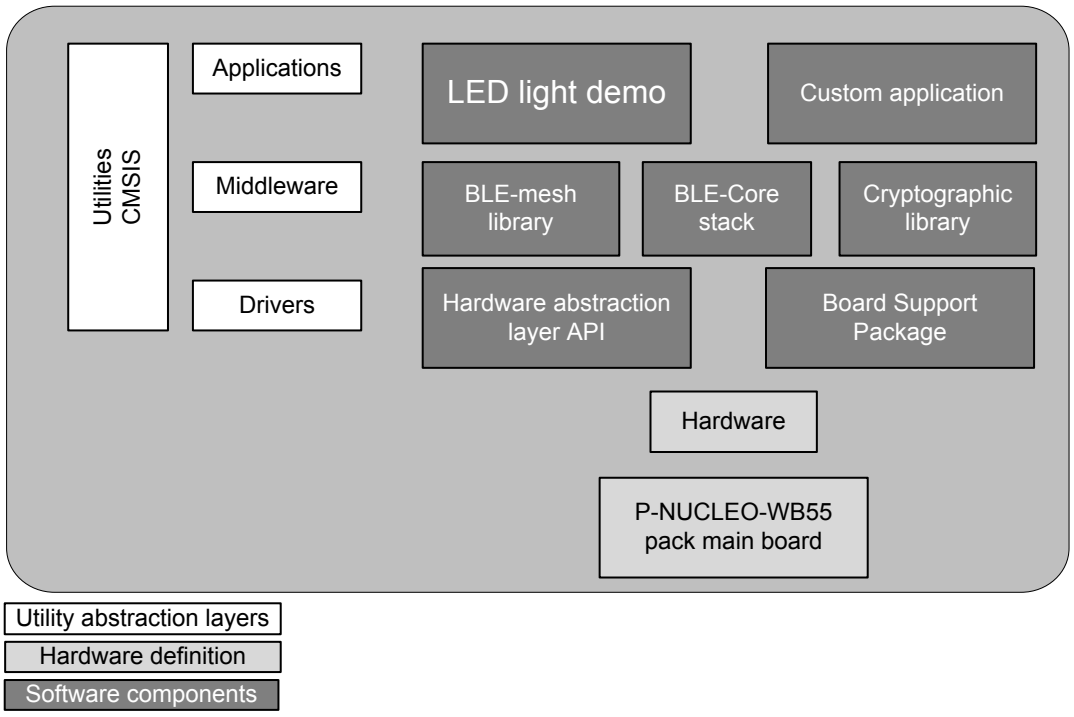
The BLE mesh library is delivered as a binary and can be considered as a mesh profile with mesh models using the ACI-HCI layer to interact with the BLE stack.

The BLE mesh profile defines fundamental requirements to enable an interoperable BLE mesh networking solution for Bluetooth® Low Energy wireless technology.

BLE mesh models defines models (along with the associated states and messages) that are used to define basic functionality of nodes on a mesh network.

BLE mesh applications can be implemented over the BLE mesh library and the BLE mesh middleware. The firmware architecture is illustrated in the figure below.

Figure 33. Firmware architecture



4.1 Stack features

The stack feature definition is given in Table 4 and the feature values are given in Table 5

Table 4. Feature definition

Feature	Decription
BLE mesh version	1.0
Node types	Relay
	Proxy
	Friend
	Low-power node (LPN)
	Embedded provisioner
GATT services	PB-ADV
	PB-GATT
Security	ECDH based provisioning of devices using P-256 elliptic curve
	AES-CCM authentication in network and application layers using separate keys AES-128

Table 5. **Feature values**

Feature	Maximum value
Key	
Network key number	3
Application key number	3
Models	
Bluetooth® SIG number	7
VENDOR number	1
Element number	8
Security	
ECC private key length	32
ECC public key length	64
ECC secret key length	32
Mesh Network ID length	8

4.2 STM32WB BLE mesh application – middleware application

4.2.1 BLE mesh configuration header file

Each project in the `STM32_WPAN/app` subfolder contains a mesh application configuration header file called `mesh_cfg_usr.h`. This file is used to set up the features and other properties related to the node.

- Proxy, relay and friend features are set up with the definition files listed in Table 6

Table 6. **Proxy, relay and friend features definition header files**

Header file	Definition
`#define ENABLE_RELAY_FEATURE`	Defines the capacity to relay the mesh messages.
`#define ENABLE_PROXY_FEATURE`	Defines the capacity to be proxy connected with a mesh smartphone application.
`#define ENABLE_FRIEND_FEATURE`	Defines the capacity to established a friendship withLow-power nodes.
`#define ENABLE_LOW_POWER_FEATURE`	Defines the low-power feature for the low-power node. This feature does not support other features (proxy, relay or friend)

- Macros used in user defined serial print data for models without low-power features enabled, by default the following traces are enabled:

Table 7. Trace macro file headers

File headers	Definition
`#define TF_GENERIC`	Related to Generic Server models.
`#define TF_GENERIC_CLIENT`	Related to Generic Client models.
`#define TF_SENSOR`	Related to Sensor Server models.
`#define TF_LIGHT`	Related to Light Server models.
`#define TF_LIGHT_CLIENT`	Related to Light Client models.
`#define TF_LIGHT_LC`	Related to Light Control Server models.
`#define TF_VENDOR`	Related to Vendor ServerVendor Server models.
`#define TF_CONFIG_CLIENT`	Related to the Configuration Client model used by provisioner node.
`#define TF_LPN_FRND`	Related to Low-power node and Friend node management.
`#define TF_PROVISION`	Related to provisioning process.
`#define TF_HANDLER`	Related to interrupt handler.
`#define TF_INIT`	Related to node initialization.
`#define TF_MISC`	Related to others.
By default the following traces are disabled:	
`#define TF_COMMON`	Related to common (middleware).
`#define TF_GENERIC_M`	Related to Generic Server (middleware).
`#define TF_GENERIC_CLIENT_M`	Related to Generic Client (middleware).
`#define TF_SENSOR_M`	Related to Sensor Server (middleware).
`#define TF_LIGHT_M`	Related to Light Server (middleware).
`#define TF_LIGHT_CLIENT_M`	Related to Light Client (middleware).
`#define TF_LIGHT_LC_M`	Related to Light Light Control Server (middleware).
`#define TF_VENDOR_M`	Related to Vendor Server (middleware).
`#define TF_CONFIG_CLIENT_M`	Related to Configuration Client (middleware).
`#define TF_NEIGHBOUR`	Related to neighbor table management (middleware).
`#define TF_MEMORY`	Related to memory management with mesh library (middleware).
`#define TF_BEACON`	Related to user beacon.
`#define TF_SERIAL_CTRL`	Related to serial control commands.

- Definition of the device name during provisioning.
- Model enabling macros are defined by bitmap such as the following:
  - Bit N - 1 is dedicated to the element N...
  - Bit 2 is dedicated to the element 3
  - Bit 1 is dedicated to the element 2
  - Bit 0 is dedicated to the element 1.

  Where N is the APPLICATION_NUMBER _OF_ELEMENTS.
- For example to enable the Generic ON/OFF Server model on element 2 and 3: `#define ENABLE_GENERIC_MODEL_SERVER_ONOFF (6).`

- Macros for the number of sensors present
- Macros for the sensor occupancy presence, related to Light LC model
- Macros for the sensor publication
- Macros for the pulse width modulation time period, used for external LED control.
- `#define ENABLE_SAVE_MODEL_STATE_NVM` to save the model states in Flash memory.
- `#define SAVE_MODEL_STATE_FOR_ALL_MESSAGES` to save the models states in Flash memory for all messages (if enabled, `SAVE_MODEL_STATE_POWER_FAILURE_DETECTION` must be disabled).
- `#define SAVE_MODEL_STATE_POWER_FAILURE_DETECTION`to save model states in Flash memory on power failure detection (if enabled `SAVE_MODEL_STATE_FOR_ALL_MESSAGES` must be disabled).
- Macros for the numbers of bytes dedicated for generic model and light model saved in Flash memory.
- Macros for the selection of the number of LEDs and type of lighting used for the application depending on the following conditions:
  - ```
    if
        the board has one LED,
        remove the comment tags on USER_BOARD_1LED.
    ```
  - ```
    if
        the board has cool warm LEDs,
        remove the comment tags on USER_BOARD_COOL_WHITE_LED.
    ```
  - ```
    if
        the board has RGB LEDs,
        remove the comment tags on USER_BOARD_RGB_LED.
    ```
  - ```
    if
        the board has RGB LEDs as well as cool warm LEDs,
        remove the comment tags on USER_BOARD_COOL_WHITE_LED and USER_BOARD_RGB_LED.
    ```

Note: The P-NUCLEO-WB55 pack board has one LED so the comment tags on `USER_BOARD_1LED` are removed by default.

- Macros for pulse width modulation support for external LED control.
- Macros to enable the serial user interface using UART.
- `#define APPLICATION_NUMBER_OF_ELEMENTS` macros to set the number of element, by default set to 1.
- `#define USER_SIG_MODELS_MAX_COUNT` macros to set the maximum number of models per element, by default set to 7.
- `#define USER_VENDOR_MODELS_MAX_COUNT` macros to set hte maximum number of Vendor models per element, by default set to 1.
- Macros to set the 16-bit: company identifier assigned by the Bluetooth® SIG, vendor assigned product identifier, vendor assigned product version.
- Macros for the maximum application packet size set to 160.
- Macros for the definition of the MAC address: static random, external, internal unique number.
- Macros for a minimum gap between successive transmissions, varies from 10 to 65535 (set to 50).
- Macros for the provision bearer supported by the BLE mesh: PB ADV and or PB GATT.
- Macros for out of band features.
- Macros for the number of Low-power nodes that can be associated with Friend node varies from 1 to 10 (set to 6).
- Macros for prioritizing friendship offer with good RSSI link, varies from 0 to 3 Ref @Mesh_v1.0 (set to 1):
  - 0 → 1
  - 1 → 1.5
  - 2 → 2
  - 3 → 2.5.

- Macros for prioritizing friendship offer with good receive window factor, varies from 0 to 3 Ref @Mesh_v1.0 (set to 1):
  - 0 → 1
  - 1 → 1.5
  - 2 → 2
  - 3 → 2.5.
- Macros for the minimum packets queue size required by Low-power node, varies from 1 to 7 Ref @Mesh_v1.0 (set to 2):
  - 1 → 2
  - 2 → 4
  - 3 → 8
  - 4 → 16
  - 5 → 32
  - 6 → 64
  - 7 → 128.
- Macros for reception delay for Low-power node, varies from 0x0A to 0xFF Ref @Mesh_v1.0 (set to 150):
  - 0x0A → 10ms
  - 0xFF → 255ms.
- Macros for poll timeout value after which friendship cease to exist (unit 100ms), varies from 0x00000A to 0x34BBFF Ref @Mesh_v1.0 (set to 2000):
  - 0x00000A → 1 second
  - 0x34BBFF → 96 hours.
- Macros for the maximum receive window size acceptable to Low-power node (unit ms) varies from 0x0A to 0xFF Ref @Mesh_v1.0 (set to 255):
  - 0x0A → 10ms
  - 0xFF → 255ms.
- Macros to define the minimum friend subscription list size capability required by Low-power node, varies from 1 to 5 Ref @Mesh_v1.0 (set to 2).
- Macros for the frequency at which Low-power node would poll for a Friend node (unit 100ms) should be less than poll timeout 100 → 10s (set to 25).
- Macros for the minimum RSSI required by Low-power node should be less than equal to -60 (set to -100).
- Macros defining the number of retries to be made by Low-power node before terminating a friendship, varies from 2 to 10 (set to 10).
- Macros to enable or disable neighbor table. If defined as enabled, and the neighbor MAC address changes the neighbor appears as new neighbor. The neighbor table update can be triggered (configurable) in the following cases:
  - On receiving an unprovisioned device beacon
  - On receiving a secure network beacon
  - On receiving a mesh message: message with any TTL, message with TTL = 0.
- Macros defining the size of the neighbor table (set to 5)
- Macros for the duration in seconds (wrt appear or last refresh) defining which neighbor exists in the neighbor table (set to 20)
- Macros to enable/disable the neighbor table update with unprovisioned device beacon (set to 1)
  - 0: Disable neighbor table update with unprovisioned device beacon
  - 1: Enable neighbor table update with unprovisioned device beacon.
- Macros to enable/disable neighbor table updates with secure network beacon (set to 0):
  - 0: Disable neighbor table update with secure network beacon
  - 1: Enable neighbor table update with secure network beacon.
- Macros to enable/disable neighbor table update with TTL 0 message:
  - 0: Disable neighbor table update messages
  - 1: Enable neighbor table update with messages with 0 TTL
  - 2: Enable neighbor table update with messages with any TTL.

- Macros for pulse width modulation configuration for the P-NUCLEO-WB55 pack boards
- Macros set according the models enabled (do not modify).

4.2.2 BLE mesh middleware

In `Middlewares/ST/STM32_WPAN/ble`, the `svc` subfolders includes the BLE services and `mesh/MeshModel` for BLE mesh models. These are listed in Table 8.

Warning: *Do not modify the files in the `svc` folder*

Table 8. BLE services and mesh model files

File	Description
`svc_ctl.c`	Initializes the BLE stack and manages the application GATT service events.
`mesh.c`	Initializes the BLE mesh library where the mesh service and the mesh characteristics are created and managed, and includes: • Updating the characteristics • Receiving the notifications or write commands • Making the link between the BLE stack and the applicative part.
`common.c`	Includes functions used by mesh models middleware.
`config_client.c`	Model client middleware configuration file, used by provisioner to provision and configure a mesh node.
`generic.c`	Manages all the Bluetooth® SIG mesh Generic Server model states and messages coming from the BLE mesh library. It defines a number of generic states, messages and models that are explicitly defined as functionally non-specific. For example, many devices can be turned on and off regardless of whether they are, for example: a fan, an air conditioning unit, a light, or a power socket. All of the devices support the Generic ON/OFF states, messages, and models instead of having separate ON/OFF states, messages, and models for each type of device.
`generic_client.c`	Manages all the Bluetooth® SIG MESH Generic Client models states and messages for the BLE mesh library.
`light.c`	Manages all the Bluetooth® SIG mesh Light Server model states and messages coming from the BLE mesh library. It defines a set of functionalities related to lighting control. This includes lights which can be dimmed as well as color changing lights which can be tuned.
`light_client.c`	Manages all the Bluetooth® SIG mesh Lighting Client model states and messages for the BLE mesh library.
`light_lc.c`	Manages all the Bluetooth® SIG mesh Light Control Server model states and messages coming from the BLE mesh library. This light control model manages specific behaviors triggered by sensors, such as turning lights "on" or "off" based on presence or balancing a light level based on ambient light conditions, dimming lights after a period of inactivity and eventually turning lights "off".
`sensors.c`	Manages all the Bluetooth® SIG mesh Sensor Server model states and messages coming from the BLE mesh library. It defines a standard way of interfacing with sensors. It supports the set of sensors from any device without needing specific states, messages, and models defined for each application area.

File	Description
vendor.c	Manages all the Vendor Server model states and messages coming from the BLE mesh library. Models may be defined and adopted by Bluetooth SIG and may be defined by vendors. Models defined by Bluetooth® SIG are known as SIG adopted models, and models defined by vendors are known as Vendor models.
time_scene.c	Manages all the Bluetooth® SIG mesh Time and Scenes Server model states and messages coming from the BLE mesh library. It defines a set of functionalities related to time and saved states on devices. This allows any device to have a concept of time and execute a defined scene at a given time. Scenes are the stored states of a device that can be recalled using messages or at a given time. This model is still under development.
mesh_cfg.c	Initializes all the data structures associated to the models being used.

4.2.3 BLE mesh application

Three BLE mesh projects are available to create specific BLE applications. The code is located in the BLE folder as detailed in the list below:

- BLE_MeshLightingLPN
- BLE_MeshLightingPRFNode
- BLE_MeshLightingProvisioner.

A sample project is illustrated in Figure 34.

Each project contains a Core subfolder which is composed of the files listed in Table 9.

Table 9. Core subfolder files

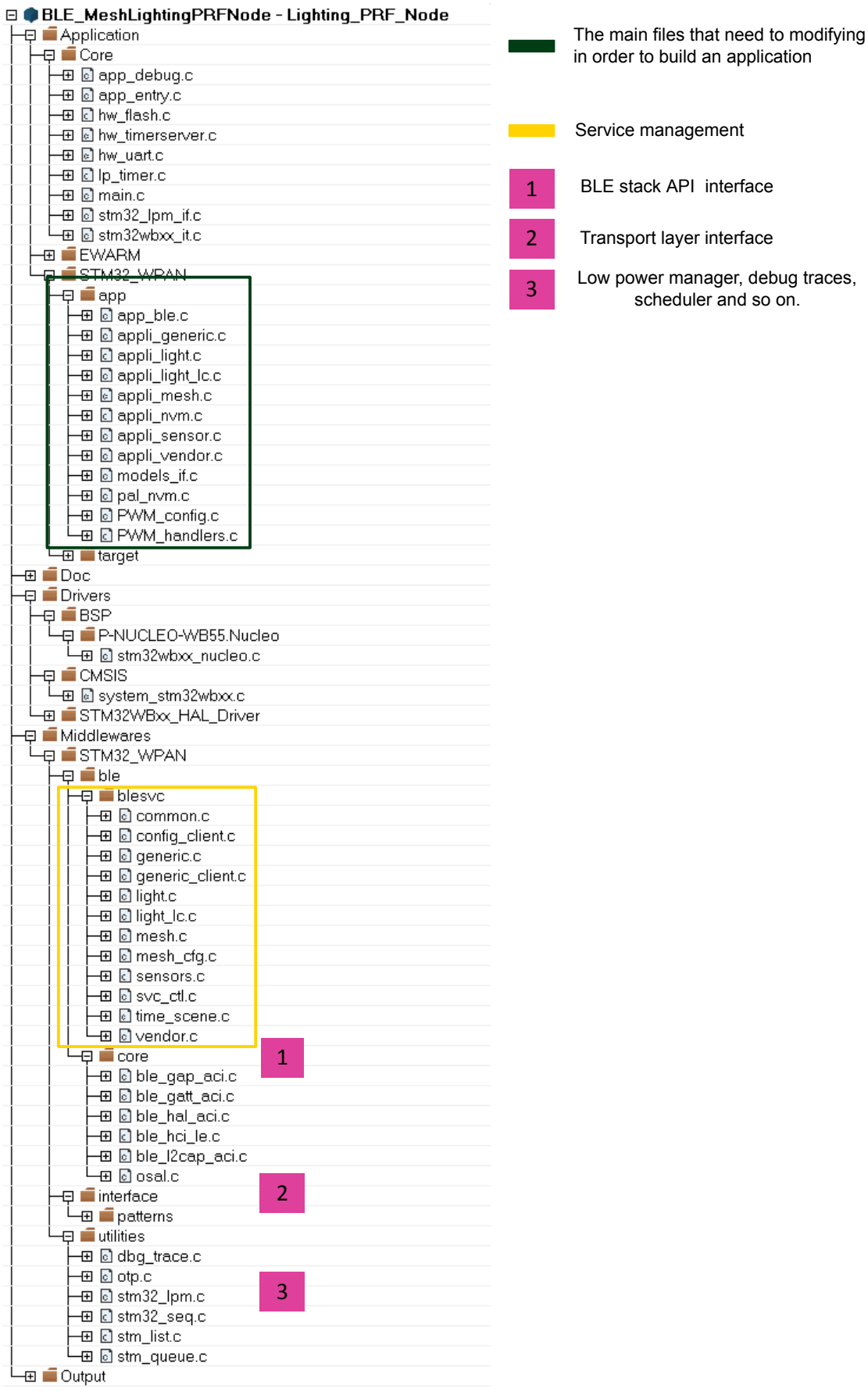
File	Description
app_debug.c	To manage the debug capabilities.
app_entry.c	To initialize the BLE transport layer and the BSP such as LEDs, buttons and so on.
hw_flash.c	Contains the Flash driver.
hw_timerserver.c	Contains the hardware timer server source file for application timers.
hw_uart.c	Initializes and manages the UART and low power UART.
lp_timer.c	Initializes the low power timer.
main.c	Initializes and starts the STM32WB platform.
stm32_lpm_if.c	Initializes the layer function which manages the change between the Low-power modes such as Stop, Sleep and so on.
stm32wbxx_it.c	Contains the main interrupt service routines. This file provides the template for all exception handlers and peripheral interrupt service routines.
system_stm32wbxx.c	Contains the CMSIS Cortex® device peripheral access layer system source file.

Each project contains a STM32_WPAN/app subfolder contains the files in Table 10.

Table 10. STM32_WPAN/App subfolder files

Files	Description
app_ble.c	Initializes the BLE application, manages the GAP and the connection such as the advertise, the scan and so on.
appli_config_client.c	Application interface for Configuration Client Mesh model.
appli_generic.c	Generic model applications: such as turning "on" and "off" a node, regardless of whether it is for example: a fan, an air conditioning unit, a light, or a power socket.
appli_light.c	Light model applications: such as dimming lights as well as tunable and color changing lights.
appli_light_client.c	Light model client application (like sending command to light dimming as well as tunable and color changing lights).
appli_light_lc.c	Light Control model applications to produce specific behaviors triggered by sensors as previously described.
appli_mesh.c	Manages all the mesh library callback functions, and the user button associated actions.
appli_nvm.c	Manages all the Flash memory accesses for saving all the provision data, model application data and provisioner data for the provisioner feature.
appli_sensor.c	Sensor model application.
appli_vendor.c	Vendor model application.
models_if.c	Manages the interface with the model and schedules all the associated processes.
pal_nvm.c	Defines the set of functions used in the appli_nvm.c file.
PWM_config.c	Gives a set of functions that can be used to configure external RGB LEDs using pulse width modulation.
PWM_handlers.c	Gives a set of functions that are used to manage external RGB LEDs using pulse width modulation.

Figure 34. **STM32Cube_FW_WB BLE mesh-LightingPRFNode project structure**



4.2.4 BLE mesh middleware by file

Generic Server model functionalities

The `generic.c` file contains a set of functions called when Generic Server messages are received for:

- Generic ON/OFF model
- Generic Level model
- Generic Power ON/OFF model
- Generic Default Transition Time Server model.

The Generic Server model file location is illustrated in the figure below and defined in Table 11.

Figure 35. Generic Server model code

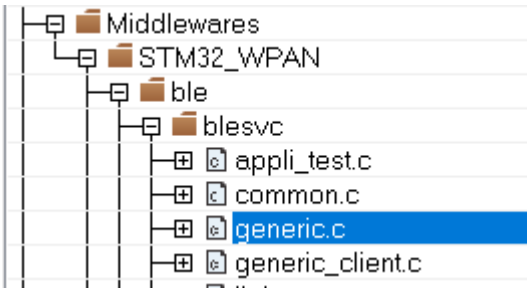


Table 11. Generic Server model code functions

Function	Description
`Generic_OnOff_Set()`	Calls for both acknowledged and unacknowledged Generic ON/OFF set messages.
`Generic_OnOff_Status()`	Calls for Generic ON/OFF status message.
`Generic_Level_Set()`	Calls for both acknowledged and unacknowledged Generic Level Set Ack messages.
`Generic_Delta_Set()`	Calls for both acknowledged and unacknowledged Generic Level Delta Set messages.
`Generic_Move_Set()`	Calls for both acknowledged and unacknowledged Generic Move Set messages.
`Generic_Level_Status()`	Calls for Generic Level Status message.
`Generic_PowerOnOff_Set()`	Calls for both acknowledged and unacknowledged Generic Power ON/OFF Set messages.
`Generic_PowerOnOff_Status()`	Calls for Generic Power ON/OFF status message.
`Generic_DefaultTransitionTime_Set()`	Calls for both acknowledged and unacknowledged Generic Default Transition Time Set messages.
`Generic_DefaultTransitionTime_Status()`	Calls for Generic Default Transition Time status message.
`GenericModelServer_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`GenericModelServer_GetStatusRequestCb()`	Callback from the library to send response to the message from peer.
`GenericModelServer_ProcessMessageCb()`	Callback from the library whenever a Generic model message is received.
`Generic_TransitionBehaviour()`	Generic Default Transition Time behavior used for the Generic ON/OFF model when the transition time is received in the message.
`Generic_GetStepValue()`	Calculates values for the transition time.

Function	Description
Generic_Process()	Execute the transition state machine for particular generic model.
Generic_Publish()	Publish command for the Generic model used while long press button.
GenericOnOffStateUpdate_Process()	Updates the parameters of the Generic ON/OFF model in application file from the temporary parameter in the model file.
GenericLevelStateUpdate_Process()	Updates the parameters of the Generic Level model in the application file from the temporary parameter in model file.
LightActual_GenericOnOffBinding()	Reverse data binding between the Generic ON/OFF and Light Lightness Actual. This function sets the ON/OFF status of the light when the Light Lightness actual value is set.
LightActual_GenericLevelBinding()	Data binding between the Generic Level and the Light Lightness Actual. This function sets the actual Light Lightness value at the time of Generic Level Set.
Light_CtlTemp_GenericLevelBinding()	Data binding between the Generic Level and the Light CTL temperature. This function sets theGeneric Level value at the time of the CTL temperature value set.
Light_HslHue_GenericLevelBinding()	Data binding between the Generic Level and the Light HSL. This function sets the Generic Level value at the time of setting the HSL hue value.
Light_HslSaturation_GenericLevelBinding()	Data binding between the Generic Level and the Light HSL. This function sets the Generic Level value at the time the HSL saturation value is set.
Light_LC_GenericOnOffBinding()	Reverse data binding between the Generic ON/OFF and the Light LC ON/OFF. This function sets the Generic ON/OFF status of the Light when the Light LC ON/OFF is set.
Generic_OnOffDefaultTransitionValue()	Called in Generic ON/OFF when the default transition time is enabled.
Generic_LevelDefaultTransitionValue()	Called in Generic Level when the default transition time is enabled.
Generic_Client_OnOff_Status()	Called when the Generic ON/OFF Clientstatus of the model is received on the client.
Generic_Client_Level_Status()	Called when the status of the Generic Level Client model is received on the client.
Generic_Client_PowerOnOff_Status()	Called when the status of the Generic Power ON/OFF Client model is received on the client.
Generic_Client_DefaultTransitionTime_Status()	Called when the status of the Generic Default Transition Time Client model is received on the client.
A set of weak functions defined to support the original function if they are not included in firmware. When there is no use of this function for application development purpose.	

Generic Client model functionalities

The `generic_client.c` file contains a set of functions called when Generic Client messages have to be sent for:

- Generic ON/OFF model
- Generic Level model
- Generic Power ON/OFF model
- Generic Default Transition Time Client models.

The file location is illustrated in the figure below and described in Table 14:

Figure 36. **Generic Client model code**

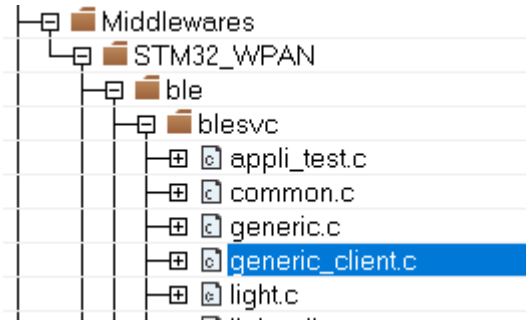


Table 12. **Generic Client model code functions**

Function	Description
`GenericClient_OnOff_Get()`	Calls for message Generic ON/OFF Get.
`GenericClient_OnOff_Set()`	Calls for acknowledged message Generic ON/OFF Set.
`GenericClient_OnOff_Set_Unack()`	Calls for unacknowledged Generic ON/OFF Set message.
`GenericClient_Level_Get()`	Calls for Generic Level Get message.
`GenericClient_Level_Set()`	Calls for Generic Level Set Ack message.
`GenericClient_Level_Set_Unack()`	Calls for Generic Level Set UnAck message.
`GenericClient_Delta_Set()`	Calls for acknowledged Generic Level Delta Set message.
`GenericClient_Delta_Set_Unack()`	Calls for Generic Level Delta Set UnAck message.
`GenericClient_Move_Set()`	Calls for acknowledged Generic Move Set message.
`GenericClient_Move_Set_Unack()`	Calls for unacknowledged Generic Move Set message.
`GenericClient_PowerOnOff_Get()`	Calls for Generic On Power Up Get message.
`GenericClient_PowerOnOff_Set()`	Calls for acknowledged Generic On Power Up Set message.
`GenericClient_PowerOnOff_Set_Unack()`	Calls for Generic Power ON/OFF Set UnAck message.
`GenericClient_DefaultTransitionTime_Get()`	Calls for Generic Default Transition Time Get message.
`GenericClient_DefaultTransitionTime_Set()`	Calls for acknowledged Generic Default Transition Time Set message.
`GenericClient_DefaultTransitionTime_Set_Unack()`	Calls for Generic Default Transition Time Set UnAck message.
`GenericModelClient_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`GenericModelClient_GetStatusRequestCb()`	Callback from the library to send response to the message from peer.
`GenericModelClient_ProcessMessageCb()`	Callback from the library whenever a Generic model message is received.

Light Server model functionalities

The `light.c` file contains a set of functions called when Light Server messages are received.
The file location is illustrated in the figure below:

Figure 37. **Light Server model code**

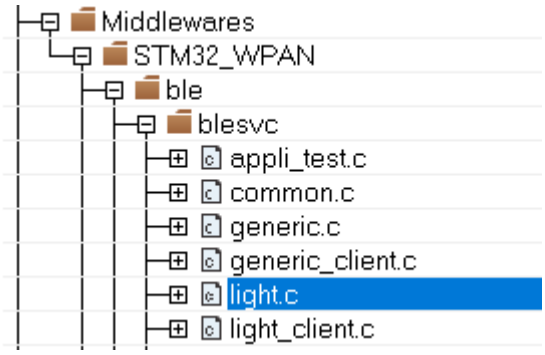


Table 13. **Light Server model code functions**

Function	Description
`Light_Publish_Reset()` `Light_Publish_Add()`	Used for the publication list management.
`Light_Lightness_Set()`	Calls for both acknowledged and unacknowledged Light Lightness messages. The acknowledgment is taken care of by the library.
`Light_Lightness_Status()`	Calls for Light Lightness status message.
`Light_Lightness_Linear_Set()`	Calls for both acknowledged and unacknowledged Light Lightness messages. The acknowledgment is taken care of by the library.
`Light_Lightness_Linear_Status()`	Calls for Light Lightness Linear status message.
`Light_Lightness_Last_Set()`	Calls for both acknowledged and unacknowledged Light Lightness messages. The acknowledgment is taken care of by the library.
`Light_Lightness_Last_Status()`	Calls for Light Lightness Last status message.
`Light_Lightness_Default_Set()`	Calls for both acknowledged and unacknowledged Light Lightness Default Set messages. The acknowledgment is taken care of by the library.
`Light_Lightness_Default_Status()`	Calls for Light Lightness Default status message.
`Light_Lightness_Range_Set()`	Calls for both acknowledged and unacknowledged Light Lightness Range Set messages. The acknowledgment is taken care of by the library.
`Light_Lightness_Range_Status()`	Calls for Light Lightness Range status message.
`Light_Lightness_Ctl_Set()`	Calls for both acknowledged and unacknowledged Light Lightness CTL message. The acknowledgment is taken care of by the library.
`Light_Lightness_Ctl_Status()`	Calls for Light Lightness CTL status message.
`Light_Lightness_CtlTemperature_Set()`	Calls for both acknowledged and unacknowledged Light Lightness CTL Temperature message. The acknowledgment is taken care of by the library.
`Light_Lightness_CtlTemperature_Status()`	Calls for Light Lightness CTL Temperature status message.
`Light_Lightness_CtlTemperature_Range_Set()`	Calls for both acknowledged and unacknowledged Light Lightness CTL Temperature Range message. The acknowledgment is taken care of by the library.
`Light_Lightness_CtlTemperature_Range_Status()`	Calls for Light Lightness CTL Temperature Range status message.

Function	Description
Light_Lightness_CtlDefault_Set()	Calls for both acknowledged and unacknowledged Light Lightness CTL Default Set message. The acknowledgment is taken care of by the library.
Light_Lightness_CtlDefault_Status()	Calls for Light Lightness CTL Default status message.
Light_Lightness_Hsl_Set()	Calls for both acknowledged and unacknowledgedLight Lightness HSL Set message. The acknowledgment is taken care of by the library.
Light_Lightness_Hsl_Status()	Calls for Light Lightness HSL status message.
Light_Lightness_HslHue_Set()	Calls for both acknowledged and unacknowledged Light LightnessHSL Hue Set message. The acknowledgment is taken care of by the library.
Light_Lightness_HslHue_Status()	Calls for Light Lightness HSL Hue Status message.
Light_Lightness_HslSaturation_Set()	Calls for both acknowledged and unacknowledged Light Lightness HSL Saturation Set message. The acknowledgment is taken care of by the library.
Light_Lightness_HslSaturation_Status()	Calls for Light Lightness HSL Saturation status message.
Light_Lightness_HslDefault_Set()	Calls for both acknowledged and unacknowledged Light Lightness HL Default Set message. The acknowledgment is taken care of by the library.
Light_Lightness_HslDefault_Status()	Calls for Light Lightness HSL Default status message.
Light_Lightness_HslRange_Set()	Calls for both acknowledged and unacknowledged Light Lightness HSL Range Set message. The acknowledgement is taken care of by the library.
Light_Lightness_HslRange_Status()	Calls for Light Lightness HSL Range Status message.
Light_Lightness_HslTarget_Status()	Calls for Light Lightness HSL Target Status message.
LightModelServer_GetLightOpcodeTableCb()	Calls from the library to send model opcode table info to the library.
LightModelServer_GetStatusRequestCb()	Calls from the library to send response to the message from peer.
LightModelServer_ProcessMessageCb()	Calls function from the library whenever a light model message is received.
Light_Client_Lightness_Status()	Calls when status of the model received on the client.
Light_Client_Lightness_Linear_Status()	Calls when status of the model received on the client.
Light_Client_Lightness_Last_Status()	Calls when status of the model received on the client.
Light_Client_Lightness_Default_Status()	Calls when status of the model received on the client.
Light_Client_Lightness_Range_Status()	Calls when status of the model received on the client.
Light_Client_Ctl_Status()	Calls when status of the model received on the client.
Light_Client_CtlTemperature_Range_Status()	Calls when status of the model received on the client.
Light_Client_CtlDefault_Status()	Calls when status of the model received on the client.
Light_Client_CtlTemperature_Status()	Calls when status of the model received on the client.
Light_Client_Hsl_Status()	Calls when status of the model received on the client.
Light_Client_HslDefault_Status()	Calls when status of the model received on the client.
Light_Client_HslRange_Status()	Calls when status of the model received on the client.
Light_Client_HslTarget_Status()	Calls when status of the model received on the client.
Light_Client_HslHue_Status()	Calls when status of the model received on the client.

Function	Description
Light_Client_HslSaturation_Status()	Calls when status of the model received on the client.
Light_TransitionBehaviourSingle_Param()	used for the Light Lightness model when transition time is received in the message. This function is used for single parameter transition.
Light_TransitionBehaviourMulti_Param()	used for the Light Lightness model when transition time is received in message. Used for the multiple parameters in transition.
Model_BindingPublishStatus()	Publishes the status of the node to the publish address. The Publishing of all the model states depends on the model message received and supporting binding with other models.
Light_GetStepValue()	Calculates parameters for transition time for light model.
Lighting_Process()	Executes the transition state machine for particular light model.
LightLightnessStateUpdate_Process()	Updates the parameters of the Light Lightness model in the application file from Temporary parameter in model file.
Light_LC_LightnessStateUpdate_Process()	Updates the parameters of the Light Lightness LC model in application file from Temporary parameter in model file.
LightLinearStateUpdate_Process()	Updates the parameters of the Light Lightness linear model in application file from Temporary parameter in model file.
LightCtlStateUpdate_Process()	Updates the parameters of Light CTL model in application file from the temporary parameter in model file.
LightCtlTemperatureStateUpdate_Process()	Updates the parameters of the Light CTL Temperature model in application file from the temporary parameter in model file.
LightHslStateUpdate_Process()	Updates the parameters of the Light HSL model in application file from the temporary parameter in model file.
LightHslHueStateUpdate_Process()	Updates the parameters of the Light HSL hue model in application file from the temporary parameter in model file.
LightHslSaturationStateUpdate_Process()	Updates the parameters of the Light HSL Saturation model in application file from the temporary parameter in model file.
Light_Ctl_LightActual_Binding()	Function for binding the data of the actual lightness and Light CTL Set.
Light_Lightness_Linear_Binding()	Function for binding the data of the actual lightness and linear lightness is implicit binding with Generic ON/OFF state.
Light_Lightness_Binding()	Function for binding the data of the actual lightness and linear lightness is implicit binding with Generic ON/OFF state.
GenericOnOff_LightActualBinding()	Data binding between the Generic ON/OFF and the Light Lightness Actual. This function sets the actual light lightness value at the time of Generic Power ON/OFF Set.
GenericLevel_LightBinding()	Data binding between the Generic Level and the Light Lightness Actual. This function sets the actual light lightness value at the time of Generic Level Set.
Light_Actual_LinearBinding()	Used for binding the data of the actual lightness and linear lightness. This function changes the value of linear lightness as the actual lightness value is set.
Light_Linear_ActualBinding()	Used for binding the data of the actual lightness and linear lightness. This function changes the value of the Actual lightness as Linear lightness value is set.
Light_Actual_RangeBinding()	Used for binding the data of the actual lightness and lightness range this function set the value of the Actual lightness according to the min range and max range value.
Light_CtlTemperature_Binding()	Used for binding the data of the CTL temperature with other models.

Function	Description
`Light_CtlTemperature_TempRangeBinding()`	Used for binding the data of the CTL temperature and CTL temperature range this function changes the value of the CTL temperature according to the min range and max range value.
`GenericLevel_CtlTempBinding()`	Data binding between the Generic Level and the CTL temperature set.
`Light_HslLightness_LightnessActualBinding()`	Data binding between the HSL lightness and the lightness actual. This function sets the lightness actual value at the time of HSL lightness value set.
`Light_HSL_HUE_Binding()`	Used for binding the data of the HSL hue parameter with other models.
`Light_HSL_Saturation_Binding()`	Used for binding the data of the HSL saturation parameter with other models.
`GenericLevel_HslSaturationBinding()`	Data binding between the Generic level and the HSL hue set.
`GenericLevel_HslHueBinding()`	Data binding between the Generic level and the HSL hue set.
`Light_HslHue_RangeBinding()`	Used for binding the data of the HSL hue value and the HSL hue range. This function changes the value of the HSL hue according to the min range and max range value.
`Light_HslSaturation_RangeBinding()`	Used for binding the data of the HSL saturation value and HSL saturation range. This function changes the value of the HSL saturation according to the min range and max range value.
`Light_Linear_Ligth_LC_binding()`	Used for binding the data of the Light_LC_lightness output data with the Light lightness linear data.
`Light_ActualLightness_HslLightnessBinding()`	Data binding between the actual lightness and the HSL lightness . This function sets the HSL lightness value at the time of lightness Actual value set.
`Light_LightnessDefaultTransitionValue()`	Calls in the Light lightness when the default transition time is enabled.
`Light_CTLDefaultTransitionValue()`	Calls in the Light CTL set when the default transition time is enabled.
`Light_CTLTemperatureDefaultTransitionValue()`	Calls in the Light CTL temperature set when the default transition time is enabled.
• A set of weak functions defined to support the original function if they are not included in firmware. When there is no use of this function for application development purpose.	

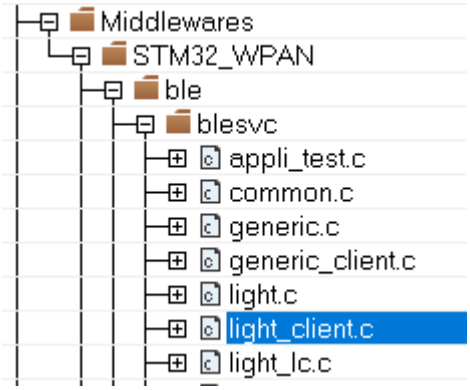
Lighting Client model functionalities

The `light_client.c` file contains a set of functions called when Light Client messages have to be sent for:

- Light Lightness procedure
- Light Lightness Linear procedure
- Light Lightness Last procedure
- Light Lightness Default procedure
- Light Lightness Range Client procedure.
- Light LC procedure (LC stands for light control)

Illustrated in the figure below.

Figure 38. **Lighting Client model code**



For each procedure, there are four call functions:

- `Set`: called for acknowledged Set message
- `Set Unack`: called for unacknowledged Set message
- `Get`: called to send get message
- `Status`: called when status received

The three callback functions are available outlined in Table 14:

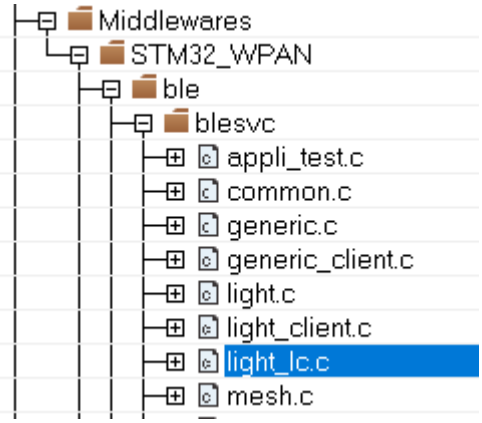
Table 14. **Lighting Client model code functions**

Function	Description
`LightModelClient_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`LightModelClient_GetStatusRequestCb()`	Callback from the library to send response to the message from peer.
`LightModelClient_ProcessMessageCb()`	Callback from the library whenever a light model message is received.

