BLE Mesh
BL654 Sample *smart*BASIC Application

Application Note *29.3.31.8-MESH310-8 - rel 1*

1 INTRODUCTION

If you are migrating a smartBASIC application which works with an older mesh firmware then plase refer to the migration section in this document which highlights the changes you have to make to your application.

In July of 2017, the Bluetooth SIG released *Mesh Profile Specification v1.0* describing a Mesh Profile running on top of any BLE device which is v4.0 or newer.

The following are the goals of this document:

- Provide an overview of BLE mesh from an application perspective by introducing you to some early beta BLE mesh functionality in the Laird BL654 module
- Demonstrate how to utilize it in a sample *smart*BASIC application by testing the functionality over the UART using a light switch client and server example.

Given the *smart*BASIC implementation, the mesh explanation is in the context of the event-driven *smart*BASIC programming paradigm.

The mesh functionality described in this application note is for **testing and demonstrative purpose only**. It is **not fit for production** as it is built using the v3.1.0 release of the BLE mesh SDK from Nordic Semiconductor. This release allows provisioning of devices using either an iOS or Android device using an application from Nordic called *nRF Mesh*.

As this is based on the v3.1.0 release of the SDK from Nordic, Laird reserves the right to change the specifics of the API that is used to expose the Mesh functionality and is described in this application note. Potential changes will be in line with any changes that Nordic may introduce as they continue to work on the stack towards an eventual feature-rich production release which they intend to have approved by the Bluetooth SIG. This implies that Laird will inherit that certification and pass it along to customers who use a Laird module that incorporates mesh. The certification may extend to Sig adopted models as and when smartBASIC applications implementing those models are available.

1.1 Low Power Node Overview

Ble Mesh is power hungry and that is because it uses a managed flood adverts mechanism for propogating messages to all the nodes in a netwrok, and to do that the specification recommends that the radio receiver be switched on for nearly 100% of the time. Given that on average most chipsets consume around 5milliAmps when the radio is listening for packets then battery operation is feasible only when massive capacities are used and will not provide months of operation.

This is not unique to just mesh. Theoretically the same problem applies when a BLE device is in a GATT connection, but it manages to operate from coin cell batteries simply because when in a connection, it knows exactly when an incoming message is going to arrive and so only switches on its radio at those times. This results in very low duty cycles which results in extremely low power consumption.

This low duty cycling paradigm is used to given the Low Power Node feature in BLE Mesh.

Simply put, a low power device relies on a Friend node, which is not battery powered, to listen for incoming messages on its behalf. The friend will cache all incoming messages intended for the Low Power Node in a message queue.

The Low Power Node will then read that message queue as and when it needs to by polling the friend. This means that the low power node can keep its radio switched off and like in a GATT connection only switch on the radio when interogating the

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

Friend for messages to be transferred from that queue. Hence with this type of operation the average power consumption can get to very low values similar to GATT connections.

The BLE Mesh specification has defined a protocol for a low power node to seek a friend and then poll for messages from it and is taken care of by the underlying stack.

2 REQUIREMENTS

To use this sample application, you need the following:

- DVK-BL654 (development kits) – Minimum of three. Ideally you will need a total of five to view the interaction with multiple on/off servers
  One of the five dev kits is used as a sniffer for mesh adverts to get a better understanding of mesh operation because it shows reassuring activity (such as unprovisioned beacons). The sniffer device can be a BL652 devkit because the *smart*BASIC application runs as-is on both platforms.
- An optional Nordic PCA10056 devkit to be used as a Friend Node if you want to try the Low Power Node example.
- PC with spare USB ports (using a USB hub, if appropriate)
- UwTerminalX – available for Windows, Linux and Mac: https://github.com/LairdCP/UwTerminalX/releases
- Engineering mesh-capable firmware for the BL654
  The response to the AT command “AT I 3” will have the word ‘MESH310’.
- Sample command manager *smart*BASIC applications demonstrating mesh functionality called cmd.manager.mesh.sb
- Sample apps *$autorun$.mesh.light.switch.client.sb*, *$autorun$.mesh.light.switch.server.sb* and *$autorun$.mesh.light.switch.server.lpn.sb* which are included in the firmware zip file.
  The app *$autorun$.mesh.light.switch.server.lpn.sb* application demonstrates the light switch server application when operating as low power node.
- An optional MeshSniff *smart*BASIC application called *$autorun$.mesh.sniff.sb* which is included in the firmware zip file. This is used to sniff and display mesh-related advert packets which can be loaded onto a BL652 or BL654 devkit. See section 11 for more details
- An iOS or Android smartphone with the latest OS and the latest application called *nRF Mesh* installed from either Apple Store or Google Play respectively.

Note: For the purposes of this document, we assume you have familiarised yourself with compiling/loading *smart*BASIC applications

---

The switches and jumpers on the BL654 devkits shall be configured as per the photograph below.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

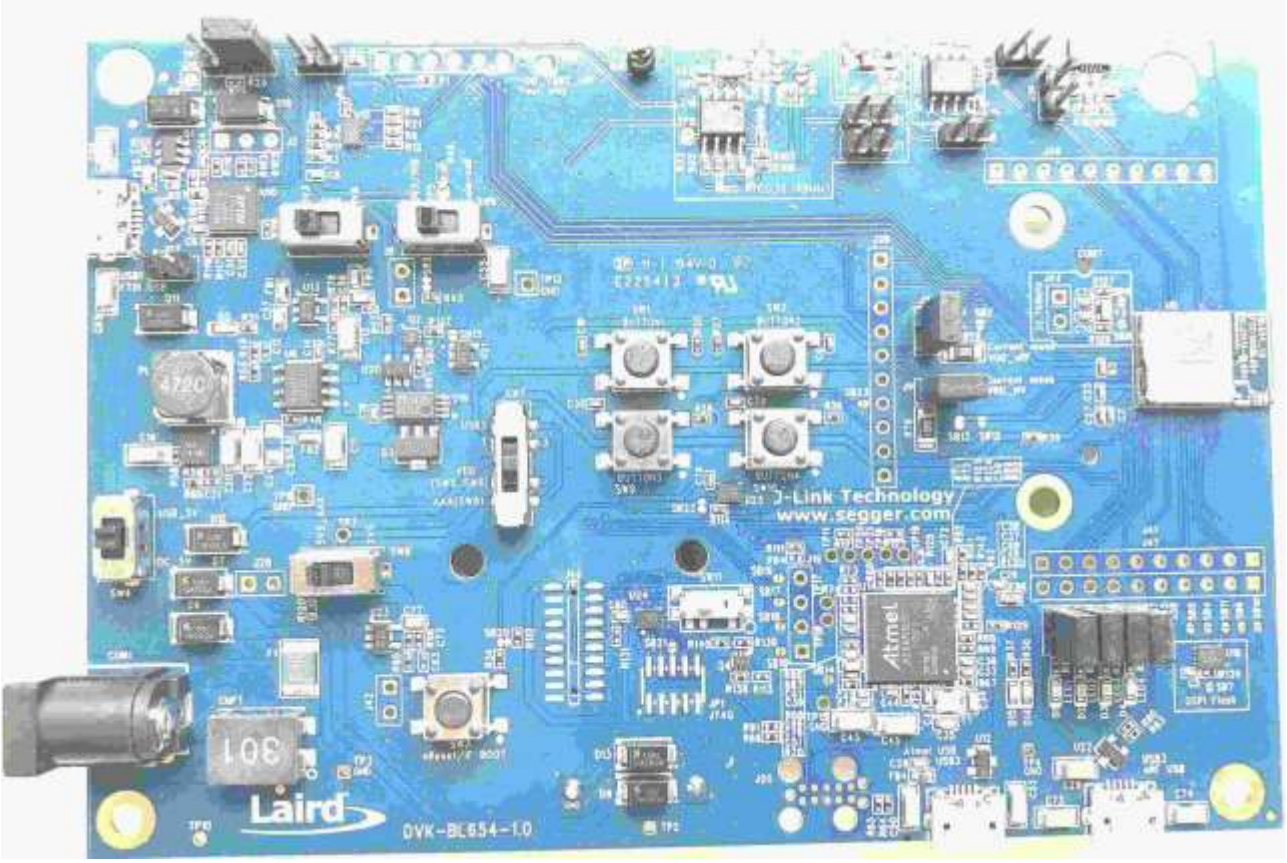


Figure 1: BL654 development kit

WARNING: If the four jumpers on the bottom right of the board are missing then the four LEDs that are below them will not operate. This mesh demo relies on showing those LED being switched on and off.

3 RELEASE SPECIFIC NOTES

This application note describes mesh functionality exposed via Laird's *smart*BASIC programming language implemented on top of Nordic Semiconductor's Mesh SDK which is at release level 3.1.0. This release is not fully functional and offers the advert bearer, relay functionality and Low Power node features. There is no ability to offer Friend capability as that will not be available until a future release of the Nordic SDK.

Please refer to Nordic Semiconductor's release notes for more details about the 3.1.0 SDK.

4 DEMO DESCRIPTION

The examples demonstrated in this application note are light switch client and server devices with provisioning of all devices using an iOS or Android smartphone. It will also optionally demonstrate Low Power Node capability.

The client device implements the client behavior of a Nordic Semiconductor custom Light Switch model and likewise the server device implements the server behavior of the Light Switch model. The Low Power Node capability is also demostrated using a low power node variant of the light switch server application.

In the overview section, a model is described as an array of opcode and associated handlers. The Nordic custom Light Switch model consists of the following opcodes and the recipient for each opcode.

Table 1: Nordic custom light switch model opcodes and opcode recipients

Opcode Name	Opcode	Role	Msg Data

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

SIMPLE_ON_OFF_OPCODE_SET	0xC10059	Server		0/1	tid
SIMPLE_ON_OFF_OPCODE_GET	0xC20059	Server			
SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE	0xC30059	Server		0/1	tid
SIMPLE_ON_OFF_OPCODE_STATUS	0xC40059		Client	0/1	

* 0059 – Nordic Semiconductor Company ID

tid – Transaction ID

The *tid* data field is just an incrementing number which wraps at 0xFF to 0x00; Nordic did not attached any specific meaning to it. It is incremented each time a SET or SET_UNRELIABLE message is sent.

The example demonstrated in this appnote shows provisioning of up to three light switch servers and a single light switch client.

The example is best described using the following figure (Figure 2).

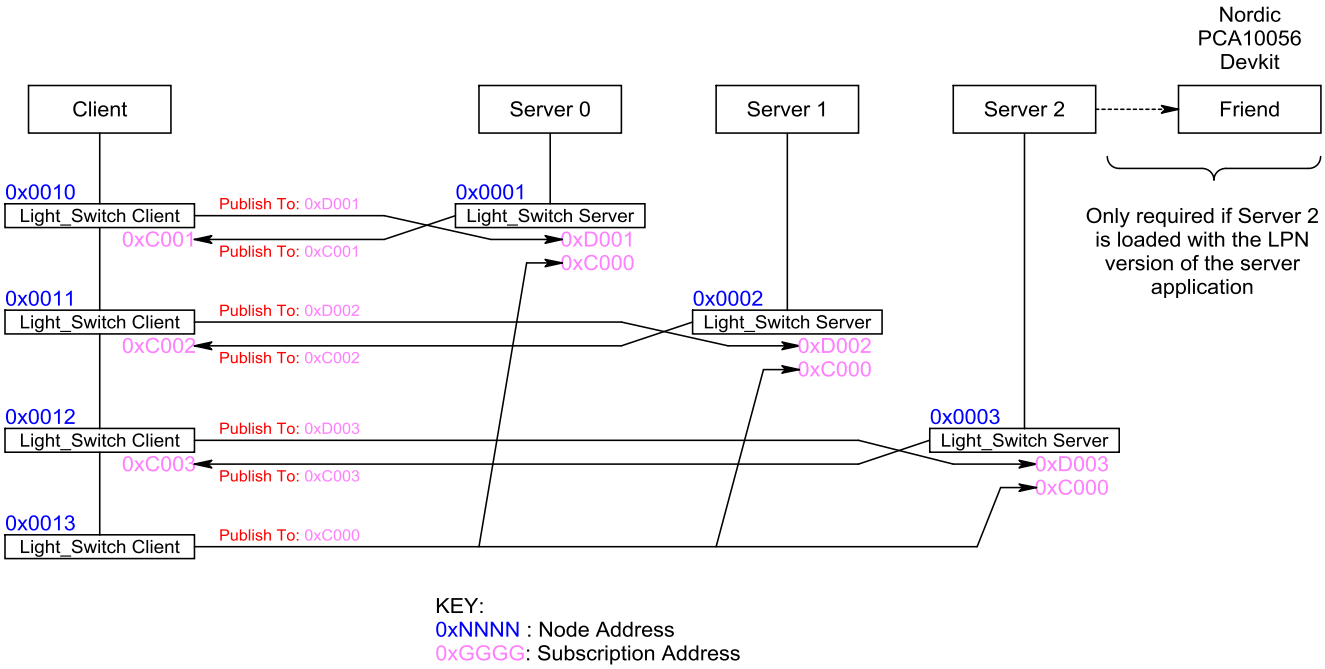


Figure 2: Demo client/server structure

Figure 2 shows five devkits labelled Client, Server 0, Server 1, Server 2 and Friend. Each server contains a single element implementing Nordic's custom Light Switch Model server roles and the client contains four elements each containing a single Light Switch Model client. The Friend is loaded with the Zephyr RTOS based Friend firmware provided in binary form and loaded using the batch file supplied which is appropritely named.