Light Light Control Server (Light LC Server) model functionalities

The `light_lc.c` contains a set of functions called when Light Light Control Server messages are received. The file location is illustrated in the figure below:

Figure 39. **Light LC Server model code**



For each Light Control Mode, Light Control Occupancy, Light Control Light OnOff and Light Control Property messages there are the following functions:

- `Set`: called for both acknowledged and unacknowledged messages.
- `Status`: called for the status messages.

The file also contains the following functions detailed in Table 15:

Table 15. Light LC Server model functions

Function	Description
`Light_LC_SetPropertyID_value()`	Sets the value of the Light Control Property ID.
`Light_LC_GetPropertyID_value()`	Gets the value of the Light Control Property ID.
`Get_LengthInByteProprtyValue()`	Function returns the length of the Light Control Property ID value.
`Light_LC_ModelServer_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`Light_LC_ModelServer_GetStatusRequestCb()`	Callback from the library to send response to the message from peer.
`Light_LC_ModelServer_ProcessMessageCb()`	Callback function from the library whenever a Light LC model message is received.
`Light_LC_Fsm()`	State machine for the Light LC controller, which works on the light on off event and sensor occupancy event. It publishes the light linear set command for the nodes, values are decided according to the controller state.
`Light_LC_TransitionBehaviourSingle_Param()`	Used for the Light LC Lightness model when transition time is received in the message.This function is used for single parameter transition.
`Light_LC_GetStepValue()`	Calculates values for the transition time.
`Get_TimeToWait()`	Calculates the time to wait for any condition
`Light_LC_OnOff_Generic_OnOffBinding()`	Data binding between the light LC on off and the Generic On Off. This function sets the Generic On Off value at the time of the light LC On Off set.
`GenericOnOff_Light_LC_Binding()`	Data binding between Generic On Off and the light LC On Off. This function sets the light LC On Off value at the time of the generic On Off set.
`Light_control_Process()`	Executes the transition state machine for particular Light LC model and state machine Light LC application.
`Light_LC_LuxLevelOutputValue()`	Returns the lightness value from the lux sensors.
`Light_LC_MaxLightnessValue()`	Returns the maximum value after comparison.

Mesh middleware and library initialization

The `mesh.c` file contains two functions. The file location is illustrated in the figure below:

Figure 40. **Mesh middleware and library initialization**

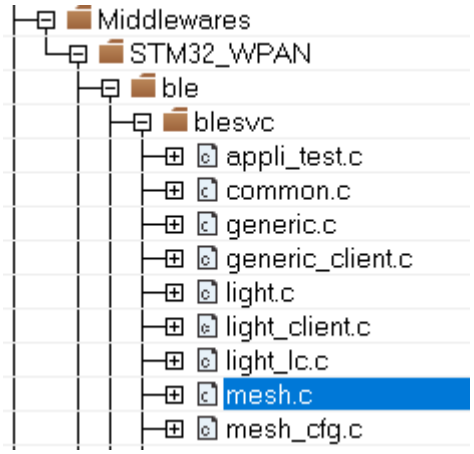


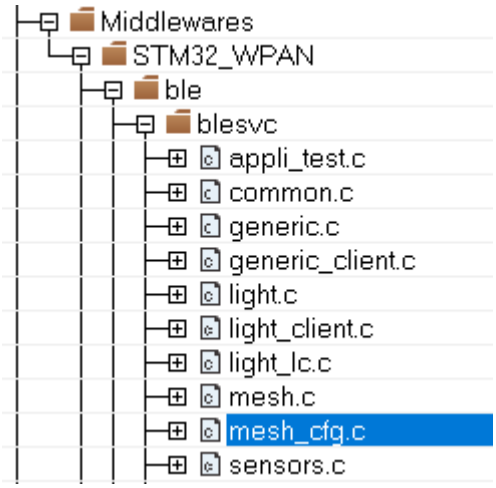
Table 16. **Mesh middleware and library initialization**

Function	Description
`MESH_Init()`	See Section 5.4 Initialization and main application loop for details.
`Appli_ConfigurationInfoDump()`	Dumps the information on the node configuration.

Mesh model configuration

The `mesh_cfg.c` file includes the following files and functions. The file location is illustrated in the figure below:

Figure 41. **Mesh models configuration**



The files that are included dependents on the user requirements:

Table 17. Mesh model configuration included dependent files

Files	Description
serial_if.c	Included for the defines ENABLE_USART and ENABLE_SERIAL_CONTROL or ENABLE_UT or ENABLE_SERIAL_INTERFACE are enabled.
serial_ctrl.c	Included for defines ENABLE_USART and ENABLE_SERIAL_CONTROL are enabled
serial_ut.c	Included for the defines ENABLE_USART and ENABLE_UT are enabled.
appli_test.c	Included for the defines ENABLE_USART and ENABLE_APPLI_TEST are enabled.
serial_prvn.c	Included for the defines ENABLE_USART and ENABLE_SERIAL_PRVN are enabled.

These files are used to send commands between provisioned and configured nodes on an USART external terminal. The command types can be:

- Bluetooth® SIG models (“atcl” commands)
- Read Vendor commands (“atvr” commands)
- Write Vendor commands (“atw” commands)
- Upper Tester commands (“atut” commands), can be used for Bluetooth® SIG Profile Tuning Suite during certification.

The supported functions are:

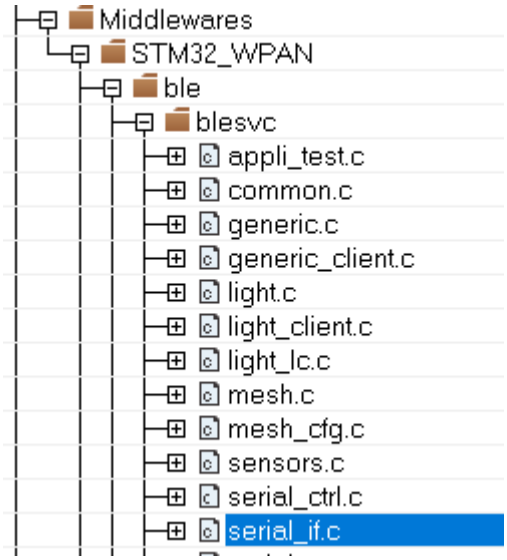
Table 18. Supported models

Models	Description
ApplicationInitVendorModel()	Initializes the vendor model list.
ApplicationInitSigModelList()	Initialize the list of the Bluetooth® SIG models.
ApplicationGetSigModelList()	Provides the list of the Bluetooth® SIG models to the calling function.
ApplicationGetCLIENTSigModelList()	Provides the list of the client Bluetooth® SIG models to the calling function.
ApplicationSetNodeSigModelList()	Sets the list of the client Bluetooth® SIG models to the calling function.
ApplicationGetVendorModelList()	Provides the calling function with the list of the vendor models.
ApplicationChkSigModelActive()	Checks if a specific server model is active in the server model list.
ApplicationChkVendorModelActive()	Checks if a specific model server is active in the vendor model server list.

Serial interface

The `serial_if.c` file includes the following functions. The file location is illustrated in the figure below and described in Table 19:

Figure 42. Serial interface



The file also contains the following functions:

Table 19. Serial interface functions

Function	Defiition
`Serial_RxCpltCallback()`	USART Rx transfer complet callback.
`Serial_Uart_Rx_Task()`	USART Rx task.
`UartReadChar()`	Wait until the receipt of a character from an UART or JTAG, then convert it into ASCII and return the character.
`Serial_GetString()`	Gets the user input from the serial port.
`Serial_InterfaceProcess()`	Processes data coming from serial port.
`Serial_CharToHexConvert()`	Converts ASCII character to hexadecimal number.
`Serial_Init()`	Initializes the Rx from UART.
`BLEMesh_PrintStringCb()`	Callback function to print data serially.
`BLEMesh_PrintDataCb()`	Callback function to print a data array on screen LSB first.

Serial control

The `serial_ctrl.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 20:

Figure 43. Serial control

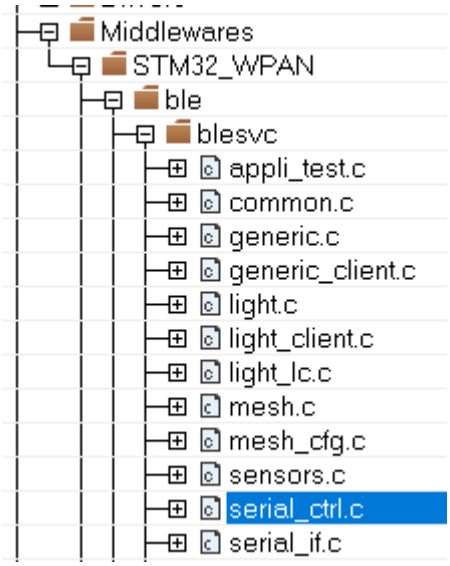


Table 20. Serial control functions

Function	Description
`SerialCtrlVendorRead_Process()`	Parses and sends the user vendor read command.
`SerialCtrlVendorWrite_Process()`	Parses and sends the user Vendor Write command.
`SerialCtrl_Process()`	Parses and sends the Bluetooth® SIG models command.
`SerialCtrl_GetMinParamLength()`	Returns the minimum number of parameters required by a particular opcode.
`SerialCtrl_GetData()`	Extracts the function parameter from the string.

See A.6 for more details.

Serial upper tester

The `serial_ut.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 21.

Figure 44. **Serial upper tester**

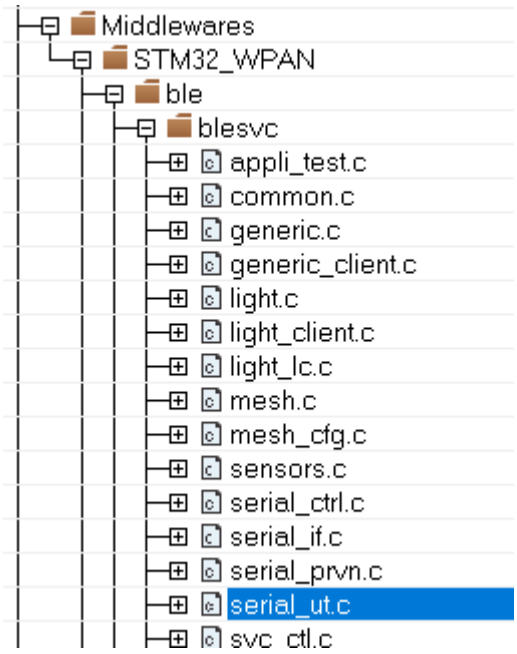


Table 21. **Serial upper tester funcitons**

Function	Description
`SerialUt_Process()`	Extracts the command and variables from the received string and passes it to the mesh library.
`SerialUt_GetFunctionIndex()`	Returns the calculated index of the command received by the user.
`SerialUt_doubleHexToHex()`	Converts two 4-bit hex integers to one 8-bit hex integer.
`BLEMesh_UpperTesterDataProcess()`	For testing purpose, if implemented in the mesh library, the linker would replace weak linking from here.

See B.4 for more details.

Serial provisioner

The `serial_prvn.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 22.

Figure 45. **Serial provisioner**

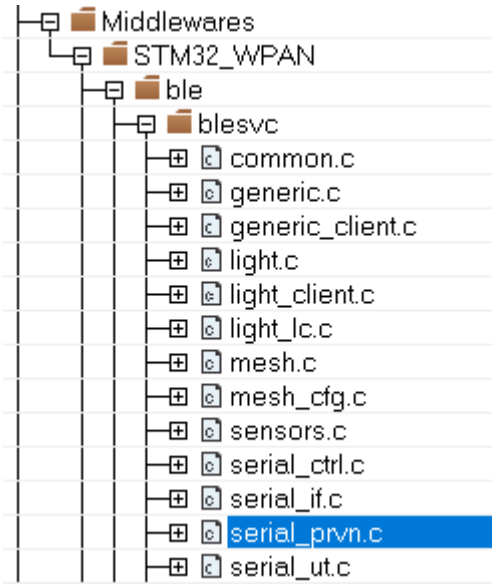


Table 22. **Serial provisioner functions**

Function	Description
`SerialPrvn_Process()`	Used to parse the string given by the user.
`SerialPrvn_ProvisionDevice()`	Returns starts the provisioning of one of the devices.
`SerialPrvn_UnProvisionDevice()`	Un-provisions one of the devices.
`SerialPrvn_ScanDevices()`	Scans and prints unprovisioned devices.
`SerialPrvn_ProvisioningStatusUpdateCb()`	Used to update the status of the provisioning from the application.

Application test

The `appli_test.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 23.

This file contains a set of functions used for robustness tests and timing measurements.

Figure 46. **Application test**

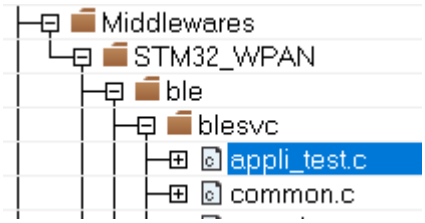


Table 23. **Application test functions**

Function	Description
`SerialResponse_Process()`	Extract the command and variables from the received string and passes it to BLE mesh library.
`SerialResponse_GetFunctionIndex()`	Returns the calculated index of the command received by the user.

Function	Description
SerialResponse_doubleHexToHex()	Converts two 4-bit hex integers to one 8-bit hex integer.
Test_ApplicationTest_Set01()	Calculates the round trip time for the packet send and received response.
Test_ApplicationTest_Set02()	Sends a number of commands to the receiver node and then checks the number of commands received by receiver.
Read_CommandCount()	Reads the number of commands received successfully by the receiver.
Packet_ResponseTimeStamp()	Calculates all the average packet send and receive times.
Test_ApplicationTest_Set03()	Sends the number of bytes and get back the same data in response.
processDelay()	Sets a delay between two commands.
Test_Process()	Runs the testing functions for packet response time.

Sensor Server model functionalities

The sensor.c file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 24.

This file contains a set of functions called when Sensor Server messages are received for the Sensor Server Model.

Figure 47. Sensor Server model

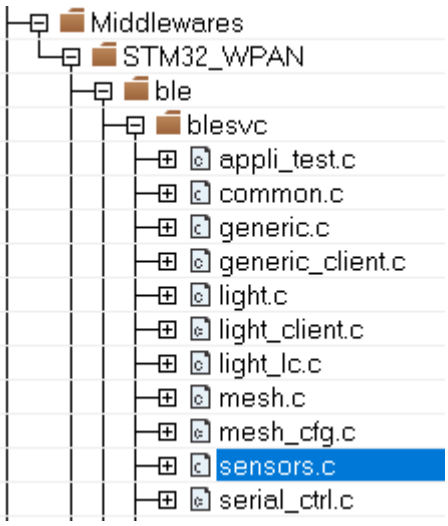


Table 24. Sensor Server model

Function	Description
Sensor_DescriptorGet()	Sensor_DescriptorGet callback, response to be sent only to client node and not to publish group.
Sensor_DescriptorStatus()	Sensor_DescriptorStatus message as per model specification v1.0.1, sent in response of Descriptor Get.
Sensor_CadenceGet()	Sensor_CadenceGet callback, response to be sent only to client node and not to publish group.
Sensor_CadenceSet()	Sensor_CadenceSet callback, response to be sent to both client node and publish group.
Sensor_CadenceSetUnack ()	Sensor_CadenceSetUnack callback, response to be sent to both client node and publish group.

Function	Description
`Sensor_CadenceStatus()`	`Sensor_CadenceStatus` message as per model specification v1.0.1, Sent in response of Cadence Set, Cadence Get or Cadence SetUnack to publish group.
`Sensor_SettingsGet()`	Sensor_SettingsGet callback, response only to be sent to client node and not to pulish group.
`Sensor_SettingsStatus()`	Sensor_SettingsStatus message as per model specification v1.0.1, Sent in response of Settings Get.
`Sensor_SettingGet()`	Sensor_SettingGet callback, response only to be sent to client node and not to publish group
`Sensor_SettingSet()`	Sensor_Setting_Set callback, response only to be sent to both client node and publish group
`Sensor_SettingSetUnack ()`	Sensor_SettingSetUnack callback, response to be sent to both client node and publish group.
`Sensor_SettingStatus()`	Sensor_SettingStatus message as per model specification v1.0.1, Sent in response of Setting Get, Setting Set.
`Sensor_Get()`	Sensor_Get callback, response only to be sent to client node and not to pulish group
`Sensor_Status()`	Sensor_Status message as per model specification v1.0.1, Sent in response of Sensor Get Marshalled sensor data.
`Sensor_ColumnGet()`	Sensor_ColumnGet callback, response only to be sent to client node and not to pulish group
`Sensor_ColumnStatus()`	Sensor_ColumnStatus message as per model specification v1.0.1, Sent in response of Sensor Column Get
`Sensor_SeriesGet()`	Sensor_SeriesGet callback, response only to be sent to client node and not to pulish group
`Sensor_SeriesStatus()`	Sensor_SeriesStatus message as per model specification v1.0.1, sent in response of Sensor Series Get Series data lying between Raw X1 and Raw X2 to be sent.
`SensorModelServer_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`SensorModelServer_GetStatusRequestCb()`	Callback from the library to send the response to the message from peer.
`SensorModelServer_ProcessMessageCb()`	Callback function from the library whenever a sensor model message is received.
A set of weak functions defined to support the original function if they are not included in firmware. When there is no use for this function for application development purpose.	

Vendor Server model functionalities

The `vendor.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 25. Vendor Server model functions.

This file contains a set of functions called when Vendor Server messages are received for the Vendor Server model.

Figure 48. Vendor Server model

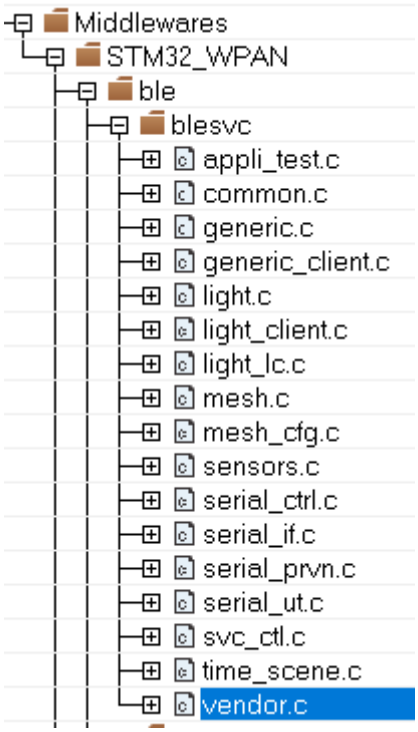


Table 25. Vendor Server model functions

Function	Description
`Vendor_WriteLocalDataCb()`	Callback function called when an action is required on the node itself.
`Vendor_ReadLocalDataCb()`	Callback function called when some data is required from the node.
`Vendor_OnResponseDataCb()`	Callback function called when some data is send by the node to the application.
`Vendor_Process()`	State machine for Vendor model.
`Vendor_Publish()`	Publish command for Vendor model.
`Vendor_TestRemoteData()`	Test command with Vendor model used to calculate the time of packet to reach at destination.
`Vendor_TestCounterInc()`	Test command with Vendor model used to calculate the time of packet to reach at destination.
`VendorModel_PID1_GetOpcodeTableCb()`	Callback from the library to send the model opcode table info to library.
`VendorModel_PID1_GetStatusRequestCb()`	Callback from the library to send a response to the message from peer.
`VendorModel_PID1_ProcessMessageCb()`	Callback function from the library whenever a Generic model message is received.

Time and scene server model functionalities

The `time_scene.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in Table 26. Time and scene server model functions.

This file contains a set of functions called when Time and Scene Server messages are received for the time and scene server model.

Figure 49. Time and scene server model

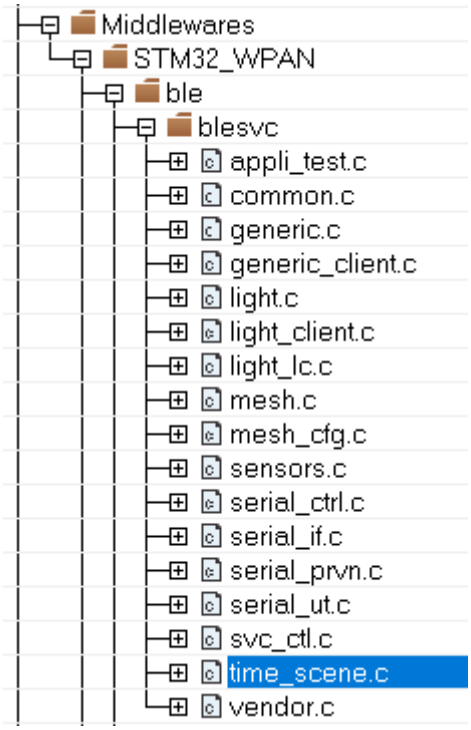


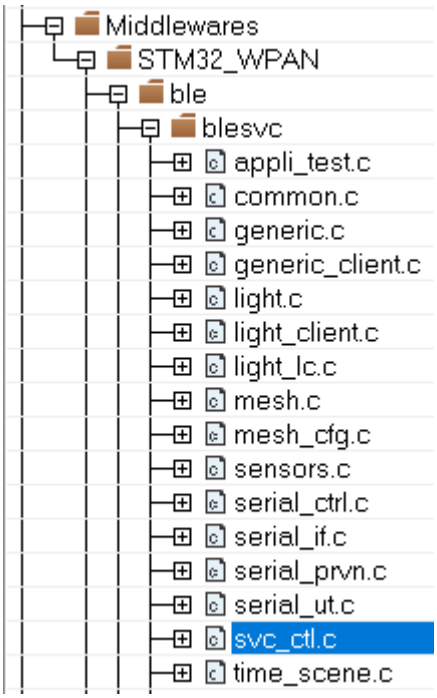
Table 26. Time and scene server model functions

Function	Description
`Time_SceneModelServer_GetOpcodeTableCb()`	Callback from the library to send model opcode table info to library.
`Time_SceneModelServer_GetStatusRequestCb()`	Callback from the library to send response to the message from peer.
`Time_SceneModelServer_ProcessMessageCb()`	Callback function from the library whenever a light model message is received.

Service controller code functionalities

The `svc_ctl.c` file includes the following functions. The file location is illustrated in the figure below and the functions are described in .

Figure 50. Service controller code



The file contains the `SVCCTL_Init()` function. This function calls the sub-functions listed in Table 27.

Table 27. SVCCTL_Init() functions

Function	Description
`mesh_Init()`	BLE mesh library initialization.
`SVCCTL_RegisterSvcHandler()`	registers service event handler. This is a function which receives the GATT events from the `svc_ctl.c` and redirects them to the BLE mesh library
`SVCCTL_RegisterCltHandler()`	registers client event handler(1).

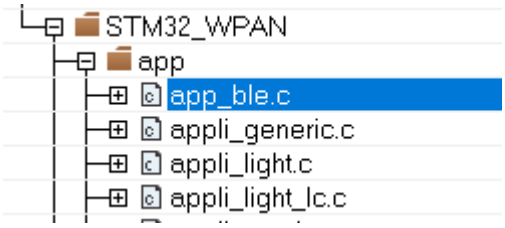
1. *This function is not applicable with mesh projects, mesh client aspects are managed in a different way by the BLE mesh library.*

4.2.5 BLE mesh application by files

Mesh application initialization

The `app_ble.c` file includes the following functions. The file location is illustrated in the figure below.

Figure 51. BLE application code



This file contains the `APP_BLE_Init()` function that initializes the BLE mesh peripheral. This `APP_BLE_Init()` function calls the following sub-functions see Table 28.

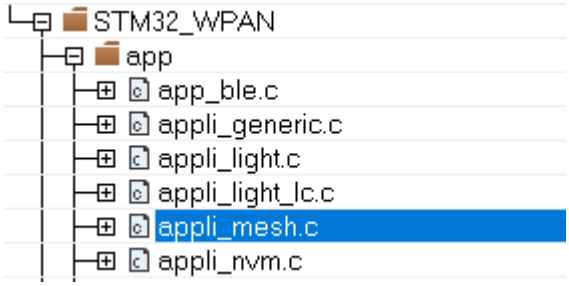
Table 28. APP_BLE_Init() sub-functions called

Function	Description
`SHCI_C2_BLE_Init()`	Initializes the BLE stack on Cortex®CM0+.
`Ble_Hci_Gap_Gatt_Init()`	Initializes the HCI, GATT, and GAP layers.
`SVCCTL_Init()`	Initializes the BLE mesh service in the mesh library.
`Appli_Init ()`	Calls the `HCI_Event_CB()` implemented in the mesh library.

Mesh application control

The `appli_mesh.c` file includes the following functions. The file location is illustrated in the figure below.

Figure 52. Mesh based application code



- `Appli_Init()` initializes the BLE mesh application, Create Timers.

- Receives and reacts to the calls of the mesh library callback functions as define in Table 29.

Table 29. Mesh library callback functions

Function	Description
`Appli_BleStackInitCb()`	Not used.
`Appli_BleStackInitCb()`	Initializes the BLE transmit power.
`Appli_BleSetUUIDCb():`	Sets the BLE mesh UUID value.
`Appli_BleSetProductInfoCB()`	Sets the advertised company product information .
`Appli_BleUnprovisionedIdentifyCb()`	Indicates that an unprovisioned node ID is received.
`Appli_BleGattConnectionCompleteCb()`	Indicates that a GATT connection is established.
`Appli_BleGattDisconnectionCompleteCb()`	Indicates a GATT disconnection.
`Appli_BleSetNumberOfElementsCb()`	Checks the number of element supported per node.
`Appli_BleAttentionTimerCb()`	Indicates that the node is advertising.
`Appli_BleOutputOOBAuthCb()`	Indicates "Out Of Band" information is received.
`Appli_BleInputOOBAuthCb()`	Provides "Out Of Band" information to the BLE mesh library.
`Appli_BleDisableFilterCb()`	Filters the reply status.
`BLEMesh_PbAdvLinkCloseCb()`	Indicates a PB-ADV link closure.
`BLEMesh_PbAdvLinkOpenCb()`	Indicates a open PB-ADV link.
`Appli_LedStateCtrlCb()`	Manages the application blue LED.
`BLEMesh_UnprovisionCallback()`	Indicates the unprovisioning of the node by provisioner.
`BLEMesh_ProvisionCallback()`	Indicates the provisioning of the node by provisioner.
`BLEMesh_ConfigurationCallback()`	Indicates the node configuration is complete, by provisioner.
`BLEMesh_FnFriendshipEstablishedCallback()`	Friendship established by Friend node.
`BLEMesh_FnFriendshipClearedCallback()`	Friendship cleared by Friend node.
`BLEMesh_LpnFriendshipEstablishedCallback()`	Friendship established by low-power node.
`BLEMesh_LpnFriendshipClearedCallback()`	Friendship cleared by low-power node.
`BLEMesh_NeighborAppearedCallback()`	New neighbor appeared.
`BLEMesh_NeighborRefreshedCallback()`	New neighbor refresh.
`BLEMesh_CustomBeaconReceivedCallback()`	No mesh message management if the feature `ENABLE_CUSTOM_BEACON` is enabled.
`BLEMesh_CustomBeaconGeneratorCallback()`	Generation of none mesh message example.

- Manages the user SW1 button action as defined in Table 30.

Table 30. SW1 button action

Action	Definition
Short press	The following events take place: – Publish `Generic_OnOff_Set_Unack` message if the feature `GENERIC_SERVER_MODEL_PUBLISH` is enabled. – Publish Blue LED command to Vendor model if the feature `VENDOR_CLIENT_MODEL_PUBLISH` is enabled. – Publish the `GenericClient_OnOff_Set_Unack` message if the feature `GENERIC_CLIENT_MODEL_PUBLISH` is enabled.
Long press	– Publish the `LightClient_Lightness_Set_Unack` message if the feature `LIGHT_CLIENT_MODEL_PUBLISH` is enabled. – Publish the Generic Client Level Set Unacknowledged message if the feature `GENERIC_CLIENT_MODEL_PUBLISH` is enabled.

- Calls the `Mesh_task()` for mesh profile and model processes when scheduled by the sequencer and re schedules these processes when needed:

Table 31. Mesh_task() rescheduling processes

Process	Description
`BLEMesh_Process()`	Mesh library profile processes.
`BLEMesh_ModelsProcess()`	Middleware mesh model processes.

- Calls the `Appli_Task()` for mesh application processes when scheduled by the sequencer and re schedule these processes when needed:
  - `Appli_Process()`: this function contains several sub-functions as listed in Table 32. Appli_Process() subfunctions:

Table 32. Appli_Process() subfunctions

Function	Description
`AppliNvm_Process()`	Flash memory accesses for the mesh model state if the feature `ENABLE_SAVE_MODEL_STATE_NVM` is enabled.
`Appli_LowPowerProcess()`	Not used, low-power management is done by the low-power manager and the sequencer.
`Appli_OobAuthenticationProcess()`	"Out Of Band" authentication process if the feature `USER_OUTPUT_OOB_APPLI_PROCESS` is enabled.

Generic Server models application

The `appli_generic.c` file contains a set of callback application functions called when a Generic Server messages are received for:

- Generic ON/OFF model
- Generic Level model
- Generic Power ON/OFF model
- Generic Default Transition Time Server model.

It includes the following functions described in Table 33. The file location is illustrated in the figure below.

Figure 53. Generic server models application

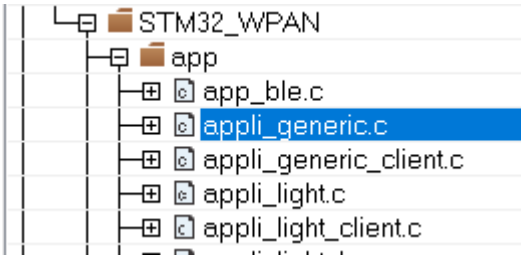


Table 33. Generic Server model functions

Function	Description
`Appli_Generic_OnOff_Status()`	Callback for an application when a Generic ON/OFF message is received.
`Appli_Generic_Level_Set()`	Callback for an application when a Generic Level message is received.
`Appli_Generic_Delta_Set()`	Callback for an application when a Generic Level Delta message is received.
`Appli_Generic_Move_Set()`	Callback for an application when a Generic Level Move message is received.
`Appli_Generic_Level_Status()`	Callback for an application when a Generic Level Move message is received.
`Appli_Generic_PowerOnOff_Set()`	Callback for an application when a Generic Power ON/OFF set message is received.
`Appli_Generic_PowerOnOff_Set()`	Callback for an application when a Generic Power ON/OFF set message is received.
`Appli_Generic_DefaultTransitionTime_Set()`	Callback for an application when a Generic Power ON/OFF set message is received.
`Appli_Generic_DefaultTransitionTime_Status()`	Callback for an application when a Generic Power ON/OFF set message is received.
`Appli_Generic_GetOnOffState()`	Callback for an application when a Generic ON/OFF Status message is to be provided.
`Appli_Generic_GetOnOffValue()`	Callback for an application to get the value for the Generic ON/OFF.
`Appli_Generic_GetLevelStatus()`	Callback for an application when a Generic Level Status message is to be provided.
`Appli_Generic_GetPowerOnOffStatus()`	Callback for an application when a Generic Get Power Status message is to be provided.
`Appli_Generic_GetDefaultTransitionStatus()`	Callback for an application when a Generic Default Transition Status message is to be provided.

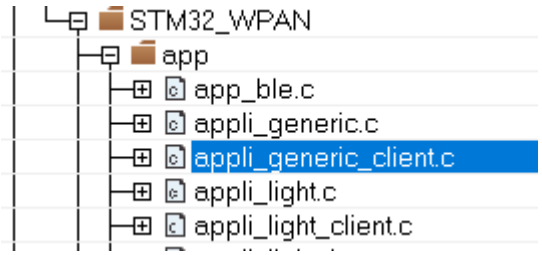
Generic Client models application

The `appli_generic_client.c` file contains a set of application callback functions called when Generic Client messages have to be sent for:

- Generic ON/OFF model
- Generic Level model
- Generic Power ON/OFF model
- Generic Default Transition Time Client model.

The file location is illustrated in the figure below.

Figure 54. Generic Client models application



Light Server models application

The `appli_light.c` file contains a set of application functions called when Light Server messages are received for:

- Light Lightness model
- Light Lightness Setup model
- Light CTL model
- Light CTL Temperature model
- Light CTL Setup model
- Light HSL model
- Light HSL Hue model
- Light HSL Saturation model
- Light HSL Setup Server model.

The file location is illustrated in the figure below. The function are described in Table 34

Figure 55. Light Server models application

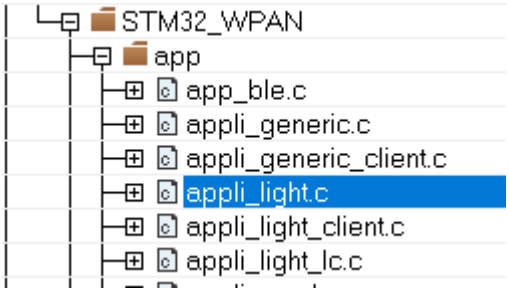


Table 34. Light Server models application functions

Function	Description
`Appli_Light_Lightness_Set()`	Callback for an application when a `Light_Lightness_Set` message is received.
`Appli_Light_Lightness_Status()`	Callback for an application when a `Light_Lightness_Status` message is received.
`Appli_Light_Lightness_Linear_Set()`	Callback for an application when a `Light_Lightness_Linear_Set` message is received.
`Appli_Light_Lightness_Linear_Status()`	Callback for an application when a `Light_Lightness_Linear_Status` message is received.
`Appli_Light_Lightness_Default_Set()`	Callback for an application when a `Light_Lightness_Default_Set` message is received.

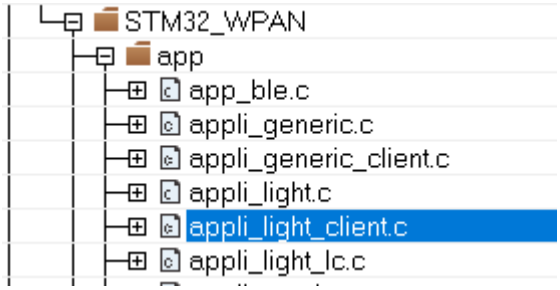
Function	Description
`Appli_Light_Lightness_Default_Status()`	Callback for an application when a `Light_Lightness_Default_Status` message is received.
`Appli_Light_Lightness_Range_Set()`	Callback for an application when a `Light_Lightness_Range_Set` message is received.
`Appli_Light_Lightness_Range_Status()`	Callback for an application when a `Light_Lightness_Range_Status` message is received.
`Appli_Light_Ctl_Set()`	Callback for an application when a `Light_CTL_Set` message is received.
`Appli_Light_Ctl_Status()`	Callback for an application when a `Light_CTL_Status` message is received.
`Appli_Light_CtlTemperature_Set()`	Callback for an application when a `Light_CTL_Temperature_Set` message is received.
`Appli_Light_CtlTemperature_Status()`	Callback for an application when a `Light_CTL` temperature Status message is received.
`Appli_Light_CtlTemperature_Range_Set()`	Callback for an application when a `Light_CTL_Temperature_Range_Set` message is received.
`Appli_Light_CtlTemperature_Range_Set()`	Callback for an application when a `Light_CTL_Temperature_Range_Status` message is received.
`Appli_Light_CtlDefault_Set()`	Callback for an application when a `Light_CTL_Default_Set` message is received.
`Appli_Light_CtlDefault_Status()`	Callback for an application when a `Light_CTL_Default_Status` message is received.
`Appli_Light_Hsl_Set()`	Callback for an application when a `Light_HSL_Set` message is received.
`Appli_Light_Hsl_Status()`	Callback for an application when a `Light_HSL_Status` message is received.
`Appli_Light_HslHue_Set()`	Callback for an application when a `Light_HSL_Hue_Set` message is received.
`Appli_Light_HslHue_Status()`	Callback for an application when a `Light_HSL_Hue_Status` message is received.
`Appli_Light_HslSaturation_Set()`	Callback for an application when a `Light_HSL_Saturation_Set` message is received.
`Appli_Light_HslSaturation_Status()`	Callback for an application when a `Light_HSL_Saturation_Status` message is received.
`Appli_Light_HslDefault_Set()`	Callback for an application when a `Light_HSL_Default_Set` message is received.
`Appli_Light_HslDefault_Status()`	Callback for an application when a `Light_HSL_Default_Status` message is received.
`Appli_Light_HslRange_Set()`	Callback for an application when a `Light_HSL_Range_Set` message is received.
`Appli_Light_HslRange_Status()`	Callback for an application when a `Light_HSL_Range_Status` message is received.
`Appli_Light_GetLightnessStatus()`	Callback for an application to get the application values in the middleware used for the transition change.

Function	Description
Appli_Light_GetLightnessLinearStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetLightnessDefaultStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetLightnessLastStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetLightnessRangeStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetCtlLightStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetCtlTeperatureStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetCtlTemperatureRange()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetCtlDefaultStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslHueStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslSaturationStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslDefaultStatus()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslSatRange()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_GetHslHueRange()	Callback for an application to get the application values in the middleware used for the transition change.
Appli_Light_PwmInit()	Initializes the pulse width modulation. This is used for the external RGB LED.
RgbF_Create()	Converts the RGB values into real values.
HSL2RGB_Conversion()	Convert the HSL values into RGB values.
Rgb_LedOffState()	Called while using CTL, turns all the RGB pulse width modulation into the off state for an application.
Ctl_LedOffState()	Called while using HSL, turns all the cool Warm pulse width modulation into the off state for an application.
Light_UpdateLedValue()	Sets the values for pulse width modulation for the external RGB LED.

Light Client models application

The `appli_light_client.c` file contains a set of application functions called when Light Client messages are received for Light Lightness, Light CTL, Light HSL Client models. The file location is illustrated in Figure 56.

Figure 56. **Light Client models application**



Light Light Control Server (Light LC Server) models application

The appli_light_lc.c file: This file contains a set of application functions called when Light LC Server messages are received for Light LC Server model. The file location is illustrated in the figure below.

Figure 57. **Light LC Server models application**

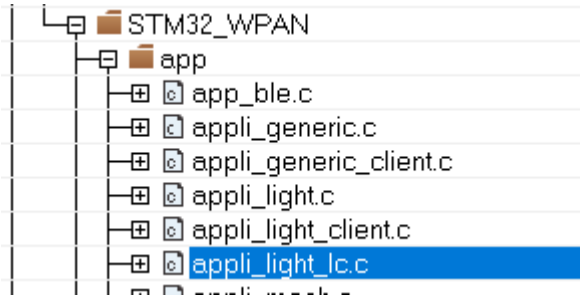


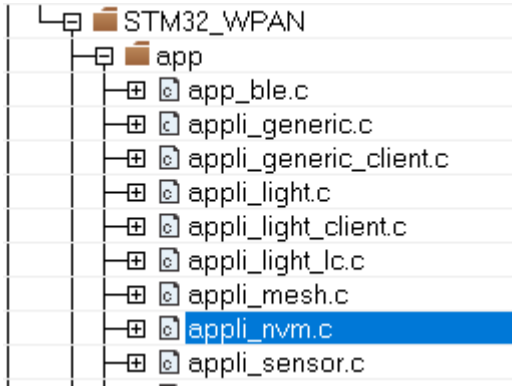
Table 35. **Light LC Server models application functions**

Function	Definitiop
`Appli_Light_LCMode_Set()`	Callback for an application when a `Light_LC_Mode Set` message is received.
`Appli_LightLC_OM_Set()`	Callback for an application when a `Light_LC_Mode_Occupancy` model Set message is received.
`Appli_LightLC_OnOff_Set()`	Callback for an application when a `Light_LC_OnOff_Set` message is received.
`Get_AmbientLuxLevelOutput()`	Calculates the Ambient Lux level output from the ambient sensor.
`Light_LC_LuxLevelPIRegulator()`	Calculates all the parameter Kid,kpu,kiu,kpd and returns the Light Lightness Linear value.
`Appli_LightLC_Get_ModeStatus()`	Callback for an application to get the application values in the middleware used for transition change.
`Appli_LightLC_Get_OMModeStatus()`	Callback for an application to get the application values in the middleware used for transition change.
`Appli_LightLC_Get_OnOffStatus()`	Callback for an application to get the application values in the middleware used for transition change.

Application Flash accesses

The `appli_nvm.c` file contains a set of application functions used to save or read to the Flash memory, for example: the state of the models, and provisioning data just for the provisioner node use case. The file location is illustrated in Figure 58.

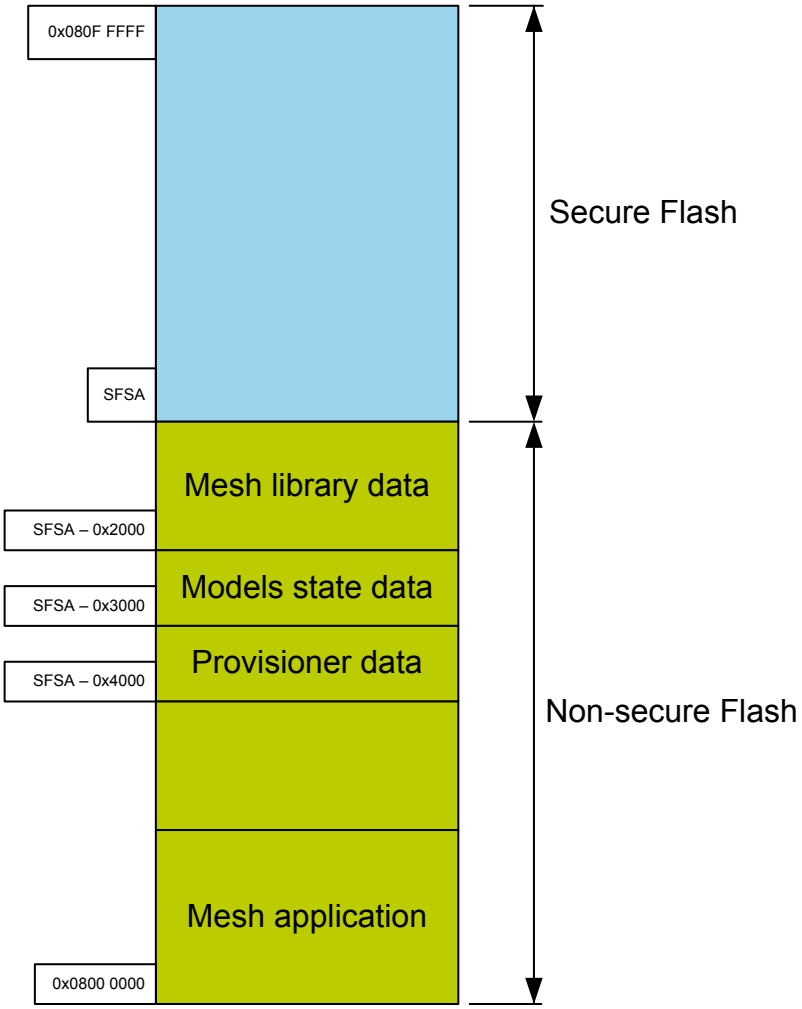
Figure 58. **Flash accesses application**



The SFSA option byte gives the last non secure Flash address. As shown in Figure 59, from this address:

- Two sectors are reserved for mesh library data:
  - Network and applications keys
  - Elements and model profile data.
- One sector is reserved to record the state of the models that are used.
- One sector is reserved for the provisioner device keys.

Figure 59. **Application Flash mapping**



Sensor Server model application

The `appli_sensor.c` file contains a set of application functions called when Sensor Server messages are received for Sensor Server and Sensor Setup models. The file location is illustrated in the figure below.

Figure 60. **Sensor Server application**

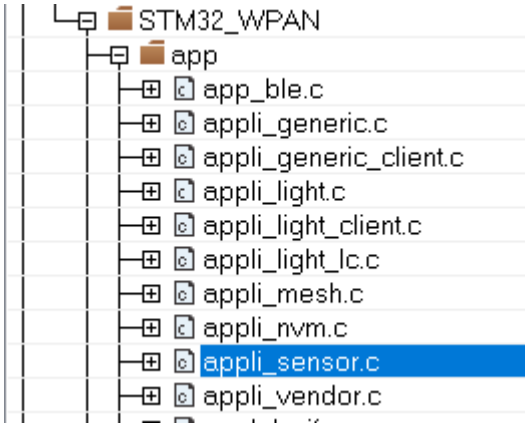


Table 36. Sensor Server application functions

Function	Description
`Appli_Sensor_Cadence_Set()`	Callback for an application when a `Sensor_Cadence_Set` message is received.
`Appli_Sensor_Cadence_Get()`	Callback for an application when a `Sensor_Cadence_Set` message is received.
`Appli_Sensor_Setting_Set()`	Callback for an application when a `Sensor_Setting_Set` message is received.
`Appli_Sensor_Data_Status()`	Callback for an application when a `Sensor_Data_Status` message is received.
`Appli_Sensor_Descriptor_Status()`	Callback for an application when a `Sensor_Descriptor_Status` message is received.
`Appli_Sensor_Column_Status()`	Callback for an application when a `Sensor_Column_Satus` message is received.
`Appli_Sensor_Series_Status()`	Callback for an application when `Sensor_Series_Satus` message is received.
`Sensor_Process()`	Continuously monitor the sensors for the publishing, data monitoring.
`Sensor_LC_Light_Publish()`	Check for the occupancy in the location and send the status message with the occupancy value, when the interrupt is detected.
`Read_Sensor_Data()`	Read the particular sensor value which are called inside.
`Sensor_Publication_Process()`	Process to publish the sensor data according to the given conditions.
`SensorDataPublish()`	Publish the sensor data according to the Sensor Publication process.
`Appli_Sensor_GetSetting_IDStatus()`	Callback for an application when the sensor setting numbers and row value status message is to be provided.
`Check_Property_ID()`	Used for checking the Property id of the sensor available in table.
`Appli_Sensor_Init()`	Callback for initialization of the sensor device.

Sensor Client model application

The `appli_sensor_client.c` file contains a set of application functions called when `Sensor_Client` messages are received for Sensor Client model.

Vendor Server model application

The `appli_vendor.c` file contains a set of application functions called when `Vendor_Server` messages are received for Vendor Server model. The file location is illustrated in the figure below and the functions are described in Table 37.

Figure 61. **Vendor Server application**

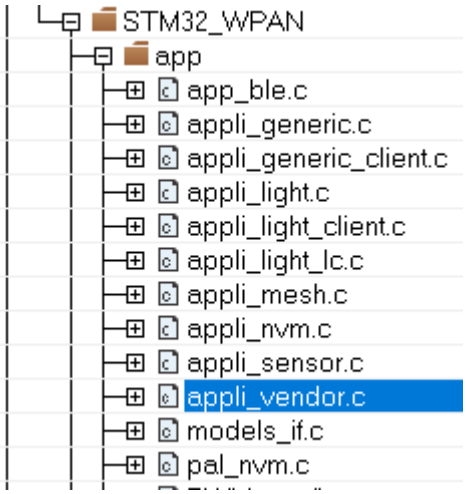


Table 37. **Vendor Server application functions**

Function	Description
`Appli_Vendor_DeviceInfo()`	Callback for an application when a Vendor Device Info command is received.
`Appli_Vendor_Test()`	Callback for an application when a Vendor Test command is received.
`Appli_Vendor_LEDControl()`	Callback for an application when a Vendor LED Control command is received.
`Appli_Vendor_Data_write()`	Callback for an application when a Vendor Data Write command is received.
`Appli_GetTestValue()`	Callback for an application when a Vensor Application Test command is received then status message is to be provided.

Model interface

The `models_if.c` file: contains a set of functions used for the models interface. The file location is illustrated in the figure below.

Figure 62. **Model interface**

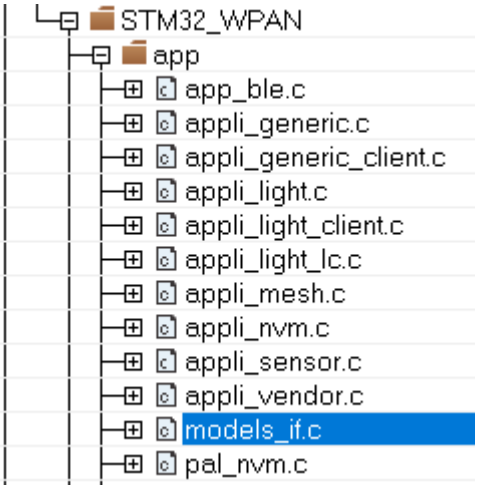


Table 38. **Model interface functions**

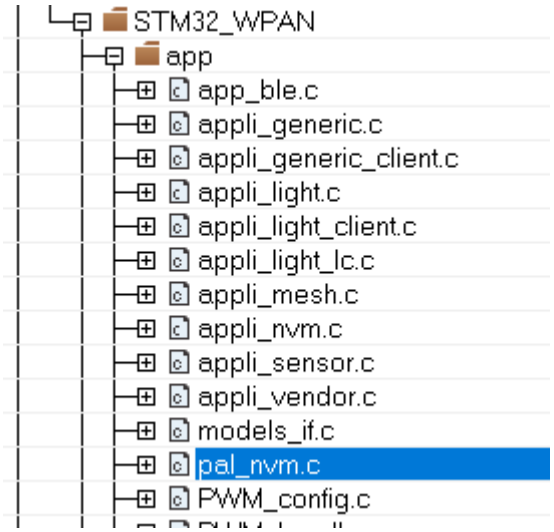
Function	Description
`GetApplicationVendorModels()`	Used to get a function access to configure the vendor model information for the mesh library.
`BLEMesh_ModelsInit()`	This function is composed of four sub-functions: • Gives the mesh library access to a set of model management functions that give the "opcodes" of the commands, the status request and the message processing. • Loads and restores the generic model states from the Flash memory. • Loads the provisioner data from the Flash memory (if the provisioner feature is enabled). • Initialize the sensors used with the sensor model.
`BLEMesh_ModelsProcess()`	Calls the model processes used to manage every model.
`BLEMesh_ModelsCommand()`	This function is composed of three sub-functions: • Publishes the Generic ON/OFF Set UnAck message if the feature GENERIC_SERVER_MODEL_PUBLISH is enabled. • Publishes the blue LED command to the VENDOR model if the feature VENDOR_CLIENT_MODEL_PUBLISH is enabled. • Publishes the Generic ON/OFF Set UnAck message if the feature GENERIC_CLIENT_MODEL_PUBLISH is enabled.
`BLEMesh_ModelsGetElementNumber()`	Gets the element number of a model.
`BLEMesh_ModelsCheckSubscription()`	Checks the subscription of elements to group address for selected model.
`BLEMesh_ModelsDelayPacket()`	Used for the vendor model to schedule a packet to be sent with randomized send timestamp.
`BLEMesh_ModelsSendDelayedPacket()`	Used for the vendor model. If "send timestamp" is reached send all packets.
`MeshClient_SetRemotePublication()`	Used when sending client model commands. This function sets a remote publication for the given model and node address.

Flash interface

The `pal_nvm.c` file (illustrated in the figure below) contains a set of functions to manage the Flash accesses by:

- The mesh library
- The application for the model states
- The provisioner.

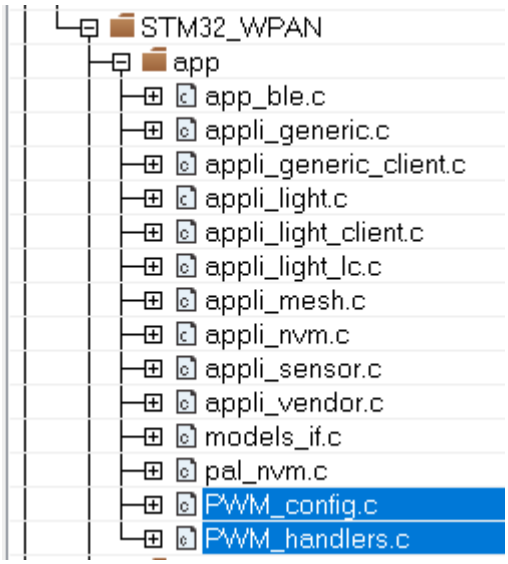
Figure 63. **Flash interface**



Pulse width modulation interface

The pulse width modulation interface is managed by two files illustrated in the figure and outlined below.

Figure 64. **Pulse width modulation interface**



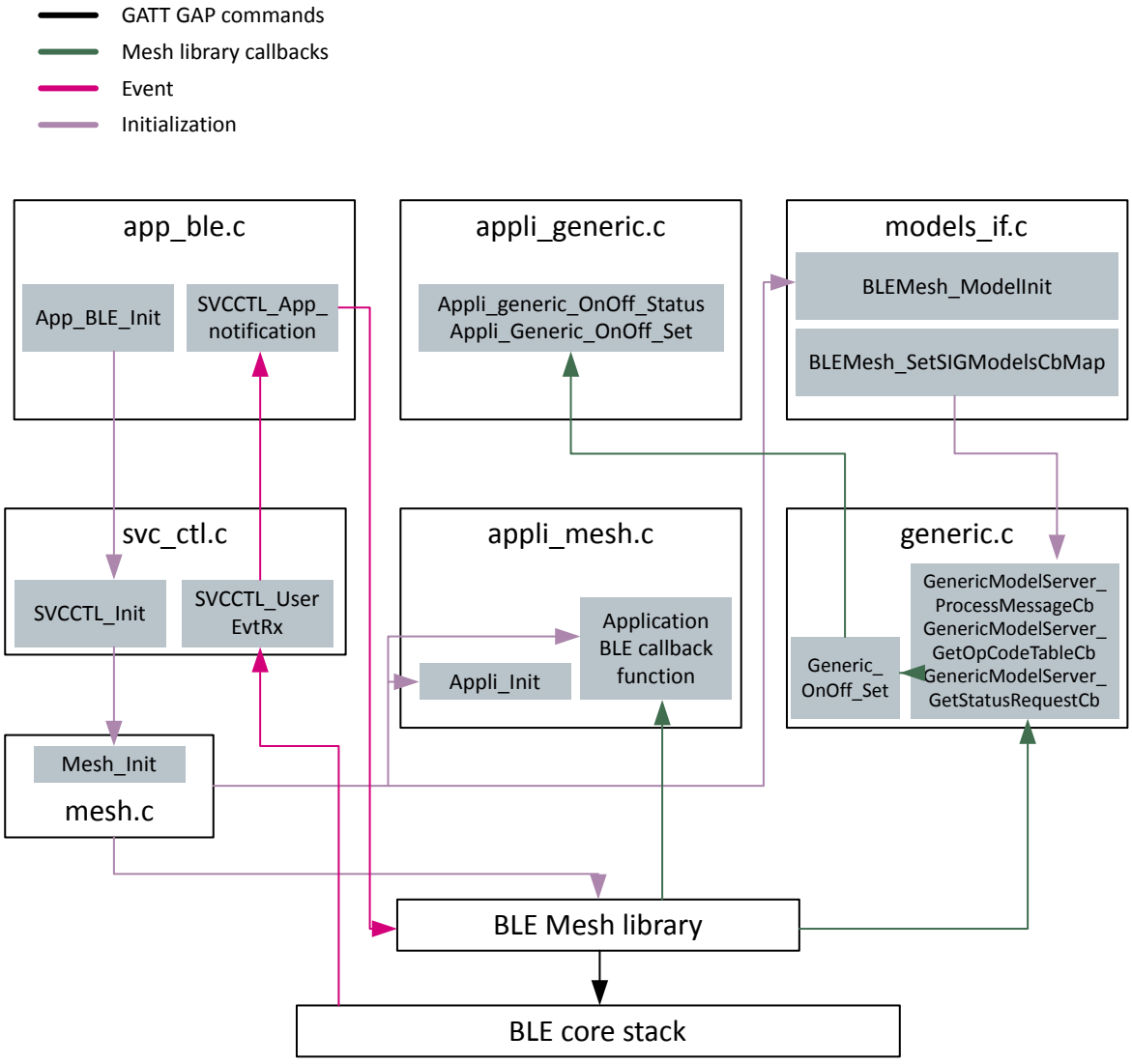
- The `PWM_config.c` file contains a set of functions used to configure a pulse width modulation which manages an external RGB LED.
- The `PWM_handlers.c` file contains a set of functions used to manage a pulse width modulation.

4.2.6 Generic ON/OFF Server model interaction

The following figure shows how the Generic ON/OFF Server model is impacted by:

- The middleware
- The application
- The mesh library
- The BLE stack

Figure 65. Example of Generic ON/OFF Server model interaction



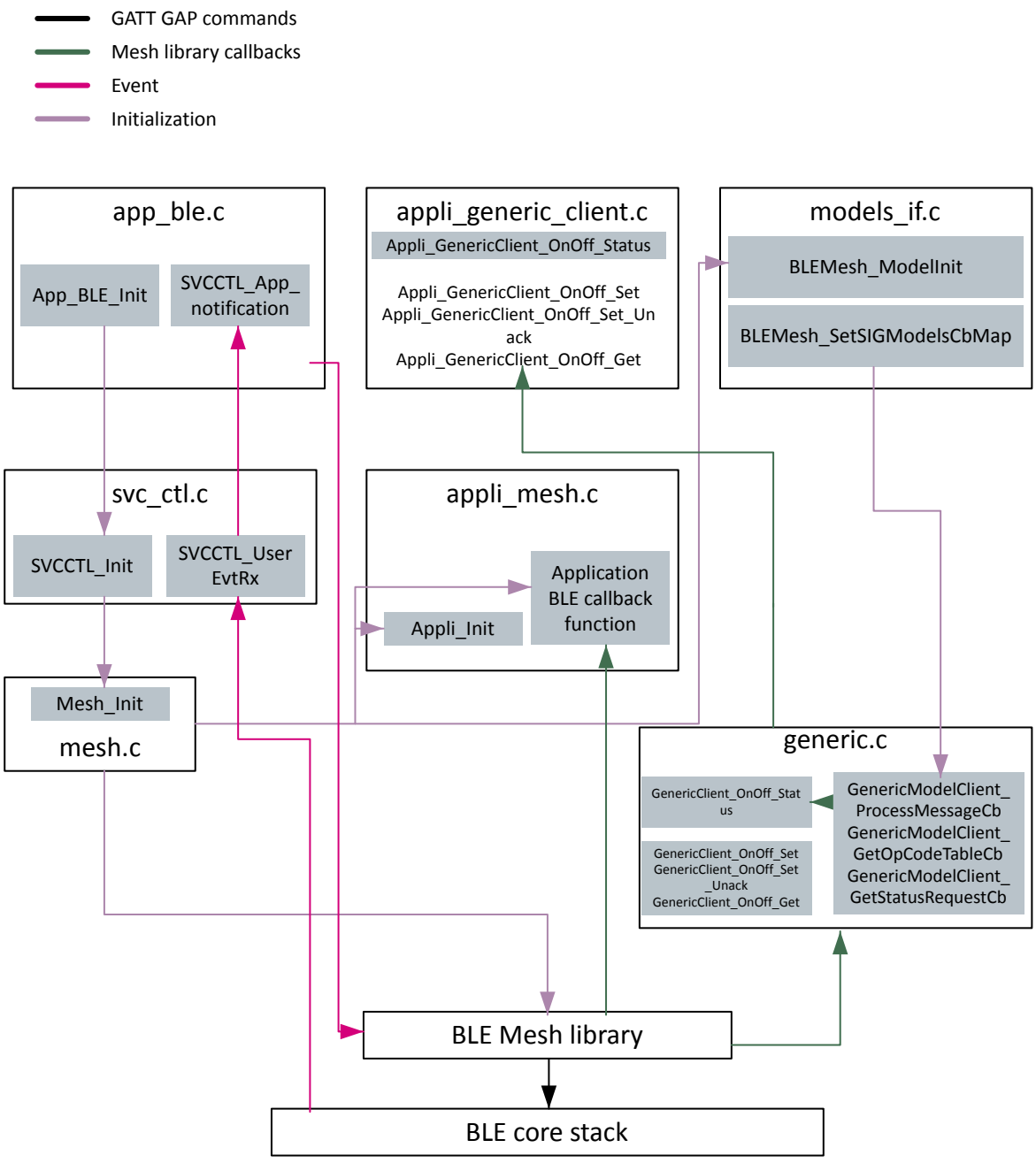
`Appli_Generic_OnOff_Satus()` and `Appli_Generic_OnOff_Set()` callbacks are used to manage respectively the Generic ON/OFF status values displaying the status value on an external terminal, and the LED state according the new status value set.

4.2.7 Generic ON/OFF Client model interaction

The following figure shows how the Generic ON/OFF Client model is impacted by:

- The middleware
- The application
- The mesh library
- The BLE stack

Figure 66. Example of Generic ON/OFF Client model interaction



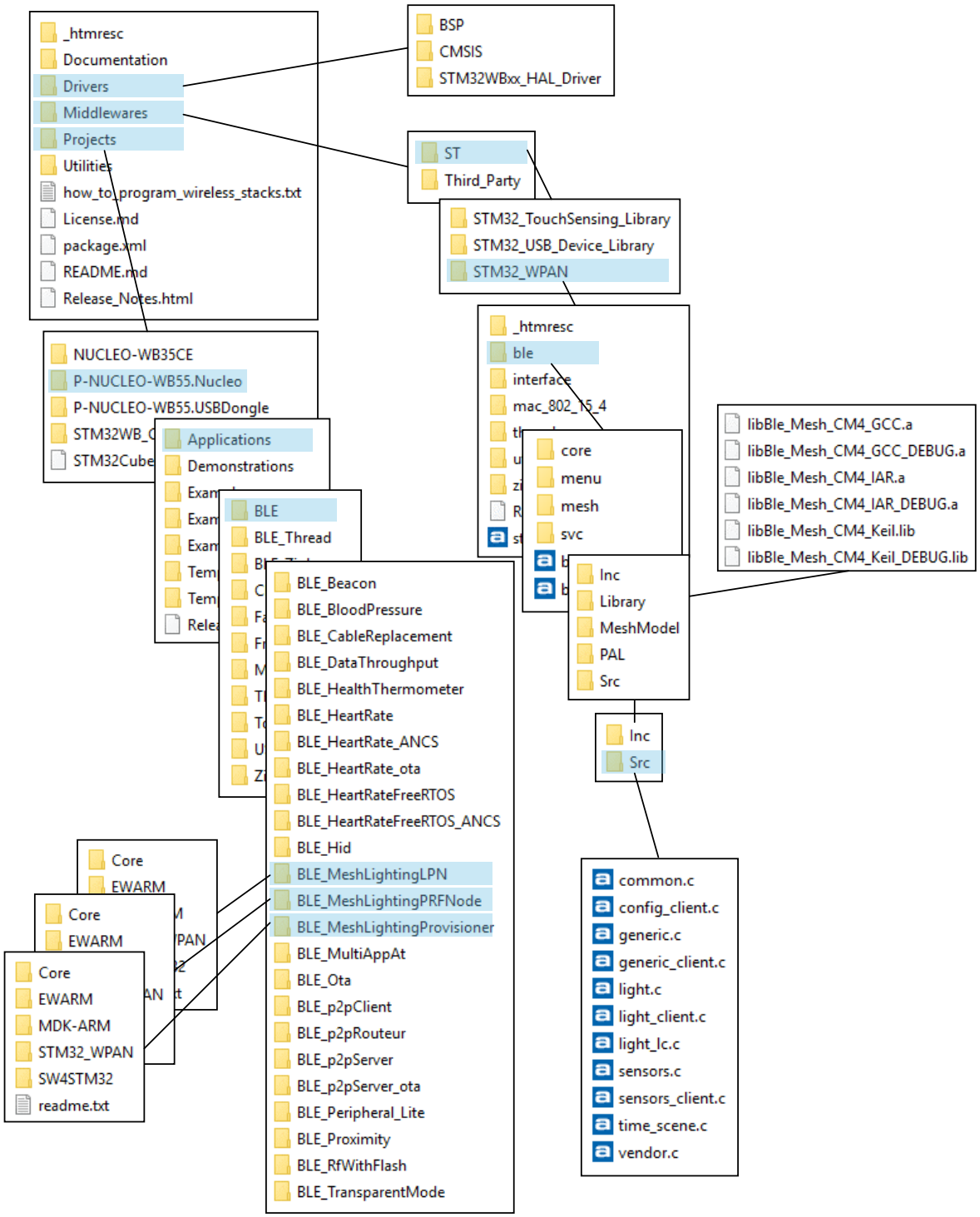
`Appli_Generic_OnOff_Satus()` and `Appli_Generic_OnOff_Set()` callbacks are used to manage respectively; the Generic ON/OFF status value displaying the status value on an external terminal, and the LED state according the new status value.

4.3 Firmware package

The firmware package is structured as follows and illustrated in Figure 67:

- A Documentation folder with a set of compiled HTML files generated from the source code together with the detailed description of all the software components and APIs
- A Drivers folder with HAL and board specific drivers to support the hardware platforms and components together with the CMSIS vendor-independent hardware abstraction layer for the Arm® Cortex®-M processor.
- A middleware folder with the mesh and BLE communication libraries. Horizontal interaction between layer components is directly handled by calling the feature APIs, while vertical interaction with the low level drivers is managed through specific callbacks and static macros implemented in the library system call interface.
- Projects folders containing the workspaces for IAR Systems®, Keil® and STM32CubeIDE Embedded Workbench integrated development environments for the P-NUCLEO-WB55 pack hich includes both the NUCLEO and DONGLE boards. The EWARM, MDK-ARM and STM32CubeIDE folders containing respectively the workspace for IAR Systems® Embedded, Keil® and STM32CubeIDE. The source files in the folders bind the firmware layers to implement the functions that demonstrates the mesh over BLE functionality.

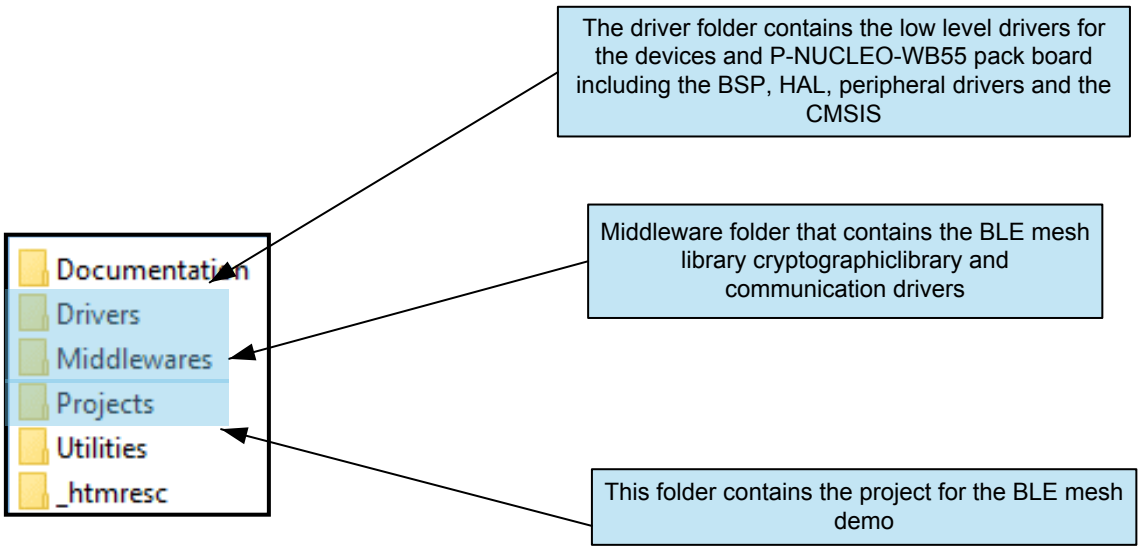
Figure 67. **Folders, sub-folders and content of the package**



4.3.1 Root folder

The figure below shows the root folder structure of the firmware package.

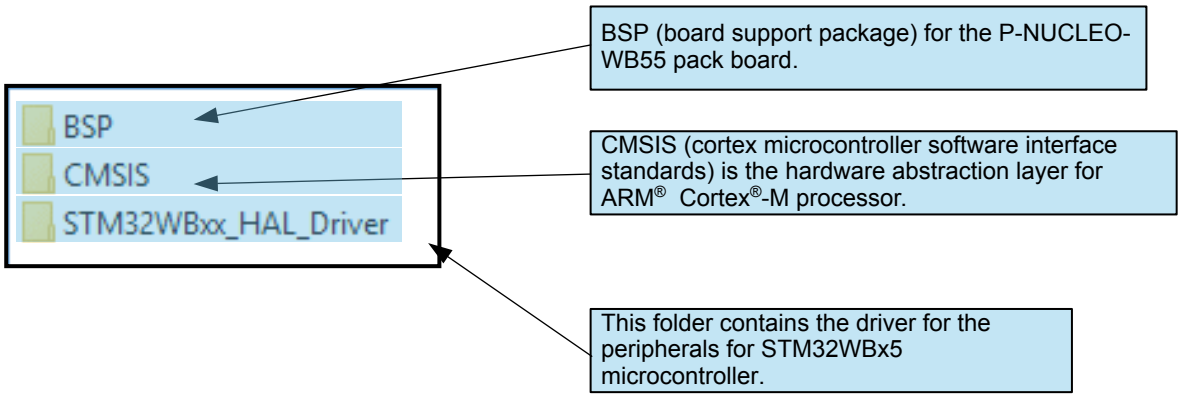
Figure 68. Root folder structure



4.3.2 Driver folder

This folder contains all low level drivers including peripheral drivers and HAL drivers corresponding to the hardware as illustrated in the figure below.

Figure 69. Driver folder



4.3.3 Internal project folder

The Nucleo (see Figure 70) and the Dongle (see Figure 71) project folders contain the projects for IAR Systems®, Keil® and STM32CubeIDE IDEs as illustrated in the figure below.

Figure 70. Nucleo project folder

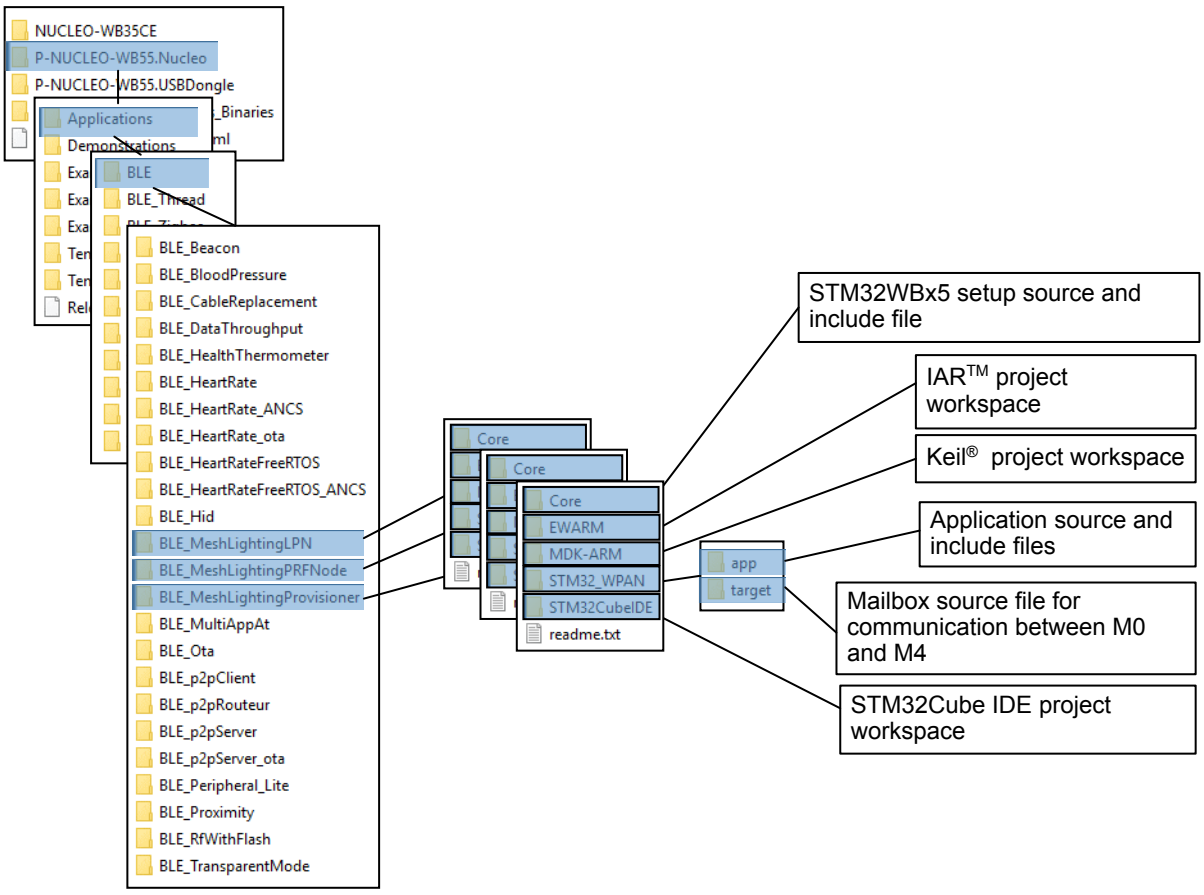
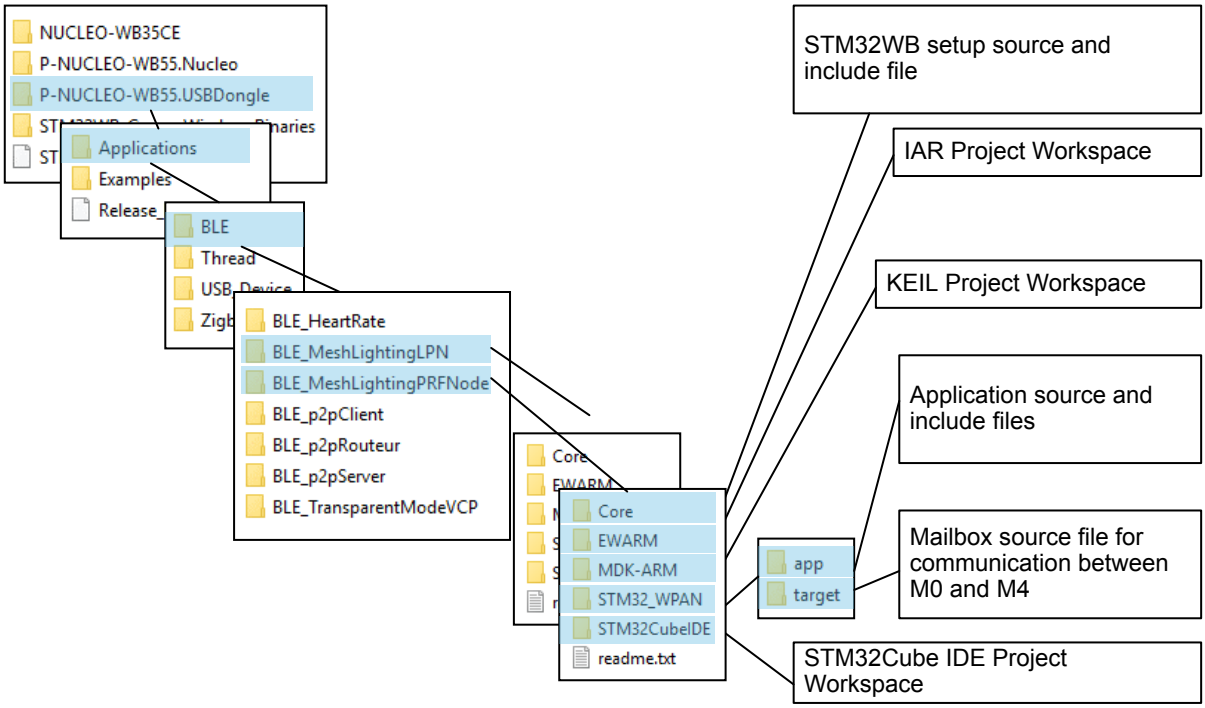


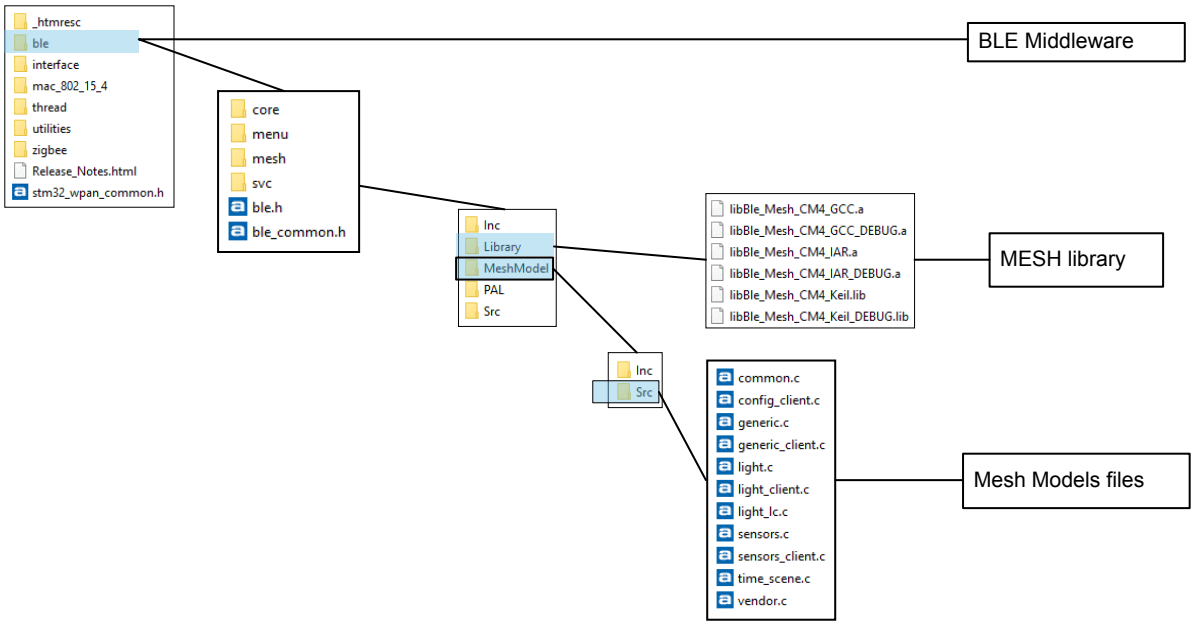
Figure 71. **Dongle project folder**



4.3.4 Middleware folder

The middleware folder contains the middleware project including pre-compiled mesh library for the STM32WBx5 as illustrated in the figure below.

Figure 72. **Middleware folder**



5 Firmware initialization and configuration

This section describes the API functions for application development based on mesh network over Bluetooth® Low Energy devices.

The mesh over BLE library manages the following features:

- Creates the mesh network between nodes.
- Handles the unicast, broadcast addressing.
- Manages the relay feature: all the packets with a destination address for other nodes are re-transmitted.
- Communication with advanced feature devices, such as provisioning and proxy service.
- Handles the platform specific communication.

The user application handles the following:

- Initialization of the mesh stack;
- User callbacks required for the application;
- Application handling.

The following sections describe the requirements for the firmware initialization and configuration.

5.1 Setting the transmit power of a node

The node transmit power is defined by initializing a callback to the mesh library. It runs as follows:

```
{  Appli_BleStackInitCb,
  Appli_BleSetTxPowerCb,
  Appli_BleGattConnectionCompleteCb,
  Appli_BleGattDisconnectionCompleteCb,
  Appli_BleUnprovisionedIdentifyCb,
  Appli_BleSetUUIDCb,
  Appli_BleSetNumberOfElementsCb};
```

The `Appli_BleSetTxPowerCb()` calls the `aci` function to set the power
`aci_hal_set_tx_power_level(uint8_t En_High_Power, uint8_t PA_Level);`
By default, +4 dbm is configured in the nodes for the STM35WBx5, this can be changed by the user.

5.2 MAC address management

Each node in the mesh network requires its unique MAC address. The following options are available to configure the node MAC addresses.

Table 39. MAC address management

Option	MAC address Management	Comments
1	Using external MAC address	User can program the nodes with desired unique MAC address. This is stored at specific location in the flash. It is the user's responsibility to ensure that the programmed MAC address in the device is compliant with the Bluetooth® communication requirements.
2	Using the unique device serial number	It is possible to configure the MAC address of the device using the unique serial number available in each device. This is the default setting.
3	Using static random MAC address	It is possible to configure the MAC address of device using the static random MAC address

5.3 Initialization of application callbacks

The application configuration starts by initializing the callbacks required for the different events and functionalities. These callbacks are used in the BLE mesh library to call the functions based on specific events or the library state machine.

```
{
  Vendor_WriteLocalDataCb,
  Vendor_ReadLocalDataCb,
  Vendor_OnResponseDataCb
};
/* Callbacks used by BLE-Mesh library */
BLEMesh_SetVendorCbMap(&vendor_cb);
```

The structure `MOBLE_VENDOR_CB_MAP` is used to initialize the vendor model for the application implementation. The function `BLEMesh_SetVendorCbMap(&vendor_cb);` is used to initialize the vendor callbacks in the library.

5.4 Initialization and main application loop

This procedure develops an application for mesh over BLE on the STM32WBx5 platforms. The procedure is to be carried out in the following steps:

1. Call the `HAL_Init()` API is used to initialize the HAL library. It must be the first instruction to be executed in the main program (before calling any other HAL function). This API performs the following operations:
   - Configure the Flash prefetch, instruction and Data caches.
   - Configures the SysTick to generate a 1 millisecond interrupt, which is clocked by the MSI.

Note: At this stage, the clock is not yet configured, therefore the system is running from the internal MSI at 4 MHz.

   - Set NVIC Group Priority to 4.
   - Calls the `HAL_MspInit()` callback function defined in user file `stm32wbxx_hal_msp.c` to do the global low level hardware initialization.
2. Call the `Reset_Device()` API to reset the backup domain and IPCC.
3. Call the `Config_HSE()` API to configure HSE clock capacitor tuning according option bytes (OTP).
4. Call the `SystemClock_Config()` API. This API configures:
   - The system clock source
   - The AHBCLK, APBCLK dividers
   - The flash latency
   - The PLL settings (when required).
5. Call the `PeriphClock_Config ()` API for USB_USER usage if enabled.
6. Call the `Init_Exti ()` API to initialize the external interrupts.
7. Call the `Init_RTC ()` API to initialize the real time counter.
8. Call the `__HAL_RCC_CRC_CLK_ENABLE()` API to enable the peripheral clock.