When the client devkit is powered up, it registers four elements, each containing the same model (Nordic Light Switch Client Model, Model ID=0x00590001). It starts to advertise an unprovisioned beacon and GATT Mesh Provisioning Service advert and contains the same device UUID in both which always remains the same.

When an unprovisioned server is powered up, it registers a single element with a single model (Nordic Light Switch Server Model, Model ID=0x00590000). It starts to advertise an unprovisioned beacon and GATT Mesh Provisioning Service advert and contains the same device UUID in both which always remains the same.

The Nordic smartphone application *nRF Mesh* on an iOS or Android device is used to provision and configure the client and all servers. It will also provision the Friend Node if you decide to try that.

The client's first model is provisioned so that it has a node address of 0x0010, publish address of 0xD001, and subscribes to group address 0xC001 (all addresses in range 0xC000 to 0xFF00 are group addresses). The second model is provisioned so that it has a node address of 0x0011, publish address of 0xD002, and subscribes to group addresses 0xC002. The third model

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

is provisioned so that it has a node address of 0x0012, publish address of 0xD003, and subscribes to group address 0xC003. Finally, the fourth model is provisioned so that it publishes to group address 0xC000 (to which all servers will subscribe) and does not subscribe to any address.

- Server 0 is provisioned so that it has a node address of 0x0001, publish address of 0xC001, and subscribes to group address 0xC000 (which is the publish address of the fourth model in the client) and group addess 0xD001 (which is publish address of first model in the client).
- Server 1 is provisioned so that it has a node address of 0x0002, publish address of 0xC002, and subscribes to group address 0xC000 (which is the publish address of the fourth model in the client) and group address 0xD002 (which is the publish address of the second model in the client)
- Server 2 is provisioned so that it has a node address of 0x0003, publish address of 0xC003, and subscribes to group address 0xC000 (which is the publish address of the fourth model in the client) and group address 0xD003 (which is the publish address of the third model in the client).

The provisioning means first model in client controls server 0, second model in client controls server 1, third model in client controls server 2, and finally fourth model in client controls all three servers because they subscribe to its publish address.

At any time, if the client wants to set the on/off state of a server, it publishes a message with opcode SIMPLE_ON_OFF_OPCODE_SET or SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE. The former results in a response message with opcode SIMPLE_ON_OFF_OPCODE_STATUS but the latter does not.

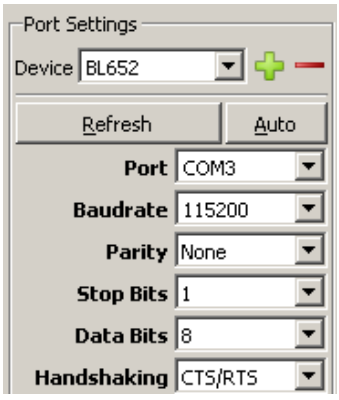
The *smart*BASIC sample app *$autorun$.mesh.light.switch.client.sb* is an application that implements the client device and is programmed so that the Laird devkit BUTTON1 controls the LED on server 0, BUTTON2 controls the LED on server 1, BUTTON3 controls the LED on server 2, and BUTTON4 controls the LED on all servers at the same time.

The *smart*BASIC sample app *$autorun$.mesh.light.switch.server.sb* is an application that implements the server device and is programmed so that the Laird devkit BUTTON1 locally toggles the state of LED and also publishes the state of the LED so that all subscribers are informed of the local state.

5 BL654 Development Kit Firmware Load

To set up each development kit with the engineering mesh firmware, locate the mesh firmware zip file, unzip into a folder, and follow these steps for all of the dev kits:

1. Connect your BL654 development kit to your PC via the USB Micro cable. The power LED illuminates when the board is receiving power.
2. Open UwTerminalX.
3. In the Config tab, set the parameters and COM port associated with your development board.



4. Click **OK** to advance to the Terminal tab.
5. Use UwTerminalX to return the BL654 to factory defaults using the command ***at&f**** as shown in Figure 3.

   If you are using a new development board with the sample application, you may need to remove the autorun jumper on J12 and press the reset button to exit out of the sample application and then issue the *at&f** command to erase the file system and all non-volatile data.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

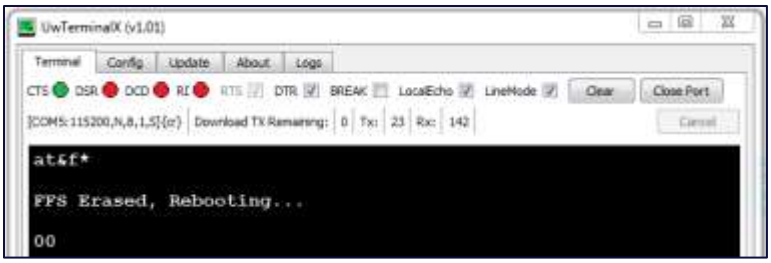


Figure 3: Returning the BL654 to factory defaults

6. Close UwTerminalX.
7. In the folder where the Mesh firmware was unzipped, locate the *_DownloadFirmwareUart.bat* file and launch it.

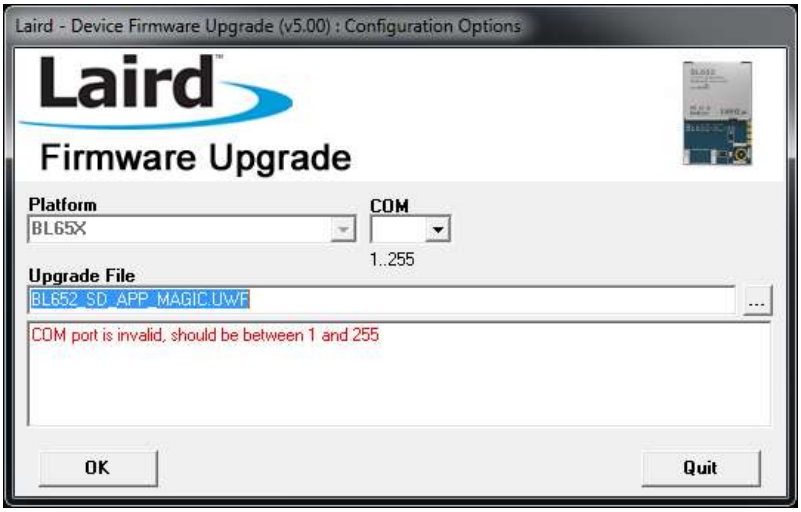


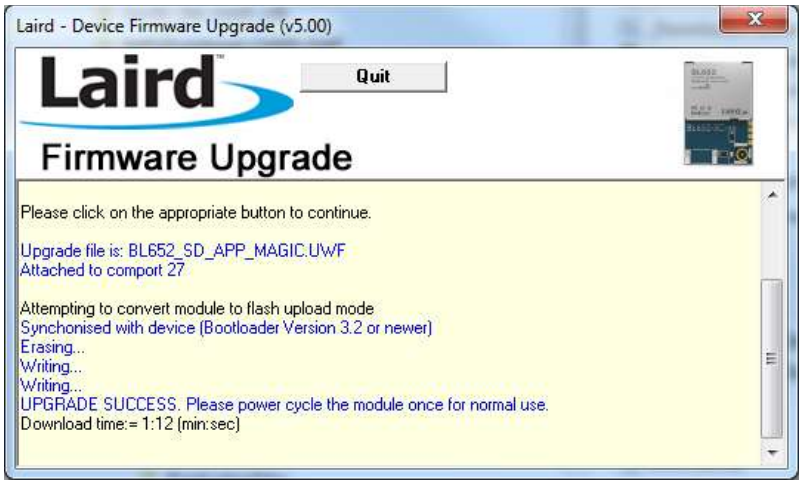
Figure 4: Launch _DownloadFIrmwareUart.bat file

8. In the COM field, enter the same comport number as was used in step 3 and confirm that the message *COM port is invalid, should be between 1 and 255* disappeared.
9. Click **OK**.

Figure 5: Valid COM port is entered

10. Click **Proceed**.
11. When the upgrade is complete, click **Quit**.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610



12. Open UwTerminalX.
13. In the Config tab, set the parameters and COM port associated with your development board.
14. Click **OK** to advance to the Terminal tab.
15. Send the command AT I 3 and confirm the following response, where nn is 10 or higher:
    10 3 29.1.1.0-MESH211-nn

6 BL654 DEVELOPMENT KIT SMARTBASIC APP LOAD

If you have five boards then label them: Client, Server 0, Server 1, Server 2, and Sniff.

For boards labelled *Client*, *Server 0*, *Server 1* (optional), and *Server 2* (optional), perform the following steps:

1. For the *Client* board, load the Mesh *smart*BASIC example application. Use the right-click menu and select **XCompile + load**.

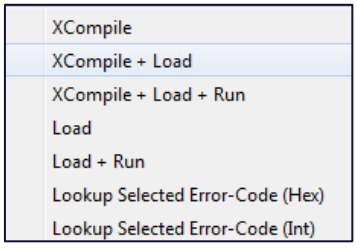


Figure 6: Select XCompile + Load

2. Select the *$autorun$.mesh.light.switch.client.sb* file which is located in the *MeshApps* subfolder.

   It should take approximately ten seconds for the mesh program to load. Run the mesh example by typing *at+run "$autorun$"* followed by Enter or pressing the reset button.

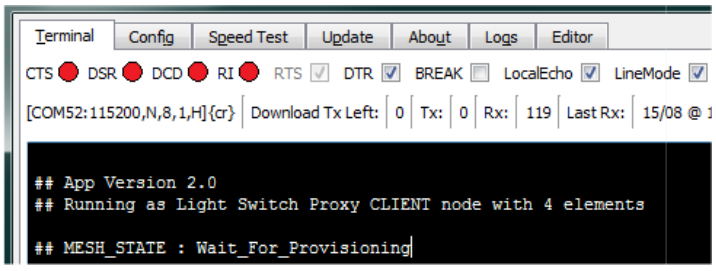


Figure 7: Running the mesh program

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

For the board labelled *Sniff*, perform the following steps:

1. Load the Mesh Sniff *smart*BASIC example application. Use the right-click menu and select **XCompile + Load**.

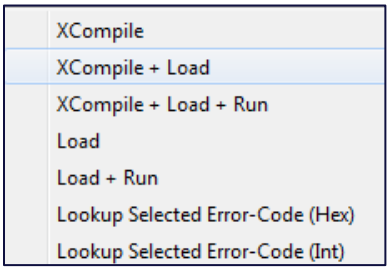


Figure 8: Select XCompile + Load

2. Select the *$autorun$.mesh.sniff.sb* file which is located in the *MeshApps* subfolder.

   It should take approximately ten seconds for the mesh program to load. Run the sniff example by typing *$autorun$* followed by Enter or pressing the reset button.