9. Call the `APPE_Init()` API to initialize the LEDs, the buttons and all transport layers. This is done using the following steps:
   a. Call the `SystemPower_Config()` API to configure the system Low-power mode.
   b. Call the `HW_TS_Init()` API to initialize the timer server.
   c. Call the `APPD_Init()` API to configure UART for debugging trace.
   d. Call the `UTIL_LPM_SetOffMode()` API to turn off the Low-power mode
   e. Call the `Led_Init()` API to Initialize the LED.
   f. Call the `Button_Init()` API to initialize the button.
   g. Initialize the Flash memory base addresses used to provision the data, models application data and embedded provisioner data according OTP SFSA value (secure Flash start address).
   h. Call the `LpTimerInit()` API to initialize the low-power timer if LOW_POWER_FEATURE is enabled.
   i. Call the `appe_Tl_Init()` API to initialize all transport layers. All transport layer initializations call the mesh initialization `MESH_Init()` with:
      - Call for the `Appli_CheckBdMacAddr()` API to check the validity of the MAC address.
      - If the MAC address is not valid, then the firmware is stuck in a `while(1)` loop with the LED continuously blinking.
      - Initialization of the hardware callback functions for the BLE hardware. Done by updating:

```
MOBLE_USER_BLE_CB_MAP user_ble_cb =
 { Appli_BleStackInitCb,
 Appli_BleSetTxPowerCb,
 Appli_BleGattConnectionCompleteCb,
 Appli_BleGattDisconnectionCompleteCb,
 Appli_BleUnprovisionedIdentifyCb,
 Appli_BleSetUUIDCb,
 Appli_BleSetProductInfoCB,
 Appli_BleSetNumberOfElementsCb,
 Appli_BleDisableFilterCb}
;
```

        These APIs are used to create an application interface for the BLE radio initialization and TxPower configuration.
      - Initialization of the BLE mesh library by calling `BLEMesh_Init(&BLEMeshlib_Init_params)`

        Done by updating the structure containing mesh library Initialization info data:

```
const Mesh_Initialization_t BLEMeshlib_Init_params =
{
 bdaddr,
 &FnParams,
 &LpnParams,
 MESH_FEATURES,
 &DynBufferParam
};
```

        In the event of an error, the demo firmware prints a message on the VCOM port terminal window opened by the board USB connection. The `Appli_LedBlink()` API also causes the LED to blink continuously.
      - Initialize of the BT SIG models list table according models enabled in the `mesh_cfg_usr.h` file.
      - Initialize of the Vendor model list table according Vendor models enabled in the `mesh_cfg_usr.h` file.
      - Initialize the application using the following APIs:
        - Create an `UnprovisionedDeviceBeaconApp()` task called by the associated timer `discoverTimer_Id` to stop unprovisioned node advertising after 10 minutes to save on power consumption.
        - Initialize the serial interface if enabled in the `mesh_cfg_usr.h` file. The serial interface allows BT SIG Model command messages to be sent via the virtual COM port.

- Initialize an external GPIO interrupt for the power failure detection feature. It must be enabled in the `mesh_cfg_usr.h` file.
- Enable the clocking of two hardware timers used in pulse width modulation mode to control external RGB LED. The PWM timers are only available without low-power and if enabled in the `mesh_cfg_usr.h`.
- Create the `AppliMeshSW1Task()` task to manage the user SW1 BUTTON.
- Create the `LowPowerNodeApiApp()` task called by the associated timer `lowPowerNodeApiTimer_Id` to manage the low-power node advertising and scanning after provisioning.

◦ Check whether the device has been provisioned or not. A provisioned device has network keys and other parameters configured in the internal Fash memory. Checks can be performed with `BLEMesh_IsUnprovisioned()` API.
- If the node is unprovisioned, use the `BLEMesh_InitUnprovisionedNode()` API to initialize it.
- If the device is already provisioned, then `BLEMesh_InitprovisionedNode()` API initializes the device.

◦ Print the messages to the terminal window for the nodes that are being initialized. The messages also prints the MAC address assigned to the node.

◦ Call `Appli_CheckForUnprovision()` to:
- Check the button state. To initialize the node in the unprovisioned state, hold down the user SW1 button. When the unprovisioning SW1 button sequence is detected, the `BLEMesh_Unprovision()` API erases all the network parameters configured in the device internal memory. Once unprovisioning is complete, the board must be reset.
- Create and register the `Mesh_Task()` in the sequencer to manage all the MESH processes. At the end of the task, the task is set for the next scheduling again in the sequencer.
- Set the `Mesh_Task()` in the sequencer.
- Create and register the `Appli_Task()` in the sequencer to manage all the application processes:

Table 40. Appl_task functions

Functiona	Description
`AppliNvm_Process()`	Flash model data management if enabled in the `mesh_cfg_usr.h` file.
`Appli_OobAuthenticationProcess()`	Out of band authentication process if enabled in the `mesh_cfg_usr.h` file.
`AppliPrvnNvm_Process()`	Provisions the embedded Flash data management if enabled in the `mesh_cfg_usr.h` file.
`Appli_ConfigClient_Process()`	embedded provisioner configuration process reserved for the provisioner.
`Appli_SelfConfigurationProcess()`	embedded provisioner self configuration process reserved for the provisioner.
The task ends with the set of the task in the Sequencer for the next scheduling.	

- Set the Attention Timer callback

◦ Set the `Appli_Task()` in the Sequencer.

◦ Set the UUID for the board

◦ Display:
- the mesh application version
- the mesh library version
- the BLE stack version
- the FUS version

- the BD MAC address
- the UUID address.

◦ Initialize all the models enabled in the `mesh_cfg_usr.h file (vendor, generic , lighting, sensor models)` triggered by the events `BLEMesh_ModelsInit();`

◦ Initialize PWM LED value to default value if external RGB LEDs are available.

10. Process MoBLE and HCI events in `while(1)` loop. The application must call `SCH_Run()` in the `while(1)` loop as frequently as possible. The sequencer schedules all the registered tasks related to the BLE mesh processes. The sequencer also manages the Low-power mode when it is enabled in the `app_conf.h` file.

6 Mesh networking information

6.1 Local and remote concept

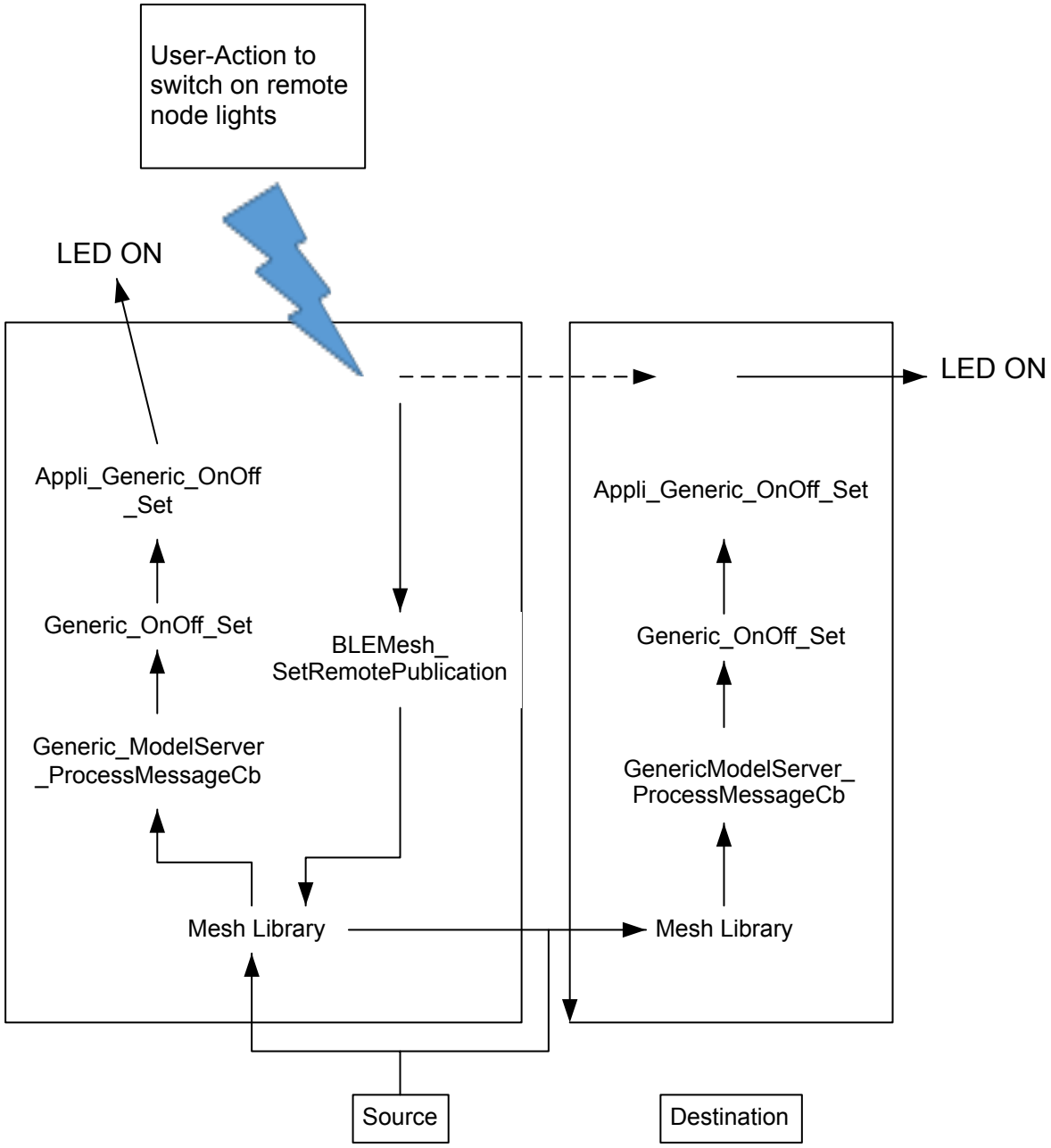
Remote actions or operations refer to actions on other network nodes, while local actions refer to the resources on the same network node.

For example, if the user wants to control the LEDs on the BLE mesh app nodes with a Generic ON/OFF model, click the icon button to invoke the `BLEMesh_SetRemotePublication` action from the app. This causes a `GenericModelServer_ProcessMessageCb` action on the node.

When a group of LEDs need to be controlled on the board, use the above process also. The LEDs on the addressed nodes toggle when the button on the board is pressed.

The same message received by the node which needs to turn the LEDs on requires a `GenericModelServer_ProcessMessageCb` action on that node. This is shown in the figure below for the `SetRemote/ProcessMessageCb` actions on different nodes in the network.

Figure 73. **SetRemote/ProcessMessageCb actions with Generic ON/OFF model**



6.2 Acknowledged and unacknowledged messages

By default, all messages in the mesh network are configured as unacknowledged. The difference between acknowledged and unacknowledged messages is the response to the message. For example, a write message to a node may have a response in acknowledged communication. Whereas, in unacknowledged communication, the response may not be there.

Unacknowledged messages must be used in the mesh network to avoid redundant network message exchanges, which may impact the system performance.

6.3 GATT connection/disconnection node

Each node in the network connects to a smartphone through the GATT interface.

When this connection is established, the node becomes a “proxy” The proxy bridges the commands and responses between the mesh network and the smartphone. The connection status with the smartphone is managed by the following callbacks:

- `Appli_BleGattConnectionCompleteCb;`
- `Appli_BleGattDisconnectionCompleteCb;`

These are initialized during the main loop. It may be interesting to understand which node is connected to the smartphone when there are many nodes nearby.

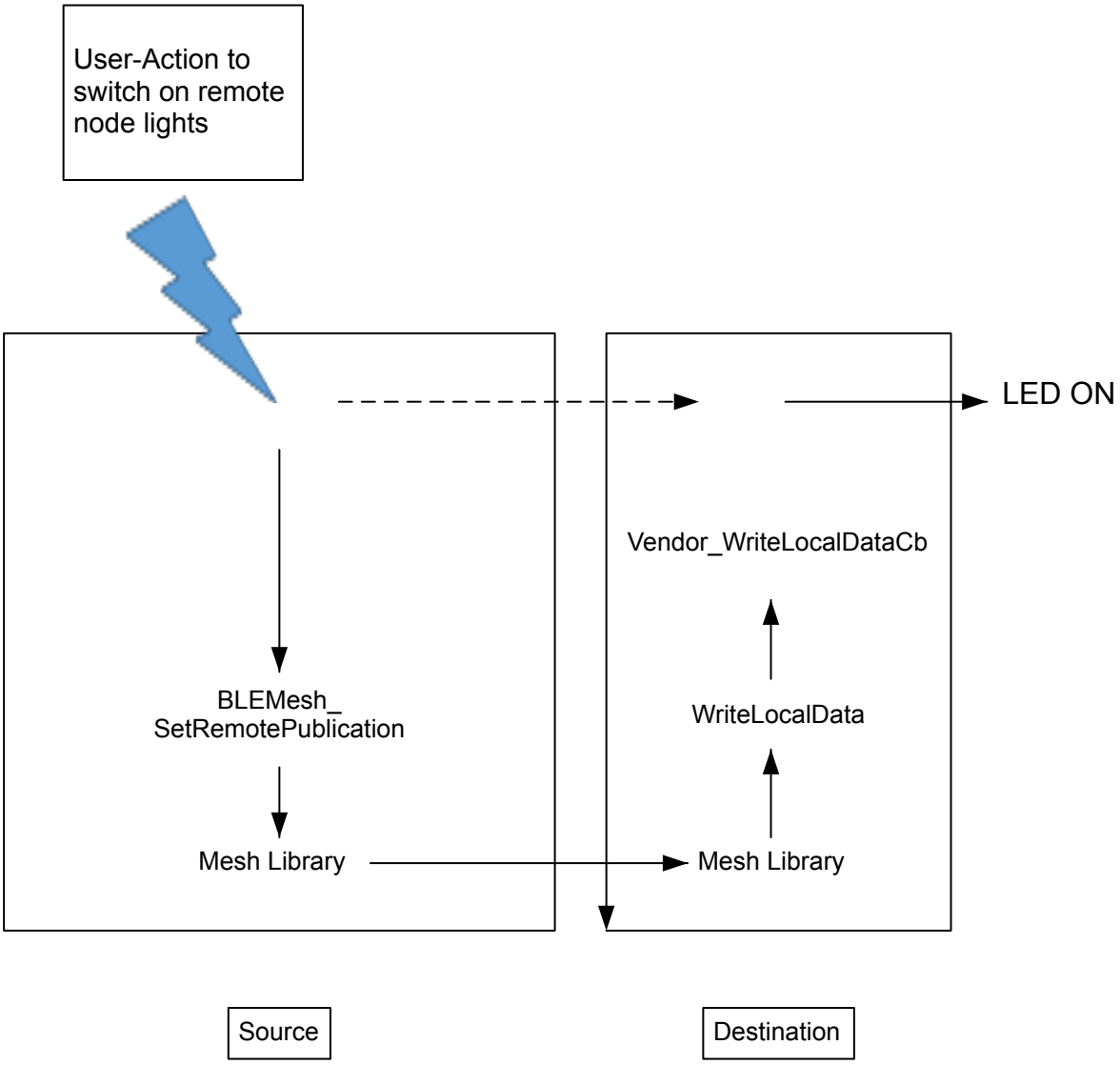
During provisioning, the GATT connection is established with the node which needs to be provisioned. If the smartphone moves out of the range of the Proxy node, it establishes a new connection with the next available node. LED indication (LED2) is used to show the proxy connection.

6.4 Vendor model write command from remote node

A Vendor model command from a remote node or from a smartphone to an addressed node invokes a `WriteLocalData` callback.

This callback can be used to process the commands or data received by the network. In the application demo, the `Vendor_WriteLocalDataCb` function is the callback where data or commands are processed. The command/data flow is illustrated in the picture below.

Figure 74. **Vendor model write command data flow**

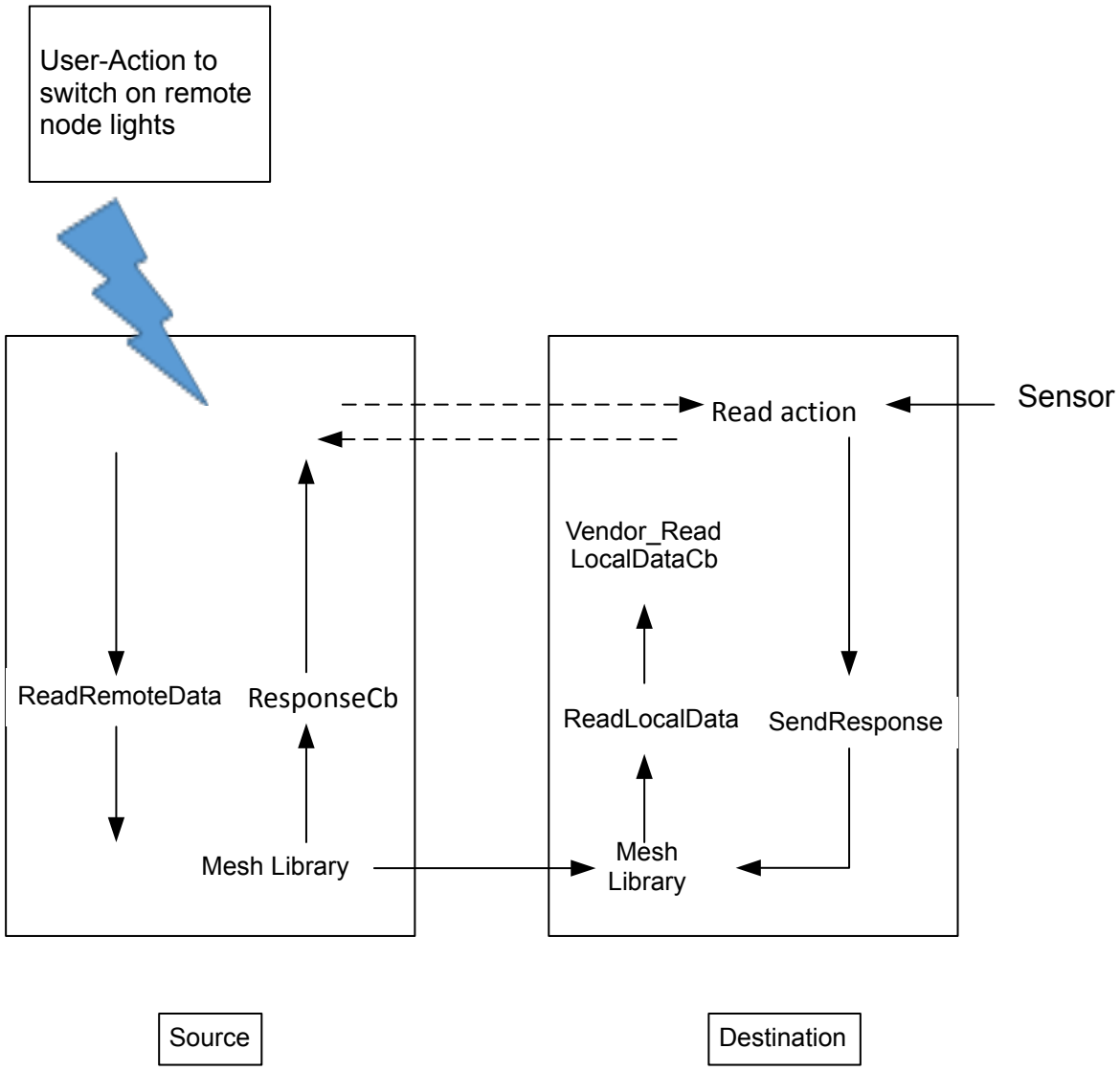


The response data from the node is sent through `SendResponse` function.

6.5 Vendor model read command from a remote node

A "Vendor model Read" command from a remote node or from a smartphone to an addressed node, uses the `Vendor_ReadLocalDataCb` callback. This callback can be used to read some information that is requested by a remote node and provides the data back to the remote node. In the application demo,the `Vendor_ReadLocalDataCb` function is the callback where the read commands is processed. The command/ data flow is illustrated in the figure below.

Figure 75. Vendor model read command from a remote node



The response data from the node is sent via the `BLEMesh_SendResponse` function.

6.6 Application functions and callbacks

6.6.1 User interface and indications

Table 41. Appli_LedCtrl

Function	Description
Prototype	`void Appli_LedCtrl(void)`
Behavior description	Makes the onboard LED flash. This function is used at power-on and to draw attention to an error condition
Input parameter	None
Output parameter	void

6.6.2 User and button interface

Table 42. Appli_ShortButtonPress

Function	Description
Prototype	`static void Appli_ShortButtonPress(void)`
Behavior description	Call when a button is pressed for short duration
Input parameter	void
Output parameter	void

Table 43. Appli_LongButtonPress

Function	Description
Prototype	`static void Appli_LongButtonPress(void)`
Behavior description	Call when a button is pressed for long duration
Input parameter	void
Output parameter	void

Table 44. Appli_UpdateButtonState

Function	Description
Prototype	`static void Appli_UpdateButtonState(int isPressed)`
Behavior description	Update the button status
Input parameter	Int isPressed
Output parameter	void

6.6.3 Device BLE configuration type interface

This section explains the device configuration functions to be used in the network which are available to the application developer.

Table 45. Appli_BleSetTxPowerCb

Function	Description
Prototype	`MOBLE_RESULT Appli_BleSetTxPowerCb()`
Behavior description	This callback sets the transmission power of BLE radio. This function calls. `aci_hal_set_tx_power_level`. By default, the power level is set to +dbm
Input parameter	`none`
Output parameter	`MOBLE_RESULT` status of result

Table 46. Appli_BleSetUUIDCb

Function	Description
Prototype	`MOBLE_RESULT Appli_BleSetUUIDCb( MOBLEUINT8 *uuid_prefix_data);`
Behavior description	This callback sets the UUID value
Input parameter	`MOBLEUINT8 *uuid_prefix_data` pointer of UUID buffer data
Output parameter	`MOBLE_RESULT` status of result

Table 47. Appli_BleSetProductInfoCB

Function	Description
Prototype	`MOBLE_RESULT Appli_BleSetProductInfoCB( MOBLEUINT8 *company_product_info);`
Behavior description	This callback sets the CID , PID and VID values
Input parameter	`MOBLEUINT8 *company_product_info` pointer on the vendor fill product information
Output parameter	`MOBLE_RESULT` status of result

Table 48. Appli_BleUnprovisionedIdentifyCb

Function	Description
Prototype	`void Appli_BleUnprovisionedIdentifyCb( MOBLEUINT8 data);`
Behavior description	Callback for unprovisioned node identification
Input parameter	`MOBLEUINT8 data` node identification
Output parameter	`MOBLE_RESULT` status of result

Table 49. Appli_BleGattConnectionCompleteCb

Function	Description
Prototype	`void Appli_BleGattConnectionCompleteCb(void)`
Behavior description	This function is called when a GATT connection is detected by the node. The application uses this callback to indicate to the user the node is connected to the smartphone
Input parameter	`void`
Output parameter	`void`

Table 50. Appli_BleGattDisconnectionCompleteCb

Function	Description
Prototype	`void Appli_BleGattDisconnectionCompleteCb(void)`
Behavior description	This function is called when a GATT disconnection is detected by the node. The application uses this callback to indicate to the user that node is no longer connected to the smartphone
Input parameter	`void`
Output parameter	`void`

Table 51. Appli_BleSetNumberOfElementsCb

Function	Description
Prototype	`MOBLEUINT8 Appli_BleSetNumberOfElementsCb(void)`
Behavior description	If the number of elements defined on application side in the `mesh_cfg_usr.h` file fits with the number of elements in the mesh library, return this number else return mesh library element number if over, or 1 if null.
Input parameter	`void`
Output parameter	`MOBLEUINT8` number of element defined on the application side in the `mesh_cfg_usr.h`.

Table 52. Appli_BleAttentionTimerCb

Function	Description
Prototype	`MOBLE_RESULT Appli_BleAttentionTimerCb(void);`
Behavior description	Application Attention Timer Callback function
Input parameter	`void`
Output parameter	`MOBLE_RESULT` status of result

Table 53. Appli_BleOutputOOBAuthCb

Function	Description

Prototype	`void Appli_BleOutputOOBAuthCb(`
	`MOBLEUINT8* output_oob,`
	`MOBLEUINT8 size);`

Behavior description	Call back function to give Output OOB information
Input parameter	`MOBLEUINT8* output_oob` pointer on output OOB information `MOBLEUINT8 size` size of output OOB information
Output parameter	`void`

Table 54. Appli_BleInputOOBAuthCb

Function	Description
Prototype	`void Appli_BleInputOOBAuthCb(` `MOBLEUINT8 size);`
Behavior description	Call back function to provide Input OOB information
Input parameter	`MOBLEUINT8 size` size of input OOB information
Output parameter	`void`

Table 55. Appli_BleDisableFilterCb

Function	Description
Prototype	`MOBLEUINT8 Appli_BleDisableFilterCb(void)`
Behavior description	Call back function to provide disable filter state
Input parameter	`void`
Output parameter	`void`

Table 56. BLEMesh_PbAdvLinkCloseCb

Function	Description
Prototype	`void BLEMesh_PbAdvLinkCloseCb(void)`
Behavior description	Call back function called when PB-ADV link is closed
Input parameter	`void`
Output parameter	`void`

Table 57. Appli_LedStateCtrlCb

Function	Description
Prototype	`MOBLE_RESULT Appli_LedStateCtrlCb(` `MOBLEUINT16 ctrl)`
Behavior description	Callback function Sets the state of the bulb
Input parameter	`MOBLEUINT16 ctrl` which sets the lighting state of LED
Output parameter	`MOBLE_RESULT` status of result

6.6.4 Vendor model network data communication functions

The functions explained below help the developer manage the network data communication and take the appropriate actions.

Table 58. Vendor_WriteLocalDataCb

Function	Description

Prototype	`MOBLE_RESULT Vendor_WriteLocalDataCb(`
	`MODEL_MessageHeader_t *pmsgParams,`
	`MOBLEUINT8 command,`
	`MOBLEUINT8 const *data,`
	`MOBLEUINT32 length,`
	`MOBLEBOOL response)`

Behavior description	Call back function when an action is required on the node itself.
Input parameter	`MODEL_MessageHeader_t *pmsgParams`: pointer to structure of message header for parameters: • elementIndex, src, dst addresses, TTL, RSSI, NetKey and AppKey Offset. • `MOBLEUINT8 command`: receiving command code. • `MOBLEUINT8 const *data`: pointer to the data received from peer_addr. • `MOBLEUINT32 length`: length of the data. • `MOBLEBOOL response`: 1 if command acknowledgment is expected by peer_addr.
Output parameter	`MOBLE_RESULT` status of result

Table 59. Vendor_ReadLocalDataCb

Function	Description

Prototype	`MOBLE_RESULT Vendor_WriteLocalDataCb(`
	`MODEL_MessageHeader_t *pmsgParams,`
	`MOBLEUINT8 command,`
	`MOBLEUINT8 const *data,`
	`MOBLEUINT32 length,`
	`MOBLEBOOL response)`

Behavior description	Call back function when an action is required on node itself.
Input parameter	`MODEL_MessageHeader_t *pmsgParams`: pointer to the parameters of the message header structure : • elementIndex, src, dst addresses, TTL, RSSI, NetKey and AppKey Offset • `MOBLEUINT8 command`: receiving command code • `MOBLEUINT8 const *data`: pointer to the data received from peer_addr • `MOBLEUINT32 length`: length of the data • `MOBLEBOOL response`: 1 if command acknowledgment is expected by peer_addr
Output parameter	`MOBLE_RESULT` status of result

6.6.5 MAC address configuration

Table 60. Appli_CheckBdMacAddr

Function	Description
Prototype	`int Appli_CheckBdMacAddr(void)`
Behavior description	Checks MAC address validity
Input parameter	`void`
Output parameter	int status of result

Table 61. Appli_GetMACFromUniqueNumber

Function	Description
Prototype	`#ifdef INTERNAL_UNIQUE_NUMBER_MAC` `static void Appli_GetMACfromUniqueNumber(void)`
Behavior description	Reads the OTP BD address of the device and generates the MAC address from it.
Input parameter	`void`
Output parameter	`void`

6.6.6 BLE mesh node configuration

The following tables list the functions to configure the node to be used in the network.

Table 62. BLEMesh_InitUnprovisionedNode

Function	Description
Prototype	`MOBLE_Result BLEMesh_InitUnprovisionedNode(void)`
Behavior description	Initializes unprovisionned nodes.
Input parameter	`void`
Output parameter	MOBLE_RESULT status of result.

Table 63. BLEMesh_InitProvisionedNode

Function	Description
Prototype	`MOBLE_Result BLEMesh_InitProvisionedNode(void)`
Behavior description	Initializes provisionned nodes.
Input parameter	`void`
Output parameter	MOBLE_RESULT status of result.

Table 64. BLEMesh_IsUnprovisioned

Function	Description
Prototype	`MOBLEBOOL BLEMesh_IsUnprovisioned(void)`
Behavior description	Check if the node is configured as an unprovisioned node.
Input parameter	`void`
Output parameter	`MOBLEBOOL` MOBLE_TRUE if the node is configured as an unprovisioned node. MOBLE_FALSE otherwise.

Table 65. BLEMesh_Unprovision

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_Unprovision(void)`
Behavior description	Check if node configures as an unprovisioned node.
Input parameter	`void`
Output parameter	`MOBLEBOOL` MOBLE_TRUE if the node is configured as an unprovisioned node. MOBLE_FALSE otherwise.

Table 66. BLEMesh_BleHardwareInitCallBack

Function	Description
	`MOBLE_RESULT BLEMesh_BleHardwareInitCallBack( MOBLE_USER_BLE_CB_MAP const * _cb);`
Behavior description	BLE Hardware init callback.
Input parameter	`MOBLE_USER_BLE_CB_MAP const * _cb` pointer on callback functions.
Output parameter	`MOBLE_RESULT` status of the result.

Table 67. BLEMesh_GetUnprovisionState

Function	Description
Prototype	`MOBLEUINT8 BLEMesh_GetUnprovisionState(void)`
Behavior description	Get the provisioning process state.
Input parameter	`MOBLE_USER_BLE_CB_MAP const * _cb` pointer on callback functions.
Output parameter	`MOBLEUINT8` state of the provisioning process.

Table 68. BLEMesh_GetAddress

Function	Description
Prototype	`MOBLE_ADDRESS BLEMesh_GetAddress (void)`
Behavior description	Gets node mesh address. If the node is unprovisioned then `MOBLE_ADDRESS_UNASSIGNED` is returned.
Input parameter	`void`
Output parameter	`MOBLE_ADDRESS` if the node is unprovisioned return. `MOBLE_ADDRESS_UNASSIGNED` else returns the node mesh address.

Table 69. BLEMesh_GetPublishAddress

Function	Description
Prototype	`MOBLE_ADDRESS BLEMesh_GetPublishAddress( MOBLEUINT8 elementIndex, MOBLEUINT32 modelId);`
Behavior description	Gets the element node publish address.
Input parameter	`MOBLEUINT8 element index` Element index. `MOBLEUINT32 modelId` BT SIG model identifier.
Output parameter	`MOBLE_ADDRESS` element node publish address of the node.

Table 70. BLEMesh_GetSubscriptionAddress

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_GetSubscriptionAddress( MOBLE_ADDRESS *addressList, MOBLEUINT8 *sizeOfList, MOBLEUINT8 elementIndex, MOBLEUINT32 modelId)`
Behavior description	Gets the node subscription addresses.
Input parameter	`MOBLE_ADDRESS *addressList` pointer on the subscription address list found. `MOBLEUINT8 *sizeOfList` pointer on the size of the subscription address list found. `MOBLEUINT8 elementIndex` element index. `MOBLEUINT32 modelId` BT SIG model identifier.
Output parameter	`MOBLE_RESULT` status of result.

Table 71. BLEMesh_SetTTL

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetTTL ( MOBLEUINT8 ttl)`
Behavior description	Sets the default TTL value. When a message is sent to the mesh network, it contains a TTL field. The user must call this function to set the TTL value used during message transmission.
Input parameter	`MOBLEUINT8 ttl` TTL value. The supported values are 0-127.
Output parameter	MOBLE_RESULT status of the result.

Table 72. BLEMesh_GetTTL

Function	Description
Prototype	`MOBLEUINT8 BLEMesh_GetTTL (void)`
Behavior description	Get default TTL value.
Input parameter	`void`
Output parameter	`MOBLEUINT8` default TTL value.

Table 73. BLEMesh_SetNetworkTransmitCount

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetNetworkTransmitCount (MOBLEUINT8 count)`
Behavior description	Set Network Transmit Count value. When the message is sent to mesh network, it is replicated `NetworkTransmitCount` + 1 times. The user shall call this function to set the network transmit value used during message transmission.
Input parameter	`MOBLEUINT8 count`, network transmit value. Supported values are 1-8.
Output parameter	`MOBLE_RESULT` status of the result.

Table 74. BLEMesh_GetNetworkTransmitCount

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_GetNetworkTransmitCount (void)`
Behavior description	Returns the default `NetworkTransmitCount` value.
Input parameter	`void`
Output parameter	`MOBLE_RESULT` status of the result.

Table 75. BLEMesh_SetRelayRetransmitCount

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetRelayRetransmitCount (MOBLEUINT8 count)`
Behavior description	Set Relay Retransmit Count value. When the message is relayed by the mesh network relay, it is replicated `RelayRetransmitCount` + 1 times. User must call this function to set Relay Retransmit value used during message transmission.
Input parameter	`MOBLEUINT8 count` Relay Retransmit value. Supported values are 1-8.
Output parameter	`MOBLE_RESULT` status of result. .

Table 76. BLEMesh_GetRelayRetransmitCount

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_GetRelayRetransmitCount(void)`
Behavior description	Get Relay Retransmit Count value
Input parameter	`void`
Output parameter	`MOBLEUINT8` Default Relay Retransmit Count value.

Table 77. BLEMesh_SetRelayFeatureState

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetRelayFeatureState( MOBLEUINT8 state)`
Behavior description	Enable or disable the relay feature. Features can only be changed if it is supported.

Function	Description
Input parameter	MOBLEUINT8 state values: • 0 - disable • 1 - enable.
Output parameter	MOBLE_RESULT status of result.

Table 78. BLEMesh_SetFriendFeatureState

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetFriendFeatureState( MOBLEUINT8 state)`
Behavior description	Enable or disable the friend feature. This feature is only changed if it is supported.
Input parameter	MOBLEUINT8 state values: • 0 - disable • 1 - enable.
Output parameter	MOBLE_RESULT status of the result.

Table 79. BLEMesh_SetLowPowerFeatureState

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetLowPowerFeatureState( MOBLEUINT8 state);`
Behavior description	Disable low power feature only if it is supported and enabled. The low power feature cannot be enabled using BLEMesh_SetLowPowerFeatureState.
Input parameter	MOBLEUINT8 state values: • 0 - disable • 1 - enable.
Output parameter	MOBLE_RESULT status of the result.

Table 80. BLEMesh_GetFeatures

Function	Description
Prototype	`MOBLEUINT16 BLEMesh_GetFeatures(void)`
Behavior description	Get features state.
Input parameter	void
Output parameter	MOBLEUINT16 feature state bitmap: • Bit 0 for the relay feature: – 0 - disabled – 1 – enabled. • Bit 1 for the proxy feature: – 0 - disabled – 1 – enabled. • Bit 2 for the friend feature: – 0 - disabled – 1 – enabled. • Bit 3 for the low power feature: • 0 - disabled

Function	Description
	• 1 - enabled.

Table 81. BLEMesh_TrsptIsBusyState

Function	Description
Prototype	`MOBLEBOOL BLEMesh_TrsptIsBusyState(void)`
Behavior description	Get status of transmission in process state.
Input parameter	`void`
Output parameter	`MOBLEBOOL` gives the status of the packet transmission: • MOBLE_TRUE: for a pending transmission • MOBLE_FALSE: no pending transmission.

Table 82. BLEMesh_SetHeartbeatCallback

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetHeartbeatCallback (MOBLE_HEARTBEAT_CB cb)`
Behavior description	Set the callback for handling heartbeat messages.
Input parameter	`MOBLE_HEARTBEAT_CB cb` Callback
Output parameter	`MOBLE RESULT` status of the result.

Table 83. BLEMesh_SetAttentionTimerCallback

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetAttentionTimerCallback (MOBLE_ATTENTION_TIMER_CB cb)`
Behavior description	Set callback for attention timer. To be used for attention during provisioning and for health model. For devices which want to implement actions corresponding to the attention timer, set callback else do not set callback.
Input parameter	`MOBLE_ATTENTION_TIMER_CB cb` Callback
Output parameter	`MOBLE RESULT` status of the result.

Table 84. BLEMesh_UnprovisionCallback

Function	Description
Prototype	`void BLEMesh_UnprovisionCallback(MOBLEUINT8 reason)`
Behavior description	Callback on unprovision by provisioner.
Input parameter	`MOBLEUINT8 reason` unprovisioning reason.
Output parameter	`void`

Table 85. BLEMesh_ProvisionCallback

Function	Description
Prototype	`void BLEMesh_ProvisionCallback(void)`
Behavior description	Callback on provision by provisioner.
Input parameter	`void`
Output parameter	`void`

Table 86. BLEMesh_PbAdvLinkOpenCb

Function	Description
Prototype	`void BLEMesh_PbAdvLinkOpenCb(void)`
Behavior description	Call back function called when PB-ADV link is opened.
Input parameter	`void`
Output parameter	`void`

Table 87. BLEMesh_PbAdvLinkCloseCb

Function	Description
Prototype	`void BLEMesh_PbAdvLinkCloseCb (void)`
Behavior description	Call back function called when PB-ADV link closed.
Input parameter	`void`
Output parameter	`void`

Table 88. BLEMesh_ProvisionRemote

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_ProvisionRemote(MOBLEUINT8 uuid[16]);`
Behavior description	Provisioning of a node from the provisioner.
Input parameter	`MOBLEUINT8 uuid[16]` UUID of the unprovisioned node.
Output parameter	`MOBLE RESULT` status of the result.

Table 89. BLEMesh_CreateNetwork

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_CreateNetwork(MOBLEUINT8 *devKey)`
Behavior description	Creates credentials for the provisioner.
Input parameter	`MOBLEUINT8 *devKey` pointer on a device key.
Output parameter	`MOBLE RESULT` status of the result.

Table 90. BLEMesh_SetSIGModelsCbMap

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetSIGModelsCbMap( const MODEL_SIG_cb_t* pSig_cb, MOBLEUINT32 count);`
Behavior description	Set BT SIG model callback map.
Input parameter	`const MODEL_SIG_cb_t* pSig_cb` pointer on the callback map. If NULL, nothing is done. `MOBLEUINT32 count` number of models defined in the application.
Output parameter	`MOBLE RESULT` status of the result.

Table 91. GetApplicationVendorModels

Function	Description
Prototype	`void GetApplicationVendorModels( const MODEL_Vendor_cb_t** pModelsTable, MOBLEUINT32*VendorModelscount);`
Behavior description	`Get the Application Vendor Models.`
Input parameter	`const MODEL_Vendor_cb_t** pModelsTable` pointer in the pointers table of the Vendor model callbacks. `MOBLEUINT32* VendorModelscount` pointer on the number of Vendor modells defined in application.
Output parameter	`void`

Table 92. BLEMesh_GetSleepDuration

Function	Description
Prototype	`MOBLEUINT32 BLEMesh_GetSleepDuration(void)`
Behavior description	Returns the sleep duration.
Input parameter	`void`
Output parameter	`MOBLEUINT32` sleep duration in milliseconds.

Table 93. BLEMesh_UpperTesterDataProcess

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_UpperTesterDataProcess( MOBLEUINT8 testFunctionIndex, MOBLEUINT8* testFunctionParm);`
Behavior description	Upper tester data process used for BT SIG profile tuning test certification.
Input parameter	`MOBLEUINT8 testFunctionIndex` test function indexes used to set the context of the BT SIG profile tuning test certification. `MOBLEUINT8* testFunctionParm` pointer on the test function parameters.
Output parameter	`MOBLE RESULT` status of the result.

Table 94. BLEMesh_PrintStringCb

Function	Description
Prototype	`void BLEMesh_PrintStringCb(const char *message)`
Behavior description	String print callback function.
Input parameter	`const char *message` pointer on string to be printed.
Output parameter	`MOBLE_RESULT` status of the result.

Table 95. BLEMesh_PrintDataCb

Function	Description

Prototype	`void BLEMesh_PrintDataCb(`
	`MOBLEUINT8* data,`
	`MOBLEUINT16 size)`

Behavior description	Data print callback function.
Input parameter	`MOBLEUINT8* data` pointer on data to be printed. `MOBLEUINT16 size` size of Data to be printed.
Output parameter	`void`

Table 96. BLEMesh_FnFriendshipEstablishedCallback

Function	Description

Prototype	`void`
	`BLEMesh_FnFriendshipEstablishedCallback(`
	`MOBLE_ADDRESS lpnAddress,`
	`MOBLEUINT8 lpnReceiveDelay,`
	`MOBLEUINT32 lpnPollTimeout,`
	`MOBLEUINT8 lpnNumElements,`
	`MOBLE_ADDRESS lpnPrevFriendAddress)`

Behavior description	Friend node callback corresponding to a friendship established with a low-power node.
Input parameter	`MOBLE_ADDRESS lpnAddress` address of corresponding low-power node, `MOBLEUINT8 lpnReceiveDelay` receives the delay of the low-power node (unit ms), `MOBLEUINT32 lpnPollTimeout` poll timeout of the low-power node (unit 100 ms), `MOBLEUINT8 lpnNumElements` number of elements of the low-power node, `MOBLE_ADDRESS lpnPrevFriendAddress` previous friend address of low-power node (can be an invalid address).
Output parameter	`void`

Table 97. BLEMesh_FnFriendshipClearedCallback

Function	Description

Prototype	`void`
	`BLEMesh_FnFriendshipClearedCallback(`
	`MOBLEUINT8 reason,`
	`MOBLE_ADDRESS lpnAddress);`

Behavior description	Friend node callback corresponding to a friendship cleared with low-power node.

Function	Description
Input parameter	`MOBLEUINT8 reason` reason for friendship clear: • 0: reserved • 1: friend request received from existing low-power node (friend) • 2: low-power node poll timeout occurred • 3: friend clear received. `MOBLE_ADDRESS lpnAddress` address of corresponding low-power node.
Output parameter	`void`

Table 98. BLEMesh_LpnFriendshipEstablishedCallback

Function	Description
Prototype	`void BLEMesh_LpnFriendshipEstablishedCallback( MOBLE_ADDRESS fnAddress)`
Behavior description	low-power node callback corresponding to a friendship established with Friend node.
Input parameter	`MOBLE_ADDRESS fnAddress` address of corresponding Friend node.
Output parameter	`void`

Table 99. BLEMesh_LpnFriendshipClearedCallback

Function	Description
Prototype	`void BLEMesh_LpnFriendshipClearedCallback( MOBLEUINT8 reason, MOBLE_ADDRESS fnAddress)`
Behavior description	low-power node callback corresponding to a friendship cleared with a Friend node.
Input parameter	`MOBLEUINT8 reason` reason of friendship clear: • 0: reserved • 1: No response received from Friend node. `MOBLE_ADDRESS fnAddress` address of corresponding Friend node.
Output parameter	`void`

Table 100. BLEMesh_LpnDisableScan

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_LpnDisableScan(void)`
Behavior description	Disable continuous scan Applicable only for provisioned low power feature enabled node.
Input parameter	`void`
Output parameter	`MOBLE RESULT` status of the result

Table 101. BLEMesh_StopAdvScan

Function	Description
Prototype	`void BLEMesh_StopAdvScan(void)`

Function	Description
Behavior description	Stops ongoing scan (if in scan mode) and advertisement (if in advertising mode).
Input parameter	`void`
Output parameter	`void`

Table 102. BLEMesh_SetProvisioningServAdvInterval

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetProvisioningServAdvInterval( MOBLEUINT16 interval)`
Behavior description	Set advertising interval of provisioning service: • 0 value results in stop • Default value: 1000 ms. Actual value is the interval value plus random (16).
Input parameter	`MOBLEUINT16 interval` is the advertising interval (ms), minimum interval value is 100 ms.
Output parameter	`MOBLE RESULT` status of the result.

Table 103. BLEMesh_SetUnprovisionedDevBeaconInterval

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetUnprovisionedDevBeaconInterval( MOBLEUINT16 interval)`
Behavior description	Set advertising interval of unprovisioned device beacon: • 0 value results in stop • Default value: 1000 ms. Actual value is the interval value plus random (16).
Input parameter	`MOBLEUINT16 interval` is the advertising interval (ms), min interval value is 100 ms.
Output parameter	`MOBLE RESULT` status of the result

Table 104. BLEMesh_SetProxyServAdvInterval

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetProxyServAdvInterval( MOBLEUINT16 interval)`
Behavior description	Set advertising interval of proxy service: • 0 value results in stop • Default value: 1000 ms. Actual value is the interval value plus random (16).
Input parameter	`MOBLEUINT16` is the advertising interval (ms), min interval value is 100 ms.
Output parameter	`MOBLE RESULT` status of the result.

Table 105. BLEMesh_SetSecureBeaconInterval

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetSecureBeaconInterval( MOBLEUINT16 interval)`
Behavior description	Set the connection interval of secure network beacon. Default value: 10000 ms. Actual value is the interval value plus random (128).
Input parameter	`MOBLEUINT16 interval` connection interval (ms) of beacons, min interval value is 10000 ms.
Output parameter	MOBLE RESULT status of the result.

Table 106. BLEMesh_SetCustomBeaconInterval

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetCustomBeaconInterval( MOBLEUINT16 interval)`
Behavior description	Sets the interval of custom beacon: • 0 value results in stop. • Actual value is the interval value plus random (128).
Input parameter	`MOBLEUINT16 interval` connection interval (ms) of the beacons, min interval value is 10000 ms.
Output parameter	`MOBLE RESULT` status of the result.

Table 107. BLEMesh_CustomBeaconGeneratorCallback

Function	Description
Prototype	`void BLEMesh_CustomBeaconGeneratorCallback( void* buffer, MOBLEUINT8* size)`
Behavior description	Sets the custom beacon data. If the size is greater than 31 bytes, beacon is rejected .
Input parameter	`void* buffer` pointer on the beacon data buffer. It includes: • length • adtype • data. `MOBLEUINT8* size` size of buffer.
Output parameter	void

Table 108. BLEMesh_CustomBeaconReceivedCallback

Function	Description
Prototype	`void BLEMesh_CustomBeaconReceivedCallback( const MOBLEUINT8* bdAddr, const MOBLEUINT8* data, MOBLEUINT8 length, MOBLEINT8 rssi)`

Function	Description
Behavior description	Callback to receive non-mesh beacons. Beacons are received only if a received beacon ad type is not mesh message, mesh beacon, or PB-ADV.
Input parameter	`const MOBLEUINT8* bdAddr` MAC address, `const MOBLEUINT8* data` pointer on data, `MOBLEUINT8 length` length of data, `MOBLEINT8 rssi` Received Signal Strength Indicator.
Output parameter	`void`

Table 109. ApplicationGetSigModelList

Function	Description
Prototype	`MOBLEUINT8 ApplicationGetSigModelList( MOBLEUINT16* pModels_sig_ID, MOBLEUINT8 elementIndex)`
Behavior description	This function provides the Server BT SIG models list.
Input parameter	`MOBLEUINT16* pModels_sig_ID` pointer of the array to be filled with SIG Models list, `MOBLEUINT8 elementIndex` index of the element for the model list.
Output parameter	`MOBLEUINT8` Count of the SIG model servers enabled in the application.

Table 110. ApplicationGetCLIENTSigModelList

Function	Description
Prototype	`MOBLEUINT8 ApplicationGetCLIENTSigModelList(MOBLEUINT16* pModels_sig_ID, MOBLEUINT8 elementIndex)`
Behavior description	It provides the list of the Client SIG models to the calling function.
Input parameter	`MOBLEUINT16* pModels_sig_ID` pointer of the array to be filled with Client SIG Models list, `MOBLEUINT8 elementIndex` index of the element for model list.
Output parameter	`MOBLEUINT8` Count of the SIG Model Clients enabled in the Application.

Table 111. BLEMeshSetSelfModelList

Function	Description
Prototype	`MOBLEUINT8 BLEMeshSetSelfModelList( MOBLEUINT8 numberOfElements)`
Behavior description	It sets the list of the SIG Models in the MESH library
Input parameter	`MOBLEUINT8 numberOfElements` number of elements set by user in `mesh_cfg_usr.h`
Output parameter	`MOBLEUINT8` Count of the SIG Model Clients enabled in the Application.

Table 112. ApplicationGetVendorModelList

Function	Description
Prototype	`MOBLEUINT8 ApplicationGetVendorModelList( MOBLEUINT32* pModels_vendor_ID, MOBLEUINT8 elementIndex)`
Behavior description	It provides the list of the Vendor models
Input parameter	`MOBLEUINT32* pModels_vendor_ID` pointer of the array to be filled with the Vendor model list, `MOBLEUINT8 elementIndex` is the index of the element for model list
Output parameter	`MOBLEUINT8` is the count of the Vendor models enabled in the application

Table 113. ApplicationChkSigModelActive

Function	Description
Prototype	`MOBLEBOOL ApplicationChkSigModelActive( MOBLEUINT16 modelID, MOBLEUINT8 elementIndex)`
Behavior description	Checks if a specific Model Server is active in the Model Server list.
Input parameter	`MOBLEUINT16 modelID` Model Server ID received for the checking function, `MOBLEUINT8 elementIndex` index of element for which active model checking is needed.
Output parameter	`MOBLEBOOL` True or False, if the Server ID matches with the list.

Table 114. ApplicationChkVendorModelActive

Function	Description
Prototype	`MOBLEBOOL ApplicationChkVendorModelActive( MOBLEUINT32 modelID, MOBLEUINT8 elementIndex)`
Behavior description	Checks if a specific Vendor model Server is active in the Model Server list
Input parameter	`MOBLEUINT16 modelID` Model Server ID received for the checking function, `MOBLEUINT8 elementIndex` index of element for which active model checking is needed.
Output parameter	`MOBLEBOOL` True or False, if the Server ID matches with the list.

Table 115. ApplicationGetConfigServerDeviceKey

Function	Description
Prototype	`MOBLE_RESULT ApplicationGetConfigServerDeviceKey( MOBLE_ADDRESS src, const MOBLEUINT8 **ppkeyTbUse)`
Behavior description	Provides the device key to the node from Application.
Input parameter	`MOBLE_ADDRESS src` node address node address, `const MOBLEUINT8 **ppkeyTbUse` pointer on device key.

Function	Description
Output parameter	`MOBLE_RESULT` status of the result.

Table 116. BLEMesh_NeighborAppearedCallback

Function	Description

Prototype	`void BLEMesh_NeighborAppearedCallback(`
	`const MOBLEUINT8* bdAddr,`
	`MOBLEBOOL provisioned,`
	`const MOBLEUINT8* uuid,`
	`MOBLE_ADDRESS networkAddress,`
	`MOBLEINT8 rssi)`

Behavior description	New neighbor appears as callback in neighbor table.
Input parameter	`const MOBLEUINT8* bdAddr` MAC address of neighbor, `MOBLEBOOL provisioned` is a provisioned or unprovisioned neighboring device, `const MOBLEUINT8* uuid` uuid of neighbor. NULL if not available, `MOBLE_ADDRESS networkAddress` network address of neighbor. MOBLE_ADDRESS_UNASSIGNED if not available, `MOBLEINT8 rssi` last updated Received Signal Strength Indicator value.
Output parameter	`void`

Table 117. BLEMesh_NeighborRefreshedCallback

Function	Description

Prototype	`void BLEMesh_NeighborRefreshedCallback(`
	`const MOBLEUINT8* bdAddr,`
	`MOBLEBOOL provisioned,`
	`const MOBLEUINT8* uuid,`
	`MOBLE_ADDRESS networkAddress,`
	`MOBLEINT8 rssi)`

Behavior description	New neighbor appears callback in neighbor table.
Input parameter	`const MOBLEUINT8* bdAddr` MAC address of neighbor, `MOBLEBOOL provisioned` is a provisioned or unprovisioned neighboring device, `const MOBLEUINT8* uuid` uuid of neighbor. NULL if not available, `MOBLE_ADDRESS networkAddress` network address of neighbor. MOBLE_ADDRESS_UNASSIGNED if not available, `MOBLEINT8 rssi` last updated Received Signal Strength Indicator value.
Output parameter	`void`

Table 118. BLEMesh_GetNeighborState

Function	Description

Prototype	`MOBLE_RESULT BLEMesh_GetNeighborState(`
	`neighbor_params_t* pNeighborTable,`
	`MOBLEUINT8* pNoOfNeighborPresent)`

Behavior description	Get neighbor table status.
Input parameter	`neighbor_params_t* pNeighborTable` pointer to application buff is updated with neighbor table parameters, `MOBLEUINT8* pNoOfNeighborPresent` pointer on number of neighbors present.
Output parameter	`MOBLE RESULT` status of the result.

Table 119. BLEMesh_SetFault

Function	Description

Prototype	`MOBLE_RESULT BLEMesh_SetFault(`
	`MOBLEUINT8 *pFaultArray,`
	`MOBLEUINT8 faultArraySize)`

Behavior description	Set system faults. It is be used by Health Model. Supporting all Bluetooth® assigned FaultValues.
Input parameter	`MOBLEUINT8 *pFaultArray` FaultValue Array pointer. (FaultValue Range: 0x01–0x32), `MOBLEUINT8 faultArraySize` Size of the fault array. Max supported array size is 5.
Output parameter	`MOBLE RESULT` status of the result

Table 120. BLEMesh_ClearFault

Function	Description

Prototype	`MOBLE_RESULT BLEMesh_ClearFault(`
	`MOBLEUINT8 *pFaultArray,`
	`MOBLEUINT8 faultArraySize);`

Behavior description	Clears all currently set system faults. Is used by Health Model.
Input parameter	`MOBLEUINT8 *pFaultArray` FaultValue Array pointer. (FaultValue Range: 0x01–0x32), `MOBLEUINT8 faultArraySize` Size of the fault array. Max supported array size is 5.
Output parameter	`MOBLE RESULT` status of the result

Table 121. BLEMesh_Shutdown

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_Shutdown(void)`
Behavior description	It is called to shutdown Bluetooth® Low Energy mesh library To resume the operation, `BLEMesh_Resume` should be called.
Input parameter	`void`
Output parameter	`MOBLE RESULT` status of the result

Table 122. **BLEMesh_Resume**

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_Resume(void)`
Behavior description	It is called to restore previously shutdown Bluetooth® Low Energy mesh library in order to resume library operation.
Input parameter	`void`
Output parameter	`MOBLE_RESULT` status of Result

6.6.7 BLE mesh library configuration

The following tables list the functions to configure and initialize the ST BLE Mesh library.

Table 123. **BLEMesh_Init**

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_Init(` `Const Mesh_Initialization_t* pInit_params)`
Behavior description	This function initializes the ST BLE Mesh library Other ST BLE Mesh library functions should NOT be called until the library is initialized.
Input parameter	`Const Mesh_Initialization_t* pInit_params` device name, Bdaddr,fFeatures to be supported by library, low-power node,friendship and dynamic buffer parameters.
Output parameter	`MOBLE_RESULT` status of result.

Table 124. **BLEMesh_GetLibraryVersion**

Function	Description
Prototype	`Char * BLEMesh_GetLibraryVersiont(void)`
Behavior description	To get the latest library version
Input parameter	`void`
Output parameter	`Char *` string

Table 125. **BLEMesh_GetLibrarySubVersion**

Function	Description
Prototype	`Char * BLEMesh_GetLibrarySubVersiont(void)`
Behavior description	To get the latest library sub version
Input parameter	`void`
Output parameter	`Char *` string

Table 126. **BLEMesh_Process**

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_Process(void)`
Behavior description	mesh task processing function. This function must be called in user application task.
Input parameter	`void`

Function	Description
Output parameter	MOBLE_RESULT status of result.

Table 127. BLEMesh_SetVendorCbMap

Function	Description
Prototype	`MOBLE_RESULT BLEMesh_SetVendorCbMap(` `MOBLE_VENDOR_CB_MAP const * map);`
Behavior description	Sets the vendor callback map
Input parameter	MOBLE_VENDOR_CB_MAP const * map callback map. If NULL, nothing is done.
Output parameter	MOBLE_RESULT status of result.

Table 128. BLEMesh_SetRemoteData

Function	Description

Prototype	`MOBLE_RESULT BLEMesh_SetRemoteData(`
	`MOBLE_ADDRESS peer,`
	`MOBLEUINT8 elementIndex,`
	`MOBLEUINT16 command,`
	`MOBLEUINT8 const * data,`
	`MOBLEUINT32 length,`
	`MOBLEBOOL response,MOBLEUINT8 isVendor)`

Behavior description	This function sets the remote data on the given peer. This API is depracated and replaced with BLEMesh_SetRemotePublication The user is responsible for serializing data into a data buffer. Vendor_WriteLocalDataCb callback is called on the remote device.
Input parameter	MOBLE_ADDRESS peer peer destination address, MOBLEUINT8 elementIndex element index, MOBLEUINT8 command vendor model commands, MOBLEUINT8 const * data pointer on data buffer, MOBLEUINT32 length length of data in bytes, MOBLEBOOL response response (If not '0', used to get the response. If '0', no response), MOBLEUINT8 isVendor for vendor model.
Output parameter	MOBLE_RESULT status of result.

Table 129. BLEMesh_SetRemotePublication

Function	Description

Prototype	`MOBLE_RESULT`
	`BLEMesh_SetRemotePublication(`
	`MOBLEUINT32 modelId,`
	`MOBLE_ADDRESS srcAddress,`
	`MOBLEUINT16 command,`
	`MOBLEUINT8 const * data,`
	`MOBLEUINT32 length,`
	`MOBLEBOOL response,`
	`MOBLEUINT8 isVendor)`

Behavior description	This function sets the remote publication for the given model ID to publish Address

Function	Description
	The user is responsible for serializing data into a data buffer. `Vendor_WriteLocalDataCb` callback is called on the remote device.
Input parameter	`MOBLEUINT32 modelId` BT SIG model ID, `MOBLE_ADDRESS srcAddress` element address of the node, `MOBLEUINT16 command` vendor model commands, `MOBLEUINT8 const * data` pointer on data buffer, `MOBLEUINT32 length` length of data in bytes, `MOBLEBOOL response` if 'MOBLE_TRUE', used to get the response. If 'MOBLE_FALSE', no response, `MOBLEUINT8 isVendor` for vendor model.
Output parameter	`MOBLE RESULT` status of result.

Table 130. BLEMesh_ModelSendMessage

Function	Description

Prototype	`MOBLE_RESULT`
	`BLEMesh_ModelSendMessage(`
	`MOBLE_ADDRESS srcAddress,`
	`MOBLE_ADDRESS peerAddress,`
	`MOBLEUINT32 modelId,`
	`MOBLEUINT16 opcode,`
	`const MOBLEUINT8 *pData,`
	`MOBLEUINT32 length)`

Behavior description	This function sends a message for given BT SIG model ID and node address. The user is responsible for serializing data into a data buffer.
Input parameter	`MOBLE_ADDRESS srcAddress` element address of the node, `MOBLE_ADDRESS peerAddress` address of targeted node(s), `MOBLEUINT32 modelId` BT SIG model ID, `MOBLEUINT16 opcode` opcode of message, `const MOBLEUINT8 *pData` pointer on Data buffer, `MOBLEUINT32 length` length of data in bytes
Output parameter	`MOBLE RESULT` status of result.

Table 131. BLEMesh_ReadRemoteData

Function	Description

Prototype	`MOBLE_RESULT`
	`BLEMesh_ReadRemoteData(`
	`MODEL_MessageHeader_t *pmsgParam,`
	`MOBLEUINT16 command,`
	`MOBLEUINT8 const * data,`
	`MOBLEUINT32 length)`

Behavior description	Reads remote data on the given peer User is responsible for serializing data into a data buffer. `Vendor_ReadLocalDataCb` callback is called on the remote device. It is reliable command

Function	Description
Input parameter	`MODEL_MessageHeader_t *pmsgParam` pointer to structure of message header for parameters: • elementIndex • src, dst addresses • TTL • RSSI • NetKey • AppKey Offset `MOBLEUINT16 command` vendor model commands, `MOBLEUINT8 const * data` pointer on data buffer, `MOBLEUINT32 length` length of data in bytes
Output parameter	`MOBLE_RESULT` status of result.

Table 132. BLEMesh_SendResponse

Function	Description

Prototype	`MOBLE_RESULT`
	`BLEMesh_SendResponse(`
	`MOBLE_ADDRESS peer,`
	`MOBLEUINT8 status,`
	`MOBLEUINT8 const * data,`
	`MOBLEUINT32 length)`

Behavior description	This functions sends a response on received packet. The usage of this API is depracated and replaced with `VendorModel_SendResponse`
Input parameter	`MOBLE_ADDRESS peer` peer destination address, `MOBLEUINT8 status` status of response, `MOBLEUINT8 const * data` pointer on data buffer, `MOBLEUINT32 length` length of data in bytes.
Output parameter	`MOBLE_RESULT` status of result.

Table 133. VendorModel_SendResponse

Function	Description

Prototype	`MOBLE_RESULT`
	`VendorModel_SendResponse(`
	`MOBLEUINT16 vendorModelId,`
	`MODEL_MessageHeader_t *pmsgParams,`
	`MOBLEUINT8 status,`
	`MOBLEUINT8 const * data,`
	`MOBLEUINT32 length)`

Behavior description	This function sends a response on received packet.
Input parameter	`MOBLEUINT16 vendorModelId` vendor model ID, `MODEL_MessageHeader_t *pmsgParams` pointer to structure of message header for parameters: • src, dst addresses • TTL • RSSI • NetKey • AppKey Offset,

Function	Description
	MOBLEUINT8 status status of response, MOBLEUINT8 const * data pointer on data buffer, MOBLEUINT32 length length of data in bytes. Maximum accepted length is 8. If length is zero, no associated data is sent with the report.
Output parameter	MOBLE_RESULT status of result.

Table 134. Model_SendResponse

Function	Description
Prototype	MOBLE_RESULT Model_SendResponse(MODEL_MessageHeader_t *pmsgParams, MOBLEUINT8 opcode, MOBLEUINT8 const * data, MOBLEUINT32 length)
Behavior description	BT SIG model send response on received packet.
Input parameter	MODEL_MessageHeader_t *pmsgParams Pointer to structure of message header for parameters: • src • dst addresses • TTL • RSSI • NetKey • AppKey Offset, MOBLEUINT8 opcode opcode to be send, MOBLEUINT8 const * data pointer on data buffer, MOBLEUINT32 length length of data in bytes. Maximum accepted length is 8. If length is zero, no associated data is sent with the report.
Output parameter	MOBLE_RESULT status of result.

Table 135. ConfigModel_SendMessage

Function	Description
Prototype	MOBLE_RESULT ConfigModel_SendMessage(MOBLE_ADDRESS src_peer, MOBLE_ADDRESS dst_peer, MOBLEUINT16 opcode, MOBLEUINT8 *pData, MOBLEUINT32 length, MOBLEUINT8 *pTargetDevKey);
Behavior description	Configuration model send message to the remote device.
Input parameter	MOBLE_ADDRESS src_peer peer source address. Must be a device address (0b0xxx xxxx xxxx xxxx, but not 0), MOBLE_ADDRESS dst_peer peer destination address. Must be a device address (0b0xxx xxxx xxxx xxxx, but not 0), MOBLEUINT16 opcode opcode to be send, MOBLEUINT8 *pData pointer on data buffer,

Function	Description
	MOBLEUINT32 length length of data in bytes. Maximum accepted length is 8. If length is zero, no associated data is sent with the report, MOBLEUINT8 *pTargetDevKey pointer on target device key.
Output parameter	MOBLE_RESULT status of result.

Table 136. ConfigModel_SelfPublishConfig

Function	Description
Prototype	MOBLE_RESULT ConfigModel_SelfPublishConfig(MOBLE_ADDRESS dst_peer, MOBLEUINT16 opcode, MOBLEUINT8 *pData, MOBLEUINT32 length)
Behavior description	Publishes send to the provisioner
Input parameter	MOBLE_ADDRESS dst_peer the peer destination address is the provisioner address, MOBLEUINT16 opcode opcode to be send, MOBLEUINT8 *pData pointer on data buffer, MOBLEUINT32 length length of data in bytes.
Output parameter	MOBLE_RESULT status of result.

Table 137. ConfigModel_SelfSubscriptionConfig

Function	Description
Prototype	MOBLE_RESULT ConfigModel_SelfSubscriptionConfig(MOBLE_ADDRESS dst_peer, MOBLEUINT16 opcode, MOBLEUINT8 *pData, MOBLEUINT32 length)
Behavior description	Subscription send to the provisioner
Input parameter	MOBLE_ADDRESS dst_peer the peer destination address is the provisioner address, MOBLEUINT16 opcode opcode to be send, MOBLEUINT8 *pData pointer on the data buffer, MOBLEUINT32 length length of data in bytes.
Output parameter	MOBLE_RESULT status of result.

Table 138. ConfigClient_SelfModelAppBindConfig

Function	Description
Prototype	MOBLE_RESULT ConfigClient_SelfModelAppBindConfig(MOBLE_ADDRESS dst_peer, MOBLEUINT16 opcode, MOBLEUINT8 *pData, MOBLEUINT32 length)
Behavior description	Binding Send to the provisioner

Function	Description
Input parameter	`MOBLE_ADDRESS dst_peer` the peer destination address is provisioner address, `MOBLEUINT16 opcode` opcode to be send, `MOBLEUINT8 *pData` pointer on data buffer, `MOBLEUINT32 length` length of data in bytes. Maximum accepted length is 8. If length is zero, no associated data is sent with the report.
Output parameter	`MOBLE RESULT` status of result.

Table 139. ConfigClient_SelfModelAppBindConfig

Function	Description
Prototype	`MOBLE_RESULT ConfigClient_SelfModelAppBindConfig( MOBLE_ADDRESS dst_peer, MOBLEUINT16 opcode, MOBLEUINT8 *pData, MOBLEUINT32 length)`
Behavior description	Binding Send to the provisioner.
Input parameter	`MOBLE_ADDRESS dst_peer` the peer destination address is the provisioner address, `MOBLEUINT16 opcode` opcode to be send, `MOBLEUINT8 *pData` pointer on data buffer, `MOBLEUINT32 length` length of data in bytes. Maximum accepted length is 8. If length is zero, no associated data is sent with the report.
Output parameter	`MOBLE RESULT` status of result.

7 BLE mesh models

A model defines a set of states, state transitions, state bindings, messages, and other associated behaviors. An element within a node must support one or more models, and it is the model or models that define the functionality that an element has. There are a number of models that are defined by the Bluetooth® SIG, and many of them are deliberately positioned as “generic” models, having potential utility within a wide range of device types. Essentially, models are specifications for standard software components that, when included in a product, determine what it does as a mesh device. Models are self-contained components and products incorporate several of them. Collectively, from a network point of view, models make the device what it is.

Below is the list of models supported by our solution and their respective model ID:

Table 140. List of available models

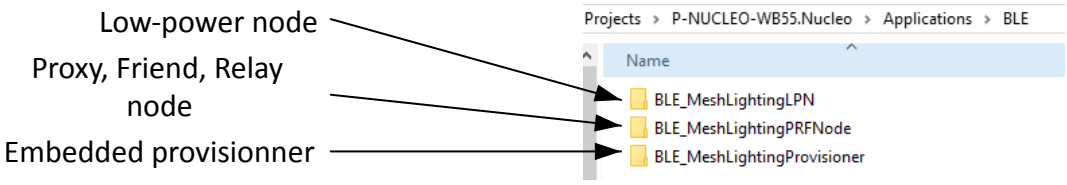
Model	SIG Model ID
Generic model	
Generic ON/OFF Server	0x1000
Generic ON/OFF Client	0x1001
Generic Level Server	0x1002
Generic Level Client	0x1003
Generic Default Transition Time Server	0x1004
Generic Default Transition Time Client	0x1005
Generic Power ON/OFF Server	0x1006
Generic Power ON/OFF Setup Server	0x1007
Generic Power ON/OFF Client	0x1008
Sensor models	
Sensor Server	0x1100
Sensor Setup Server	0x1101
Sensor Client	0x1102
Lighting models	
Light Lightness Server	0x1300
Light Lightness Setup Server	0x1301
Light Lightness Client	0x1302
Light CTL Server	0x1303
Light CTL Setup Server	0x1304
Light CTL Client	0x1305
Light CTL Temperature Server	0x1306
Light HSL Server	0x1307
Light HSL Setup Server	0x1308
Light HSL Client	0x1309
Light HSL Hue Server	0x130A
Light HSL Saturation Server	0x130B
Light LC Server	0x130F
Light LC Setup Server	0x1310
Light LC Client	0x1311

7.1 How to

Up to 7 models are supported per Element.

To enable or disable models, select the application project. The location of the projects is illustrated in Figure 76.

Figure 76. Model node locations



Then, open `mesh_cfg_usr.h` file and define the models that need enabling using the following procedure:

- Define the required model macro, for example `ENABLE_GENERIC_MODEL_SERVER_ONOFF` for Generic ON/OFF Server model.
- Select the node element with a bitmap as illustrated in the following list:
  - Bit N - 1 is dedicated to the element N...
  - Bit 2 is dedicated to the element 3
  - Bit 1 is dedicated to the element 2
  - Bit 0 is dedicated to the element 1.

  Where N is the `APPLICATION_NUMBER _OF_ELEMENTS`. The model can be enabled for several elements at the same time (refer to Section 8.2 Multiple element nodes for more information about node elements)

Generic ON/OFF Server model example

The example shown in the code sample below enables the following setup models for element 1:

```
#define ENABLE_GENERIC_MODEL_SERVER_ONOFF         (1)
#define ENABLE_GENERIC_MODEL_SERVER_LEVEL         (1)
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF         (1)
#define ENABLE_GENERIC_MODEL_SERVER_ONOFF_SETUP     (1)
//#define ENABLE_GENERIC_MODEL_SERVER_DEFAULT_TRANSITION_TIME     (1)
```

- Generic ON/OFF Server
- Generic Level Server
- Generic Power ON/OFF Server
- Generic Power ON/OFF Setup Server.

Once the files are defined and included, the next step is to build the example application and flash it on the board.

7.2 Generic models

The generics Bluetooth® mesh models are designed to be used by any kind of device, offering a set of common, generally applicable capabilities.

Available Generic Server models:

- Generic ON/OFF Server
- Generic Level Server
- Generic Default Transition Time Server
- Generic Power ON/OFF Server
- Generic Power ON/OFF Setup Server.

Available Generic Clientmodels:

- Generic ON/OFF Client
- Generic Level Client
- Generic Default Transition Time Client
- Generic Power ON/OFF Client

Generic ON/OFF model

The Generic ON/OFF models allow one device to switch other devices on or off.

The server model contains one state only: the Generic ON/OFF state.

In both cases, it is a simple Boolean state that indicates whether an element is currently switched on or off. A value of 0x00 means it is off, and a value of 0x01 means it is on. The Generic ON/OFF Client may send Generic ON/OFF Get, Generic ON/OFF set, or Generic ON/OFF set unacknowledged messages.

Generic Level model

Some devices can be turned up or down; lights can be dimmed and the temperature of a room can be changed by turning the thermostat up or down. The Generic Level models allow control to be exercised over the level of other devices.

The Generic Level server model contains a state called generic level that can be positive or negative and has a range of -32767 to +32767.

Generic Power ON/OFF model

The Generic Power ON/OFF models enable the initial state that a device is in immediately after powering up to be configured.

For example:

- One device may need to be turned off when the model is powered up, as indicated by a value of 0x00 in the Generic ON/OFF state.
- Another device, on the other hand, may need to be turned on, with Generic ON/OFF set to 0x01.

The Generic Power ON/OFF Server model has a single state, generic on powerup which has three values defined with meanings shown below :

- 0x00: Off. After being powered up, the element is in an off state.
- 0x01: Default. After being powered up. the element is in an on state and uses default state values.
- 0x02: Restore. If a transition was in progress when powered on, the element restores the target state when powered up. Otherwise, the element restores the state it was in when powered down.
- 0x03 – 0xFF: Prohibited.

State changes can either be instantaneous or they can take place over a specified period of time. There are two ways in which a non-instantaneous state change can be initiated. Many message types support an optional field called transition time and, if included in a message, determines the time it takes for a state change to be executed. In addition, the Generic Default Transition State, which might be available in the optional generic default transition time server model, can also be a source of transition time information for state changes.

7.3 Lighting models

The Bluetooth® mesh Lighting models allow control over the on/off state of lights, their lightness, color temperature, and their color (using various color spaces). Importantly, they also provide a highly sophisticated software-based lighting controller that enables smart lighting automation scenarios.

Available Light Server models:

- Light Lightness Server
- Light Lightness Setup Server
- Light CTL Server
- Light CTL Setup Server
- Light CTL Temperature Server
- Light HSL Server
- Light HSL Hue Server
- Light HSL Saturation Server
- Light HSL Setup Server
- Light LC Server
- Light LC Setup Server

Available Lighting Client models:

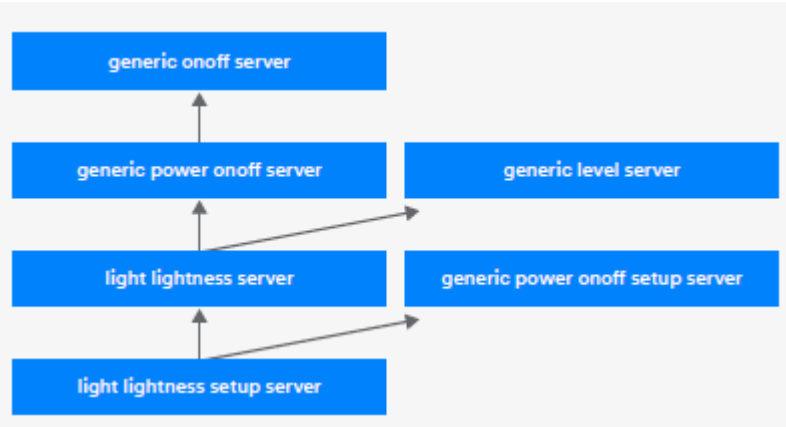
- Light Lightness Client
- Light CTL Client
- Light HSL Client
- Light LC Client

Light Lightness model

The Light Lightness models allow the lightness of a lamp to be controlled by mesh messages and events, such as powering up the device.

Figure 77 depicts the relationship the Light Lightness Server model has with other models. It extends any model depicted with an arrow pointing to it from this model directly or indirectly. It is extended by a model which has an arrow going to the light lightness server.

Figure 77. Light Lightness Server and associated models



Light LC model

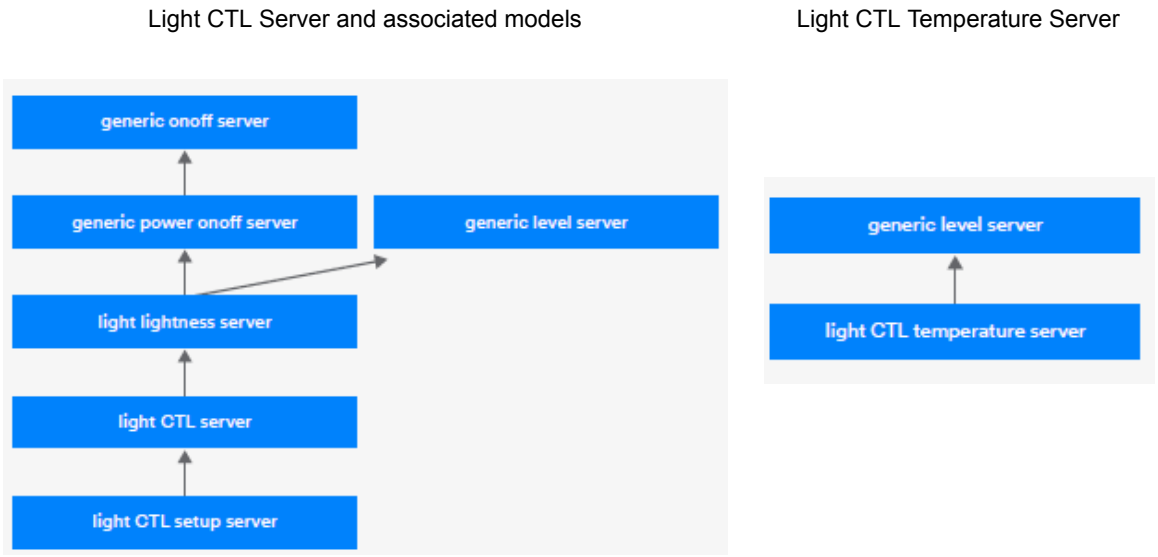
Collectively, the Lighting Control (LC) models form a lighting controller: a software component that allows sophisticated, sensor and user-driven lighting control to be set up. Occupancy and ambient light sensors are catered for so that techniques such as daylight harvesting can be implemented. As the state of the lighting controller changes, the light lightness state of the light under control progresses through a series of levels, with the transition from one level to another governed by configurable timing parameters so that changes are not abrupt and feel natural to building users.

Light CTL models

The light control (Light CTL) models allow the control of a tunable, white light source. Tunable white lights offer control over the color temperature of a white light and incorporates the most recent research into human biological and cognitive responses to light.

Figure 78 depicts the relationships the Light CTL Server model and Light CTL Temperature server model has with other models.

Figure 78. Light CTL Server and associated models (left), Light CTL Temperature Server (right)

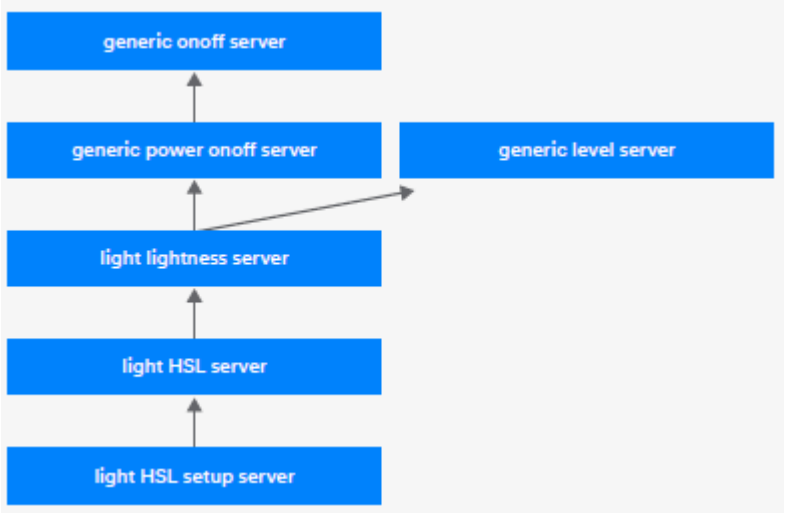


Light HSL models

The Light HSL models provide control over color-changing lights, using the hue/saturation/lightness (HSL) color representation model.

Figure 79 depicts the relationships the Light HSL Server model has with other models.

Figure 79. Light HSL Server and associated models



7.4 Sensor models

Sensors play a critical role in many mesh networking applications, including, but not limited to, that of the smart building.

They detect and report events like the changing occupancy status of rooms, and they measure attributes of the environment, sharing this data with other devices.

Sensor data is used to influence or control the operation of one particular type of device, or the data can be used to change the state of many other devices of many different types, all in one go.

Available Sensor models:

- Sensor models
- Sensor Setup Server
- Sensor Client

These models provide a generalized approach to sensor operation in a Bluetooth® mesh network and the models allow any type of sensor to communicate sensor readings to other nodes on the network. The Sensor Setup Server allows the sensor and format of its data to be configured.

The sensor models make extensive use of properties within a relatively small number of states.

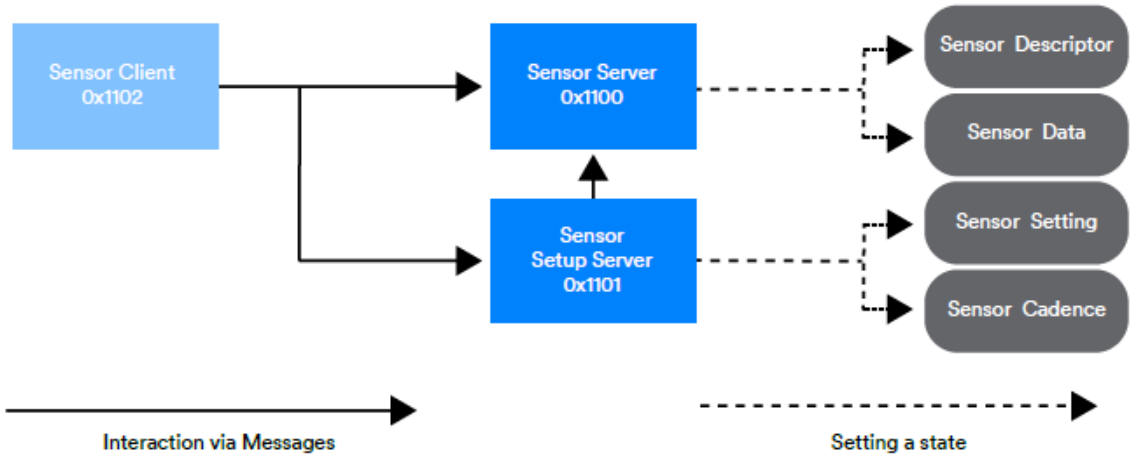
Properties differ from states in that they contain both an identifier and a value. The identifier tells us what type of data the property contains so that it is self-describing. States, on the other hand, have no explicit type identifier, and it is the model or message which contains the state that tells us the data state.

Leveraging properties has allowed the three sensor models to accommodate any type of sensor and sensor data, rather than requiring different models, messages, and states for each conceivable type of sensor that might be part of a network.

An element that implements the Sensor Server model must also have the Sensor Setup Server model, which extends it. The Sensor Client model is not related to other models and can be used standalone.

The following figure illustrates the relationship between the three sensor models.

Figure 80. Sensor related models



7.5 Vendor models

Models defined by the Bluetooth® Special Interest Group (SIG) in the Bluetooth® Mesh Model Specification are known as Bluetooth® SIG models.

Vendor companies may define their own models too, and these are known as Vendor models.

Vendor models must be used with caution and only when there is no possible way to use Bluetooth® SIG models to meet the requirements.

8 More concept around BLE mesh

The BLE mesh solution offers several features to cover a large amount of use case requirements. For example, the possibility to reduce power consumption to work with a small battery, or the fact of managing several elements on a same node.

All these features are described in the following sections.

8.1 Low-power and Friend nodes

8.1.1 Friendship definitions

Bluetooth® mesh networking includes various options to optimize power consumption and, in particular, a feature called “friendship”.

The major part of Bluetooth® mesh devices is connected to the main electricity power source and their power consumption is not a problem. But, some devices may have other constraints such as they are powered by a small battery or rely on energy harvesting.

The latter devices are most likely to leverage on the friendship concept of Bluetooth® mesh.

Friendship implementation requires two kinds of nodes, at least one Friend node, and one or more LPNs:

- Low-power feature: The ability to operate within a mesh network with significantly reduced receiver duty cycles. Minimizing the time the radio receiver is on leads to lower power consumption with the node only powering the receiver when it is strictly necessary. LPNs achieve this through establishing a friendship with a Friend node.
- Friend feature: The ability to help an LPN operate by storing messages destined for the LPN and only forwarding them to it when the LPN explicitly requests messages from the Friend node.

The principle of friendship is that a Friend node collects and stores messages intended to the associated LPNs, the LPNs then polls the friend for new messages at a lower frequency than it would otherwise need if it had to listen for messages all the time. This is how power is saved.