7 PCA10056 Development Kit Firmware Load

If you decide to try the low power node variant of in server 2, then you will need a Nordic PCA10056 devkit loaded with frimware capable of offering a Friend capability.

To load the firmware, connect the devkit to your PC (and assuming you have already played with that devkit and loaded other Nordic firmware) all you need to do is launch the batch file called “ZephyrFriendLoad.bat”. If it it successful then the devkit will be ready for provisioning and use the Nordic smartphone application to do so.

8 Launch and Test the Mesh Example

8.1 Overview

This section describes a step-by-step guide to creating and provisioning a mesh of up to four devices (we recommend four, but only two are required) implementing Nordic’s Light Switch example as per their SDK, but implemented in Laird’s easy-to-use event driven *smart*BASIC programming environment.

The devices can be provisioned using either an iOS or an Android application which is developed by Nordic Semiconductor called *nRF Mesh*. Only step-by-step instructions for provisioning using an iOS device are described in the this application note.

Warning:

Before you begin, please take a note of the revision of the BL654 devkit you are using.

The production release has the silkscreen label *DVK-BL654-1.0* at the corner where the DC jack CON1 is located.

You may have the following earlier versions: *DVK-BL654-A1* or *DVL-BL654-B0*. These earlier DVKs do not have the correct labels for the four buttons; trial and error is required to locate which button is which.

When you run this example, server 0,1 and 2 DVKs only use BUTTON 1 and the client DVK uses all four buttons.

In each, when the appropriate button is pressed, a debug message prints on the UwTerminalX screen to identify which

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

button was pressed. Experiment to locate the correct button. We then recommend you make a mark on the devkit appropriately for future reference.

8.2 Launch and Test Process Initial Steps

To launch and test the mesh example, follow these steps:

1. Connect all boards to your PC.
2. Open as many UwTerminalX instances as there are boards using the comport that your PC exposes for each board.
3. Reset each board via the reset button on the devkit. Confirm that you see the following messages on the client board (Figure 9), server boards (Figure 10), and sniff board (Figure 11).

```
## Running as Light Switch CLIENT node with 4 elements

## MESH_EVENT : Provision - Wait for
```

Figure 9: Client board message

```
## Running as Light Switch Proxy SERVER node

## MESH_EVENT : Provision - Wait for
```

Or if Low Power Node variant in Server 2 then..

```
## Running as Light Switch SERVER node - LPN

## MESH_EVENT : Provision - Wait for
```

Figure 10: Server boards message

```
MESH SNIFFER

OK
>
```

Figure 11: Sniff board message

4. If necessary, revert the boards into unprovisioned and clean states.

Note: If this is the first time you are running this test and the boards are already in a clean, unprovisioned state, you can skip this step.
If, at any time, you think the boards may have non-volatile mesh information (so aren't in an unprovisioned state) or you are uncertain, it is best practice to revert all the boards.
If you clean one board, clean all the boards.

a. In the UwTerminalX toolbar, untick the DTR checkbox.
b. Tick/untick the BREAK checkbox.
c. This resets the module and ensures that the *smart*BASIC $autorun$ application does not automatically launch.

   In this mode, the module accepts AT commands.

d. Confirm this by sending AT and seeing a *00* response.
e. Send the `AT&F 0x100000` command. This erases all flash sectors used by the Mesh stack. This is interpreted as an unprovisioned state.
f. In UwTerminalX, click **Clear**.
g. Tick the DTR checkbox.
h. Tick/untick the BREAK checkbox to reset the board.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

i. Confirm that the board's UwTerminalX displays the following:

```
## MESH_STATE : Wait_For_Provisioning
```

Figure 12: Board is waiting for provisioning

5. On the Sniff board UwTerminalX screen, tick/untick the BREAK checkbox after ensuring that the DTR checkbox is ticked.
6. Switch off the client board and all the server boards as you will be powering up and provisioning them using a smartphone one by one. This sequential startup is not necessary for the provisioning process, but it helps to make this step-by-step guide unambiguous. As you get more familiar with the *nRF Mesh* smartphone app, you will be able to ignore this and provision selected devices at will.

The following sections instruct you on actual provisioning.

8.3 Setting Up Server 0

To set up Server 0 for provisioning, follow these steps:

1. Switch on the Server 0 board and observe the following sniff board traffic:

```
01EC7380404DE0 PB-GATT[-50] (DevUUID=)457E9BDFF56F1791E04D408073EC2B15 (OOB=)0000 DevName=ls-server
01EC7380404DE0 PB-ADV [-58] (DevUUID=)457E9BDFF56F1791E04D408073EC2B15 (OOB=)0000 (URIhash=)
01EC7380404DE0 PB-GATT[-50] (DevUUID=)457E9BDFF56F1791E04D408073EC2B15 (OOB=)0000 DevName=ls-server
01EC7380404DE0 PB-GATT[-49] (DevUUID=)457E9BDFF56F1791E04D408073EC2B15 (OOB=)0000 DevName=ls-server
01EC7380404DE0 PB-ADV [-57] (DevUUID=)457E9BDFF56F1791E04D408073EC2B15 (OOB=)0000 (URIhash=)
```

Figure 13: Sniff board traffic

This traffic shows that Server 0 started advertising, that it is unprovisioned, and that it can accept provisioning either over adverts (PB-ADV) or via a BLE connection (PB-GATT).

Note that both advert forms contain the same UUID and OOB data. In your case, the UUID value will be very different as each mesh device has a unique value.

The following applies to each row:

Field 1	The Bluetooth address of the unprovisioned mesh device
Field 2	`PB_GATT[-50]` or `PB_ADV[-58]` This means this is an unprovisioned mesh beacon and the [-50] or [-58] is the RSSI value for the beacon/advert that arrived.
Field 3	`(DevUUID=)9F0CE6498091A00B4865DD9386892175` The device UUID that is factory programmed into the device and is always constant for that particular device. This will vary for you.
Field 4	`(OOB=)0000` The out-of-band bit mask which conveys how the authentication phase of the provisioning will take place. The bit mask is reproduced from the spec as follows:

Bit	Description
0	Other
1	Electronic/URI
2	2D machine-readable code
3	Bar code
4	Near Field Communication (NFC)
5	Number
6	String
7	Reserved for future use
8	Reserved for future use
9	Reserved for future use

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

	10	Reserved for future use
	11	On box
	12	Inside box
	13	On piece of paper
	14	Inside manual
	15	On device
Field 5	`(URIhash=)` This is empty and may contain an eight-hex digit hash value of a URL that would be advertised by this mesh device in a normal advert (arranged via the GATT stack). It can also be used to direct the user to a website for installation or product details. See *smart*BASIC function BleAdvertStart() in the *BL654 smartBASIC Extensions Guide* for more details.	

8.4 Provisioning Server 0 by Phone

8.4.1 iOS

To provision your iOS Server 0, follow these steps:

1. Launch the *nRF Mesh* application.
2. From the bottom tabs, select **Settings**.
3. From the Settings page Reset Mesh State row, select/tap **Forget Network** and confirm in the pop-up dialog box. This erases all states from the smartphone and allows you to start a new mesh network. It also recreates a new random network and creates three new random app keys.
4. Locate the Application Version row and ensure that the version is at least v1.0.2.
5. From the bottom tabs, select **Scanner**. The scanner screen scans for all devices that are in an unprovisioned state and offers mesh provisioning service via GATT.
6. Select the *LS P-Server* device to advance to the Node Provision screen.
7. Ensure the following:
   - the Name row shows *LS P-Server*
   - the Unicast address shows *0x0001*
   - there is a value in the Appkey 1 row

   Note: You can edit any of these rows by tapping it. This is especially relevant if you want to allocate a node address different from the one displayed.

8. On the top right, tap **Identify** to view additional information that is pertinent to the provisioning step. From that window, tap **Provision** from the top right to open the UwTerminalX window for that device. This window displays a BLE connection being established and then some *## MESH STATE* messages that show the progress of the provisioning process.

   When the new Progress row at the bottom of the phone screen reaches 100%, the screen automatically changes to the Network screen.

   Note: If the progress gets stuck before it reaches 100%, kill the phone app, power-cycle the module, and start again. Ensure that, in the Node Provision screen, the unicast address to be allocated is still correct.

   On the UwTerminal screen, the following message confirms that the address of the first node is 1 (==0x0001) and that there is one element: **`## MESH_STATE : Provisioned <Addr=1 Count=1>`**

   The Network screen displays all the nodes in this network. For now, there is only one – *LS P-SERVER : 0001* where *0001* is the node address allocated to it.

   It also has one element and three models. In the loaded *smart*BASIC application (*$autorun$.mesh.light.switch.proxy.server.sb*), we only registered one light switch server model in function *ls_server()*; the other two are the configuration and health foundation models which all devices inherit by default.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

LS P-SERVER corresponds to the #define DEVICE_NAME that exists in the loaded *smart*BASIC app.

9. Tap the LS P-Server box to advance to the Node Configuration screen. This screen shows details of the three models.
10. Tap the Simple OnOff Server row to advance to the Simple OnOff Server window.
11. Tap the APPKEY BINDING row and select **Appkey 1**. The APPKEY BINDING value now displays *Key Bound* with index *0000*.
12. Because Server 0 publishes to the group address *0xC001* (see Figure 2), do the following:
    a. In the Simple OnOff Server screen, tap to add the value for the PUBLICATION ADDRESS row to advance to the Publication Settings screen. On this screen, you see that the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xC001* and tap **Set** to return to the Publication Settings screen.
    d. On the top right of the screen, tap **Apply Publication** to accept and return to the previous screen.
13. Because the subscription address is *0xC000* and 0xD001 (see Figure 2), do the following:
    a. Tap **Add Subscription Address**.
    b. Enter *0xC000* in the new dialog box.
    c. Tap **Add**.
    d. Confirm that the Simple OnOff Server window lists *C000* as a subscription address.
    e. Tap **Add Subscription Address**.
    f. Enter *0xD001* in the new dialog box.
    g. Tap **Add**.
    h. Confirm that the Simple OnOff Server window lists *C000 & D001* as subscription addresses.
    i. On the top left of the window, tap **Back**.

    In the Node Configuration window, an icon in the third row shows two vertical arrows and a horizontal line. This shows that the model was assigned a publication and a subscription address.
14. On the top left of the window, tap **Network**.
15. On the top right of the Network window, tap **Disconnect** to advance to the appropriate UwTerminalX window and see confirmation of a BLE disconnection.

    The Sniffer UwTerminalX window now displays a new type of adverts called PROXY(NET_ID). This is basically the device now advertising a Mesh Proxy Service for the phone to get back into the network via a GATT connection, if necessary.

8.4.2 Android

TBD. Awaiting a refreshed nRF Mesh that has same iOS functionality.

8.5 Provisioning Server 1 by Phone

This step is recommended but can be skipped if you have only one server.

Turn on board Server 1 and observe the sniff board traffic that is similar to what you saw in a previous step.

8.5.1 iOS

To provision your iOS Server 1, follow these steps:

1. Launch the *nRF Mesh* application if it's not still running from the previous step.
2. From the bottom tabs, select **Scanner**. The scanner screen scans for all devices that are in an unprovisioned state and offers mesh provisioning service via GATT.
3. Select the *LS P-Server* device to advance to the Node Provision screen.
4. Ensure the following:
   - the Name row shows *LS P-Server*
   - the Unicast address shows *0x0002*

Note: You can edit any of these rows by tapping it. Change the unicast address to *0x0002*, if necessary.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

5. On the top right, tap **Identify** to view additional information that is pertinent to the provisioning step.
6. From that window, tap **Provision** from the top right to open the UwTerminalX window for that device. This window displays a BLE connection being established and then some *## MESH STATE* messages that show the progress of the provisioning process.

   When the new Progress row at the bottom of the phone screen reaches 100%, the screen automatically changes to the Network screen.