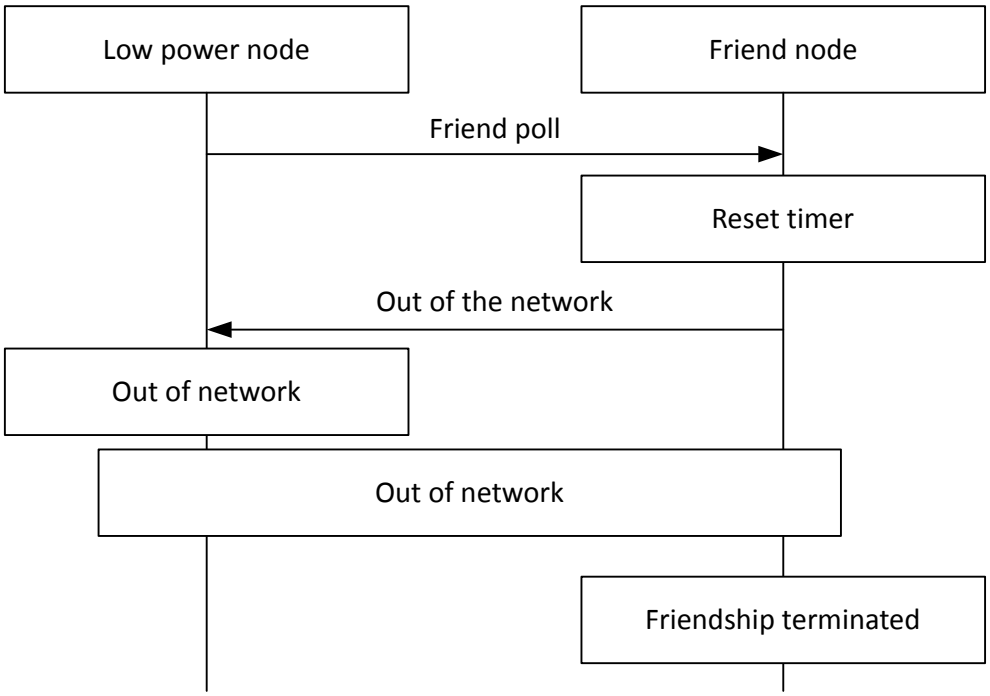
LPNs need to find Friend nodes and establish a friendship relationship with them. The procedure involved is called "friend establishment".

The following parameters are governing the behavior of LPNs, and are set during friend establishment:

- `ReceiveDelay` is the time which elapses between the LPN sending a request to the Friend node and it starting to listen for a response. This allows time for the Friend node to prepare its response and send it back.

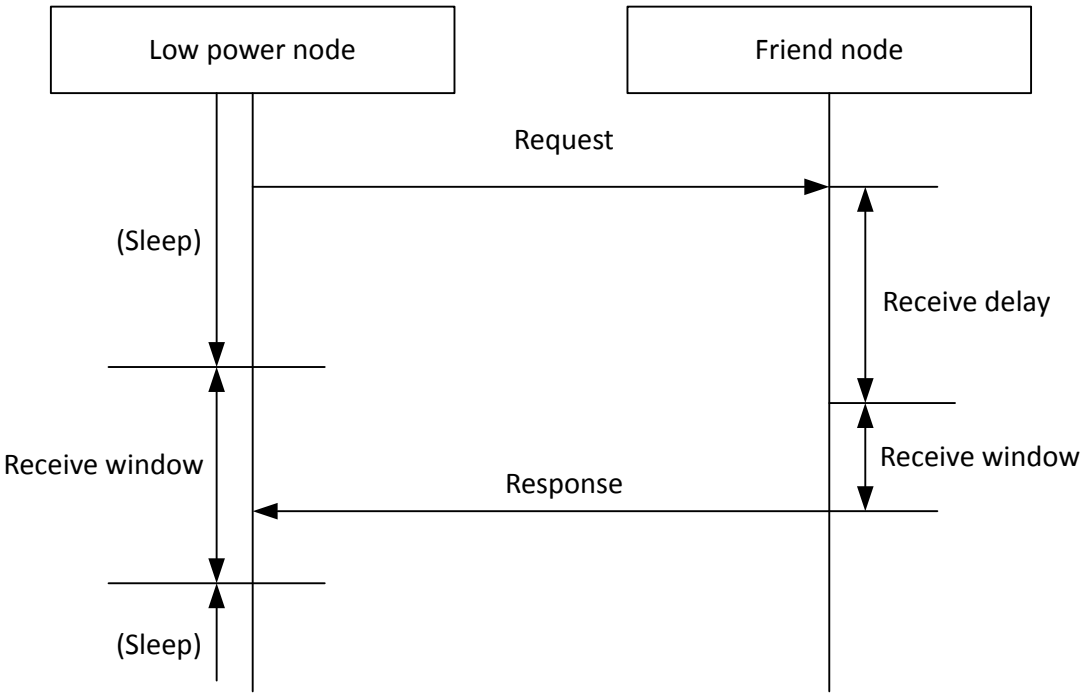
- `ReceiveWindow` is the time that the LPN spends listening for a response. Figure 81 illustrates the timing sequence involving both `ReceiveDelay` and `ReceiveWindow`.

Figure 81. ReceiveDelay and ReceiveWindow timing sequence



- `PollTimeout` establishes a maximum time which may elapse between two consecutive requests sent by the LPN to its Friend node. If no requests from an LPN are received by the Friend node before the `PollTimeout` timer expires, then the friendship is terminated. Figure 82 illustrates the timing sequence involving `PollTimeout` timing sequence.

Figure 82. PollTimeout timing sequence



The friendship establishment steps are listed below:

- The LPN publishes a friend request message. This message is not relayed and therefore only Friend nodes in direct radio range process it. Nodes which do not have the friend feature discard it. The friend request message includes the LPN's required `ReceiveDelay`, `ReceiveWindow` and `PollTimeout` parameters.
- Each nearby Friend node which supports the requirements specified in the friend request message prepares and transmit a friend offer message back to the LPN. This message includes various parameters, including the supported ReceiveWindow size, available message queue size, available subscription list size and the RSSI value measured by the Friend node.
- On receiving a friend offer message, the LPN selects a suitable Friend node by applying an implementation-specific algorithm. The algorithm is likely to consider various points. Some devices may place priority on the ReceiveWindow size to reduce power usage as much as possible, while some may be more concerned with the RSSI value in order to ensure they can maintain a good quality link with the Friend node. The precise algorithm used is left to the product developer to determine.
- After selecting a Friend node, the LPN sends a friend poll message to this Friend node.
- After receiving a friend poll message from the LPN, the Friend node replies with a friend update message, which completes the friend establishment procedure and provides the associated security parameters. At this point, friendship has been established.

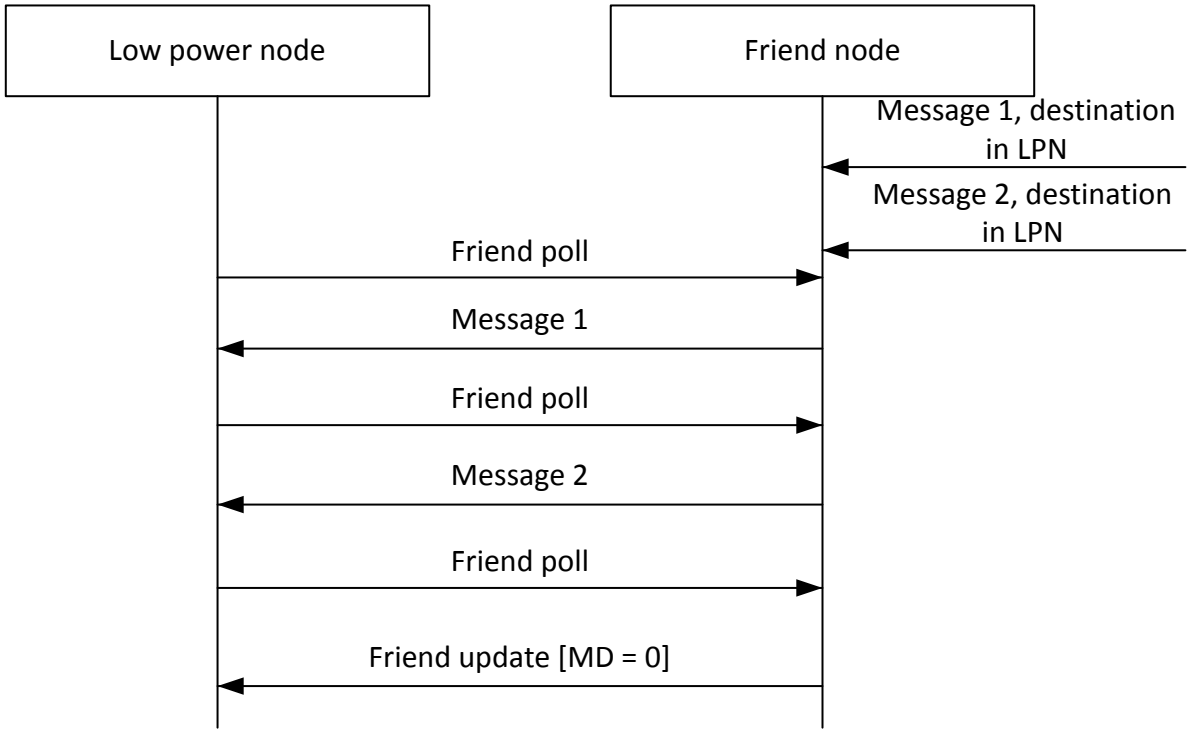
After friendship establishment, the Friend node stores all messages for the LPN in the friend queue. Collectively, these are known as stored messages.

Here are the friend messaging steps:

- When the Friend node receives a message addressed to the Friend node LPN, the Friend node buffers this message, storing it in an area called the friend queue. In Figure 83, message 1 and 2 are stored in the Friend node on behalf of the LPN.
- Periodically, the LPN enables its transceiver and sends a friend poll to the Friend node, asking for any buffered messages stored for it.
- The Friend node begins by sending one stored message back to the LPN as a response to the friend poll.

- The LPN continues sending friend poll messages after each received message from the Friend node until it receives a friend update message with the MD (more data) field set to 0. This means there are no more messages buffered for the LPN. At this point, the LPN stops polling the Friend node.

Figure 83. Friendship messages



From the security point of view, friendship uses two special security credentials:

- Master security material: Derived from the NetKey, it can also be used by the other nodes in the same network. Messages encrypted with the master security material are be decrypted by any node in the same network.
- Friend security material: Derived from the NetKey and some additional counter numbers which are generated by the LPN and the Friend node. Messages encrypted with the friend security material are only be decrypted by the friend and LPNs which possess it.

The corresponding friendship messages encrypted with the friend security material are:

- Friend poll
- Friend update
- Friend subscription list add/remove/confirm
- Stored messages that the Friend node delivers to the LPN.

The corresponding friendship messages encrypted with the master security material are:

- Friend clear
- Friend clear confirm

The messages sent from the LPN to the Friend node are encrypted with the master or friend security material depending on an application setting.

Friendship termination can occur in the any of the following conditions:

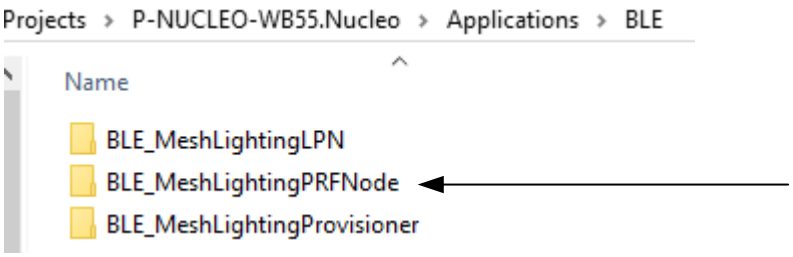
- If no friend poll, friend subscription list add or friend subscription list remove messages are received by the Friend node before the PollTimeout timer expires, the friendship is terminated.
- An LPN can initiate a friendship termination procedure by sending friend clear message to its Friend node, causing the friendship to be terminated by the Friend node.

8.1.2 Setup

Friend node

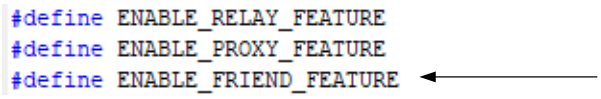
From projects workspace, select the Proxy/Relay/Friend node (BLE_MeshLightingPRFNode) project as illustrated in Figure 84.

Figure 84. Friend node project workspace selection



Open the project in an appropriate IDE and check features enabled in `mesh_cfg_usr.h` as illustrated in Figure 85

Figure 85. Friend feature selection



The number of LPNs that can be associated with a Friend node is defined by `FN_NO_OF_LPNS`, see Figure 86.

Figure 86. FN_NO_OF_LPNS location

```
/*
*  Number of Low power nodes that can be associated with Friend node
*  varies from 1 to 10
*/
#define FN_NO_OF_LPNS                                          6U
```

Note: LPN stands for Low-power node.

The number of LPN / Friend node associations is limited by:

- 1KB SRAM
- Radio time.

Some Friend node parameters must stay fixed:

- Friend receive window size: 255 ms
- Friend subscription list size: 4
- Friend max. no. of messages: 16.

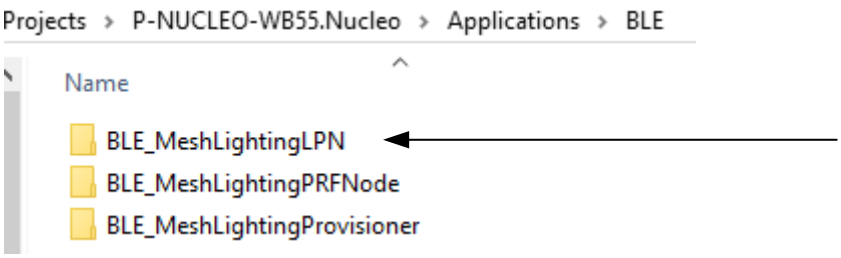
Build and flash the binary on the board.

Provision the node using ST BLE Mesh application (Android™/iOS™) or embedded provisioner (see Section 8.3).

Low-power nodes

From projects workspace, select the LPN project (BLE_MeshLightingLPN) as illustrated in Figure 87.

Figure 87. **LPN project selection**



Open the project in an appropriate IDE and check the features enabled in `mesh_cfg_usr.h` as illustrated in Figure 88.

Figure 88. **Low-power feature**

```
#define ENABLE_LOW_POWER_FEATURE
```

Note: An LPN should not support any other features (Proxy, Friend, Relay).

Build and flash binary on the board.

Provision the node using ST BLE Mesh application (Android™/iOS™) or Embedded provisioner (See Section 8.3).

Friendship demonstration

This quick BLE mesh network demonstration with LPNs requires three boards as detailed in Table 141.

Table 141. **Demonstration board breakdown**

Board	Blocks	Group Subscription list	Publication address
1	LPN1	Default	Default
2	LPN2	Default	Default
3	Friend node	Default	Default

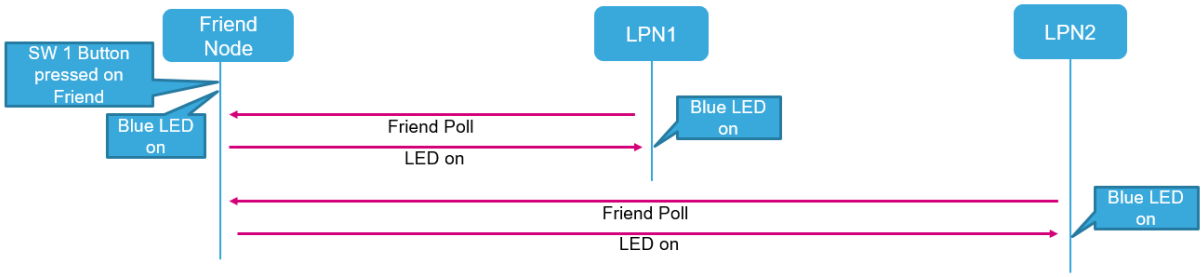
Note: After provisioning, it might take a few seconds for the friendship to be established between the friend and the LPNs.

In this demo, the friend establishes a friendship with LPN1 and LPN2 after having provisioned the three boards as illustrated in Figure 89.

Pressing SW1 button on the Friend node:

- The friend publishes LED ON command to the default group.
- The friend blue LED immediately turns on (the friend subscribed to the default group).
- The friend receives a poll from an LPN and sends LED ON to the respective LPNs.

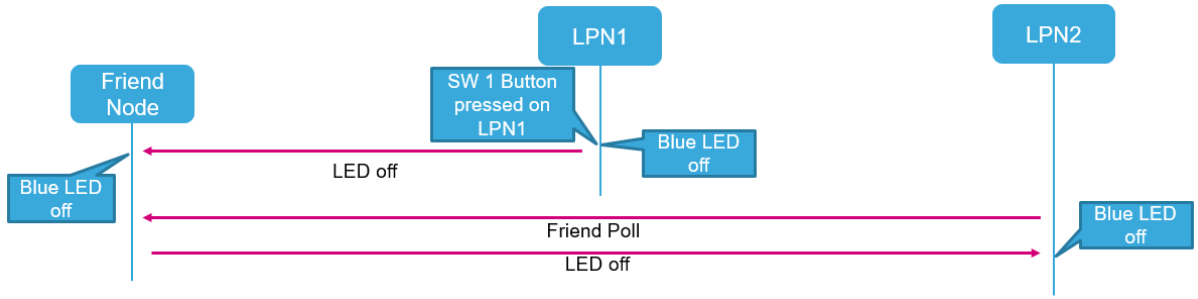
Figure 89. Pressing SW1 button on Friend node, command flow illustration



Pressing SW1 button on LPN1 and the process is illustrated in Figure 90:

- LPN1 publishes LED OFF command to the default group.
- The LPN1 blue LED turns off (LPN1 subscribed to the default group).
- LPN1 immediately forwards the LED OFF command to the default group.
- The friend receives an LED OFF command (subscribed to the default group).
- The friend also sends a LED OFF command to LPN2 (subscribed to the default group) as soon as friend receives a poll from LPN2.
- LPN2 blue LED turns off.

Figure 90. Pressing SW1 button on LPN1, command flow illustration



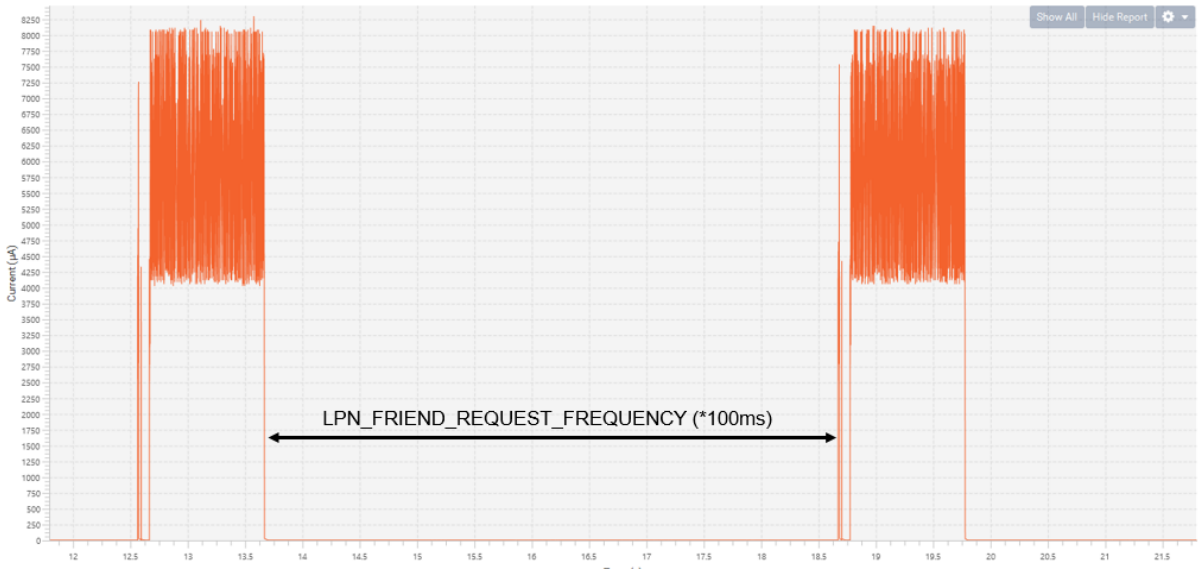
Each LPN polls the Friend node and process the commands which are received. When an LPN send a command to the network nodes, the friend receives it and process it immediately, but the other LPN on the network has to poll the Friend node to receive and process the command.

8.1.3 Power consumption profiles

This section shows and describes the different energy consumption profiles of an LPN, depending on the friendship parameters defines in the friend and LPN code.

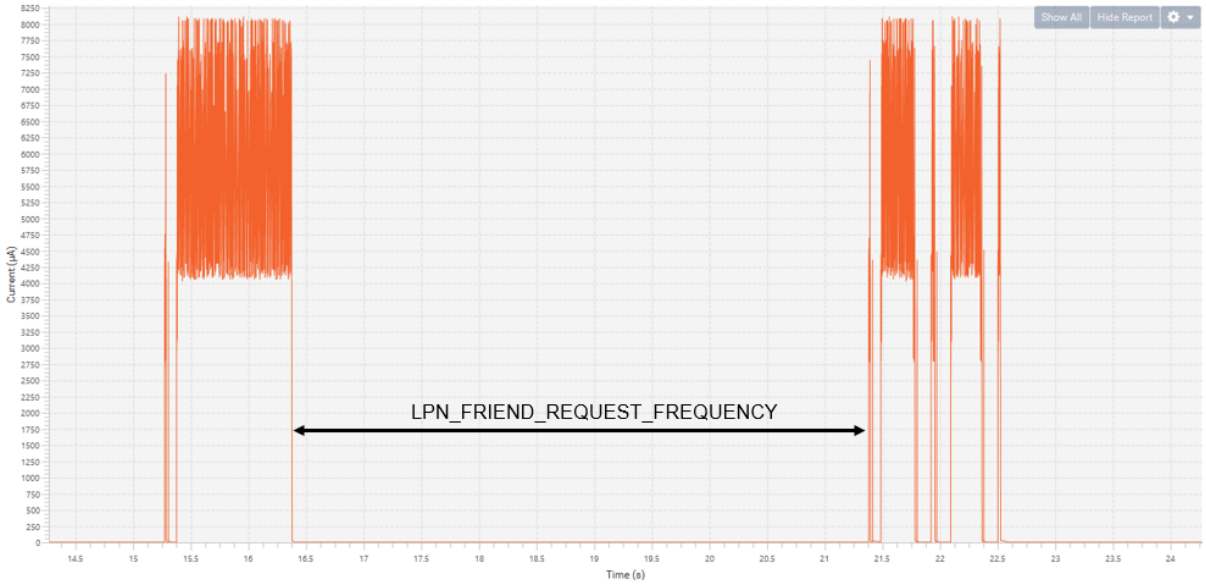
1. The LPN sends a friend request, but no answer from friend, see Figure 91.

Figure 91. Low-power friend request process with no answer



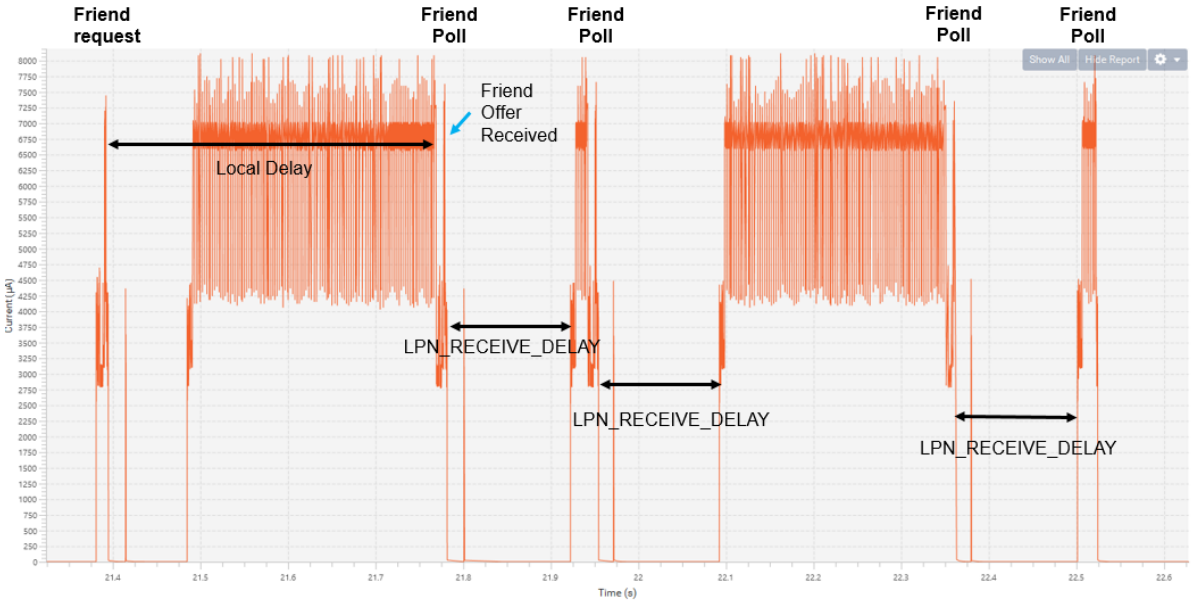
2. The LPN sends a friend request and the Friend node answers, a friendship is established, see Figure 92:

Figure 92. Low-power friend request process with answer



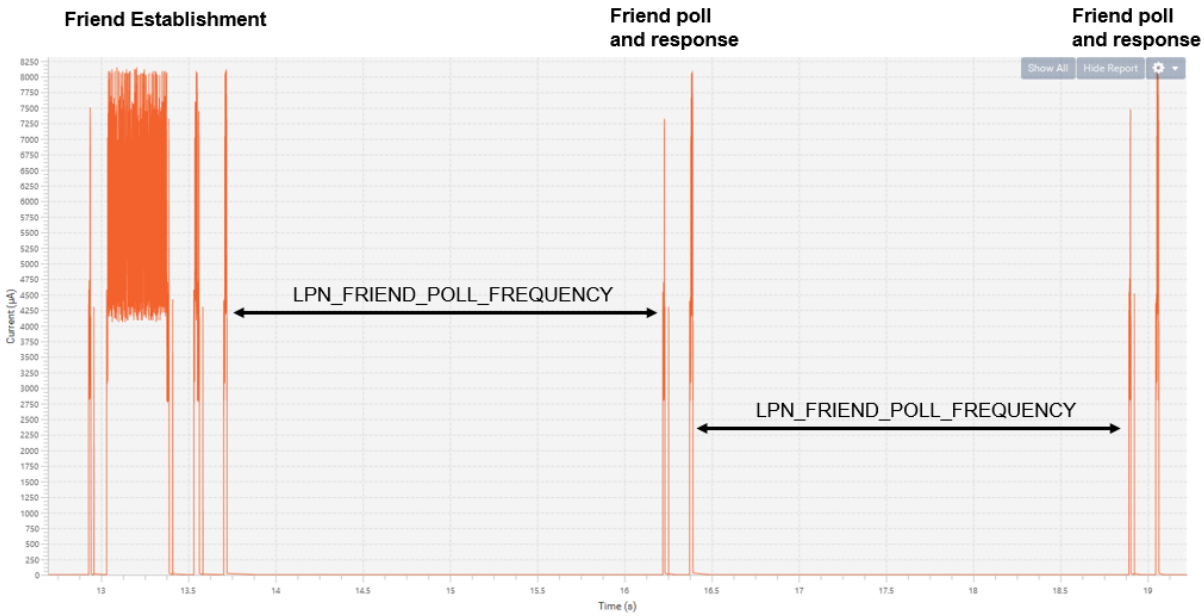
3. Details of the friendship establishment sequence, see Figure 93:

Figure 93. Friendship establishment sequence



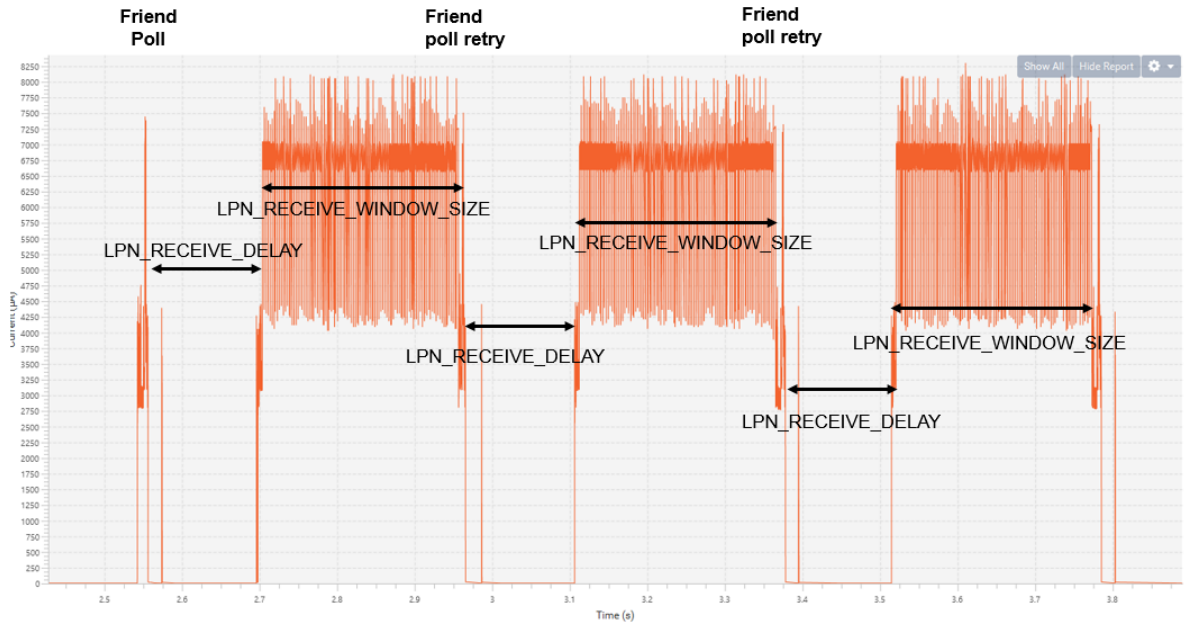
4. Friend poll after establishing a friendship, see Figure 94:

Figure 94. Friend poll after establishing a friendship



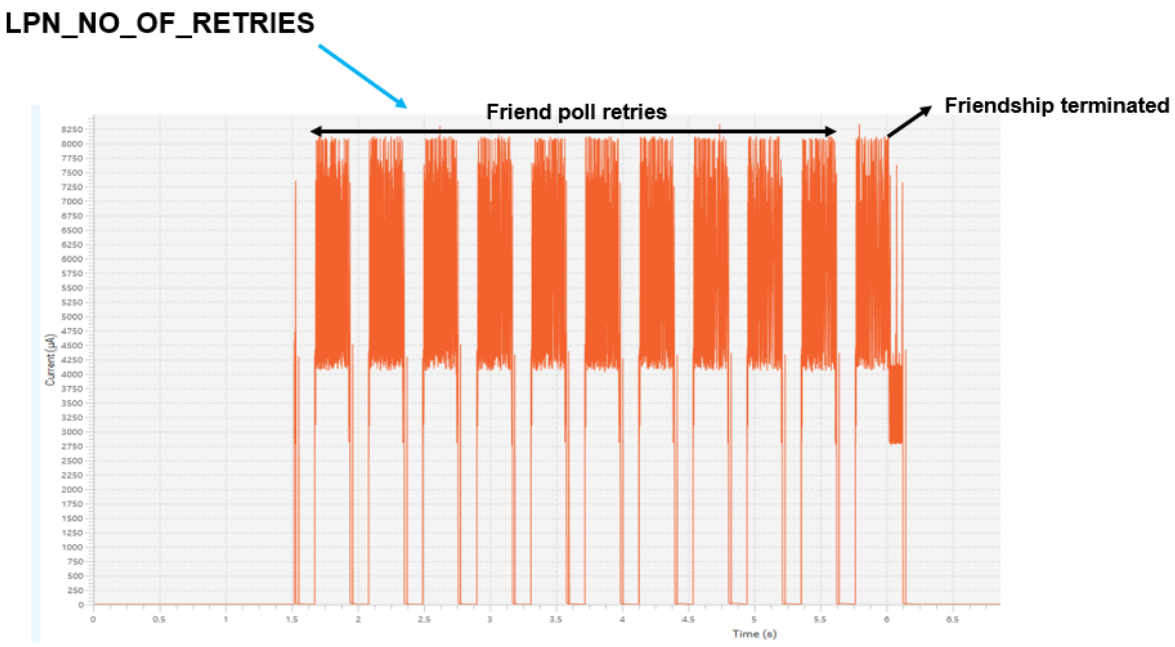
5. Friend poll retries, see Figure 95:

Figure 95. Friend poll retries



6. Friend poll retries and friendship termination, see Figure 96:

Figure 96. Friend poll retries and friendship termination



8.2 Multiple element nodes

An element is an addressable entity within a device.

This means that it is some part of a device which has at least one, independent state value which represents some conditions of the element.

ST BLE Mesh nodes must have at least one element and up to 8 elements.

The number and structure of the elements is static and does not change throughout the lifetime of a node. If needs to change, then the node must be reprovisioned first.

An element is not allowed to contain multiple instances of models that use the same message.

To implement multiple instances of overlapping models within a single node, the node must contain multiple elements.

Light fixture example

A light fixture may have two lamps, each implementing an instance of the Light Lightness Server model and an instance of the Generic Power ON/OFF Server.

This requires that the node contains two elements, one for each lamp.

To setup a node containing multiple elements, open the desired project and go to `mesh_cfg_usr.h` file. Modify the line `#define APPLICATION_NUMBER_OF_ELEMENTS` with an element number from 1 to 8.

Once the number of elements is selected the number of elements for the node, the required models must be defined for each element, using the bitmap defined as below and illustrated in :

- Bit N - 1 is dedicated to the element N...
- Bit 2 is dedicated to the element 3
- Bit 1 is dedicated to the element 2
- Bit 0 is dedicated to the element 1.

Figure 97 illustrates the element numbering definition.

Figure 97. Element numbering definition

```
#define ENABLE_GENERIC_MODEL_SERVER_ONOFF                    (1)
#define ENABLE_GENERIC_MODEL_SERVER_LEVEL                    (2)
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF              (4)
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF_SETUP        (3)
#define ENABLE_GENERIC_MODEL_SERVER_DEFAULT_TRANSITION_TIME  (7)
```

In this case:

- GENERIC_MODEL_SERVER_ONOFF is enabled for the Element 1
- GENERIC_MODEL_SERVER_LEVEL is enabled for the Element 2
- GENERIC_MODEL_SERVER_POWER_ONOFF is enabled for the Element 3
- GENERIC_MODEL_SERVER_POWER_ONOFF_SETUP is enabled for the Elements 1 and 2
- GENERIC_MODEL_SERVER_DEFAULT_TRANSITION_TIME is enabled for the 3 Elements.

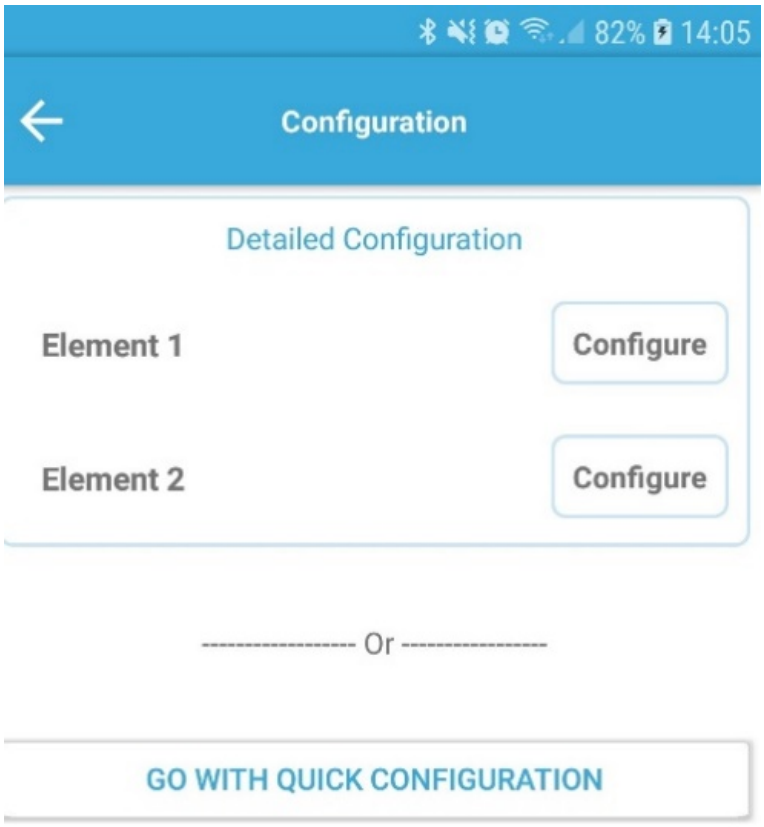
Note: Each element must have at least one model enabled.

After having defined the models for each element, build and flash the application.

Then open the smartphone application and process to provisioning and model binding.

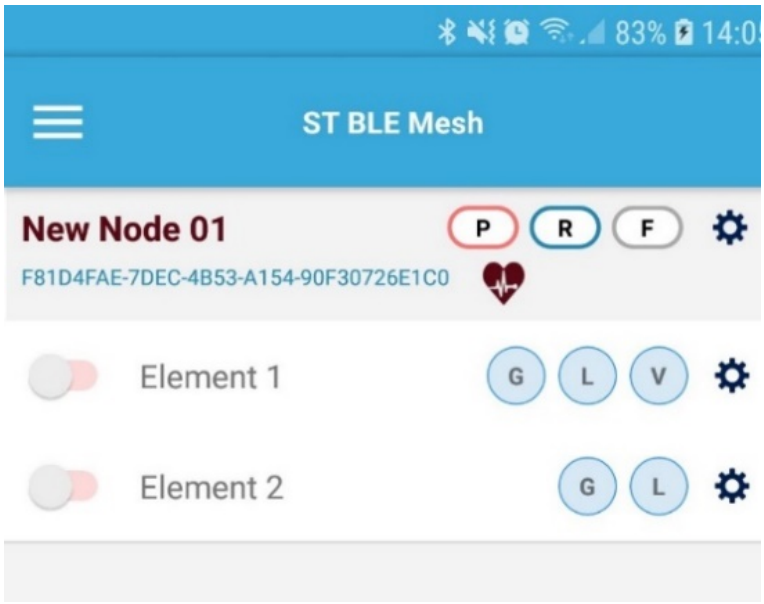
During the provisioning and configuration step, the defined elements must be visible. Click on "GO WITH QUICK CONFIGURATION" button to setup all the elements. See Figure 98

Figure 98. **Quick configuration menu**



Afterward, the interface nodes must be visible and model operations can be run on the elements from the node interfaces. See Figure 99

Figure 99. **Node interface screen**



8.3 Embedded provisioner

8.3.1 Provisioning definition

A new device which is intended to be incorporated in a mesh network is initially referred to as an unprovisioned device.

Provisioning is a process of adding this un-provisioned device to a mesh network, managed by a provisioner. The provisioner is typically a smartphone or other mobile computing device.

Being provisioned, the device is now referred to as a node and it has the ability to communicate with other nodes in the network. Provisioning has made the device a member of this particular mesh network.

Figure 100 describes the link setup using an ID between a provisioner and an unprovisioned device.

Figure 100. Link setup process

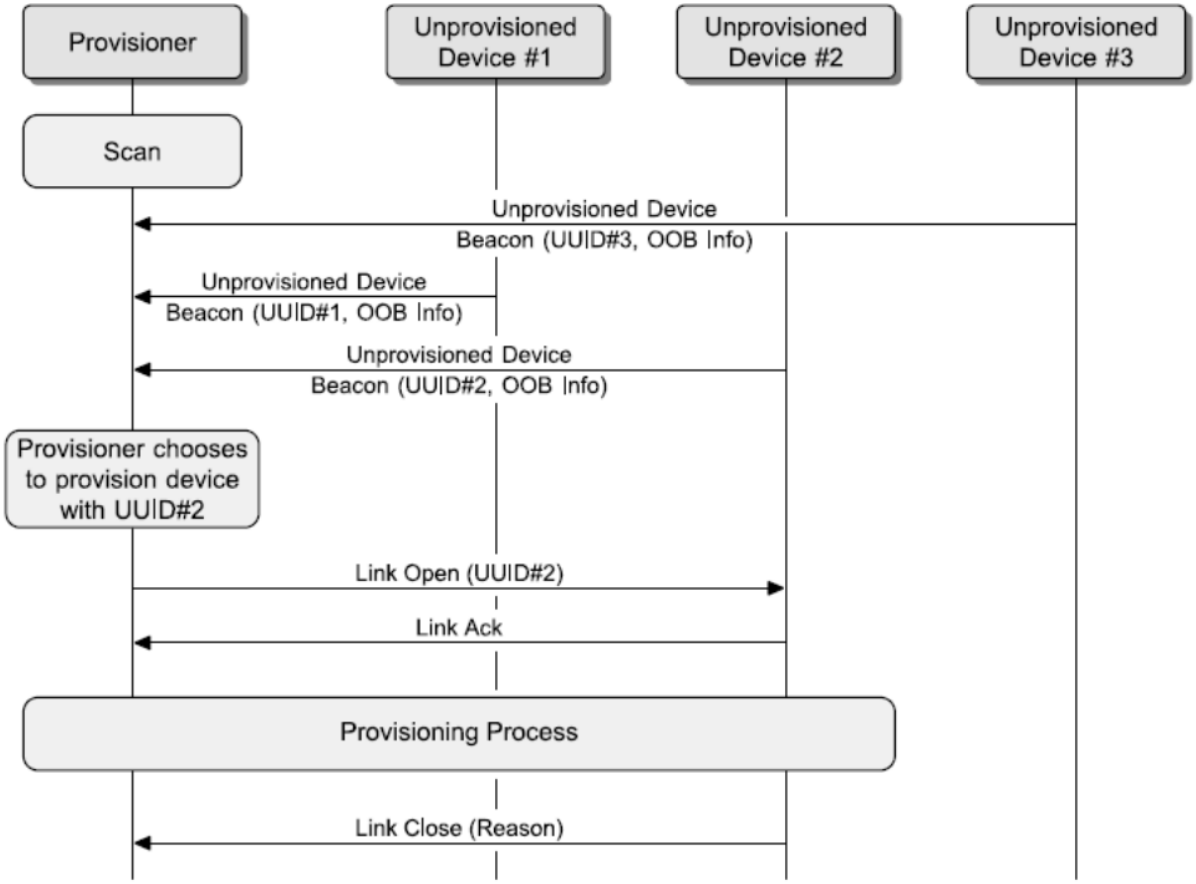
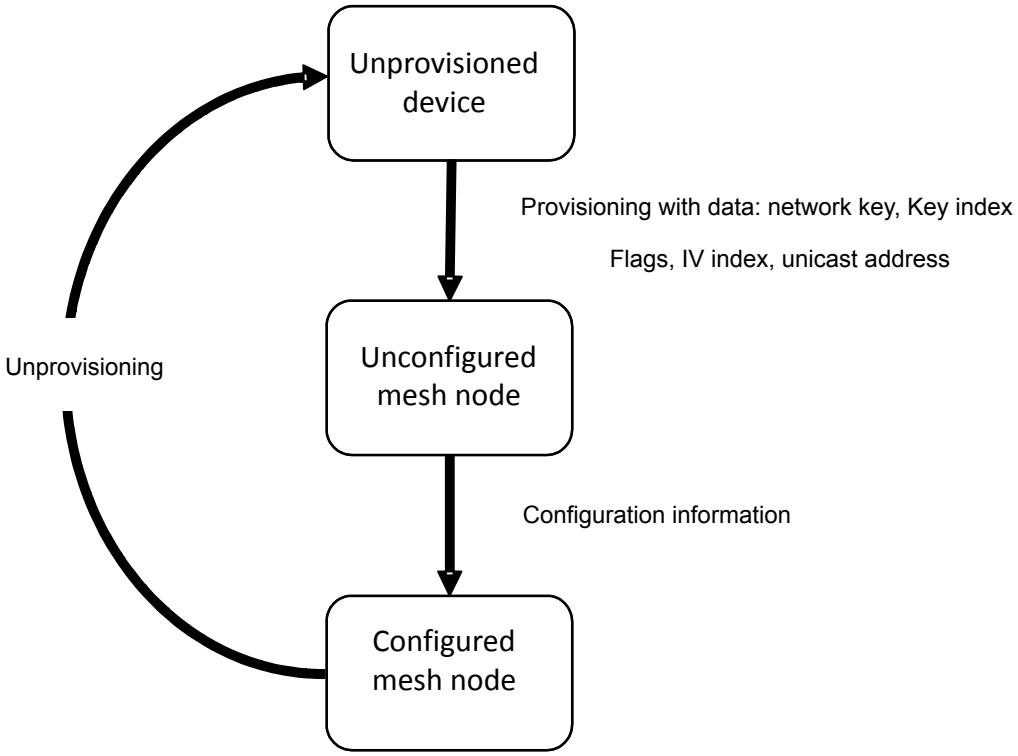


Figure 101 shows the configuration states of a BLE mesh node.

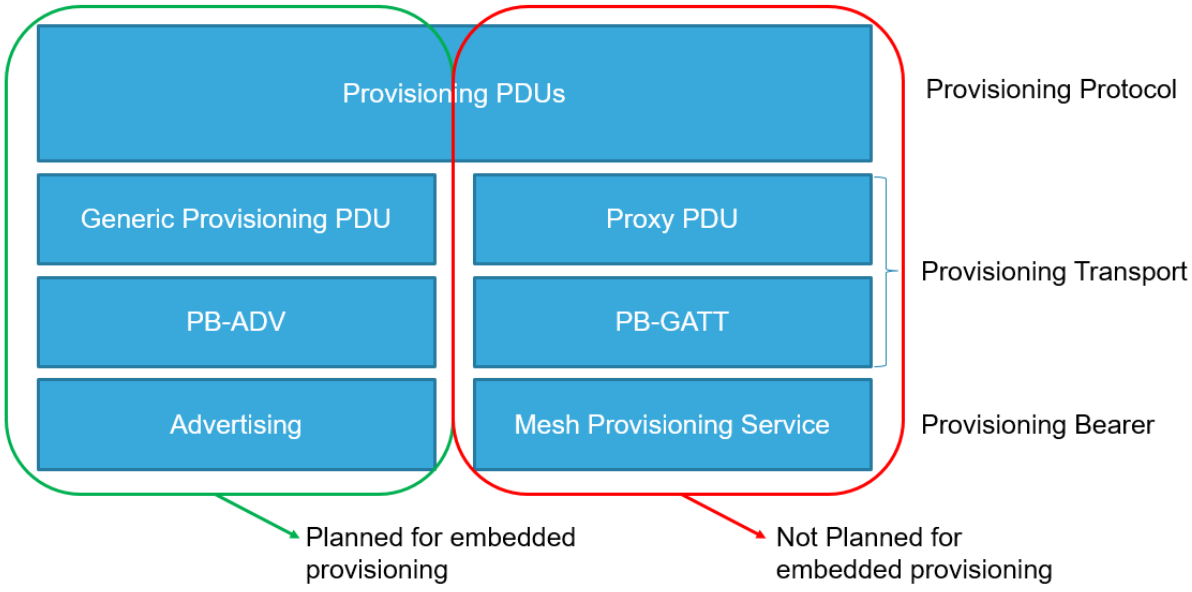
Figure 101. **BLE mesh node state machine**



The Bluetooth® mesh specification defines the provisioning protocol, which defines PDUs used to communicate between a provisioner and a new, unprovisioned device during the provisioning process.

Figure 102 illustrates the provisioning protocol stack alongside the full Bluetooth® mesh protocol stack.

Figure 102. **BLE mesh provisioning protocol stack**



From bottom to top, these are the components:

Provisioning bearer

A provisioning bearer layer enables the transport of provisioning PDUs between a provisioner and an unprovisioned device. Two provisioning bearers are defined:

- PB-ADV: A provisioning bearer used to provision a device over the Bluetooth® advertising channels. The PB-ADV bearer is used for transmitting generic provisioning PDUs. A device supporting PB-ADV should perform passive scanning with a duty cycle as close to 100% as possible in order to avoid missing any incoming generic provisioning PDUs.
- PB-GATT: A provisioning bearer used to provision a device using Bluetooth® mesh proxy PDUs from the proxy protocol. The proxy protocol enables nodes to send and receive network PDUs, mesh beacons, proxy configuration messages, and provisioning PDUs over a connection-oriented, Bluetooth® Low Energy (LE) bearer. PB-GATT encapsulates provisioning PDUs within GATT operations, involving the GATT provisioning service and it is provided for use when a provisioner does not support PB-ADV.

Provisioning protocol

The provisioning protocol defines requirements for provisioning PDUs, behavior, and security. Understanding the provisioning protocol helps the user select an approach to authenticate based on application requirements.

8.3.2 Embedded Provisioner principle

Embedded provisioning idea is to use a nucleo board as provisioner instead of a smartphone or other mobile computing device.

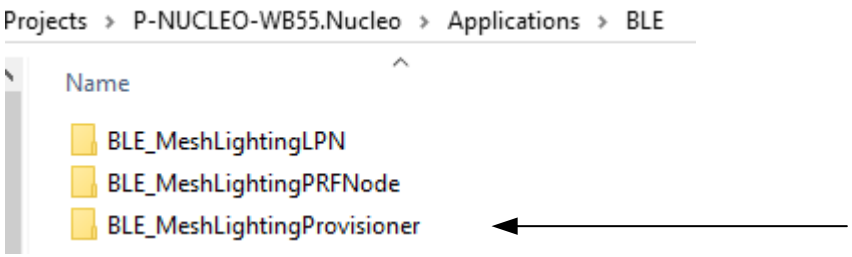
The board used as provisioner does not need to be provisioned as it creates itself the network parameters.

The board is then be able to scan all unprovisioned devices around and provision them.

8.3.3 Setup

From the projects workspace, select the embedded provisioner projectm see Figure 103:

Figure 103. Embedded provisioner project workspace selection



Open the project in an appropriate IDE and check the features enabled in `mesh_cfg_usr.h`:

Manually configured provisioner

Refer to the sample code below:

```
#define EANBLE_RELAY_FEATURE
#define EANBLE_PROXY_FEATURE
#define EANBLE_FRIEND_FEATURE
//#define EANBLE_PROVISIONER_FEATURE
#define DYNAMIC_PROVISIONER
```

If the `DYNAMIC_PROVISIONER` is selected, the provisioner configuration must be done manually following the steps below:

- Build and flash the project into your provisioner board
- Switch on the board

- The following welcome message is displayed on the serial terminal :

```
Next NVM Address 080c7000
Unprovisioned device
[…]
BLE-Mesh Lighting Demo v1.12.008
BLE-Mesh Library v01.12.008
BLE Stack v1.6.0 Branch=0 Type=3
FUS v1.0.2
BD_MAC Address = [c0]:[e1]:[26]:[07]:[f3]:[90]
UUID Address = [f8] [1d] [4f] [ae] [7d] [ec] [4b] [53] [a1] [54] [90] [f3] [07] [26]
[e1] [c0]
```

- Write the following instruction in the serial terminal to define one node as the ROOT node for network creation: `ATEP ROOT` (case insensitive).

  This performs the model binding and configuration.
- Write the following instruction in serial terminal to scan unprovisioned devices in range: `ATEP SCAN` (case insensitive).

  The log on terminal indicates unprovisioned devices with their corresponding UUIDs:

```
Device-0 -> F81D4FAE7DEC4B53A15490F30726E1C0
Device-1 -> F81D4FAE7DEC4B53A1542B350826E1C0
Test command executed successfully
```

  Here two devices (device-0 and device-1) are unprovisioned in the network
- Write the following instruction to start provisioning of device-0: `ATEP PRVN-0` (case insensitive)

  The log will appear on terminal indicating the device-0 is provisioned by the provisioner by making the PB-ADV connection link:

```
Test command executed successfully
PB-ADV Link opened successfully
Device Key: 46 82 e9 3e 20 […]
App Key: e6 bd d2 95 2d 81 85 7c a4 2e […]
Device is provisioned by provisioner
```

  The log in the provisioned node terminal indicates that the node is provisioned by the provisioner.
- Finally the PB-ADV connection link is closed and device-0 is configured and models bound.

Pre-configured provisioner

In the case of a provisioner which is already configured, see the code example below:

```
#define EANBLE_RELAY_FEATURE
#define EANBLE_PROXY_FEATURE
#define EANBLE_FRIEND_FEATURE
#define EANBLE_PROVISIONER_FEATURE
//#define DYNAMIC_PROVISIONER
```

If `ENABLE_PROVISIONER_FEATURE` is defined, the provisioner node is automatically be configured at board initialization.

The following process allows the user to create a mesh network:

- Build and flash your project into your provisioner board.
- Switch on the board.
- Write the following instruction in the serial terminal to scan for unprovisioned devices in range: `ATEP SCAN` (case insensitive)
- The log on terminal indicates unprovisioned devices with their corresponding UUIDs:

```
Device-0 -> F81D4FAE7DEC4B53A15490F30726E1C0
Device-1 ->F81D4FAE7DEC4B53A1542B350826E1C0
Test command executed successfully
```

  In this case, two devices (device-0 and device-1) are unprovisioned in the network

- Write the following instruction to start provisioning of device-0: `ATEP PRVN-0` (case insensitive)

  Where 0 indicates device 0.

  The following log appears on terminal indicating the device-0 is provisioned by the provisioner by making the PB-ADV connection link:

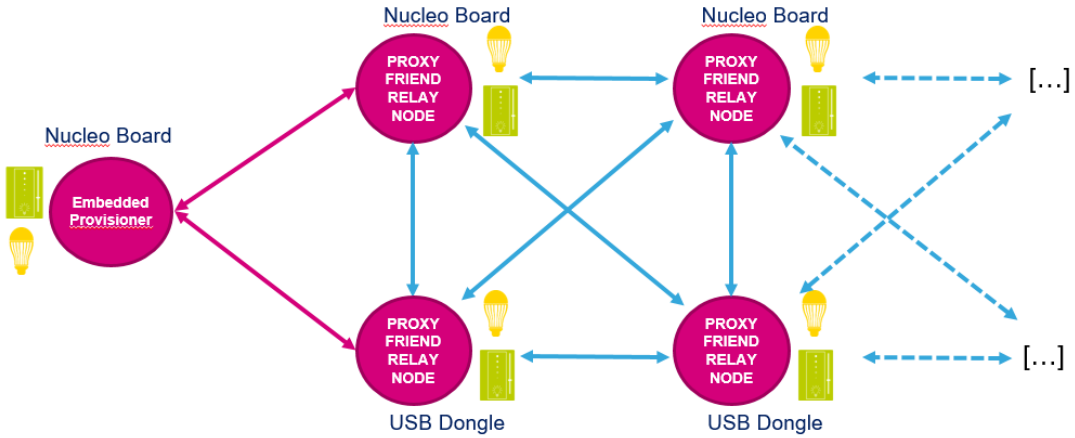
```
Test command executed successfully
PB-ADV Link opened successfully
Device Key: 46 82 e9 3e 20 […]
App Key: e6 bd d2 95 2d 81 85 7c a4 2e […]
Device is provisioned by provisioner
```

  The log on the provisioned node terminal indicates that the node is provisioned by the provisioner.

- Finally the PB-ADV connection link is closed and device-0 is configured and models bound.

After these steps, the network is setup as in Figure 104.

Figure 104. BLE-mesh network illustration



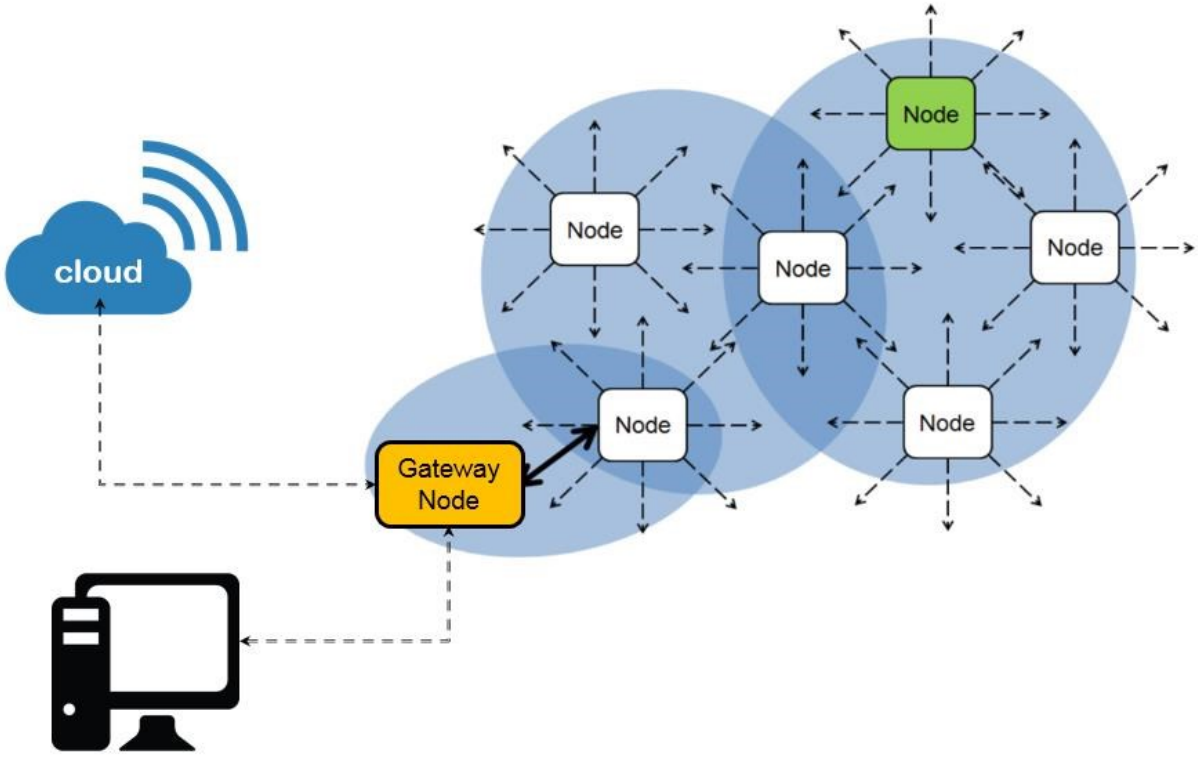
Appendix A Bluetooth® Low Energy mesh serial gateway

A.1 Introduction

The objective of this appendix describes how to use Bluetooth® Low Energy mesh serial gateway. The application demonstrates how to setup a gateway node. The gateway node is a Bluetooth® Low Energy device connected to a PC or another higher level device that has the capacity to manage serial communication. This node can be used to send or receive information from other receiver nodes present on the mesh network.

This node provides an interface between the user's higher device and the Bluetooth® Low Energy mesh. This is illustrated in the figure below.

Figure 105. Bluetooth® Low Energy mesh serial gateway



A.2 Getting started

See Section 3 Getting started for more details.

A.3 Serial gateway initialization

This procedure initializes the serial control gateway which is required to control of the BLE mesh via the virtual COM port.

In the BLE mesh firmware go to `mesh_cfg_usr.h` and replace the following:

```
/* Enables the serial interface using Uart */
#define ENABLE_SERIAL_INTERFACE 0
#define ENABLE_UT 0
#define ENABLE_SERIAL_CONTROL 0
#define ENABLE_APPLI_TEST 0
```

With

```
/* Enables the serial interface using Uart */
#define ENABLE_SERIAL_INTERFACE 1
#define ENABLE_UT 0
#define ENABLE_SERIAL_CONTROL 1
#define ENABLE_APPLI_TEST 0
```

A.4 Configuration of models to print data

This procedure initializes the required BLE mesh models to print data on the virtual COM port.
In BLE mesh firmware go to `mesh_cfg_usr.h` replace the following:

```
#define TF_GENERIC 0
#define TF_LIGHT 0
#define TF_SENSOR 0
#define TF_VENDOR 0
#define TF_NEIGHBOUR 0
#define TF_LPN_FRND 0
#define TF_ELEMENTS 0
#define TF_PROVISION 0
#define TF_HANDLER 1
#define TF_INIT 1
#define TF_ADDRESS 0
#define TF_MISC 0
#define TF_SERIAL_CTRL 0
```

with

```
#define TF_GENERIC 1
#define TF_LIGHT 1
#define TF_SENSOR 0
#define TF_VENDOR 1
#define TF_NEIGHBOUR 0
#define TF_LPN_FRND 0
#define TF_ELEMENTS 0
#define TF_PROVISION 1
#define TF_HANDLER 1
#define TF_INIT 1
#define TF_ADDRESS 0
#define TF_MISC 0
#define TF_SERIAL_CTRL 1
```

A.5 Enable all generic and light models

In BLE mesh firmware go to `mesh_cfg_usr.h` and replace the following:

```
/* Define the following Macros to enable the usage of the Server Generic Client*/
#define ENABLE_GENERIC_MODEL_SERVER_ONOFF
#define ENABLE_GENERIC_MODEL_SERVER_LEVEL
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF8SETUP
//#define ENABLE_GENERIC_MODEL_SERVER_DEFAULT_TRANSITION_TIME
```

With:

```
/* Define the following Macros to enable the usage of the Server Generic Client*/
#define ENABLE_GENERIC_MODEL_SERVER_ONOFF
#define ENABLE_GENERIC_MODEL_SERVER_LEVEL
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF
#define ENABLE_GENERIC_MODEL_SERVER_POWER_ONOFF8SETUP
//#define ENABLE_GENERIC_MODEL_SERVER_DEFAULT_TRANSITION_TIME
```

And replace:

```
/* Define the following Macros to enable the usage of the Light model*/
#define ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS
#define ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS_SETUP
//#define ENABLE_LIGHT_MODEL_SERVER_CTL
//#define ENABLE_LIGHT_MODEL_SERVER_CTL_SETUP
//#define ENABLE_LIGHT_MODEL_SERVER_CTL_TEMPERATURE
//#define ENABLE_LIGHT_MODEL_SERVER_HSL
//#define ENABLE_LIGHT_MODEL_SERVER_HSL_SETUP
//#define ENABLE_LIGHT_MODEL_SERVER_HSL_HUE
//#define ENABLE_LIGHT_MODEL_SERVER_HSL_SATURATION
```

With:

```
/* Define the following Macros to enable the usage of the Light model*/
#define ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS
#define ENABLE_LIGHT_MODEL_SERVER_LIGHTNESS_SETUP
#define ENABLE_LIGHT_MODEL_SERVER_CTL
#define ENABLE_LIGHT_MODEL_SERVER_CTL_SETUP
#define ENABLE_LIGHT_MODEL_SERVER_CTL_TEMPERATURE
#define ENABLE_LIGHT_MODEL_SERVER_HSL
#define ENABLE_LIGHT_MODEL_SERVER_HSL_SETUP
#define ENABLE_LIGHT_MODEL_SERVER_HSL_HUE
#define ENABLE_LIGHT_MODEL_SERVER_HSL_SATURATION
```

A.6 Serial control commands

After the board is connected and the terminal window is opened, press the reset button. If reset state print message is displayed the system is ready to control mesh with serial commands.

A serial command is the string which when entered properly calls:

`SerialCtrl_Process()` in `serial_ctrl.c` which breaks the string into different parameters. To do so `SerialCtrl_GetMinParamLength()` is called, it compares the opcode entered in the string to the opcode table of each model. It then recall :

- Peer address
- Opcode
- Data (opcode parameters).

All this data is passed through `BleMesh_SetRemoteData()` to the library.

A.6.1 Sensor Setup Server model command format

A valid control command must be entered in the format detailed in the table below. This model does not include the vendor model which are defined in Section A.6.2 and Section A.6.3 .

Table 142. Mesh model control commands format

Identifier	Destination address	Op code	Data
ACTL	4 characters	4 characters	Variable length

Press enter after completing the string of the above parameter. The parameters are defined in Table 143.

Table 143. **Mesh control parameter definition**

Parameter	Description
Identifier	Fixed keyword used to identify the type of command.
Destination Address	Defines one of the following network device target addresses: • Unicast • Broadcast • Multicast.
Op Code	Model specifier to be used to execute the command.
Data	Specifies the selected model properties.

A.6.2 Vendor Write model command format

Press enter after completing the parameter string defined in Section A.6.1 .

Table 144. **Vendor Write parameter definition**

Parameter	Descrption
Identifier	Fixed keyword used to identify Vendor Write command.
Destination Address	Defines one of the following network device target addresses: • unicast • broadcast • multicast.
Op Code	Specifies the model to be used to execute the command.
Data	Specifies the subcommand and model properties.

A.6.3 Vendor read model command format

Press enter after completing the parameter string defined in Section A.6.1 .

Table 145. **Vendor read arameter definition**

Parameter	Descrption
Identifier	Fixed keyword used to identify vendor read command.
Destination Address	Defines one of the following network device target addresses: • unicast • broadcast • multicast.
Op Code	Specifies the model to be used to execute the command.
Data	Specifies the subcommand and model properties.

A.6.4 Generic ON/OFF set format

A valid control command must be entered in the format detailed in the table below.

Table 146. Mesh control commands format

Identifier	Destination address	Op code	Data
ATVW	4 characters	4 characters	Variable length

Press enter after completing the string of the above parameter. The parameters are defined in the table below.

Table 147. Parameter definition

Parameter	Description
Identifier	Fixed keyword used to identify the type of command.
Destination Address	Defines one of the following network device target addresses: • unicast • broadcast • multicast.
Op Code	Model specifier to be used to execute the command.
Data	Specifies the selected model properties.

A.6.5 Generic Level set data format

Data must be entered as described in the following table.

Table 148. Generic Level set data format

Field	Size (Bytes)	Description	Example
On/Off	1	Target value of generic On/Off state.	00 / 01
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.6 Generic Level Delta set format

Data must be entered as described in the following table.

Table 149. Generic Level Delta set data format

Field	Size (Bytes)	Description	Example
Level	2	Target value of generic level state.	0x0000 – 0x7FFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.7 Generic Power ON/OFF set data format

Data must be entered in the following format.

Table 150. Generic Power ON/OFF set data format

Field	Size (Bytes)	Description	Example
On power up	1	Target value of generic on power up state.	00 / 01

A.6.8 Generic Default Transition Time Server set format

Data must be entered in the following format.

Table 151. Generic Default Transition Time Server set data format

Field	Size (Bytes)	Description	Example
Transition time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE

A.6.9 Light Lightness setting data

This data includes:

- Format for light lightness set
- Light lightness linear set
- Light lightness default set
- Light lightness range set.

Data must be entered in the following format.

Table 152. Light Lightness data format

Field	Size (Bytes)	Description	Example
Lightness	2	Target value of actual light lightness state.	0x0000 – 0xFFFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.10 Light CTL set data format

Data must be entered in the following format.

Table 153. Light CTL set data format

Field	Size (Bytes)	Description	Example
CTL lightness	2	Target value of light lightness CTL state.	0x0000 – 0xFFFF
CTL temperature	2	Target value of light temperature CTL state.	0x0320 – 0x4E20
CTL delta UV	2	Target value of light delta UV CTL state.	0x8000 – 0x7FFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.11 Light CTL Temperature Range set data format

Data must be entered in the following format.

Table 154. Light CTL Temperature Range data format

Field	Size (Bytes)	Description	Example
Range Min	2	Min temp. range	0x0320
Range Max	2	Max. temp. range	0x4E20

A.6.12 Light CTL Default Set data format

Data must be entered in the following format.

Table 155. Light CTL default data format

Field	Size (Bytes)	Description	Example
CTL Lightness	2	Target value of Light CTL Lightness State.	0x0000 – 0xFFFF
CTL Temperature	2	Target value of Light CTL Temperature State.	0x0320 – 0x4E20
CTL Delta UV	2	Target value of Light CTL Delta UV State.	0x8000 – 0x7FFF

A.6.13 Light HSL Set data format

Data must be entered in the following format.

Table 156. Light HSL Set data format

Field	Size (Bytes)	Description	Example
HSL Lightness	2	Target value of light HSL lightness state.	0x0000 – 0xFFFF
HSL Hue	2	Target value of light HSL hue state.	0x0000 – 0xFFFF
HSL Saturation	2	Target light HSL saturation state.	0x0000 – 0xFFFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition time	1	Time take to change from present state to target state (Optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.14 Light HSL Range Set data format

Data must be entered in the following format.

Table 157. Light HSL Range Set data format

Field	Size (Bytes)	Description	Example
Hue range	2	HSL range set	0x0000 – 0xFFFF
Saturation range	2	Saturation range set	0x0000 – 0xFFFF

A.6.15 Light HSL Default Set data format

Data must be entered in the following format.

Table 158. Light HSL Default Set data format

Field	Size (Bytes)	Description	Example
Lightness	1	Value of light lightness default state.	0x00 – 0xFF
Hue	1	Value of light HSL hue default state.	0x00 – 0xFF
Saturation	1	Value of light HSL hue saturation default state.	0x00 – 0xFF

A.6.16 Light HSL Hue Set data format

Data must be entered in the following format.

Table 159. Light HSL Hue Set data format

Field	Size (Bytes)	Description	Example
Hue	2	Value of light HSL hue state.	0x0000 – 0xFFFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition Time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.17 Light HSL Saturation Set data format

Data must be entered in the following format.

Table 160. light HSL saturation data format

Field	Size (Bytes)	Description	Example
Saturation	2	Value of Light HSL Saturation State.	0x0000 – 0xFFFF
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition Time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.18 Light LC Mode Set data format

Data must be entered in the following format.

Table 161. Light LC mode data format

Field	Size (Bytes)	Description	Example
Mode	1	The target value of the light LC mode state	00 – 01

A.6.19 Light LC occupancy mode set data format

Data must be entered in the following format.

Table 162. Light LC Occupancy data format

Field	Size (Bytes)	Description	Example
Mode	1	The target value of the light LC occupancy mode state	00 – 01

A.6.20 Light LC Light OnOff Set data format

Data must be entered in the following format.

Table 163. Light LC Light OnOff Set data format

Field	Size (Bytes)	Description	Example
Light OnOff	1	The target value of the Light LC Light OnOff state	0x00 – 0x01
TID (transaction identifier)	1	Indicate whether the message is a new message or a retransmission of a previously sent message.	0x00 – 0xFF
Transition Time	1	Time take to change from present state to target state (optional).	0x00 – 0xFE
Delay	1	Message execution delay in milliseconds (optional).	-

A.6.21 Light LC Property Set data format

Data must be entered in the following format.

Table 164. Light LC Property Set data format

Field	Size (Bytes)	Description	Example
Light LC property ID	2	Property ID identifying a Light LC property.	0x00 – 0xFFFF
Light LC property value	Variable, 1 to 4 bytes	Raw value for the Light LC property	0x00 – 0xFFFFFFFF

A.6.22 Generic Power ON/OFF set data format

Data must be entered in the following format.

Table 165. Generic Power On/Off set data format

Field	Size (Bytes)	Description	Example
OnPowerUp	1	The value of the generic OnPowerUp state: • 0x00 Off. After being powered up, the element is in an off state. • 0x01 Default. After being powered up, the element is in an on state and uses default state values. • 0x02 Restore. If a transition was in progress while being powered down, the element restores the target state when it is powered up. Otherwise the element restores the state it was in when it was powered down. • 0x03-0xFF Prohibited	0x00 – 0x02

A.6.23 Sensor Descriptor Get data format

Data must be entered in the following format.

Table 166. Sensor descriptor get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor (optional)	-

A.6.24 Sensor Cadence Get data format

Data must be entered in the following format.

Table 167. Sensor Cadence Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-

A.6.25 Sensor Cadence Set data format

Data must be entered in the following format.

Table 168. Sensor Cadence Set data format

Field	Size (bits)	Description	Example
Property ID	16	Property ID for the sensor.	-
Fast Cadence Period Divisor	7	Divisor for the Publish Period.	-
Status Trigger Type	1	Defines the unit and format of the status trigger delta fields.	-
Status Trigger Delta Down	Variable	Delta down value that triggers a status message.	-
Status Trigger Delta Up	Variable	Delta up value that triggers a status message.	-
Status Min Interval	8	Minimum interval between two consecutive status messages.	-
Fast Cadence Low	Variable	Low value for the fast cadence range.	-
Fast Cadence High	variable	High value for the fast cadence range.	-

A.6.26 Sensor Settings Get data format

Data must be entered in the following format.

Table 169. Sensor Settings Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-

A.6.27 Sensor Setting Get data format

Data must be entered in the following format.

Table 170. Sensor Setting Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-

Field	Size (Bytes)	Description	Example
Sensor Setting Property ID	2	Setting Property ID identifying a setting within a sensor.	-

A.6.28 Sensor Setting Set data format

Data must be entered in the following format.

Table 171. Sensor Setting Set data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-
Sensor Setting Property ID	2	Setting the property ID identifying a setting within a sensor.	-
Sensor Setting Raw	variable	Raw value for the setting.	-

A.6.29 Sensor Get data format

Data must be entered in the following format.

Table 172. Sensor Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor (optional)	-

A.6.30 Sensor Column Get data format

Data must be entered in the following format.

Table 173. Sensor Column Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-
Raw Value X	variable	Raw value identifying a column	-

A.6.31 Sensor Series Get data format

Data must be entered in the following format.

Table 174. Sensor Series Get data format

Field	Size (Bytes)	Description	Example
Property ID	2	Property ID for the sensor.	-
Raw Value X1	variable	Raw value identifying a starting column. (optional)	-
Raw Value X1	variable	Raw value identifying an ending column. (C.1)	-

A.6.32 Command List

Table 175. Generic command list

Identifier	Destination address(range)	Op Code	Data	Description
ATCL	0x0001-0xFFFF	0x8201	-	Generic ON/OFF Get

Identifier	Destination address(range)	Op Code	Data	Description
ATCL	0x0001-0xFFFF	0x8202	0x01 00	Generic ON/OFF Set Ack
ATCL	0x0001-0xFFFF	0x8203	0x01 00	Generic ON/OFF Set UnAck
ATCL	0x0001-0xFFFF	0x8205	-	Generic Level Get
ATCL	0x0001-0xFFFF	0x8206	0x3FFF 00	Generic Level Set Ack
ATCL	0x0001-0xFFFF	0x8207	0x3FFF 00	Generic Level Set UnAck
ATCL	0x0001-0xFFFF	0x8209	0xFFFFFFFF 00	Generic Level Delta Set
ATCL	0x0001-0xFFFF	0x820A	0xFFFFFFFF 00	Generic Level Delta Set UnAck
ATCL	0x0001-0xFFFF	0x820B	0xFFFF 00	Generic Level Delta Move Set
ATCL	0x0001-0xFFFF	0x820C	0xFFFF 00	Generic Level Delta Move Set UnAck
ATCL	0x0001-0xFFFF	0x8211	-	Generic Power ON/OFF Get
ATCL	0x0001-0xFFFF	0x8213	0x01	Generic Power ON/OFF Set
ATCL	0x0001-0xFFFF	0x8214	0x01	Generic Power ON/OFF Set UnAck
ATCL	0x0001-0xFFFF	0x820D	-	Generic Default Transition Time Get
ATCL	0x0001-0xFFFF	0x820E	0x02	Generic Default Transition Time Set
ATCL	0x0001-0xFFFF	0x820F	0x02	Generic Default Transition Time Set UnAck
ATCL	0x0001-0xFFFF	0x824B	-	Light Lightness Get
ATCL	0x0001-0xFFFF	0x824C	0xFFFF 00	Light Lightness Set
ATCL	0x0001-0xFFFF	0x824D	0xFFFF 00	Light Lightness Set UnAck
ATCL	0x0001-0xFFFF	0x824F	-	Light Lightness Linear Get

Table 176. Light lightness and CTL command list

Identifier	Destination address (range)	Op code	Data	Description
ATCL	0x0001-0xFFFF	0x8250	0xFFFF 00	Light Lightness Linear Set
ATCL	0x0001-0xFFFF	0x8251	0xFFFF 00	Light Lightness Linear Set UnAck
ATCL	0x0001-0xFFFF	0x8255	-	Light Lightness Default Get
ATCL	0x0001-0xFFFF	0x8259	0xFFFF	Light Lightness Default Set
ATCL	0x0001-0xFFFF	0x825A	0xFFFF	Light Lightness Default Set UnAck
ATCL	0x0001-0xFFFF	0x8257	-	Light Lightness Range Get
ATCL	0x0001-0xFFFF	0x825B	0xFFFF FFFF	Light Lightness Range Set
ATCL	0x0001-0xFFFF	0x825C	0xFFFF FFFF	Light Lightness Range Set UnAck
ATCL	0x0001-0xFFFF	0x825D	-	Light CTL Get
ATCL	0x0001-0xFFFF	0x825E	0x7FFF4E19FFFF 00	Light CTL Set
ATCL	0x0001-0xFFFF	0x825F	0x7FFF4E19FFFF 00	Light CTL Set UnAck
ATCL	0x0001-0xFFFF	0x8262	-	Light CTL Temperature Range Get
ATCL	0x0001-0xFFFF	0x826B	0x4E200320	Light CTL Temperature Range Set
ATCL	0x0001-0xFFFF	0x826C	0x4E200320	Light CTL Temperature Range Set UnAck
ATCL	0x0001-0xFFFF	0x8267	-	Light CTL Default Get
ATCL	0x0001-0xFFFF	0x8269	0x7FFF4E19FFFF	Light CTL Default Set
ATCL	0x0001-0xFFFF	0x826A	0x7FFF4E19FFFF	Light CTL Default Set UnAck
ATCL	0x0001-0xFFFF	0x826D	-	Light HSL Get

Identifier	Destination address (range)	Op code	Data	Description
ATCL	0x0001-0xFFFF	0x8276	0xFFFFFFFFFFFF	Light HSL Set
ATCL	0x0001-0xFFFF	0x8277	0xFFFFFFFFFFFF	Light HSL Set UnAck
ATCL	0x0001-0xFFFF	0x827D	-	Light HSL Range Get
ATCL	0x0001-0xFFFF	0x8281	0xFFFFFFFF	Light HSL Range Set
ATCL	0x0001-0xFFFF	0x8282	0xFFFFFFFF	Light HSL Range Set UnAck
ATCL	0x0001-0xFFFF	0x827B	-	Light HSL Default Get
ATCL	0x0001-0xFFFF	0x827F	0xFFFFFF	Light HSL Default Set
ATCL	0x0001-0xFFFF	0x8280	0xFFFFFF	Light HSL Default Set UnAck
ATCL	0x0001-0xFFFF	0x826E	-	Light HSL Hue Get
ATCL	0x0001-0xFFFF	0x826F	0xFFFF 00	Light HSL Hue Set
ATCL	0x0001-0xFFFF	0x8270	0xFFFF 00	Light HSL Hue Set UnAck
ATCL	0x0001-0xFFFF	0x8272	-	Light HSL Saturation Get
ATCL	0x0001-0xFFFF	0x8273	0xFFFF 00	Light HSL Saturation Set
ATCL	0x0001-0xFFFF	0x8274	0xFFFF 00	Light HSL Saturation Set UnAck
ATCL	0x0001-0xFFFF	0x8291	-	Light LC Mode Get
ATCL	0x0001-0xFFFF	0x8292	0x01	Light LC Mode Set
ATCL	0x0001-0xFFFF	0x8293	0x01	Light LC Mode Set unAck
ATCL	0x0001-0xFFFF	0x8295	-	Light LC OM Get
ATCL	0x0001-0xFFFF	0x8296	0x01	Light LC OM Set
ATCL	0x0001-0xFFFF	0x8297	0x01	Light LC OM Set unAck
ATCL	0x0001-0xFFFF	0x8299	-	Light LC Light OnOff Get
ATCL	0x0001-0xFFFF	0x829A	0x01 01	Light LC Light OnOff Set
ATCL	0x0001-0xFFFF	0x829B	0x01 01	Light LC Light OnOff Set unAck
ATCL	0x0001-0xFFFF	0x829D	0xFFFF	Light LC Property Get
ATCL	0x0001-0xFFFF	0x0062	0xFFFF 00	Light LC Property Set
ATCL	0x0001-0xFFFF	0x0063	0xFFFF 00	Light LC Property Set unAck
ATCL	0x0001-0xFFFF	0x8230	0xFF	Sensor Descriptor Get
ATCL	0x0001-0xFFFF	0x8231	0xFFFF	Sensor Get
ATCL	0x0001-0xFFFF	0x8232	0xFFFF 01	Sensor Column Get
ATCL	0x0001-0xFFFF	0x8233	0xFFFF 01 FF	Sensor Series Get
ATCL	0x0001-0xFFFF	0x8234	0xFFFF	Sensor Cadence Get
ATCL	0x0001-0xFFFF	0x0055	Variable as defined in Table 177	Sensor Cadence Set
ATCL	0x0001-0xFFFF	0x0056	Variable as defined in Table 177	Sensor Cadence Set UnAck
ATCL	0x0001-0xFFFF	0x8235	0xFFFF	Sensor Settings Get
ATCL	0x0001-0xFFFF	0x8236	0xFFFF FFFF	Sensor Setting Get
ATCL	0x0001-0xFFFF	0x0059	0xFFFF FFFF 00	Sensor Setting Set
ATCL	0x0001-0xFFFF	0x005A	0xFFFF FFFF 00	Sensor Setting Set UnAck
ATVW	0x0001-0xFFFF	0x0003	0x01	Vendor LED ON
ATVW	0x0001-0xFFFF	0x0003	0x02	Vendor LED OFF
ATVW	0x0001-0xFFFF	0x0003	0x03	Vendor LED Toggle

Identifier	Destination address (range)	Op code	Data	Description
ATVW	0x0001-0xFFFF	0x0003	0x06 0000-0x7FFF	Vendor LED Intensity
ATVR	0x0001-0xFFFF	0x0002	0x01	Board Type
ATVR	0x0001-0xFFFF	0x0002	0x02	Library version
ATVR	0x0001-0xFFFF	0x0002	0x03	Sub library Version
ATVR	0x0001-0xFFFF	0x0003	-	Vendor LED Status

Table 177. Data format for the sensor cadence set lines

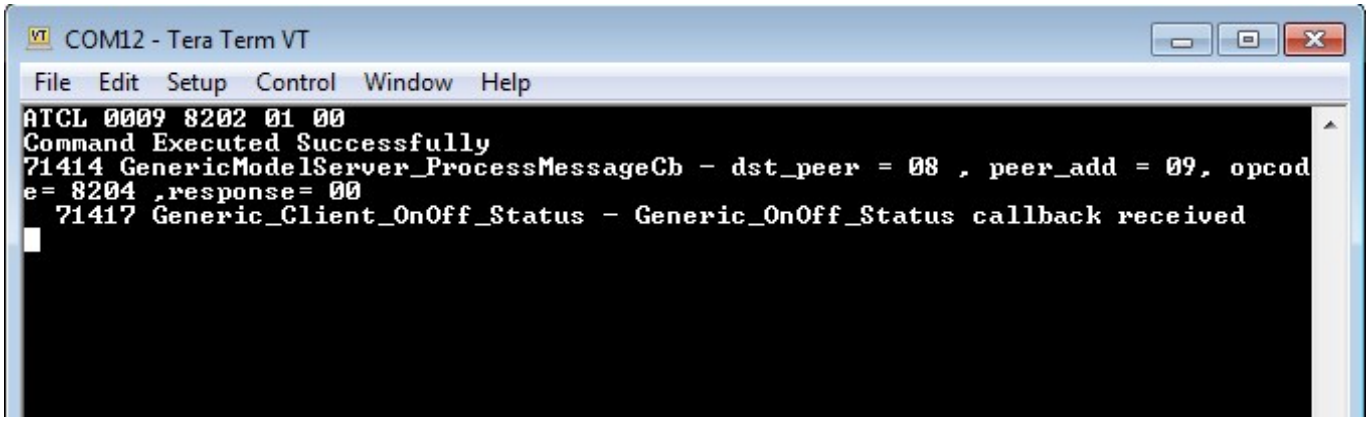
Property ID	Status trigger type (1 bit), fast cadence period divisor (3 bits)	Fast cadence period divisor	Status trigger delta down	Status trigger delta up	Set minimum interval	Fast cadence low	Fast cadence high
0xFFFF	0b1+0x7F	Variable	Variable	Variable	0x1A	Variable	Variable

A.7 Example of Generic ON/OFF set ack

Two nodes with unicast address, 8 and 9 are used for testing.