   Note: If the progress gets stuck before it reaches 100%, kill the phone app, power-cycle the module, and start again. Ensure that, in the Node Provision screen, the unicast address to be allocated is still correct.

   The Network screen displays all the nodes in this network. For now, there is only one – *LS P-SERVER : 0002* where *0002* is the node address allocated to it.

   It also has one element and three models. In the loaded *smart*BASIC application (*$autorun$.mesh.light.switch.proxy.server.sb*), we only registered one light switch server model in function *ls_server()*; the other two are the configuration and health foundation models which all devices inherit by default.

   LS P-SERVER corresponds to the #define DEVICE_NAME that exists in the loaded *smart*BASIC app.
7. Tap the LS P- SERVER : 0002 box to advance to the Node Configuration screen. This screen shows details of the three models.
8. Tap the Simple OnOff Server row to advance to the Simple OnOff Server window.
9. Tap the APPKEY BINDING row and select **Appkey 1**. The APPKEY BINDING value now displays *Key Bound* with index *0000*.
10. Because Server 1 publishes to the group address *0xC002* (see Figure 2), do the following:
    a. In the Simple OnOff Server screen, tap to add the value for the PUBLICATION ADDRESS row to advance to the Publication Settings screen. On this screen, you see that the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xC002* and tap **Set** to return to the Publication Settings screen.
    d. On the top right of the screen, tap **Apply Publication** to accept and return to the previous screen.
11. Because the subscription address is *0xC000* & *0xD002* (see Figure 2), do the following:
    a. Tap **Add Subscription Address**.
    b. Enter *0xC000* in the new dialog box.
    c. Tap **Add**.
    d. Confirm that the Simple OnOff Server window lists *C000* as a subscription address.
    e. Tap **Add Subscription Address**.
    f. Enter *0xD002* in the new dialog box.
    g. Tap **Add**.
    h. Confirm that the Simple OnOff Server window lists *C000 and D002* as subscription addresses.
    i. On the top left of the window, tap **Back**.

       In the Node Configuration window, an icon in the third row shows two vertical arrows and a horizontal line. This shows that the model was assigned a publication and a subscription address.
12. On the top left of the window, tap **Network**.
13. On the top right of the Network window, tap **Disconnect** to advance to the appropriate UwTerminalX window and see confirmation of a BLE disconnection.

    The Sniffer UwTerminalX window now displays more adverts called PROXY(NET_ID) with different MAC addresses. This is basically the device now advertising a Mesh Proxy Service for the phone to get back into the network via a GATT connection, if necessary.

8.5.2 Android

TBD. Awaiting a refreshed nRF Mesh that has same iOS functionality.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

8.6 Provisioning Server 2 by Phone

This step is recommended but can be skipped if you have only two servers.

Turn on board Server 2 and observe the sniff board traffic that is similar to what you saw in a previous step.

Note: if you have loaded the LPN variant of the smartBASIC application then the provisioning process is no different so just fiollow the same process that foillows.

8.6.1 iOS

To provision your iOS Server 2, follow these steps:

1. Launch the *nRF Mesh* application if it's not still running from the previous step.
2. From the bottom tabs, select **Scanner**. The scanner screen scans for all devices that are in an unprovisioned state and offers mesh provisioning service via GATT.
3. Select the *LS P-Server* device to advance to the Node Provision screen.
4. Ensure the following:
   - the Name row shows LS P-Server
   - the Unicast address shows 0x0003

   Note: You can edit any of these rows by tapping it. Change the unicast address to *0x0003*, if necessary.

5. On the top right, tap **Identify** to view additional information that is pertinent to the provisioning step.
6. From that window, tap **Provision** from the top right to open the UwTerminalX window for that device. This window displays a BLE connection being established and then some *## MESH STATE* messages that show the progress of the provisioning process.

   When the new Progress row at the bottom of the phone screen reaches 100%, the screen automatically changes to the Network screen.

   Note: If the progress gets stuck before it reaches 100%, kill the phone app, power-cycle the module, and start again. Ensure that, in the Node Provision screen, the unicast address to be allocated is still correct.

   The Network screen displays all the nodes in this network. For now, there is only one – *LS P-SERVER : 0003* where *0003* is the node address allocated to it.

   It also has one element and three models. In the loaded *smart*BASIC application (*$autorun$.mesh.light.switch.proxy.server.sb*), we only registered one light switch server model in function *ls_server()*; the other two are the configuration and health foundation models which all devices inherit by default.

   LS P-SERVER corresponds to the #define DEVICE_NAME that exists in the loaded *smart*BASIC app.

14. Touch Tap the LS P- SERVER : 0003 box to advance to the Node Configuration screen. This screen shows details of the three models.
15. Tap the Simple OnOff Server row to advance to the Simple OnOff Server window.
16. Tap the APPKEY BINDING row and select **Appkey 1**. The APPKEY BINDING value now displays *Key Bound* with index *0000*.
17. Because Server 2 publishes to the group address *0xC003* (see Figure 2), do the following:
    a. In the Simple OnOff Server screen, tap to add the value for the PUBLICATION ADDRESS row to advance to the Publication Settings screen. On this screen, you see that the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xC003* and tap **Set** to return to the Publication Settings screen.
    d. On the top right of the screen, tap **Apply Publication** to accept and return to the previous screen.
18. Because the subscription address is *0xC000 & 0xD003* (see Figure 2), do the following:
    a. Tap **Add Subscription Address**.
    b. Enter *0xC000* in the new dialog box.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

c. Tap **Add**.
d. Confirm that the Simple OnOff Server window lists *C000* as a subscription address.
e. Tap **Add Subscription Address**.
f. Enter *0xD003* in the new dialog box.
g. Tap **Add**.
h. Confirm that the Simple OnOff Server window lists *C000 & D003* as subscription addresses.
i. On the top left of the window, tap **Back**.

In the Node Configuration window, an icon in the third row shows two vertical arrows and a horizontal line. This shows that the model was assigned a publication and a subscription address.

19. On the top left of the window, tap **Network**.
20. On the top right of the Network window, tap **Disconnect** to advance to the appropriate UwTerminalX window and see confirmation of a BLE disconnection.

The Sniffer UwTerminalX window now displays more adverts called PROXY(NET_ID) with different MAC addresses. This is basically the device now advertising a Mesh Proxy Service for the phone to get back into the network via a GATT connection, if necessary.

8.6.2 Android

TBD. Awaiting a refreshed nRF Mesh that has same iOS functionality.

8.7 Provisioning the Client by Phone

Turn on the board Client and observe the sniff board traffic that is similar to what you saw in a previous step.

8.7.1 iOS

To provision your iOS Client, follow these steps:

1. Launch the nRF Mesh application if it's not still running from the previous step.
2. From the bottom tabs, select Scanner. The scanner screen scans for all devices that are in an unprovisioned state and offers mesh provisioning service via GATT.
3. Select the *LS P-CLIENT* device to advance to the Node Provision screen.
4. Check the following:
   - the Name row shows *LS P-CLIENT*
   - the Unicast address now shows *0x0004*

Note: You can edit any of these rows by tapping it.

---

5. Change the Unicast address to *0x0028*.
6. On the top right, tap **Identify** to view additional information that is pertinent to the provisioning step.
7. From that window, tap **Provision** from the top right to open the UwTerminalX window for that device. This window displays a BLE connection being established and then some *## MESH STATE* messages that show the progress of the provisioning process.

   When the new Progress row at the bottom of the phone screen reaches 100%, the screen automatically changes to the Network screen.

Note: If the progress gets stuck before it reaches 100%, kill the phone app, power-cycle the module, and start again. Ensure that, in the Node Provision screen, the unicast address to be allocated is still correct.

The UwTerminal window displays the following message:
`## MESH_STATE : Provisioned <Addr=40 Count=4>.` This confirms that the address of the first node is 40 (==0x0028) and that there are four elements.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

The Network screen displays all the nodes in this network. For now, there is only one – *LS P-CLIENT : 0028* where *0028* is the node address allocated to it.

It also has four elements and six models. In the loaded *smart*BASIC application (*$autorun$.mesh.light.switch.proxy.client.sb*), we only registered four light switch client models in function *ls_client()*; the other two are the configuration and health foundation models which all devices inherit by default.

The *LS P-CLIENT* corresponds to the #define DEVICE_NAME that exists in the loaded *smart*BASIC application.

8. Tap the LS P- CLIENT : 0028 box to advance to the Node Configuration screen. This screen shows details of the six models.
9. Tap the Simple OnOff Server row to advance to the Simple OnOff Server window.
10. Tap the APPKEY BINDING row and select **Appkey 1**. The APPKEY BINDING value now displays *Key Bound* with index *0000*.
11. Because the client's first model publishes to the address *0xD001*, do the following: (see Figure 2), do the following:
    a. In the Simple OnOff Server screen, tap to add the value for the PUBLICATION ADDRESS row to advance to the Publication Settings screen. On this screen, you see that the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xD001* and tap **Set** to return to the Publication Settings screen.
    d. On the top right of the screen, tap **Apply Publication** to accept and return to the previous screen.
12. Because the subscription address is *0xC001*(see Figure 2), do the following:
    a. Tap **Add Subscription Address**.
    b. Enter *0xC001* in the new dialog box.
    c. Tap **Add**.
    d. Confirm that the Simple OnOff Client window lists *C001* as a subscription address.
    e. On the top left of the window, tap **Back**.
    f. Tap the second row (Simple OnOff Client) to advance to the Simple OnOff Client) window.
    g. Tap the APPKEY BINDING row and select **Appkey 1**. The APPKEY BINDING value now displays *Key Bound* with an index *0000*.
13. Because the client's second model publishes to the address *0xD002* (see Figure 2), do the following:
    a. In the Simple OnOff Client window, tap to add the value for the PUBLICATION ADDRESS row. This reveals the Publication Settings window where the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xD002*.
    d. Tap **Set** to return to the Publication Settings window.
    e. On the top right of the window, tap **Apply Publication**.
14. Because the client's second model has the subscription address *0xC002* (see Figure 2), do the following:
    a. Tap **Add Subscription Address**.
    b. Enter *0xC002*.
    c. Tap **Add**.
    d. Confirm that the Simple OnOff Client window lists *C002* as a subscription address.
    e. On the top left of the window, tap **Back**.
    f. Tap the third row (Simple OnOff Client) to advance to the SimpleOnOff Client window.
    g. Tap the APPKEY BINDING row and select **Appkey 1**. This returns you to the previous window and the APPKEY BINDING value now shows *Key Bound* with index *0000*.
15. Because the client's third model publishes to the address *0xD003* (see Figure 2), do the following:
    a. In the Simple OnOff Client window, tap to add the value for the PUBLICATION ADDRESS row. This reveals the Publication Settings window where the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xD003*.
    d. Tap **Set** to return to the Publication Settings window.
    e. On the top right of the window, tap **Apply Publication**.
16. Because the client's third model has the subscription address *0xC003* (see Figure 2), do the following:

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

a. Tap **Add Subscription Address**.
b. Enter *0xC003*.
c. Tap **Add**.
d. Confirm that the Simple OnOff Client window lists *C003* as a subscription address.
e. On the top left of the window, tap **Back**.
f. Tap the fourth row (Simple OnOff Client) to advance to the SimpleOnOff Client window.
g. Tap the APPKEY BINDING row and select **Appkey 1**. This returns you to the previous window and the APPKEY BINDING value now shows *Key Bound* with index *0000*.

17. Because the client's forth model publishes to the address *0xC000* (see Figure 2), do the following:
    a. In the Simple OnOff Client window, tap to add the value for the PUBLICATION ADDRESS row. This reveals the Publication Settings window where the first row has a default publication address of *0xCEEF*.
    b. Tap **0xCEEF**.
    c. In the pop-up dialog box, enter *0xC000*.
    d. Tap **Set** to return to the Publication Settings window.
    e. On the top right of the window, tap **Apply Publication**.
    f. On the top left of the window, tap **Back**.

    In the Node Configuration window, an icon in the first three Simple OnOff Client model rows shows two vertical arrows and a horizontal line. This shows that the model was assigned a publication and a subscription address. The fourth has an icon with only an up arrow and the horizontal line which implies that it has a publication address but not a subscription address.
18. On the top left of the window, tap **Network**.
19. On the top right of the Network window, tap **Disconnect** to advance to the appropriate UwTerminalX window and see confirmation of a BLE disconnection.

    The Sniffer UwTerminalX window now displays more adverts called PROXY(NET_ID) with different MAC addresses. This is basically the device now advertising a Mesh Proxy Service for the phone to get back into the network via a GATT connection, if necessary.

8.7.2 Android

TBD. Awaiting a refreshed nRF Mesh that has same iOS functionality.

8.8 Testing Nordic's Simple OnOff Server and Client Mesh Model

In all four devkits, locate the four LEDs. They are located near the area where there are two USB sockets close together. There are four two-pin headers with the following labels: J26, J37, J45, and J39. Ensure that all four of these have a jumper which connects the four LEDs to the appropriate GPIO pins on the module.

All four devices are now provisioned so you can test Nordic's Simple OnOff server and client mesh model. To do this, follow these steps:

1. On the client, press BUTTON 1. Its UwTerminal window displays the following

```
## BUTTON 1
## Sent Mesh Message

## SIMPLE_ON_OFF_OPCODE_STATUS
## SIMPLE_ON_OFF_OPCODE_STATUS
```

   This confirms that BUTTON 1 was pressed and that a mesh message was sent. It also displays two status messages: 1) the direct replay to the SET message (reliable) and 2) the status that was published so that all subscribers are aware of the change.

   LED1 should now be ON.

   The following displays on Server 0's UwTerminal window:

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

```
## SIMPLE_ON_OFF_OPCODE_SET
```

This displays when a message with the opcode SET is received.

LED 1 should now be ON.

2. On Server 0, press BUTTON 1. It's UwTerminal window then displays the following:

```
## BUTTON 1
## Sent Mesh Message
```

This confirms that BUTTON ! was pressed and that a mesh message was sent.

LED1 on that devkit should now be OFF.

The following displays on the client's UwTerminal window:

```
## SIMPLE_ON_OFF_OPCODE_STATUS
```

This displays when a STATUS message is received.

LED 1 should now be OFF.

Note the following:

- Pressing BUTTON 2 on the client devkit controls LED1 on Server 1.
- Pressing BUTTON 3 on the client devkit controls LED1 on Server 2.
- Pressing BUTTON 4 on the client devkit controls LED1 on all server devkits and locally LED1 to LED3 follow those states.

9 MESH-RELATED *SMART*BASIC FUNCTIONS AND EVENTS

This section describes the functions and events that were added to this engineering firmware release. Because it is based on v2.1.1 of the Nordic Mesh SDK, Laird reserves the right to change or delete any functions and events listed in this section.

9.1 Mesh-related AT Commands

9.1.1 AT&F 0x100000

This AT command is used to delete all mesh-related flash sectors so that all state information is deleted. This results in the device reverting to the unprovisioned state and so it begins sending unprovisioned beacons.

9.2 Mesh-related Result Codes

Many of the new functions return a result code. There is a lookup feature in UwTerminalX that describes what those failure result codes mean. The new mesh-related result codes for this alpha release are as follows:

```
UWRESULTCODE_BLE_MESH_INVALID_OPCODEID          0x60C0
UWRESULTCODE_BLE_MESH_TOO_MANY_MODELS           0x60C1
UWRESULTCODE_BLE_MESH_OPCODE_TABLE_FULL         0x60C2
UWRESULTCODE_BLE_MESH_MODEL_NOT_ADDED           0x60C3
UWRESULTCODE_BLE_MESH_PREV_MODEL_EMPTY          0x60C4
UWRESULTCODE_BLE_MESH_PREV_ELEMENT_EMPTY        0x60C5
UWRESULTCODE_BLE_MESH_CURRENT_MODEL_EMPTY       0x60C6
UWRESULTCODE_BLE_MESH_TOO_MANY_ELEMENTS         0x60C7
UWRESULTCODE_BLE_MESH_TABLE_EMPTY               0x60C8
UWRESULTCODE_BLE_MESH_LAST_MODEL_EMPTY          0x60C9
UWRESULTCODE_BLE_MESH_DUPLICATE_OPCODEID        0x60CA
UWRESULTCODE_BLE_MESH_INVALID_MODELHANDLE       0x60CB
UWRESULTCODE_BLE_MESH_INVALID_MODELINDEX        0x60CC
UWRESULTCODE_BLE_MESH_INVALID_PACKEDOPCODE      0x60CD
```


Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

```
UWRESULTCODE_BLE_MESH_INVALID_REPLYINFO         0x60CE
UWRESULTCODE_BLE_MESH_ALREADY_STARTED           0x60CF
UWRESULTCODE_BLE_MESH_CANNOT_BE_PROVISIONER     0x60D0
UWRESULTCODE_BLE_MESH_INVALID_DATALEN           0x60D1
UWRESULTCODE_BLE_MESH_INVALID_TIMEOUT           0x60D2
UWRESULTCODE_BLE_MESH_INV_STATIC_AUTH_DATA      0x60D3
UWRESULTCODE_BLE_MESH_LAST_ELEMENT_EMPTY        0x60D4
UWRESULTCODE_BLE_MESH_MODELS_NOTALLOWED         0x60D5
UWRESULTCODE_BLE_MESH_PROVISIONER_BUSY          0x60D6
UWRESULTCODE_BLE_MESH_ATT_MTU_TOO_SMALL         0x60D7
UWRESULTCODE_BLE_MESH_PUB_REL_PENDING           0x60D8
UWRESULTCODE_BLE_MESH_NOPENDING_PUB_REL         0x60D9
UWRESULTCODE_BLE_MESH_NOT_STARTED               0x60DA
UWRESULTCODE_BLE_MESH_INV_FEATURE_COMBO         0x60DB
UWRESULTCODE_BLE_MESH_INVALID_DURATION          0x60DC

UWRESULTCODE_BLE_MESH_INVALID_RSSI              0x60DD
UWRESULTCODE_BLE_MESH_INVALID_RXWINSZ           0x60DE
UWRESULTCODE_BLE_MESH_INVALID_SUBSLSTSZ         0x60DF
UWRESULTCODE_BLE_MESH_ALREADY_FRIENDED          0x60E0
UWRESULTCODE_BLE_MESH_LPNSTATE_NOT_IDLE         0x60E1
UWRESULTCODE_BLE_MESH_FRIEND_NOT_CONNECTED      0x60E2
```

9.3 Mesh-related Functions

9.3.1 BleMeshAddElement

When a mesh is started, it must know the number of elements the device will expose as well as the models and messages each of those elements will host. The element/mesh/message information can be viewed as a tree structure of information and that collection is referred to as in the specification as the ‘composition’.

Use this function to add elements to the container, which is initialised with an element with a location value of 0 on powerup, to which is added more instances of models and op-codes. Each element ends up getting a unique node address by the provisioner. If this function is called when the container has the default element and no models have been added yet, then the function call overrides the default and that is how the location value can be updated for the first element.

As mentioned in the description for BleMeshAddMessage() which is described later, a new element is needed if a device will end up with multiple instances of messages. The Mesh specification mandates that an element SHALL have only one instance of a message.

Note: For those familiar with how a USB device works when plugged into a host, it sends configuration data describing itself. The composition data serves a similar function in Mesh provisioning.

BleMeshAddElement(nLocation)

Returns	INTEGER : resultCode 0x0000 : Success 0x0607 : Location value not in range 0x0000 to 0xFFFF 0x60C5 : Previous element empty 0x60C6 : Current model empty 0x60C7 : Too many elements. Limit will be exceeded.
Arguments:	
nLocation	**byVAL *nLocation* AS INTEGER.** Specifies the location description as defined in the GATT Bluetooth Namespace Descriptors which can be found here and is a value in the range 0x0000 to 0xFFFF.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

	For example:- **0x0000 : unknown** 0x0101 : back 0x0107 : backup 0x0103 : bottom 0x0110 : external 0x010A : flash 0x0100 : front 0x010B : inside 0x010F : internal 0x010D : left 0x0105 : lower 0x0106 : main 0x0109 : supplementary 0x010C : outside 0x010E : right 0x0102 : top 0x0104 : upper 0x0001 to 0x00FF : just conveys the value, e.g 0x0004=fourth, 0x000A=tenth

9.3.2 BleMeshAddSigModel

An element in a device (the default added on powerup or specifically using BleMeshAddElement()) shall have one or more models. Use this function to add a model using a 16-bit SIG adopted identifier to the mesh schema. It is added to the most recently added element. A model in turn contains opcodes; a function detailed later is used to do that.

BleMeshAddSigModel(nModelId, handleModel)

Returns	INTEGER : resultCode 0x0000 : Success 0x0607 : nModelId value not in range 0x0000 to 0xFFFF 0x60C4 : Previously added model has no messages attached 0x60C1 : Too many models have been defined in total 0x60CC : handleModel is not recognised as a model handle
Arguments:	
nModelId	**byVAL *nModelId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is model ID as adopted by the Bluetooth SIG and described in the specification “Mesh Model Specification”. For example that specification defines 0x1000 as a Generic OnOff Server and 0x1001 as a Generic OnOff Client.
handleModel	**byREF *handleModel* AS INTEGER.** On Entry if this model is going to be an extension of another earlier added model then it shall be the handle of that model obtained when BleMeshAddSigModel() or BleMeshAddVendorModel() was called, **otherwise it shall contain 0**. If this model is an extension of another one, then it will share the subsciption list with the shared model. On Exit, this is an opaque handle value that the *smart*BASIC app uses to describe a model when an API will interact with a model or when a message arrives, this value will be presented to enable the developer to channel the behaviour accordingly We recommend that you store it in a global *smart*BASIC variable.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

9.3.3 BleMeshAddVendorModel

An element in a device (the default added on powerup or specifically using BleMeshAddElement()) has one or more models. Use this function to add a Model using a 32-bit vendor identifier to the mesh schema. It is added to the most recent-added element. A model in turn contains messages; a function detailed later is used to do that.

BleMeshAddVendorModel(nCompanyId, nModelId, handleModel)

Returns	INTEGER : resultCode 0x0000 : Success 0x0607 : nCompanyId value not in range 0x0000 to 0xFFFF 0x0608 : nModelId value not in range 0x0000 to 0xFFFF 0x60C4 : Previously added model has no messages attached 0x60C1 : Too many models have been defined in total 0x60CC : handleModel is not recognised as a model handle
Arguments:	
nCompanyId	**byVAL *nCompanyId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is a company ID. A member of the Bluetooth SIG can request one for free. For a full list of company identifiers see here. Where you will see for example, 0x0059 is for Nordic Semiconductor. It is VERY important that if you create a new custom model you use your own company ID and not someone else as you risk collision and thus confuse a provisioner. Also please note that if you want to interact with a Nordic defined Model then it is perfectly valid to use their company identifier here.
nModelId	**byVAL *nModelId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is model ID as adopted by the Bluetooth SIG and described in the specification “Mesh Model Specification”.
handleModel	**byREF *handleModel* AS INTEGER.** On Entry if this model is going to be an extension of another earlier added model then it shall be the handle of that model obtained when BleMeshAddSigModel() or BleMeshAddVendorModel() was called, **otherwise it shall contain 0**. If this model is an extension of another one, then it will share the subsciption list with the shared model. On Exit, this is an opaque handle value that the *smart*BASIC app shall use to describe a model when an API will interact with a model or when a message arrives, this value will be presented to enable the developer to channel the behaviour accordingly We recommend that you store it in a global *smart*BASIC variable.

9.3.4 BleMeshAddMessage

A model in a device will have one or more opcodes for messages registered so that incoming messages can be processed. Use this function to add a **packed** opcode that defines a message which is a value in up to 3-bytes long. A 3 byte packed opcode consists of companyID and a 6 bit opcode value, whereas a value which <= 0xFFFF will be a SIG defined opcode.

Note: If this function fails with BLE_MESH_DUPLICATE_OPCODEID (0x60CB) then it implies that your mesh structure is faulty. If you need a duplicate opcode, then you must add another element to the device for it to again be a unique entry. Then, since an element gets its own node address, the node address is used to differentiate which instance of opcode is being referenced.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

BleMeshAddMessage(nPackedOpcode)

Returns	INTEGER : resultCode 0x0000 : Success 0x06C3 : No models have been added to the current element 0x60C2 : Too many messages have been added. Limit will be exceeded 0x60CE : nPackedOpcode is invalid 0x60CB : Current element already has this message added
Arguments:	
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.

9.3.5 BleMeshStart

Once an Element/Model/Message composition has been defined using the functions described above, it must be registered with the Mesh stack and started. This function does that and is always done even if the device is provisioned and configured. When the mesh stack starts, it checks if the non-volatile information matches the structure defined in the tree. It will also know how to fork from there. If the non-volatile data is missing or does not match, then it puts the device into unprovisioned state and starts unprovisioned adverts. Otherwise it resumes mesh operation as a full member of a network.

Some of the parameters supplied in this function are used to configure the composition data – the information that is supplied to a provisioner so that it knows more about this device and more of that composition data is configured using the functions **BleMeshConfigInt()** and **BleMeshConfigStr()** which are described later.

After calling this function the event **EVBLEMESH_EVENT** which will have eventType WAIT_FOR_PROVISIONING or PROVISIONED will be thrown to the smartBASIC application.

If the device is provisioned then simple wait for EVBLEMESH_MESSAGERX events for incoming messages to process or based on gpio or other triggers call one of the Publish messages to send messages.

BleMeshStart(nTxPower, nCompanyId, nProductId, nVersionId, staticAuth$)

Returns	INTEGER : resultCode 0x0000 : Success 0x0608 : nCompanyId value not in range 0x0000 to 0xFFFF 0x0609 : nProductId value not in range 0x0000 to 0xFFFF 0x060A : nVersionId value not in range 0x0000 to 0xFFFF 0x060C : nDefaultTTL value not in range 0 to 127 0x60D0 : The mesh stack has already been started 0x60C9 : The mesh table tree is not empty 0x60D1 : This device cannot be a provisioner 0x60C8 : The mesh table tree is empty 0x60CA : The last model is empty in the tree
Arguments:	
nTxPower	**byVAL *nTxPower* AS INTEGER.** Specifies a value in the range -128 to 20 which is the transmit power for network advert messages.
nCompanyId	**byVAL *nCompanyId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is a company ID. A member of the Bluetooth SIG can request one for free. For a full list of company identifiers see here. Where you will see for example, 0x0059 is for Nordic Semiconductor. It is VERY important that you use your own companyID so that a provisioner better understands

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

	how to configure your device. Think of this value and the nProductId as the equivalent of the plug and play VID/PID information presented by a USB device.
nProductId	**byVAL *nProductId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is a product ID. This can be any value that you wish as you maintain a list of all the different mesh products that your produce. This is very similar to the PID value in USB world
nVersionID	**byVAL *nVersionId* AS INTEGER.** Specifies a value in the range 0x0000 to 0xFFFF which is a version ID. This can be any value that you wish.
staticAuth$	**byREF *staticAuth$* AS STRING.** This is a 16 byte string containing a key which will be randomly generated

9.3.6 BleMeshConfigInt

This function is used to specify more integer configuration parameters before the mesh functionality is started using the function **BleMeshStart()**.

BleMeshConfigInt(nCnnfigId, nConfigValue)

Returns	INTEGER : resultCode 0x0000 : Success
Arguments:	
nConfigId	**byVAL *nConfigId* AS INTEGER.** See 'nConfigValue' for more details. From 0 to 99 configId they can be set before the mesh stack is started using BleMeshStart(). From 100 to 999 configid values can only be modifed after BleMeshStart() is called and for all other configId they can be altered regardless..
nConfigValue	**byVAL *nConfigValue* AS INTEGER.** For **ConfigId= 1** A bit mask which specifies which mesh features the device will expose as follows:- Bit 0 : Relay Capability Bit 1 : Proxy Capability Bit 2 : Friend Capability Bit 3 : Low Power Node Capability Bits 4 to 31 : Reserved for future use and should be set to 0 For **ConfigId= 10** Bit mask specifying which bearers to activate for provisioning all but Low Power Nodes Bit Description 0 Advert Bearer 1 Gatt Bearer 4..31 Reserved for future use, set to 0 For **ConfigId= 11** Bit mask specifying which bearers to activate for provisioning Low Power Nodes Bit Description 0 Advert Bearer 1 Gatt Bearer 4..31 Reserved for future use, set to 0

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

For **ConfigId= 12**
16 bit OOB Info field in unprovisioned beacons
Range : 0..0xFFFF (Bit Mask as follows)

Bit	Description
0	Other
1	Electronic / URI
2	2D machine-readable code
3	Bar code
4	Near Field Communication (NFC)
5	Number
6	String
7	Reserved for Future Use
8	Reserved for Future Use
9	Reserved for Future Use
10	Reserved for Future Use
11	On box
12	Inside box
13	On piece of paper
14	Inside manual
15	On device

For **ConfigId= 13**
Output OOB size (Table 5.21 in Mesh Profile spec)
Range : 0..8 (0==Device does not support output OOB)

For **ConfigId= 14**
Output OOB Action
Range : 0..15

For **ConfigId= 15**
Input OOB size (Table 5.23 in Mesh Profile spec)
Range : 0..8 (0==Device does not support input OOB)

For **ConfigId= 16**
Input OOB Action
Range : 0..15

For **ConfigId= 20**
Minimum queue size requested in a Friend Request by a Low Power Node
Range : 1..7 (Actual size is 2 to the power of this value)

For **ConfigId= 21**
Receive Window Factor requested in a Friend Request by a Low Power Node
Range : 0..3 (Value Factor)

Value	Factor
0	1
1	1.5
2	2
3	2.5

For **ConfigId= 22**
RSSI Factor requested in a Friend Request by a Low Power Node
Range : 0..3 (Value Factor)

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

	0 1 1 1.5 2 2 3 2.5 For **ConfigId= 23** Receive Delay in milliseconds requested in a Friend Request by a Low Power Node Range : 10..255 For **ConfigId= 24** Poll Retry Count by a Low Power Node when in a freindship Range : 1..10 For **ConfigId= 1000** Enable/Disable Mesh subevent reporting in EVBLEMESH_EVENT This is a bitmask: Bit Description 0 Other Subevents 1 Low Power Node low level events

9.3.7 BleMeshConfigStr

This function is used to specify string configuration parameters before the mesh functionality is started using the function **BleMeshStart()**.

BleMeshConfigStr(nCnnfigId, sConfigValue$)

Returns	INTEGER : resultCode 0x0000 : Success
Arguments:	
nConfigId	**byVAL *nConfigId* AS INTEGER.** See 'sConfigValue$' for more details. . Negative configId and up to 99 values are config values that can be set before the mesh stack is started, that is, before BleMeshStart() is called.
sConfigValue$	**byREF *sConfigValue$* AS STRING.** For **ConfigId= 1** Provisioning Beacon Device URI

9.3.8 BleMeshMessagePublish

This function is used to publish a message with the opcode and data specified using the provisoned publish details (like destination address) of the model which is specified by the handleModel provided (the handle that was returned when either BleMeshAddSigModel() or BleMeshAddVendorModel() were called). It uses the appkey and netkey bound to the model indentified by 'handleModel'.

BleMeshMessagePublish(handleModel, nPackedOpcode, sData$)

Returns	INTEGER : resultCode 0x0000 : Success 0x60CC : handleModel is not recognised as a model handle 0x60CE : nPackedOpcode is invalid Other : Nordic stack specific

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

Arguments:	
handleModel	**byVAL *handleModel* AS INTEGER.** This is the handle of a model that was registered using BleMeshAddSigModel() or BleMeshAddVendorModel(). The destination address, appkey comes from whatever was configured for the model by a provisioner.
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.
sData$	**byREF *sData$* AS STRING.** This contains the data that will be sent as payload for the message. The specification allows this to be from 0 to 380-bytes. It will be appropriately lower if the opcode is 3-bytes long

9.3.9 BleMeshMessagePublishAcked

This function is used to publish a reliable message with all the parameters as described for the function BleMeshMessagePublish() but in addition, it takes two more parameters which correspond to the opcode of the message to expect that acknowledges receipt of this method and the maximum time to wait for that ack.

If the function returns a successful resultcode, then a reliable publish transaction shall be assumed to have started. It will terminate when an event message EVBLEMESH_ACKEDMSG_RESULT is received with a status code which conveys whether it was successful or a timeout occurred or cancelled because the function **BleMeshMessagePubAckedCancel()** had subsequengly been called.

While a reliable publish transaction is in progress, this function cannot be called for the **same** handleModel.

Note: If the model publishes to a group address, then a message is sent but the transaction is deemed to be completed immediately – do not expect to wait for the EVBLEMESH_ACKEDMSG_RESULT event.

BleMeshMessagePubAcked(handleModel, nPackedOpcode, nExpectedOpcode, nTimeoutsec, sData$)

Returns	INTEGER : resultCode 0x0000 : Success 0x60CC : handleModel is not recognised as a model handle 0x60CE : nPackedOpcode is invalid Other : Nordic stack specific
Arguments:	
handleModel	**byVAL *handleModel* AS INTEGER.** This is the handle of a model that was registered using BleMeshAddSigModel() or BleMeshAddVendorModel(). The destination address, appkey comes from whatever was configured for the model by a provisioner.
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.
nExpectedOpcode	**byVAL *nExpectedOpcode* AS INTEGER.** This is the packed opcode of the message to wait for as an acknowledgement from all subscribers of this message. For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.
nTimeoutSec	**byVAL *nTimeoutSec* AS INTEGER.** Wait for this long, in seconds, for an ack to arrive and **shall** be in the range 30 to 60 seconds.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

sData$	**byREF *sData$* AS STRING.** This contains the data that will be sent as payload for the message. The specification allows this to be from 0 to 380-bytes. It will be appropriately lower if the opcode is 3-bytes long

9.3.10 BleMeshMessagePubAckedCancel

This function is used to cancel a reliable publish transaction that was initiated for a model using the function **BleMeshMessagePubAcked()**.

BleMeshMessagePubAckedCancel(handleModel)

Returns	INTEGER : resultCode 0x0000 : Success 0x60CC : handleModel is not recognised as a model handle Other : Nordic stack specific
Arguments:	
handleModel	**byVAL *handleModel* AS INTEGER.** This is the handle of a model that was registered using BleMeshAddSigModel() or BleMeshAddVendorModel().

9.3.11 BleMeshMessageReply

This function is used to send a response to an incoming message with the opcode and data specified using the destination details (like destination address) embedded in the opaque parameter sReplyData$. This was supplied when the incoming message arrived via the event EVBLEMESH_MESSAGERX which is described later. The sReplyInfo$ will also contain the appkey that was used by the incoming message and so the response needs to use the same one.

Note: handleModel and nPackedOpcode are also supplied in the EVBLEMESH_MESSAGERX event when the incoming message arrived

BleMeshMessageReply(handleModel, nPackedOpcode, sData$, sReplyInfo$)

Returns	INTEGER : resultCode 0x0000 : Success 0x60CC : handleModel is not recognised as a model handle 0x60CE : nPackedOpcode is invalid 0x60CF : sReplyInfo$ is invalid Other : nordic stack specific
Arguments:	
handleModel	**byVAL *handleModel* AS INTEGER.** This is the handle of a model that was registered using BleMeshAddSigModel() or BleMeshAddVendorModel(). The destination address, appkey comes from whatever was configured for the model by a provisioner.
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.
sData$	**byREF *sData$* AS STRING.** This contains the data that will be sent as payload for the message. The specification allows this to be from 0 to 380-bytes. It will be appropriately lower if the opcode is 3-bytes long
sReplyInfo$	**byREF *sReplyInfo$* AS STRING.** This will have been supplied in the EVBLEMSG_OPC_MSG event and MUST be supplied unmodified from there. It is an opaque object and will be checked for modification and if so will result in a failure to send a response

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

9.3.12 BleMeshFriendConnect

When a device is in Low Power Node mode this function is used to seek a Friend which will cache incoming messages for this node while the radio is switched off to conserve power. It does so by sendng a Friend Request message as described in the Mesh Profile specification.

If a success resultcode is returned then this transation is terminate by one of two events **EVBLEMESH_FRIENDCONNECT** or **EVBLEMESH_FRIENDDISCON**.

BleMeshFriendConnect(scanDurMs, pollTimeoutMs, minRssi, minRxWinSz, maxRxWinSz, minSubsLstSz)

Returns	INTEGER : resultCode 0x0000 : Success Other : Nordic stack specific
Arguments:	
scanDurMs	**byVAL *scanDurMs* AS INTEGER.** The duration to scan for incoming Friend Offers which shall be in the range 100 to 1000.
pollTimeoutMs	**byVAL *pollTimeoutMs* AS INTEGER.** Poll Timeout in milliseconds in the range 1000 to 345599900 (which is 1 sec to 96 hours). If the low power node does not send a poll within this timeout, then the Friend will assume that the low power node has disappeared. Use function BleMeshFriendPoll() to manually initiate a poll to trigger one, but note that the stack will automatically send one out within this timeout period. This auto poll will occur at interval which is a function of this value minus the sum of Receive Delay and Receive Window multiplied by the Poll Trey count +1 which is set via BleMeshConfigInt(24)
minRssi	**byVAL *minRssi* AS INTEGER.** This is the minimum RSSI as measured by the Friend when it received the Friend Request message and will be provided in the Friend Offer message and this node will reject the offer when the rssi is below this value
minRxWinSz	**byVAL *minRxWinSz* AS INTEGER.** When a friend sends an Offer it will provide a Receive Window Size in milliseconds. The offer will be rejected if the value is less than this value.
maxRxWinSz	**byVAL *maxRxWinSz* AS INTEGER.** When a friend sends an Offer it will provide a Receive Window Size in milliseconds. The offer will be rejected if the value is bigger than this value
minSubLstSz	**byVAL *minSubLstSz* AS INTEGER.** When a friend sends an Offer it will provide a subscription list size. The offer will be rejected if the value is less than this value because otherwise the friend will not be able to listen for all the subscribed addresses for this node.

9.3.13 BleMeshFriendDisconnect

When a device is in Low Power Node mode and connected to a friend (see BleMeshFriendConnect()), this function is used to break the friendship.

If a success resultcode is returned then this transation is terminate by the event **EVBLEMESH_FRIENDDISCON**.

BleMeshFriendDisonnect()

Returns	INTEGER : resultCode 0x0000 : Success Other : Nordic stack specific
Arguments:	None

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

9.3.14 BleMeshFriendPoll

When a device is in Low Power Node mode this function is used to force a poll to see if the friend has any messages cached for it. If there are any messages then they will be presented using the normal event EVBLEMESH_MESSAGERX

BleMeshFriendPoll(pollTimeoutMs)

Returns	INTEGER : resultCode 0x0000 : Success Other : Nordic stack specific
Arguments:	
pollTimeoutMs	**byVAL *pollTimeoutMs* AS INTEGER.** Poll Timeout in milliseconds in the range up to 345599900 (which is 1 sec to 96 hours). If the value is <= 0 then an immediate manual POLL is triggered. If value is >0 and <1000 then a POLL will be triggered but an error resultcode will be returned. If greater than 1000 then a POLL gets triggered and in addition the auto POLL interval is set to this value as long as it satisfies the criteria for the poll timeout value as specified in the BleMeshFreindConnect() function

9.3.15 BleMeshUnProvAndReset

This function is used to wipe all provisioning and configuration information so that it is back into unprovisioned state. Note that effect of calling this from the application is the same as a provisioner sending a 'Node Reset' Foundation Model message.

After the information is cleared the module will perform a self-warm reset and so expect the EVBLEMESH_EVENT event with event type PROVISION_WAIT_FOR.

BleMeshUnProvAndReset()

Returns	INTEGER : resultCode 0x0000 : Success Other : Nordic stack specific
Arguments:	None

9.4 Mesh-related Events

9.4.1 EVBLEMESH_EVENT

This event is used to report events detected by the mesh stack.

Parameters:	
nEventType	**byVAL *nEventType* AS INTEGER.** This contains the event type
nContext	**byVAL *nContext* AS INTEGER.** This contains the cntext datat for the event and if the event has no payload then it will be set to 0
sContext$	**byVAL *sContext$* AS STRING.** This contains context data for the event and can be an empty string if the event type has no payload

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

The values for nEventType and associated context integer and string will be as per the table below.

If the context column is 'none' then the integer will be 0 and the string will be empty.

Value	Description	Context
100	PROVISION_WAIT_FOR	None
110	PROVISION_IDNTFY_START	**nContext** Duration in seconds that the Node will physically indicate (like a blinking LED) that it is the device that will be provisioned when provisioning is initiated
120	PROVISION_IDNTFY_STOP	None
150	PROV_PRE_STACK_DISABLE	None (will be received just before the stack is disabled so that the GATT table which contains the Mesh Provisioning Service can be destroyed and on re-enabling a new GATT table is created with the Mesh Proxy Service)
160	PROVISION_STACK_ENABLED	None (happens after the stack is restarted with Mesh Proxy Service)
190	PROVISION_ABORTED	None (happens when provisioning is aborted)
200	PROVISIONED	**sContext$** First 2-bytes = First Element Node address Second 2-bytes = Number of Elements <<Note: 2-bytes entities are little endian>>
220	FOUNDATION_CONFIG	**nContext** The foundation message identified by this opcode has updated the foundation model state. For example, if the publication info or subscription list (or any other state) is updated the application will know it has happened, but will not know what the updated value.
300	KEY_REFRESH_NOTIFICATION	None
400	IV_UPDATE_NOTIFICATION	None
800	OTHER_EVENT	Low level events. This event will only happen if enabled via BleMeshConfigInt(50,n) where n has bit 0 set. By default that bit is set to 0. **nContext** This further identifies a Low Power Node feature related event that has occurred as follows:- 0 : A mesh message has been received 1 : A mesh message transmission has been completed 4 : An authenticated network beacon is received 5 : A heartbeat message is received 6 : The heartbeat subscription parameters changed 14 : Flash operations queue is empty, and flash is stable. 15 : RX Failed 16 : SAR Failed 17 : Flash Failed

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

		18 : Config Stable 19 : Config Storage Failure 20 : Config Load Failure 27 : Mesh stack has been disabled
810	LPN_EVENT	Low Level events. Normal Low Power Node events to handle are **EVBLEMESH_FRIENDCONNECT** and **EVBLEMESH_FRIENDDISCON** This event will only happen if enabled via BleMeshConfigInt(50,n) where n has bit 1 set. By default that bit is set to 0. **nContext** This further identifies a Low Power Node feature related event that has occurred as follows:- 21 : The node received a friend offer 22 : The node received a friend update 23 : The Friend Request Timed out 24 : The node successfully polled all data from the friend note 25 : A Friendship has been successfully established 26 : A friendship has successfully been terminated
900	MALLOCFAIL_MSGRX	**nContext** A message with this opcode was received but could not be delivered to the application because of a malloc fail.

9.4.2 EVBLEMESH_MESSAGERX

This event occurs when a message arrives and needs to be processed and may result in zero or more outgoing messages.

Parameters:	
nElementIndex	**byVAL *nElementIndex* AS INTEGER.** This contains the element index 0 to N and will correspond to the elements that were added using BleMeshElementAdd()
handleModel	**byVAL *handleModel* AS INTEGER.** This contains the handle that was returned by BleMeshAddSigModel() or BleMeshAddVendorModel()
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** This contains the packed opcode. For a SIG defined opcode this shall be a value in the range 0x0000 to 0xFFFF. For a vendor defined opcode the value shall be 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID
sData$	**byREF *sData$* AS STRING.** This contains the data that arrived in the message associated with the opcode
sReplyInfo$	**byREF *sReplyInfo$* AS STRING.** This contains context data that will be used if BleMeshMessageReply() is called and should be supplied to that function unmodified. This data MUST NOT be modified in anyway by the application when it is supplied in BleMeshMessageReply()

It is expected that the *smart*BASIC application the handler will switch on the nPackedOpcode value (using the Select compound statement) and then call an appropriate function to handle the data

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

9.4.3 EVBLEMESH_ACKEDMSG_RESULT

This event occurs to signal the end of a reliable publish transaction which was initiated using the function **BleMeshMessagePublishAcked()**

Parameters:	
nElementIndex	**byVAL *nElementIndex* AS INTEGER.** This contains the element index 0 to N and will correspond to the elements that were added using BleMeshElementAdd()
handleModel	**byVAL *handleModel* AS INTEGER.** This contains the handle that was returned by BleMeshAddSigModel() or BleMeshAddVendorModel()
nResult	**byVAL *nResult* AS INTEGER.** This contains the result outcome as follows:- 0 = Transaction successfully completed 1 = Transaction failed due to a timeout 2 = Transaction Aborted due to BleMeshMessagePubAckedCancel() being called

9.4.4 EVBLEMESH_FRIENDCONNECT

This is a low power node related event which occurs to signal that an attempt to seek a friend using the function **BleMeshFriendConnect()** has succeeded and mesh messages will now be received via the event **EVBLEMESH_MESSAGERX** as a result of the automatic polling that happens. At any time the low power node can manually trigger a poll for messages queued up in a friend by calling the function **BleMeshFriendPoll()**

Parameters:	
nReceiveWinSize	**byVAL *nReceiveWinSize* AS INTEGER.** This contains the receive window size offered by the friend
nMsgQueSize	**byVAL *nMsgQuwSize* AS INTEGER.** This contains the message queue size offered by the friend
nSubscriptionListSize	**byVAL *nSubscriptionListSize* AS INTEGER.** This contains the number of subscription addresses that the friend can subscribe on behalf of the low power node. When this event occurs, the subscription list in the low power node has already been transferred to the friend.
nMeasuredRssi	**byVAL *nMeasuredRssi* AS INTEGER.** This contains the rssi of the friend request message from this low power node as measured by the friend.

9.4.5 EVBLEMESH_FRIENDDISCON

This is a low power node related event which occurs to signal that an attempt to seek a friend using the function **BleMeshFriendConnect()** has failed and the reason is provided.

It will also happen on power up if the device is a low power node so that tha application can use that event to initiate a friend connection using **BleMeshFriendConnect()** and in that case the reason will be set to 256.

Parameters:	
nReason	**byVAL *nReason* AS INTEGER.** This reason this event was generated 0 : The Low Power node actively terminated the friendship 1 : There was no response from the LPN within the Poll Timeout 2 : The Friend node did not reply to the (repeated) Friend Poll 3 : The Low Power node was not able to send transport command due to internal fault 256 : The device has just powered up

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

10 *SMART*BASIC APP CODE WALKTHROUGH

In a *smart*BASIC BLE Mesh application, the developer must only code for creating the mesh models and then sending and receiving mesh messages; the developer only has to deal with what triggers a mesh message to be sent and what happens when a mesh message is received. With that in mind, *smart*BASIC provides functions such as BleMeshMessagePubXXXX() and BleMeshMessageReply() for sending messages; since it is an event-driven language, there is an event EVBLEMESH_MESSAGERX that is thrown to the application which in turn has a handler written by the developer to process it.

In terms of which node to send to or where a message comes from, that is entirely dealt with when a device is provisioned and configured by a provisioner like a smartphone. In fact, all the coding of that aspect is completely opaque to the application layer so is not even exposed to the *smart*BASIC app developer.

With that in mind, the relevant portions of the Client and Server applications are described in the following sections.

10.1 Client : $autorun$.mesh.light.switch.client.sb

On start, the function **GpioInit()** on line circa 625 which is defined at line circa 273 where the gpio for all 4 buttons on the devkit are configured as inputs using **GpioSetFunc()** and all 4 LEDs are configured as outputs, again using **GpioSetFunc()** and then finally all 4 button gpios are configured to generate the **EVGPIOCHANx** event so that the handlers **HandlerOnButtonx()** are called when they change state.

Then the function **MeshInit ()** is called to initialise the Gatt table, and then **RegModels**() which in turn registers 4 mesh elements, each with a single 'simple onoff client' model which you see in the 'for' loop, and then finally the mesh stack is started by calling the function **BleMeshStart()**.

For incoming mesh messages, the following statement, at line circa 609, ensures that the function **HandlerMeshMessageRx()** is called.

```
OnEvent EVBLEMESH MESSAGERX          call HandlerMeshMessageRx
```

Locate the **HandlerMeshMessageRx()** function and you will see it is passed the element index, the model handle, the opcode of the message and the payload. There you see a SELECT statement to switch on the opcode value and in this case the model only listens for the STATUS message and for this demo we call the function **onStatus()** and print a message that the message has arrived.
Locate the **onStatus()** function where you will see that the data in the message is decoded to extract the state of the LED at the server that sent it and that it calls **GpioWrite()** to reflect the state of the remote LED on one of the local LEDs

Locate the **HandlerOnButtonX()** (where X== 1 or 2 or 3 or 4) and you will see that a debug message is printed to indicate that BUTTON X has been pressed and then the function **OnButton()** is called. That function is defined at circa line which in turn calls **SendSetMsg()** which in turn creates and sends the SIMPLE_ON_OFF_OPCODE_SET message.

On power-up the event **EVBLEMESH_EVENT** will always be thorwn to the app with event type 100 (PROVISION_WAIT_FOR) or 200 (Provisioned). That event is processed in function **HandlerMeshEvent()** that can be found in the included file EvMeshEvent_Handler.sblib where all that happens is that a message is printed.

10.2 Server : $autorun$.mesh.light.switch.server.sb

On start, the function **GpioInit()** on line circa 560 which is defined at line circa 260 where the gpio for button 1 on the devkit is configured as input using **GpioSetFunc()** and all 4 LEDs are configured as outputs, again using **GpioSetFunc()** and then finally button 1 gpio is configured to generate the **EVGPIOCHAN0** event so that the handlers **HandlerOnButton1()** is called when it changes state.

Then the function **MeshInit()** is called and then **RegModels()** which in turn registers 1 mesh element, with a single 'simple onoff server model at line circa 350. Then the three opcodes are registered and then finally the mesh stack is started by calling the function **BleMeshStart()** at line circa 575.

For incoming mesh messages, the following statement, towards the end of the file, ensures that the function **HandlerMeshOpcMsg()** is called.

```
OnEvent EVBLEMESH MESSAGERX          call HandlerMeshMessageRx
```


Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

Locate the **HandlerMesh()** function and you will see it is passed the element index, the model handle, the opcode of the message and the payload. There you see a SELECT statement to switch on the opcode value and in this case the model listens for the GET, SET and SET_UNRELIABLE messagew and for this demo we call the functions **OnGet(), OnSetRel()** and **OnSetUnRel()** respectively and also print a message that the message has arrived.

Locate the **OnGet()** function where you will see that the STATUS message is published – and note there is no information about who it publishes to as that is provided at provisioning time.

Locate the **OnSetRel()** function where you will see that STATUS message is sent in a reply (the destination node address is provided in the respInfo$ parameter) and also another STATUS message is published – again to a recipient that will have been configured when the device was provisioned.

Locate the **OnSetUnrel()** function where you will see that STATUS message is published – again to a recipient that will have been configured when the device was provisioned. Note there is no reply to the sender.

Locate the **HandlerOnButton1()** and you will see that a debug message is printed to indicate that BUTTON 1 has been pressed and then the locate variable **ledstate[0]** is toggled and the **GpioWrite()** is used to update the satet of LED1 as per the state of that variable and then finally the new LED state is published using **BleMeshMessagePublish()** so that all subscribers are made aware of that change of state.

On power-up the event **EVBLEMESH_EVENT** will always be thorwn to the app with event type 100 (PROVISION_WAIT_FOR) or 200 (Provisioned). That event is processed in function **HandlerMeshEvent()** that can be found in the included file EvMeshEvent_Handler.sblib where all that happens is that a message is printed.

10.3 Server : $autorun$.mesh.light.switch.server.lpn.sb

This is the application to load to experiment with a Low Power Node. The processing of mesh messages related to the models is exactly the same as per the non-lpn variant of the application except that it has the extra duty to initiate a connection to a friend so that incoming messages can be received. Without a friendship the node is useless as it cannot receive any messages to process, although it is capable of directly publishing messages without the help of a friend.

In short, the extra effort is to always maintain that it is in a friendship and that is done by calling BleMeshFriendConnect() whenever it has been sent the event EVBLEMESH_FRIENDDISCON. That means that this application has to register for an additional two low power node related messages as follows:-

```
OnEvent EVBLEMESH_FRIENDCONNECT     call HndlrMeshFrndConn
OnEvent EVBLEMESH_FRIENDDISCON      call HndlrMeshFrndDiscon
```

On receipt of the **EVBLEMESH_FRIENDISCON** event the handler **HndlrMeshFrndDiscon()** is called and all that function does is starts a timer. And when that tiemr expires the function **HandlerTimer1()** is called where it calls the function BleMeshFriendConnect().

Apart from these additional actions, the sending and receiving of messages is identical to the non-LPN variant of the server application.

On power-up the event **EVBLEMESH_EVENT** will always be thorwn to the app with event type 100 (PROVISION_WAIT_FOR) or 200 (Provisioned). That event is processed in function **HandlerMeshEventX()** which in turn invokes **HandlerMeshEvent()** that can be found in the included file EvMeshEvent_Handler.sblib where all that happens is that a message is printed.
This difference from the other two apps is purely to delay the initiatation of a friend connection if a gatt connection still exists. This is currently a workaround for an issue with related to the provisioning phase that requires a mesh stack restart and so will throw a PROVISIONED eventtype. That is normal but if the provisioning gatt connection is present then it seems to be interfered with if a friend connection is started at that point.
You will therefore notice that the FriendConnect function is invoked when the gatt connection disconnects.

11 MIGRATING SMARTBASIC APPS FROM OLDER FIRMWARE

This section describes the changes required to your smartBASIC application to port from an older version. While the mesh firmware is in experimental phase tracking Nordic's Mesh SDK which makes significant changes as it evolves Laird are forced to make changes and this section makes that task easier.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

11.1 Version Mesh310-8 from Mesh211-10

This firmware caters for Low Powr Node feature and fixes bugs related to the earlier Mesh Stack.

- The event EVBLEMESH_STATE is renamed to EVBLEMESH_EVENT which provides an extra integer paramter so the handler for it needs to cater for it. The new example app contains a file called “EvMeshEvent_Handler.sblib” which can be #included in your app to inherit the behavior as shown in the new sample apps.
- The function BleMeshConfigInt() takes has new config items that can be used to configure the behavior of the mesh node. See details here.
- Deleted functoin BleMeshSchemaNew(). Call BleMeshAddElement() instead.
- Added new function BleMeshFriendConnect()
- Added new function BleMeshFriendDisconnect()
- Added new function BleMeshFriendPoll()
- Added new function BleMeshUnProvAndReset()
- Removed event EVBLEMESH_STATE
- Added new event EVBLEMESH_EVENT
- Added new event EVBLEMESH_FRIENDCONNECT
- Added new event EVBLEMESH_FRIENDDISCON

11.2 Version Mesh211-12 from Mesh211-10

- It is now possible to add up to 8 elements per node, up from 4
- Function BleMeshStart() now takes an extra integer parameter which is used to specify the transmit power to use for mesh adverts. The default was previously 0dBm
- Function BleMeshConfigInt() no longer takes configId==0 as that configuration value can be modified by a provisioner and so allowing it to be done locally does not make sense.
- Function name BleMeshSchemaNew() has now been changed to BleMeshBeginNodeComposition()
- Function name BleMeshAddOpcode() has now been changed to BleMeshAddMessage()
- Function name BleMeshPublish() has now been changed to BleMeshMessagePublish()
- Function name BleMeshPublishReliable() has now been changed to BleMeshMessagePublishAcked()
- Function name BleMeshReply() has now been changed to BleMeshMessageReply()
- Event name EVBLEMESH_OPC_MSG has now been changed to EVBLEMESH_MESSAGERX
- Event name EVBLEMESH_PUBREL_RESULT has now been changed to EVBLEMESH_ACKEDMSG_RESULT

12 SNIFFER OPERATION

This section describes how to operate the mesh sniffer which displays mesh traffic. Messages which have content encrypted are NOT decrypted as the keys are not available for that to happen.

Load the smartBASIC application *$autorun$.mesh.sniff.sb* and on starting it any mesh related messages will be displayed similar to the UwTerminalX screenshot below.

```
01E9255B1B6277 PB-GATT[-18] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
01E9255B1B6277 PB-ADV [-31] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 (URIhash=)
01E9255B1B6277 PB-GATT[-18] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
01E9255B1B6277 PB-GATT[-17] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
01E9255B1B6277 PB-ADV [-34] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 (URIhash=)
01E9255B1B6277 PB-GATT[-18] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
01E9255B1B6277 PB-GATT[-18] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
01E9255B1B6277 PB-ADV [-33] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 (URIhash=)
01E9255B1B6277 PB-GATT[-17] (DevUUID=)869F42EC53F711E077621B5B2529B644 (OOB=)0000 DevName=LS P-SERVER
```

If there are many devices in the area you will see a lot of messages scroll up fast. If you want to temporarily stop scanning just hit the ENTER key and it will stop. Hitting the ENTER key will restart scanning.

It is also possible that you may want to monitor messages from a particular node only whose address is at the beginning of the lines. To just view advert messages from that node, hit ENTER key to suspend scanning, and enter the command

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

```
ps "01E925"
```
and then hit ENTER key to restart the scanning. You will now only see lines which have the text "01E925" in it.
Generally the command 'ps' takes a pattern which can be any sequence of characters and all the sniffer application is doing is printing a line if and only if that line contains that pattern anywhere in the line.

To abort filtered scanning, suspend scanning, enter `ps ""` and restart scanning by hitting the ENTER key.

You are free to modify the sniffer smartBASIC application in any way you want. If you think it will be useful for others we encourage you to submit to Laird for incorporation so that those changes are released to everyone on the next firmware release.

13 REFERENCES

The following documents are also accessible from the BL654 product page of the Laird website (Documentation tab):

- BL654 *smart*BASIC Extension Manual
- BL654 Datasheet
- UwTerminalX

The following documents are also accessible from the Bluetooth SIG website:

- Mesh Profile Specification v1.0
- Mesh Model Specification v1.0
- Mesh Device Properties v1.0

14 REVISION HISTORY

Version	Date	Notes	Approver
0.10.0/rel10	5 Dec 2017	Initial Release	Jonathan Kaye
2.1.1/8	17 Jul 2018	Updated to reflect firmware based on Nordic Mesh SDK V2.1.1	Jonathan Kaye
2.1.1/8-rel 2	16 Aug 2018	All sections updated as per the functionality	Jonathan Kaye
2.1.1 rel 3	24 Sep 2018	Typo correction	Mahendra Tailor
29.1.1.12-MESH211-12 Rel 1	19 Oct 2018	+ Function and Event name changes. + On mesh start the txpower can be specified to extend range between nodes. Default was 0dBm and now it can be set as high as +8dBm + Added section 10 with details of migrating a smartBASIC app from an older firmware + Added section 11 describing sniffer operation	Mahendra Tailor
29.1.1.14-MESH211-14 Rel 2	10 Jan 2019	Version change	Mahendra Tailor
29.3.31.8-MESH310-8	29 May 2019	Based on SDK v3.1.0 and added Low Power Node capability	Mahendra Tailor

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610