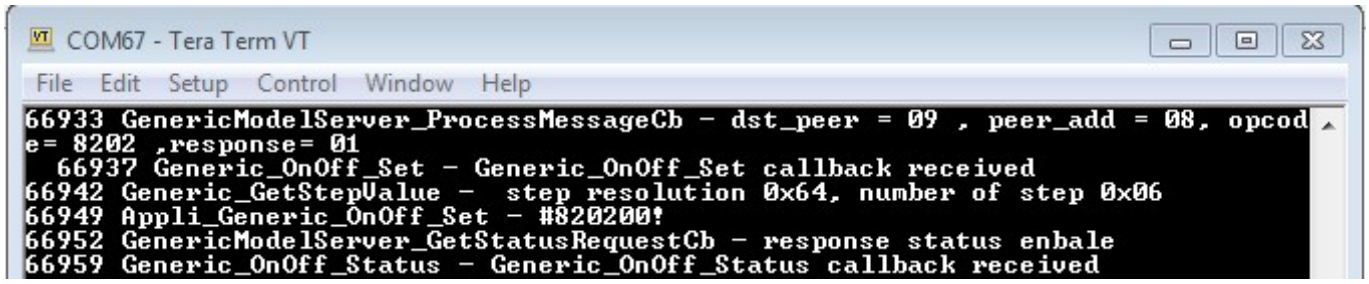
`Generic_OnOff_Set` message “ATCL 0009 8202 01 00” is sent to the node with a unicast address which is set to 9 from a node with a unicast address set to 8 and ack. The message is received on the node with a unicast address set to 8. This is illustrated in Figure 106

Figure 106. Node with unicast address – 8 send message



Node with unicast address – 9 receives it and send an ack. back to the node with unicast address – 8 as illustrated in Figure 107.

Figure 107. Node with unicast address - 9 send message

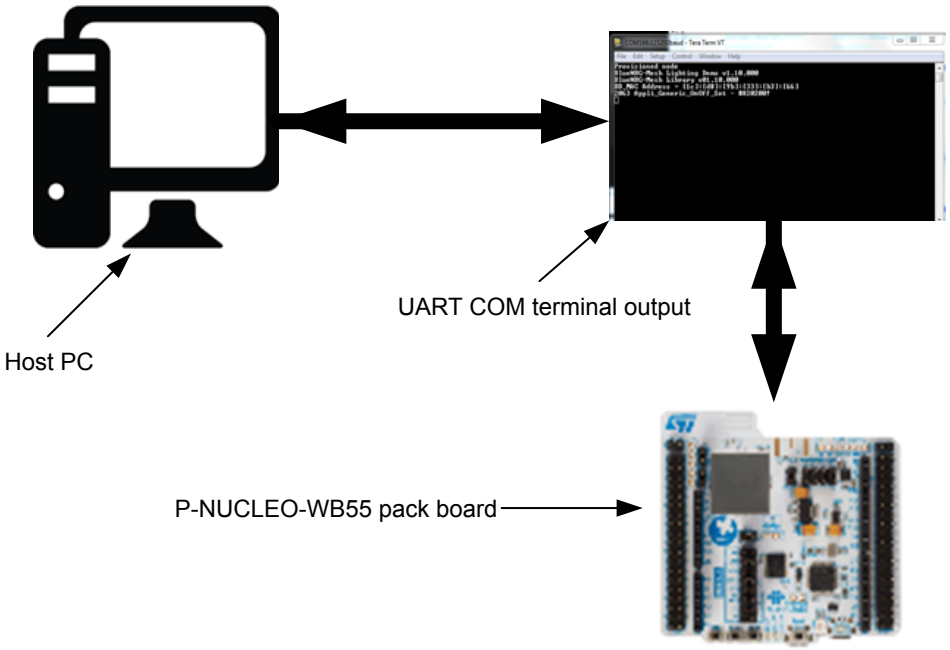


Appendix B ST BLE Mesh upper tester protocol

B.1 Introduction

The objective of this appendix describes how to use the ST BLE Mesh upper tester protocol using STMicroelectronics devices running an embedded mesh library. The upper tester protocol is an advanced utility which enables users to generate various conditions required for testing an application. It uses the power of “STMicroelectronics Virtual COM Port” and emulates the NUCLEO boards UART bus as a USB COM port as illustrated in the figure below.

Figure 108. BLE mesh network set up



B.2 Hardware and software requirements

See Section 3 Getting started section for more details.

B.3 Getting started

See Section 3 Getting started section for more details.

B.4 Enabling the upper tester

Open `mesh_cfg_usr.h` file and make sure that both of the macros are set as per the list below:

- `ENABLE_SERIAL_INTERFACE` macros is set to “1” to enable input form serial interface, and
- `ENABLE_UT` macros is set to “1” to enable upper tester command.

`ENABLE_UT` when defined includes the `serial_ut.c` file to the build project which in turn maps the testing function to the project.

B.5 Configuration of models to print data

This procedure initializes the BLE-mesh models required to print data on the virtual COM port.

In ST BlueNRG-mesh firmware go to `mesh_config_user.h` and replace the following

```
#define TF_GENERIC                      0
#define TF_LIGHT                        0
#define TF_SENSOR                       0
#define TF_VENDOR                       0
#define TF_NEIGHBOUR                    0
#define TF_LPN_FRND                     0
#define TF_ELEMENTS                     0
#define TF_ADDRESS                      0
#define TF_PROVISION                    1
#define TF_HANDLER                      1
#define TF_INIT                         1
#define TF_MISC                         0
#define TF_SERIAL_CTRL                  0
```

With:

```
#define TF_GENERIC                      0
#define TF_LIGHT                        0
#define TF_SENSOR                       0
#define TF_VENDOR                       0
#define TF_NEIGHBOUR                    0
#define TF_LPN_FRND                     0
#define TF_ELEMENTS                     1
#define TF_ADDRESS                      0
#define TF_PROVISION                    1
#define TF_HANDLER                      1
#define TF_INIT                         1
#define TF_MISC                         0
#define TF_SERIAL_CTRL                  1
```

B.6 Upper tester commands

B.6.1 Command format

A valid test command must be entered in the format following given in the table below.

Table 178. Command format

Identifier	Command	Arguments	
ATUT	Send-01	00	0001

- Identifier:
  - Specifies the type of command
  - ATUT is stated for PTS testing purposes
  - Error: "Not entered a valid parameter".
- Command:
  - Predefined keywords to perform specific action on the board
  - Error: “Wrong command test command failed”.
- Argument:
  - Additional information required by some commands
  - Error: “Wrong parameters Test command Failed. Invalid Argument”

B.6.2 Board management command

- Board reset

Table 179. Board reset command format

Identifier	Command	Argument
ATUT	set-01	-

This test case does not require any argument

Table 180. Board unprovisionning command format

Identifier	Command	Argument
ATUT	set-02	-

This test case does not require any argument

- Unprovisionning the board

B.6.3 Test specific command

- IV update - keep/remove 96-hour limit:

Table 181. IV update - keep/remove 96-hour limit format

Identifier	Command	Argument
ATUT	set 03	0/1

  - 0: Remove 96-hour limit
  - 1: Keep 96-hour limit.

- IV update - send test signal:

Table 182. IV update - send test signal format

Identifier	Command	Argument
ATUT	set 04	-

This test case does not require any argument.

- Delete/subscribe to new group address:

Table 183. Delete/subscribe to new group address format

Identifier	Command	Argument
ATUT	set 05	0/1

  - 0: Delete Subscription to group address
  - 1: Subscribe to new group address.

- Break existing friendship:

Table 184. Break existing friendship format

Identifier	Command	Argument
ATUT	set 06	-

This test case does not require any argument.

- Clears replay protection list:

Table 185. Clears replay protection list format

Identifier	Command	Argument
ATUT	set 07	-

This test case does not require any argument.

- Orders IUT to allow one Net Key index:

Table 186. Orders IUT to allow one Net Key index format

Identifier	Command	Argument
ATUT	set 08	-

This test case does not require any argument.

- Clears heartbeat subscription count:

Table 187. Clears heartbeat subscription count format

Identifier	Command	Argument
ATUT	set 09	-

This test case does not require any argument.

- Set system faults for health model:

Table 188. Set system faults for health model format

Identifier	Command	Fault argument	Argument
ATUT	set 10	00-ff	0/1

  - 0: Clear fault
  - 1: Create fault .

- Sets all node identity state for all networks:

Table 189. Clears heartbeat subscription count format

Identifier	Command	Argument
ATUT	set 11	0/1

  - 0: Enable Node Identity State for all network
  - 1: Disable Node Identity State for all network.

B.6.4 Mesh feature modification command

Table 190. **Mesh feature modification command structure**

Identifier	Command	Arguments
ATUT	set-12	01-08

Argument values:

- 01: Enable relay feature
- 02: Enable proxy feature
- 03: Enable relay and proxy feature
- 04: Enable friend feature
- 08: Enable low-power feature.

B.6.5 Packet transmission command

Transmission of network authenticated packets:

Table 191. **Network authentication packet format**

Identifier	Command	Arguments	
ATUT	send-01	<TTL>	<ADDRESS>

- TTL: 0x00-0x7F
- ADDRESS: 0x0001-0xFFFF.

Transmission of app packets to target device:

Table 192. **App packet transmission format**

Identifier	Command	Arguments	
ATUT	send-02	<SIZE>	<ADDRESS>

- TTL: 0X00-0x64
- ADDRESS: 0x0001-0xFFFF.

B.6.6 Status command

Print security credentials

Table 193. **Status command format**

Identifier	Command	Arguments
ATUT	Print-01	-

This test case does not require any argument

Note: All AT commands are case insensitive.

Appendix C ST BLE Mesh provisioner protocol

C.1 Introduction

The objective of this appendix is to describe how to use the ST BLE Mesh provisioner protocol using STMicroelectronics devices running the embedded mesh library. The provisioner protocol is an advanced utility which enables user to provision and configure a mesh node. It uses the power of “STMicroelectronics Virtual COM Port” and emulates the NUCLEO boards UART bus as a USB COM port as illustrated in Figure 109. Serial terminal screen output.

C.2 Hardware and software requirements

See Section 3 Getting started section for more details.

C.3 Getting started

See Section 3 Getting started section for more details.

C.4 Enabling the provisioner

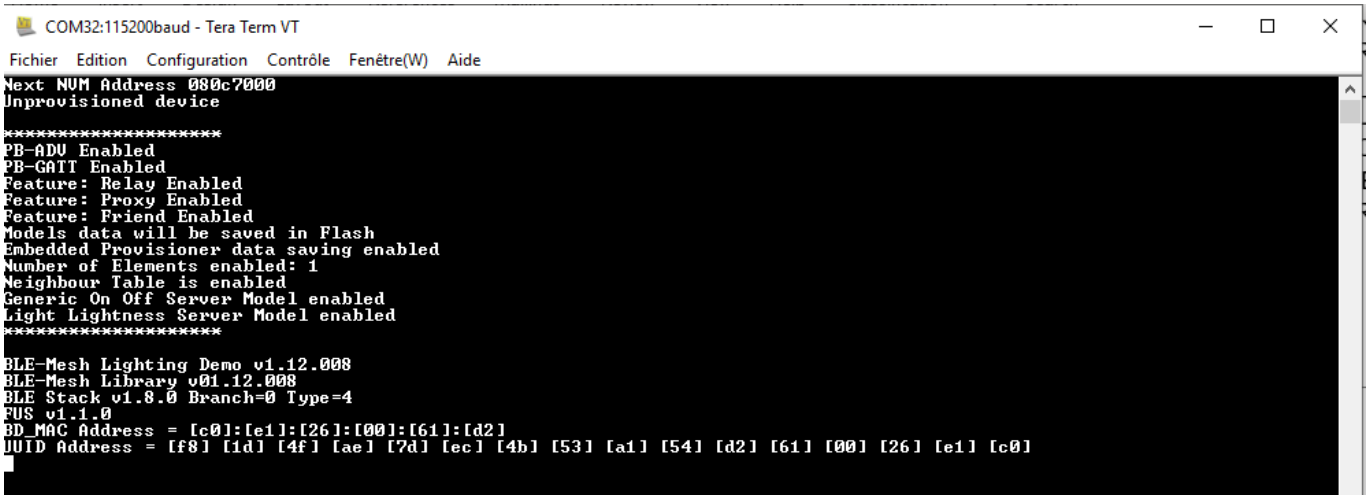
Open the `mesh_cfg_usr.h`” file make sure the macros are set as per the list below:

- `ENABLE_SERIAL_INTERFACE` macro is set to “1” to enable input from the serial interface, and
- `ENABLE_SERIAL_CONTROL` macros is set to “1”
- `ENABLE_SERIAL_PRVN` macro is set to “1”
- `ENABLE_PB_ADV` macro is set to “1”
- `ENABLE_RELAY_FEATURE` macro is set to “1”
- `ENABLE_PROXY_FEATURE` macro is set to “1”
- `ENABLE_FRIEND_FEATURE` macro is set to “1”
- `DYNAMIC_PROVISIONER` macro is set to “1”.

C.5 Starting the provisioner node

- Flash provisioner firmware in the board
- Reset the board, a welcome message is display the below log on the serial terminal indicating the board in provisioner mode qs illustrated in Figure 109:

Figure 109. Serial terminal screen output



C.6 Root node to create mesh

Write the following command on serial terminal to make a ROOT node for network creation: `atep root`

C.7 Scan Unprovisioned devices in range

Write the following command in the serial terminal to scan unprovisioned devices in the `range:atep` scan

The log on terminal indicates any unprovisioned devices with their corresponding UUIDs. This is illustrated in the figure below.

Figure 110. Unprovisioned devices in range

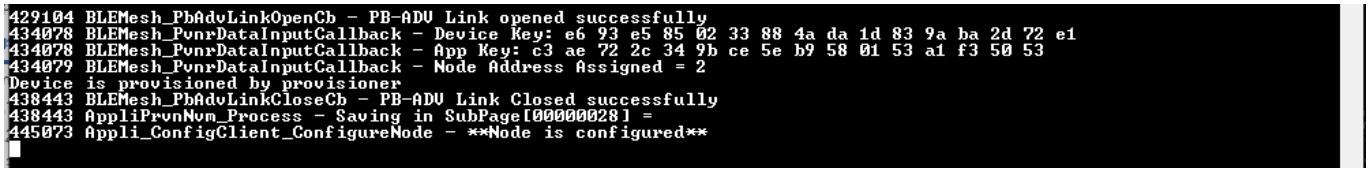
In this figure there are 2 devices (Device-0 and Device-1) which are unprovisioned in the network.

C.8 Provision and configure devices

Write the following command in the serial terminal: `atep prvn-0 2`.

This starts the provisioning and configuration of device-0. The "0" indicates it is destined for the device 0, "2" indicates the node address which the device is assigned. This is illustrated in the figure below.

Figure 111. Serial terminal provisioning information



Finally the PB-Adv connection link is closed and provisioning data is saved in Flash memory.

Device-0 has been provisioned and configured successfully by the provisioner.

C.9 Scan unprovisioned devices out of range of the provisioner

Write the following command in the provisioner serial terminal: `atep ndscan`.

This scans for unprovisioned devices out of range from the provisioner but in range of a node previously provisioned and configured by the provisioner.

Figure 112. Out of range unprovisioned devices

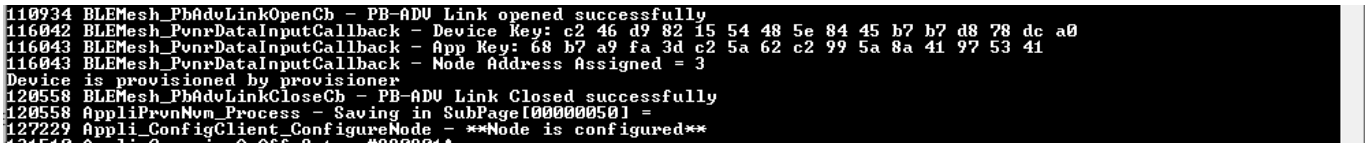
In the example illustrated above, 1 device (Device-0) is unprovisioned in the network

C.10 Provision and configure devices out of range from the provisioner

Write the following command in the provisioner serial terminal: `atep ndprvn-0 3`.

This starts the provisioning and configuration of the out of range device-0. The "0" indicates it is destined for the device 0, "3" indicates the node address which the device is assigned. This is illustrated in the figure below.

Figure 113. **Serial terminal output for out of range provisioned devices**



Revision history

Table 194. **Document revision history**

Date	Version	Changes
22-Mar-2019	1	Initial release.
15-Dec-2020	2	Document restructured to integrate and define the Bluetooth® Low Energy mesh application evolution following the Bluetooth® certifications.

Contents

1 **General information** 2
1.1 Acronyms and abbreviations 2
2 **BLE mesh technology concept** 3
2.1 Flooding protocol 3
2.2 Nodes type 3
2.3 System architecture 4
2.4 Security 4
2.5 Provisioning process 5
2.6 Elements and models 6
2.6.1 Elements 6
2.6.2 Models 6
2.6.3 Mandatory models 7
2.7 Addresses 7
2.8 Publish and subscribe 7
2.9 Messages 8
2.10 Friendship and low-power nodes 8
2.11 Mesh bearers 9
3 **Getting started** 10
3.1 Software and system requirements 10
3.2 Development kit description 10
3.2.1 P-NUCLEO-WB55 pack board presentation and setup 10
3.2.2 P-NUCLEO-WB55 board button usage 12
3.2.3 Dongle presentation and setup 12
3.2.4 Simple light system control demonstration 13
3.3 UART interface on the firmware 23
3.4 Build and install the BLE mesh application 25
3.5 Installing BLE mesh application from Cube Programmer 26
4 **STM32WB BLE mesh architecture** 28
4.1 Stack features 30
4.2 STM32WB BLE mesh application – middleware application 31

4.2.1 BLE mesh configuration header file . . . 31
4.2.2 BLE mesh middleware . . . 35
4.2.3 BLE mesh application . . . 36
4.2.4 BLE mesh middleware by file . . . 39
4.2.5 BLE mesh application by files . . . 59
4.2.6 Generic ON/OFF Server model interaction . . . 73
4.2.7 Generic ON/OFF Client model interaction . . . 74
4.3 Firmware package . . . 75
4.3.1 Root folder . . . 77
4.3.2 Driver folder . . . 77
4.3.3 Internal project folder . . . 78
4.3.4 Middleware folder . . . 79
5 Firmware initialization and configuration . . . 80
5.1 Setting the transmit power of a node . . . 80
5.2 MAC address management . . . 80
5.3 Initialization of application callbacks . . . 81
5.4 Initialization and main application loop . . . 81
6 Mesh networking information . . . 85
6.1 Local and remote concept . . . 85
6.2 Acknowledged and unacknowledged messages . . . 86
6.3 GATT connection/disconnection node . . . 87
6.4 Vendor model write command from remote node . . . 87
6.5 Vendor model read command from a remote node . . . 89
6.6 Application functions and callbacks . . . 90
6.6.1 User interface and indications . . . 90
6.6.2 User and button interface . . . 90
6.6.3 Device BLE configuration type interface . . . 91
6.6.4 Vendor model network data communication functions . . . 94
6.6.5 MAC address configuration . . . 95
6.6.6 BLE mesh node configuration . . . 95
6.6.7 BLE mesh library configuration . . . 111
7 BLE mesh models . . . 118

7.1 How to . . . 119
7.2 Generic models . . . 120
7.3 Lighting models . . . 121
7.4 Sensor models . . . 123
7.5 Vendor models . . . 123
8 More concept around BLE mesh . . . 124
8.1 Low-power and Friend nodes . . . 124
8.1.1 Friendship definitions . . . 124
8.1.2 Setup . . . 128
8.1.3 Power consumption profiles . . . 131
8.2 Multiple element nodes . . . 134
8.3 Embedded provisioner . . . 136
8.3.1 Provisioning definition . . . 136
8.3.2 Embedded Provisioner principle . . . 138
8.3.3 Setup . . . 138
Appendix A Bluetooth® Low Energy mesh serial gateway . . . 141
A.1 Introduction . . . 141
A.2 Getting started . . . 141
A.3 Serial gateway initialization . . . 141
A.4 Configuration of models to print data . . . 142
A.5 Enable all generic and light models . . . 142
A.6 Serial control commands . . . 143
A.6.1 Sensor Setup Server model command format . . . 143
A.6.2 Vendor Write model command format . . . 144
A.6.3 Vendor read model command format . . . 144
A.6.4 Generic ON/OFF set format . . . 145
A.6.5 Generic Level set data format . . . 145
A.6.6 Generic Level Delta set format . . . 145
A.6.7 Generic Power ON/OFF set data format . . . 146
A.6.8 Generic Default Transition Time Server set format . . . 146
A.6.9 Light Lightness setting data . . . 146

A.6.10 Light CTL set data format . . . 146
A.6.11 Light CTL Temperature Range set data format . . . 147
A.6.12 Light CTL Default Set data format . . . 147
A.6.13 Light HSL Set data format . . . 147
A.6.14 Light HSL Range Set data format . . . 147
A.6.15 Light HSL Default Set data format . . . 148
A.6.16 Light HSL Hue Set data format . . . 148
A.6.17 Light HSL Saturation Set data format . . . 148
A.6.18 Light LC Mode Set data format . . . 148
A.6.19 Light LC occupancy mode set data format . . . 149
A.6.20 Light LC Light OnOff Set data format . . . 149
A.6.21 Light LC Property Set data format . . . 149
A.6.22 Generic Power ON/OFF set data format . . . 149
A.6.23 Sensor Descriptor Get data format . . . 150
A.6.24 Sensor Cadence Get data format . . . 150
A.6.25 Sensor Cadence Set data format . . . 150
A.6.26 Sensor Settings Get data format . . . 150
A.6.27 Sensor Setting Get data format . . . 150
A.6.28 Sensor Setting Set data format . . . 151
A.6.29 Sensor Get data format . . . 151
A.6.30 Sensor Column Get data format . . . 151
A.6.31 Sensor Series Get data format . . . 151
A.6.32 Command List . . . 151
A.7 Example of Generic ON/OFF set ack . . . 154
Appendix B ST BLE Mesh upper tester protocol . . . 155
B.1 Introduction . . . 155
B.2 Hardware and software requirements . . . 155
B.3 Getting started . . . 155
B.4 Enabling the upper tester . . . 155
B.5 Configuration of models to print data . . . 155
B.6 Upper tester commands . . . 156
B.6.1 Command format . . . 156

B.6.2 Board management command . . . 157
B.6.3 Test specific command . . . 157
B.6.4 Mesh feature modification command . . . 159
B.6.5 Packet transmission command . . . 159
B.6.6 Status command . . . 159
Appendix C ST BLE Mesh provisioner protocol . . . 160
C.1 Introduction . . . 160
C.2 Hardware and software requirements . . . 160
C.3 Getting started . . . 160
C.4 Enabling the provisioner . . . 160
C.5 Starting the provisioner node . . . 160
C.6 Root node to create mesh . . . 161
C.7 Scan Unprovisioned devices in range . . . 161
C.8 Provision and configure devices . . . 161
C.9 Scan unprovisioned devices out of range of the provisioner . . . 161
C.10 Provision and configure devices out of range from the provisioner . . . 161
Revision history . . . 163

List of figures

Figure 1. BLE mesh topology . . . 4
Figure 2. BLE mesh library architecture . . . 4
Figure 3. Security done by provisioning . . . 5
Figure 4. Elements and Model – how device features are exposed . . . 6
Figure 5. Publish and Subscribe . . . 7
Figure 6. Messages . . . 8
Figure 7. Friendship messaging . . . 8
Figure 8. STM32WBx5 board connection with PC . . . 11
Figure 9. USB dongle board . . . 12
Figure 10. Demonstration details . . . 13
Figure 11. Demonstration illustration . . . 14
Figure 12. Sample project structure . . . 14
Figure 13. Nucleo BLE_MeshLightingPRFNode project . . . 14
Figure 14. Nucleo BLE_MeshLightingPRFNode file structure . . . 15
Figure 15. Dongle BLE_MeshLightingPRFNode project . . . 15
Figure 16. Dongle BLE_MeshLightingPRFNode file structure . . . 16
Figure 17. ST BLE device list . . . 17
Figure 18. Quick configuration screens . . . 18
Figure 19. Publication and subscription choices . . . 19
Figure 20. BLE mesh interface screen . . . 20
Figure 21. Group interface . . . 21
Figure 22. Single node demonstration illustration . . . 21
Figure 23. All node demonstration illustration . . . 22
Figure 24. SW1 LED actuation model . . . 22
Figure 25. Node unprovisioning . . . 23
Figure 26. BLE_MeshLightingProvisioner VCOM window . . . 24
Figure 27. BLE_MeshLightingPRFNode VCOM window . . . 25
Figure 28. Project type list . . . 25
Figure 29. Sample applications . . . 26
Figure 30. STM32CubeProgrammer interface . . . 26
Figure 31. Binary programming options . . . 27
Figure 32. STM32WB Bluetooth® Low Energy Architecture . . . 29
Figure 33. Firmware architecture . . . 30
Figure 34. STM32Cube_FW_WB BLE mesh-LightingPRFNode project structure . . . 38
Figure 35. Generic Server model code . . . 39
Figure 36. Generic Client model code . . . 41
Figure 37. Light Server model code . . . 42
Figure 38. Lighting Client model code . . . 46
Figure 39. Light LC Server model code . . . 46
Figure 40. Mesh middleware and library initialization . . . 48
Figure 41. Mesh models configuration . . . 48
Figure 42. Serial interface . . . 50
Figure 43. Serial control . . . 51
Figure 44. Serial upper tester . . . 52
Figure 45. Serial provisioner . . . 53
Figure 46. Application test . . . 53
Figure 47. Sensor Server model . . . 54
Figure 48. Vendor Server model . . . 56
Figure 49. Time and scene server model . . . 57
Figure 50. Service controller code . . . 58
Figure 51. BLE application code . . . 59
Figure 52. Mesh based application code . . . 59

Figure 53. Generic server models application . . . 62
Figure 54. Generic Client models application . . . 63
Figure 55. Light Server models application . . . 63
Figure 56. Light Client models application . . . 66
Figure 57. Light LC Server models application . . . 66
Figure 58. Flash accesses application . . . 67
Figure 59. Application Flash mapping . . . 68
Figure 60. Sensor Server application . . . 68
Figure 61. Vendor Server application . . . 70
Figure 62. Model interface . . . 70
Figure 63. Flash interface . . . 72
Figure 64. Pulse width modulation interface . . . 72
Figure 65. Example of Generic ON/OFF Server model interaction . . . 73
Figure 66. Example of Generic ON/OFF Client model interaction . . . 74
Figure 67. Folders, sub-folders and content of the package . . . 76
Figure 68. Root folder structure . . . 77
Figure 69. Driver folder . . . 77
Figure 70. Nucleo project folder . . . 78
Figure 71. Dongle project folder . . . 79
Figure 72. Middleware folder . . . 79
Figure 73. SetRemote/ProcessMessageCb actions with Generic ON/OFF model . . . 86
Figure 74. Vendor model write command data flow . . . 88
Figure 75. Vendor model read command from a remote node . . . 89
Figure 76. Model node locations . . . 119
Figure 77. Light Lightness Server and associated models . . . 121
Figure 78. Light CTL Server and associated models (left), Light CTL Temperature Server (right) . . . 122
Figure 79. Light HSL Server and associated models . . . 122
Figure 80. Sensor related models . . . 123
Figure 81. ReceiveDelay and ReceiveWindow timing sequence . . . 125
Figure 82. PollTimeout timing sequence . . . 126
Figure 83. Friendship messages . . . 127
Figure 84. Friend node project workspace selection . . . 128
Figure 85. Friend feature selection . . . 128
Figure 86. FN_NO_OF_LPNS location . . . 128
Figure 87. LPN project selection . . . 129
Figure 88. Low-power feature . . . 129
Figure 89. Pressing SW1 button on Friend node, command flow illustration . . . 130
Figure 90. Pressing SW1 button on LPN1, command flow illustration . . . 130
Figure 91. Low-power friend request process with no answer . . . 131
Figure 92. Low-power friend request process with answer . . . 131
Figure 93. Friendship establishment sequence . . . 132
Figure 94. Friend poll after establishing a friendship . . . 132
Figure 95. Friend poll retries . . . 133
Figure 96. Friend poll retries and friendship termination . . . 133
Figure 97. Element numbering definition . . . 134
Figure 98. Quick configuration menu . . . 135
Figure 99. Node interface screen . . . 135
Figure 100. Link setup process . . . 136
Figure 101. BLE mesh node state machine . . . 137
Figure 102. BLE mesh provisioning protocol stack . . . 137
Figure 103. Embedded provisioner project workspace selection . . . 138
Figure 104. BLE-mesh network illustration . . . 140
Figure 105. Bluetooth® Low Energy mesh serial gateway . . . 141
Figure 106. Node with unicast address – 8 send message . . . 154

Figure 107. Node with unicast address - 9 send message . . . 154
Figure 108. BLE mesh network set up . . . 155
Figure 109. Serial terminal screen output . . . 160
Figure 110. Unprovisioned devices in range . . . 161
Figure 111. Serial terminal provisioning information . . . 161
Figure 112. Out of range unprovisioned devices. . . . 161
Figure 113. Serial terminal output for out of range provisioned devices . . . 162

List of tables

Table 1. Acronyms and abbreviations . . . 2
Table 2. Button usage and LEDs management . . . 12
Table 3. Virtual communication port settings . . . 23
Table 4. Feature definition . . . 30
Table 5. Feature values . . . 31
Table 6. Proxy, relay and friend features definition header files . . . 31
Table 7. Trace macro file headers . . . 32
Table 8. BLE services and mesh model files . . . 35
Table 9. Core subfolder files . . . 36
Table 10. STM32_WPAN/App subfolder files . . . 37
Table 11. Generic Server model code functions . . . 39
Table 12. Generic Client model code functions . . . 41
Table 13. Light Server model code functions . . . 42
Table 14. Lighting Client model code functions . . . 46
Table 15. Light LC Server model functions . . . 47
Table 16. Mesh middleware and library initialization . . . 48
Table 17. Mesh model configuration included dependent files . . . 49
Table 18. Supported models . . . 49
Table 19. Serial interface functions . . . 50
Table 20. Serial control functions . . . 51
Table 21. Serial upper tester funcitons . . . 52
Table 22. Serial provisioner functions . . . 53
Table 23. Application test functions . . . 53
Table 24. Sensor Server model . . . 54
Table 25. Vendor Server model functions . . . 56
Table 26. Time and scene server model functions . . . 57
Table 27. SVCCTL_Init() functions . . . 58
Table 28. APP_BLE_Init() sub-functions called . . . 59
Table 29. Mesh library callback functions . . . 60
Table 30. SW1 button action . . . 61
Table 31. Mesh_task() rescheduling processes . . . 61
Table 32. Appli_Process() subfunctions . . . 61
Table 33. Generic Server model functions . . . 62
Table 34. Light Server models application functions . . . 63
Table 35. Light LC Server models application functions . . . 66
Table 36. Sensor Server application functions . . . 69
Table 37. Vendor Server application functions . . . 70
Table 38. Model interface functions . . . 71
Table 39. MAC address management . . . 80
Table 40. Appl_task functions . . . 83
Table 41. Appli_LedCtrl . . . 90
Table 42. Appli_ShortButtonPress . . . 90
Table 43. Appli_LongButtonPress . . . 90
Table 44. Appli_UpdateButtonState . . . 90
Table 45. Appli_BleSetTxPowerCb . . . 91
Table 46. Appli_BleSetUUIDCb . . . 91
Table 47. Appli_BleSetProductInfoCB . . . 91
Table 48. Appli_BleUnprovisionedIdentifyCb . . . 91
Table 49. Appli_BleGattConnectionCompleteCb . . . 92
Table 50. Appli_BleGattDisconnectionCompleteCb . . . 92
Table 51. Appli_BleSetNumberOfElementsCb . . . 92
Table 52. Appli_BleAttentionTimerCb . . . 92

Table 53. Appli_BleOutputOOBAuthCb . . . 93
Table 54. Appli_BleInputOOBAuthCb . . . 93
Table 55. Appli_BleDisableFilterCb . . . 93
Table 56. BLEMesh_PbAdvLinkCloseCb . . . 93
Table 57. Appli_LedStateCtrlCb . . . 93
Table 58. Vendor_WriteLocalDataCb . . . 94
Table 59. Vendor_ReadLocalDataCb . . . 94
Table 60. Appli_CheckBdMacAddr . . . 95
Table 61. Appli_GetMACFromUniqueNumber . . . 95
Table 62. BLEMesh_InitUnprovisionedNode . . . 95
Table 63. BLEMesh_InitProvisionedNode . . . 95
Table 64. BLEMesh_IsUnprovisioned . . . 96
Table 65. BLEMesh_Unprovision . . . 96
Table 66. BLEMesh_BleHardwareInitCallBack . . . 96
Table 67. BLEMesh_GetUnprovisionState . . . 96
Table 68. BLEMesh_GetAddress . . . 96
Table 69. BLEMesh_GetPublishAddress . . . 97
Table 70. BLEMesh_GetSubscriptionAddress . . . 97
Table 71. BLEMesh_SetTTL . . . 97
Table 72. BLEMesh_GetTTL . . . 97
Table 73. BLEMesh_SetNetworkTransmitCount . . . 98
Table 74. BLEMesh_GetNetworkTransmitCount . . . 98
Table 75. BLEMesh_SetRelayRetransmitCount . . . 98
Table 76. BLEMesh_GetRelayRetransmitCount . . . 98
Table 77. BLEMesh_SetRelayFeatureState . . . 98
Table 78. BLEMesh_SetFriendFeatureState . . . 99
Table 79. BLEMesh_SetLowPowerFeatureState . . . 99
Table 80. BLEMesh_GetFeatures . . . 99
Table 81. BLEMesh_TrsptIsBusyState . . . 100
Table 82. BLEMesh_SetHeartbeatCallback . . . 100
Table 83. BLEMesh_SetAttentionTimerCallback . . . 100
Table 84. BLEMesh_UnprovisionCallback . . . 100
Table 85. BLEMesh_ProvisionCallback . . . 101
Table 86. BLEMesh_PbAdvLinkOpenCb . . . 101
Table 87. BLEMesh_PbAdvLinkCloseCb . . . 101
Table 88. BLEMesh_ProvisionRemote . . . 101
Table 89. BLEMesh_CreateNetwork . . . 101
Table 90. BLEMesh_SetSIGModelsCbMap . . . 102
Table 91. GetApplicationVendorModels . . . 102
Table 92. BLEMesh_GetSleepDuration . . . 102
Table 93. BLEMesh_UpperTesterDataProcess . . . 102
Table 94. BLEMesh_PrintStringCb . . . 103
Table 95. BLEMesh_PrintDataCb . . . 103
Table 96. BLEMesh_FnFriendshipEstablishedCallback . . . 103
Table 97. BLEMesh_FnFriendshipClearedCallback . . . 103
Table 98. BLEMesh_LpnFriendshipEstablishedCallback . . . 104
Table 99. BLEMesh_LpnFriendshipClearedCallback . . . 104
Table 100. BLEMesh_LpnDisableScan . . . 104
Table 101. BLEMesh_StopAdvScan . . . 104
Table 102. BLEMesh_SetProvisioningServAdvInterval . . . 105
Table 103. BLEMesh_SetUnprovisionedDevBeaconInterval . . . 105
Table 104. BLEMesh_SetProxyServAdvInterval . . . 105
Table 105. BLEMesh_SetSecureBeaconInterval . . . 106
Table 106. BLEMesh_SetCustomBeaconInterval . . . 106

Table 107. BLEMesh_CustomBeaconGeneratorCallback . . . 106
Table 108. BLEMesh_CustomBeaconReceivedCallback . . . 106
Table 109. ApplicationGetSigModelList . . . 107
Table 110. ApplicationGetCLIENTSigModelList . . . 107
Table 111. BLEMeshSetSelfModelList . . . 107
Table 112. ApplicationGetVendorModelList . . . 108
Table 113. ApplicationChkSigModelActive . . . 108
Table 114. ApplicationChkVendorModelActive . . . 108
Table 115. ApplicationGetConfigServerDeviceKey . . . 108
Table 116. BLEMesh_NeighborAppearedCallback . . . 109
Table 117. BLEMesh_NeighborRefreshedCallback . . . 109
Table 118. BLEMesh_GetNeighborState . . . 110
Table 119. BLEMesh_SetFault . . . 110
Table 120. BLEMesh_ClearFault . . . 110
Table 121. BLEMesh_Shutdown . . . 110
Table 122. BLEMesh_Resume . . . 111
Table 123. BLEMesh_Init . . . 111
Table 124. BLEMesh_GetLibraryVersion . . . 111
Table 125. BLEMesh_GetLibrarySubVersion . . . 111
Table 126. BLEMesh_Process . . . 111
Table 127. BLEMesh_SetVendorCbMap . . . 112
Table 128. BLEMesh_SetRemoteData . . . 112
Table 129. BLEMesh_SetRemotePublication . . . 112
Table 130. BLEMesh_ModelSendMessage . . . 113
Table 131. BLEMesh_ReadRemoteData . . . 113
Table 132. BLEMesh_SendResponse . . . 114
Table 133. VendorModel_SendResponse . . . 114
Table 134. Model_SendResponse . . . 115
Table 135. ConfigModel_SendMessage . . . 115
Table 136. ConfigModel_SelfPublishConfig . . . 116
Table 137. ConfigModel_SelfSubscriptionConfig . . . 116
Table 138. ConfigClient_SelfModelAppBindConfig . . . 116
Table 139. ConfigClient_SelfModelAppBindConfig . . . 117
Table 140. List of available models . . . 118
Table 141. Demonstration board breakdown . . . 129
Table 142. Mesh model control commands format . . . 143
Table 143. Mesh control parameter definition . . . 144
Table 144. Vendor Write parameter definition . . . 144
Table 145. Vendor read arameter definition . . . 144
Table 146. Mesh control commands format . . . 145
Table 147. Parameter definition . . . 145
Table 148. Generic Level set data format . . . 145
Table 149. Generic Level Delta set data format . . . 145
Table 150. Generic Power ON/OFF set data format . . . 146
Table 151. Generic Default Transition Time Server set data format . . . 146
Table 152. Light Lightness data format . . . 146
Table 153. Light CTL set data format . . . 146
Table 154. Light CTL Temperature Range data format . . . 147
Table 155. Light CTL default data format . . . 147
Table 156. Light HSL Set data format . . . 147
Table 157. Light HSL Range Set data format . . . 147
Table 158. Light HSL Default Set data format . . . 148
Table 159. Light HSL Hue Set data format . . . 148
Table 160. light HSL saturation data format . . . 148

Table 161. Light LC mode data format . . . 148
Table 162. Light LC Occupancy data format . . . 149
Table 163. Light LC Light OnOff Set data format . . . 149
Table 164. Light LC Property Set data format . . . 149
Table 165. Generic Power On/Off set data format . . . 149
Table 166. Sensor descriptor get data format . . . 150
Table 167. Sensor Cadence Get data format . . . 150
Table 168. Sensor Cadence Set data format . . . 150
Table 169. Sensor Settings Get data format . . . 150
Table 170. Sensor Setting Get data format . . . 150
Table 171. Sensor Setting Set data format . . . 151
Table 172. Sensor Get data format . . . 151
Table 173. Sensor Column Get data format . . . 151
Table 174. Sensor Series Get data format . . . 151
Table 175. Generic command list . . . 151
Table 176. Light lightness and CTL command list . . . 152
Table 177. Data format for the sensor cadence set lines . . . 154
Table 178. Command format . . . 156
Table 179. Board reset command format . . . 157
Table 180. Board unprovisionning command format . . . 157
Table 181. IV update - keep/remove 96-hour limit format . . . 157
Table 182. IV update - send test signal format . . . 157
Table 183. Delete/subscribe to new group address format . . . 157
Table 184. Break existing friendship format . . . 158
Table 185. Clears replay protection list format . . . 158
Table 186. Orders IUT to allow one Net Key index format . . . 158
Table 187. Clears heartbeat subscription count format . . . 158
Table 188. Set system faults for health model format . . . 158
Table 189. Clears heartbeat subscription count format . . . 158
Table 190. Mesh feature modification command structure . . . 159
Table 191. Network authentication packet format . . . 159
Table 192. App packet transmission format . . . 159
Table 193. Status command format . . . 159
Table 194. Document revision history . . . 163

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved