BLE Mesh – BL652 Sample *smart*BASIC Application

Application Note *v0.10.0/rel10*

INTRODUCTION

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

This application note provides an overview of BLE mesh from an application perspective by introducing you to some early alpha BLE mesh functionality in the Laird BL652 module. This document also describes how to use 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 and is ***not fit for production***; it is built using an early alpha release of the BLE mesh SDK (version 0.10.0) from the Nordic semiconductor. That SDK from Nordic is version 0.10.0.

As this is based on the early alpha release of the SDK from Nordic, we reserve the right to change the specifics of the API that is used to expose the SDK API and is described in this application note. Nordic may introduce changes to the stack as they work towards the eventual Bluetooth SIG-approved production release.

REQUIREMENTS

The following are required for this sample *smart*BASIC application:

- Multiple (at least three) DVK-BL652 development kits – Five development kits are ideal to view the interaction with multiple on/off servers. One devkit is used as a sniffer for mesh adverts to get a better understanding of mesh operations; activity, such as unprovisioned beacons, assures you that your equipment is 'alive'.
  Alternate compatible kits are available from Chip45.
- 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 BL652 (available from the BL652 product page Software Download section)
- A sample command manager smartBASIC application demonstrating mesh functionality – ***$autorun$.mesh.light.switch.example.sb*** (included in the firmware zip file)
- A MeshSniff *smart*BASIC application – ***$autorun$.mesh.sniff.sb*** (included in the firmware zip file)

Note: For the purposes of this document, we assume you are familiar with compiling/loading *smart*BASIC applications.

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

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 experimental SDK (release level **0.10.0-Alpha**). This SDK is not fully functional and only offers the advert bearer and relay functionality. There are no proxy, friend, or low power node capabilities.

Please refer to Nordic Semiconductor's release notes for more details about the experimental 0.10.0 SDK.

Demo Description

A light switch client and server devices are used for this sample demonstration. The light switch client also incorporates provisioner functionality; this means that, when the client sees unprovisioned devices, it automatically provisions and configures a maximum of three devices.

While running the demonstration, if you are near other vendors' unprovisioned devices, attempts to provision them may occur. It is unclear in what state those devices will be left. The provisioning functionality is provided as-is from Nordic's SDK sample application; future Laird release's will provide clearer provisioning functionality.

The client device implements the client behavior of a Nordic Semiconductor custom light switch model. Likewise, the server device implements the server behavior of the light switch model.

Opcodes

A model is an array of opcodes and associated handlers. The Nordic custom light switch model consists of the following opcodes and recipients for each opcode.

Table 1: Opcodes and opcode recipients

Opcode Name	Opcode*	Role		Message Data	
SIMPLE_ON_OFF_OPCODE_SET	0xC10059	Server		0/1	tid
SIMPLE_ON_OFF_OPCODE_GET	0xC10059	Server			
SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE	0xC30059	Server		0/1	tid
SIMPLE_ON_OFF_OPCODE_SET_STATUS	0xC40059		Client	0/1	

* 0059 = Nordic Semiconductor Company ID
tid = Transaction ID

Note: The *tid* data field is simply an incrementing number which wraps at 0xFF to 0x00. Nordic does not attach any specific meaning to it. It is incremented each time a SET or SET_UNRELIABLE message is sent.

The example demonstrated in this application note only offers provisioning of up to three light switch servers.

The example is best described in Figure 1.

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

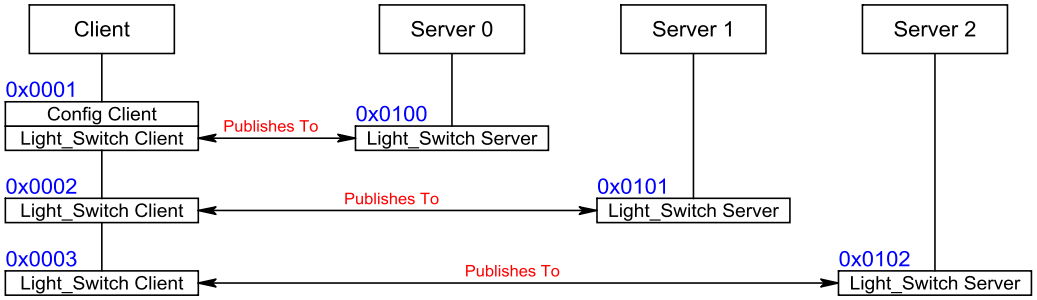


Figure 1: Provisioning of three servers

Figure 1 shows four devkits labelled Client, Server 0, Server 1, and Server 2. Each of the three servers contain a single element implementing Nordic's custom light switch model server roles. The client contains three elements: the first element consists of two models (config client and light switch client) and each of the other two elements contain a single light switch model client.

When the client devkit is initially powered up, it self-provisions, allocates to itself three node addresses (0x0001, 0x0002, and 0x0003), and gives itself a netkey with index 0 and an appkey with index 0.

When an unprovisioned server is powered up, it starts to advertise an unprovisioned beacon and contains the device UUID which always remains the same.

When the client receives an unprovisioned beacon, it immediately provisions it and give it the first available address equal to or greater than 0x0100. It also configures the first available client model to publish to that server node and configure that server with netkey and appkey. Correspondingly, it sets the publish address of the server model to the node address of the client. Hence client 0x0001 publishes to node 0x0100 and vice versa.

All this repeats as more unprovisioned servers power up until all three clients in the client device are configured to publish to a server.

At any time, if the client wants to set the on/off state of the server, it publishes a message with opcode SIMPLE_ON_OFF_OPCODE_SET or SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE. The former one (SIMPLE_ON_OFF_OPCODE_SET) results in a response message with opcode SIMPLE_ON_OFF_OPCODE_STATUS; the latter one (SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE) does not. This means that, when a SIMPLE_ON_OFF_OPCODE_SET is sent, the client sees **two** SIMPLE_ON_OFF_OPCODE_STATUS messages. One as a response to the message and the other as a result of the server publishing the new state. SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE results in a single response arising from the publish.

The *smart*BASIC sample app ***$autorun$.mesh.light.switch.example.sb*** is an application that accepts the following commands to trigger the stated actions.

Table 2: Commands to trigger the stated actions

Command	Description of Action
ms start	Starts the device as a light switch server
ms on	Change the local light state to ON
ms off	Change the local light state to OFF
mc client	Starts the device as a light switch client

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

Command	Description of Action
mc set A B	Set the light at server A to new state B and require confirmation. A==0..2 and B==0..1
mc setunrel A B C	Set the light at server A to new state B without confirmation. A==0..2 and B==0..1 and C=1..8
mc get A	Get the state of the light at server A. A==0..2

BL652 Development Kit Firmware Load

To set up **each development kit**, with the experimental mesh firmware, locate the mesh firmware zip file, unzip it into a folder, and follow these steps for **each** the devkits:

1. Connect your BL652 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 (Figure 2).

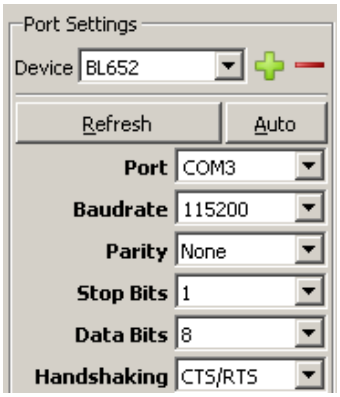


Figure 2: Config tab

4. Click **OK** to advance to the Terminal tab.
5. Use UwTerminalX to return the BL652 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; then issue the *at&f** command to erase the file system and all non-volatile data.

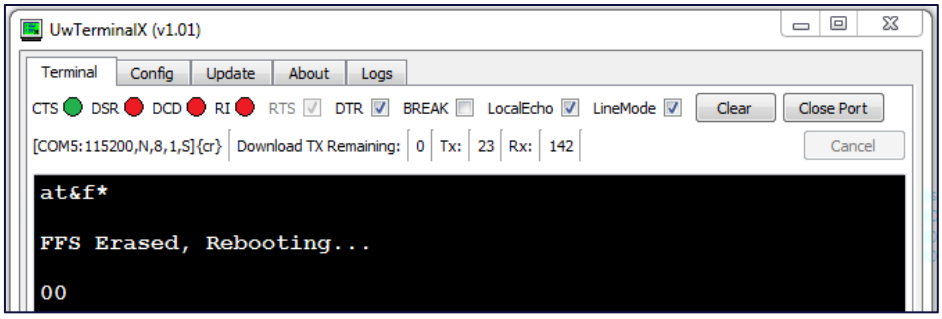


Figure 3: Return the BL652 to factory defaults

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

6. Close UwTerminalX.
7. In the folder where the mesh firmware is unzipped, locate the file and launch the following file: **_DownloadFirmwareUart.bat** to open the window in Figure 4.

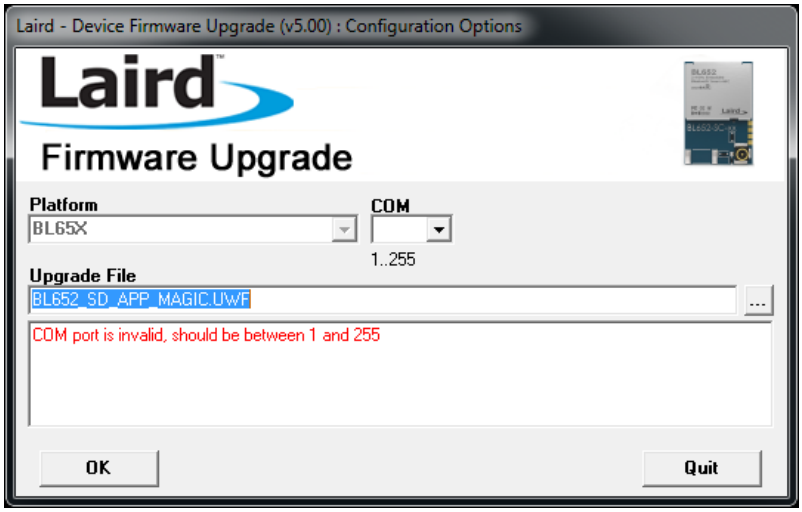


Figure 4: Firmware upgrade window

8. In the COM field, enter the same comport number previously in the Config tab (Figure 2).
9. Confirm that the message *COM port is invalid, should be between 1 and 255* is no longer displaying.
10. Click **OK** and confirm you see the following screen.

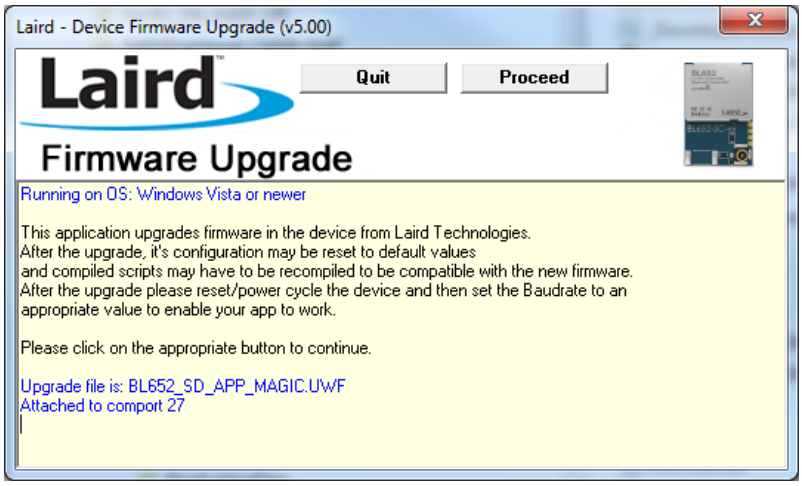


Figure 5: Resulting firmware upgrade window

11. Click **Proceed**.
12. When the following screen displays (**Figure 6**), click **Quit**.

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

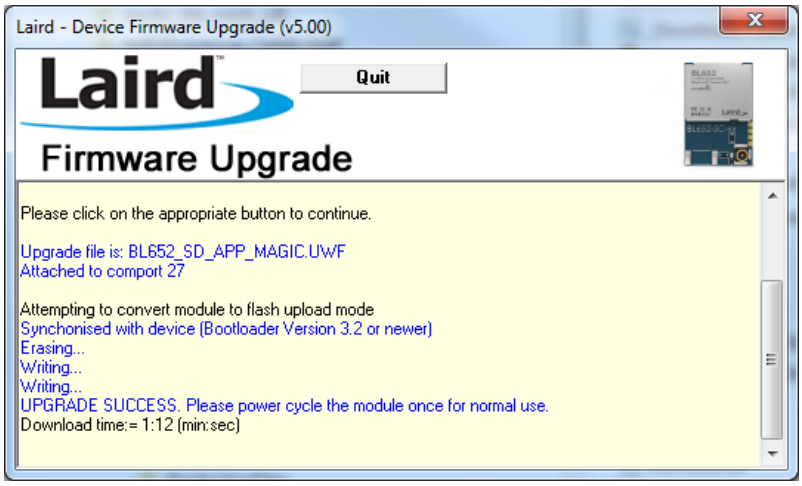


Figure 6: Click Quit

13. Open UwTerminalX.
14. In the Config tab, set the parameters and COM port associated with your development board.
15. Click **OK** to advance to the Terminal tab.
16. Send the command **AT I 3** and confirm the following response to your sent command:

    10 3 28.7.3.0-MESH-SDK0.10.0-10

BL652 Development Kit *smart*BASIC App Load

If you have five boards, then label in the following ways:

- Client
- Server 0
- Server 1
- Server 2
- Sniff

Client, Server 0, Server 1, and Server 2 Development Boards

For boards labelled Client, Server 0, and optionally if you have Server 1 and Server 2, perform the following steps:

1. Load the mesh *smart*BASIC example application – use the right-click menu to select **XCompile + Load**.

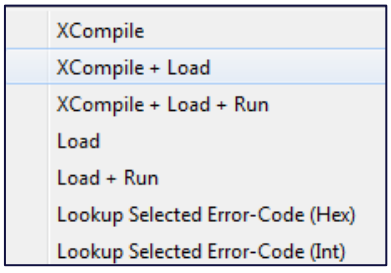


Figure 7: Right-click XCompile + Load

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

2. From the MeshApps subfolder, select the **$autorun$.mesh.light.switch.example.sb** file.

   The mesh program should take approximately ten seconds to load.

   Once loaded, you can run the mesh example by typing **at+run *"$autorun$"*** followed by a return or by pressing the reset button.

   The following message should display (Figure 8).

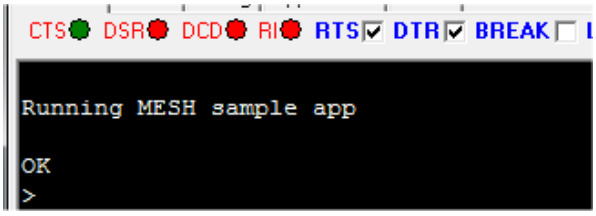


Figure 8: Running MESH sample app

Sniff Development Board

For the board labelled Sniff, perform the following steps:

1. Load the *Mesh Sniff smart*BASIC example application – use the right-click menu to select **XCompile + Load**.

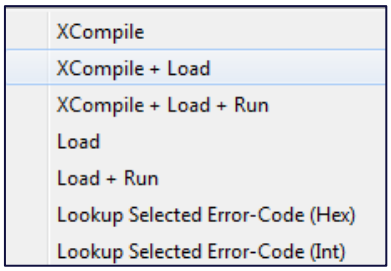


Figure 9: XCompile + Load

2. From the MeshApps subfolder, select the **$autorun$.mesh.sniff.sb** file.

   The mesh program should take approximately ten seconds to load.

3. Wait for the Mesh program to load; this should take approximately 10 seconds.

   Once loaded, you can run the sniff example by typing ***$autorun$*** followed by a return or by pressing the reset button.

   The following message should display (Figure 10).

Figure 10: Mesh sniffer is running

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

LAUNCH AND TEST THE MESH EXAMPLE

This section describes a step-by-step guide to creating and provisioning a mesh of up to four devices (only two are necessary) using Nordic's light switch example but implementing it in Laird's easy-to-use event-driven *smart*BASIC programming environment.

The next couple of sections describes the mesh-related enhancements to *smart*BASIC and then a brief code walkthrough explaining the application used in this step-by-step section.

I. Connecting and Running UwTerminalX for Each Board

To connect and run UwTerminalX for each board, 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.
4. Confirm that you see the following message for the client and server boards (Figure 11).

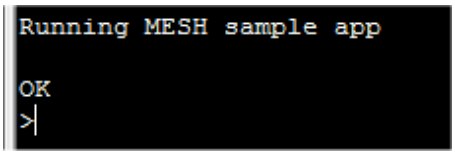


Figure 11: Client and server boards message

5. Confirm you see the following for the Sniff board (Figure 12).

Figure 12: Sniff board message

II. Putting the Boards into Clean, Unprovisioned States

Note: If this is the first time you are running the test (which means the boards are in a clean state), you can skip this step.

If the boards *possibly* have some non-volatile mesh information, we recommend that you use this step to revert ***all*** of the boards to an unprovisioned and clean state.

To put each board into a clean state, follow these steps:

1. In UwTerminalX, untick the DTR checkbox in the toolbar.
2. Tick/untick the BREAK checkbox. This resets the module and starts it up so that the *smart*BASIC **$autorun$** application does not automatically launch.
3. Send **AT** to confirm that the module now accepts AT commands. You should receive an 00 response.
4. Send the **AT I 0x100000** command (note the five zeroes) to erase all flash sectors used by the Mesh stack.
5. In UwTerminalX, click **Clear**.

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

6. Tick the DTR checkbox.
7. Tick/untick the BREAK checkbox to reset the board. Confirm that UwTerminalX displays the following (Figure 13).

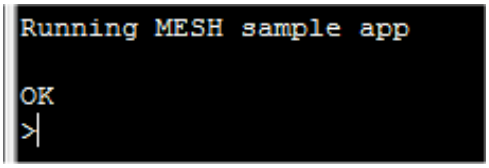


Figure 13: Mesh sample app is running

III. Sniff Board UwTerminalX Screen Confirmation

On the sniff board UwTerminalX screen, confirm that there is no activity other than the MESH SNIFFER message. If this isn't the case, ensure that the DTR checkbox is ticked, clear the screen, and tick/untick the BREAK checkbox.

IV. Starting the Mesh on the Server 0 Board

Start the mesh on the Server 0 board by sending the **ms start** command. The sniff board traffic should display the following (Figure 14) which shows that Server 0 started advertising the fact that it is unprovisioned.

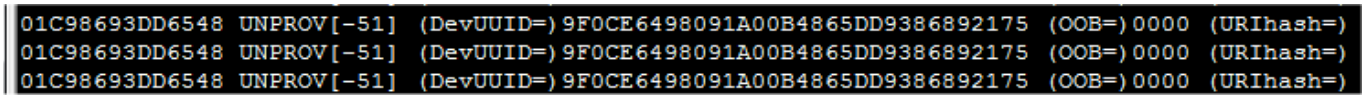
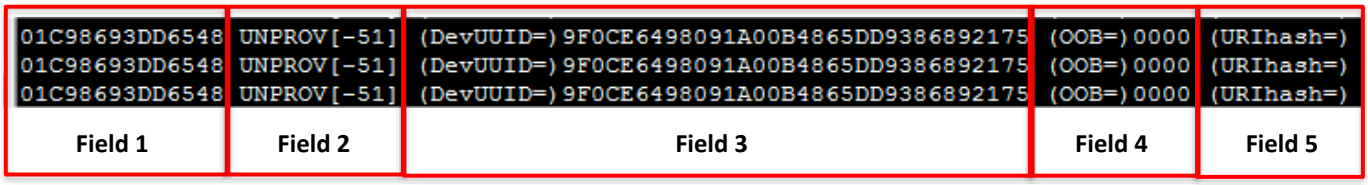


Figure 14: Sniff board traffic



Field 1	Bluetooth address of the unprovisioned mesh device.
Field 2	Indicates that this is an unprovisioned mesh beacon. The **[-51]** is the RSSI value for the beacon that arrived.
Field 3	The device UUID which is factory-programmed into the device. It remains constant for this particular device.
Field 4	The Out-of-Band bit mask which conveys how the authentication phase of the provisioning takes place. The bit mask is reproduced from the following specs:

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

Field 5	(URIhash=) This is currently empty. It would contain an eight-hex digit hash value of a URL that is advertised by this mesh device in a normal advert (arranged via the GATT stack). It can be used to direct the user to a website for installation or product details. See ***smart*BASIC function BleAdvertStart()** in the *BL652 smartBASIC Extensions User Guide* for more details. You can access this guide from the BL652 product page – Documentation section.

V. Provisioning by Client

You can now provision the first mesh device (Server 0). To do this, follow these steps:

1. Click **Clear** on the UwTerminalX screen that is attached to the sniff board.
2. Start the mesh on-board client by sending it the **mc start** command.

   Once the provisioning and configuration is complete, the following displays on the Server 0 UwTerminalX window:

```
>
STATE : Wait_For_Provisioning
STATE : Provisioning_Started
STATE : Prov_Static_Req
STATE : Provisioned <Addr=256 Count=1>
```

Figure 15: Provisioning complete for Server 0

   The final line shows the allocated node address – **256** (0x0100). The count equals 1 because the server only has one element.

   The client UwTerminalX window displays the following:

```
>
STATE : Unprovisioned_recv <Ctx=9F0CE6498091A00B4865DD9386892175>
STATE : Provisioning_Started
STATE : Prov_Static_Req
STATE : Config_Start
STATE : Config_Done
```

Figure 16: Client UwTerminalX window

   The first line shows the UUID of the device being provisioned. The final line (Config_Done) indicates that the device is fully configured.

While everything described above is happening, a lot of traffic will display on the sniff board's UwTerminalX window as follows.

```
01C98693DD6548 UNPROV[-51] (DevUUID=)9F0CE6498091A00B4865DD9386892175 (OOB=)0000 (URIhash=)
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkOPN
(parms=)9F0CE6498091A00B4865DD9386892175
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkOPN
(parms=)9F0CE6498091A00B4865DD9386892175
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkOPN
(parms=)9F0CE6498091A00B4865DD9386892175
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkACK (parms=)
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkACK (parms=)
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)00 (PDU=)START (seg#=)0 (len=)2 (fcs=)14 (data=)0000
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)00 (PDU=)ACK
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)80 (PDU=)START (seg#=)0 (len=)12 (fcs=)87
(data=)010100010001000000000000
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)80 (PDU=)ACK
```


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

```
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)01 (PDU=)START (seg#=)0 (len=)6 (fcs=)B4 (data=)020000010000
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)01 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)START (seg#=)2 (len=)65 (fcs=)9B
(data=)032F55E941D0980450551D4B742D267F3CE253E3
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)CONT  (seg#=)1
(data=)0D61EC9E7E5D1249BB0D5B0F0F3D6C727ED5AB527BDFBD
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)CONT  (seg#=)2
(data=)D2437EA3B09C3B3722BFA93770CF5266876E2490B96A
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)02 (PDU=)ACK
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)81 (PDU=)START (seg#=)2 (len=)65 (fcs=)38
(data=)03EA1ACFB50910D70FFCE51C969A7D1C61D14BF3
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)81 (PDU=)CONT  (seg#=)1
(data=)9B11D6034D23B9774A1A624ED98A0703BA116B0CF90E58
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)81 (PDU=)CONT  (seg#=)2
(data=)25AF0110F54A65A4C50AEB15B97049849CADE3E58F7E
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)81 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)03 (PDU=)START (seg#=)0 (len=)17 (fcs=)3D
(data=)05853081AE7FA16A5D94C0572F292B1CB5
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)03 (PDU=)ACK
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)82 (PDU=)START (seg#=)0 (len=)17 (fcs=)D4
(data=)055F377F9750EA80DA07297C5F5AD23A4A
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)82 (PDU=)ACK
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)04 (PDU=)START (seg#=)0 (len=)17 (fcs=)A2
(data=)06946B0A985A9CC0DD5956CD1B93418BB3
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)04 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)83 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)05 (PDU=)START (seg#=)1 (len=)34 (fcs=)14
(data=)078AB19D80BF8668D7E27F8F7494DE2D9801F5B9
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)05 (PDU=)CONT  (seg#=)1 (data=)0DCA14A2ABBBB4C730C34D155305
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)05 (PDU=)ACK
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)84 (PDU=)START (seg#=)0 (len=)1 (fcs=)3E (data=)08
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)84 (PDU=)ACK
01D103B7F3C38A     MSG[-40] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)C042AB938A4C (dst/pdu/mic=)7C8D01377F1D2A9C01F536417E8C
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01C98693DD6548     MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)DFB06C7085F6
(dst/pdu/mic=)6B681F0CF8AD98DD4D5F4DFE8FBC51AC83D790A9400B
01C98693DD6548     MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)F5FABABBEFE8
(dst/pdu/mic=)4FDCB79882826390692AB4C50D7C0890B5B9119DB9E8
. . .
. . .
. . .
 (dst/pdu/mic=)FC28C6BAA70CB2059199A9FDC27EEC687BE129A41985
01C98693DD6548     MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)2534C3B244A0
(dst/pdu/mic=)47C708D4C9FED1EEDB2A08AF0D62AF6CA9C4
01D103B7F3C38A     MSG[-40] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)4C0B512FA5E1
(dst/pdu/mic=)B077D91D2CC2A7F21977C26CB1A02CF81589D34368
01D103B7F3C38A     MSG[-42] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)DCFA6B83887D
(dst/pdu/mic=)869C8A471851EA0A3CB5D8CADEFFEA23C8
01C98693DD6548     MSG[-50] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)0E4646BC8D00
(dst/pdu/mic=)78686F340E5C4B8A4F0C27201848DCC068612A9247DB
01D103B7F3C38A SECNET[-48] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
```

The first line is the unprovisioned beacon which the client detects. In this example, without any user interaction, it unconditionally assumes that the device should be provisioned. Because of this, the second line displays the following:

```
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkOPN (parms=)9F0CE6498091A00B4865DD9386892175
```

This is the provisioning protocol starting with a Link Open message (LinkOPN).

Note that the last field – labelled (params=) – is the device UUID that was received in the preceding line. It must add that device UUID because there could be many devices in an unprovisioned state; even though all devices receive that message, only the device with the matching UUID acknowledges it.

```
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkACK (parms=)
```


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

Around the place you see the following – the shared secret is being generated using ECDH (Elliptical Curve Diffie-Hellman):

```
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)START (seg#=)2 (len=)65 (fcs=)9B
(data=)032F55E941D0980450551D4B742D267F3CE253E3
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)CONT  (seg#=)1 (data=)0D61EC9E7E5D1249BB0D5B0F0F3D6C727ED5AB527BDFBD
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)02 (PDU=)CONT  (seg#=)2 (data=)D2437EA3B09C3B3722BFA93770CF5266876E2490B96A
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)02 (PDU=)ACK
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)81 (PDU=)START (seg#=)2 (len=)65 (fcs=)38
(data=)03EA1ACFB50910D70FFCE51C969A7D1C61D14BF3
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)81 (PDU=)CONT  (seg#=)1 (data=)9B11D6034D23B9774A1A624ED98A0703BA116B0CF90E58
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)81 (PDU=)CONT  (seg#=)2 (data=)25AF0110F54A65A4C50AEB15B97049849CADE3E58F7E
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)81 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)03 (PDU=)START (seg#=)0 (len=)17 (fcs=)3D (data=)05853081AE7FA16A5D94C0572F292B1CB5
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)03 (PDU=)ACK
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)82 (PDU=)START (seg#=)0 (len=)17 (fcs=)D4 (data=)055F377F9750EA80DA07297C5F5AD23A4A
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)82 (PDU=)ACK
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)04 (PDU=)START (seg#=)0 (len=)17 (fcs=)A2 (data=)06946B0A985A9CC0DD5956CD1B93418BB3
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)04 (PDU=)ACK
```

Then, the provisioning data corresponding to the following information is sent:

ECDHSecret		
Key Index	Net Key 4 (128 bit)	
IV Index	Unicast Addr Primary Node =Y	Flags

This corresponds to the following traffic:

```
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)03 (PDU=)START (seg#=)0 (len=)17 (fcs=)3D (data=)05853081AE7FA16A5D94C0572F292B1CB5
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)03 (PDU=)ACK
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)82 (PDU=)START (seg#=)0 (len=)17 (fcs=)D4 (data=)055F377F9750EA80DA07297C5F5AD23A4A
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)82 (PDU=)ACK
01D103B7F3C38A PB-ADV[-48] (LinkID=)E4196052 (Trans#=)04 (PDU=)START (seg#=)0 (len=)17 (fcs=)A2 (data=)06946B0A985A9CC0DD5956CD1B93418BB3
01C98693DD6548 PB-ADV[-51] (LinkID=)E4196052 (Trans#=)04 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)83 (PDU=)ACK
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)05 (PDU=)START (seg#=)1 (len=)34 (fcs=)14
(data=)078AB19D80BF8668D7E27F8F7494DE2D9801F5B9
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)05 (PDU=)CONT  (seg#=)1 (data=)0DCA14A2ABBBB4C730C34D155305
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)05 (PDU=)ACK
01C98693DD6548 PB-ADV[-50] (LinkID=)E4196052 (Trans#=)84 (PDU=)START (seg#=)0 (len=)1 (fcs=)3E (data=)08
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)84 (PDU=)ACK
```

Finally, the device is in the provisioned state (as shown in Figure 15) and the provisioning link is closed as follows:-

```
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
01D103B7F3C38A PB-ADV[-40] (LinkID=)E4196052 (Trans#=)00 (PDU=)BCTRL (opcode=)LinkCLS (parms=)00
```

Then as per Figure 15, the device is to be configured with publish and subscription information which corresponds to the traffic as follows:

```
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)DFB06C7085F6 (dst/pdu/mic=)6B681F0CF8AD98DD4D5F4DFE8FBC51AC83D790A9400B
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)F5FABABBEFE8 (dst/pdu/mic=)4FDCB79882826390692AB4C50D7C0890B5B9119DB9E8
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)ABD0B7C61961 (dst/pdu/mic=)0A282DC31C5F303C7D88BA13559DDC1A21B9BD4EE285
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)EA6B138C3058 (dst/pdu/mic=)BF48DC62B1442451F4E6C567C15D0A316365D0A0F853
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)93F3C4B83ED8 (dst/pdu/mic=)81BCA027530DD77AF6E4B0CC5CCCEF32FF6F6ADD957D
```

And . . .

```
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)5944C441B39B (dst/pdu/mic=)FC28C6BAA70CB2059199A9FDC27EEC687BE129A41985
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)2534C3B244A0 (dst/pdu/mic=)47C708D4C9FED1EEDB2A08AF0D62AF6CA9C4
01D103B7F3C38A    MSG[-40] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)4C0B512FA5E1 (dst/pdu/mic=)B077D91D2CC2A7F21977C26CB1A02CF81589D34368
01D103B7F3C38A    MSG[-42] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)DCFA6B83887D (dst/pdu/mic=)869C8A471851EA0A3CB5D8CADEFFEA23C8
01C98693DD6548    MSG[-50] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)0E4646BC8D00 (dst/pdu/mic=)78686F340E5C4B8A4F0C27201848DCC068612A9247DB
01D103B7F3C38A SECNET[-48] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
```

These are normal mesh-encrypted advert packets where the Netkey provided in the provisioning phase is used.

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

Once a device is provisioned, the Sniff board displays the following traffic:

```
01D103B7F3C38A SECNET[-48] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)E13D11FC8F0B (dst/pdu/mic=)7C6B45753C1B2835F415D69F9FB6FA
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-52] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-50] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)F6E4AE8AFABA (dst/pdu/mic=)9212DBA44A653C04A800BAE8A62CD4
01D103B7F3C38A SECNET[-48] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-50] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)4DFC971D4F27 (dst/pdu/mic=)7C66FD75BB357B613497A3B467F91E
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)096AC4BD5134 (dst/pdu/mic=)3245A17D24C100C6ABDFC3C61A09BA
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)95D227625D7C (dst/pdu/mic=)21CFDC3A397DCE60D1CA5451759D1F
01D103B7F3C38A SECNET[-48] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-50] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)ED3D8E6F755D (dst/pdu/mic=)75320B60460406DDA0293D213A5F60
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01D103B7F3C38A SECNET[-40] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
01C98693DD6548    MSG[-51] (iv=)0 (nid=)4 (ctl/ttl/seq/src=)A53E4C962069 (dst/pdu/mic=)D40D1640DF1471FD194033B07F1249
```

You will see many lines such as the following:

```
01C98693DD6548 SECNET[-51] (Flags=)00 (NetworkID=)46ADFFAFF97A2B98 (IV=)00000000 (Auth=)4B0AA051BD1F0934
```

These are Secure Network beacons and, in this case, are sent by all provisioned devices to broadcast that it has been provisioned. You can configure these beacons to be disabled; this is controlled by the provisioner.

VI. Provisioning the Other (Optional) Servers

To provision and configure the other two servers (which are optional), send the **ms start** command to each one and confirm that you see the same set of traffic on their respective UwTerminalX windows.

VII. Send ON/OFF Messages from Client to Server 0

The following steps demonstrate how to send a switch ON command from the client to Server 0.

Using the client's UwTerminalX window, send the **mc set 0 1** command where **0** identifies the server number and the second parameter (**1**) requests an ON state (**0** indicate a request for an OFF state).

You can also send the set command by pressing and releasing BUTTON1 on the devkit.

On the server side, upon receipt of the SET opcode, the generic event EVBLEMESH_OPC_MSG is generated and the *smart*BASIC application handler prints the following (the LED1 also changes its state accordingly).

```
EVBLEMSG_OPC_MSG elem:0 hndl:4D444C00 opc:C10059 (SET) data:0100
 ## BleMeshReply() called
 ## BleMeshPublish() called
```

- The opcode is C10059 which, according to Figure 1, is SIMPLE_ON_OFF_OPCODE_**SET**
- The data is 0100 where the first byte is the state requested to be set.
  The second byte, 00, is Nordic's choice of sending a transaction number. This is a Nordic decision – the spec does NOT care about the content of the message, but does it if an adopted message as per the Mesh Model specification is used. Remember… in that case, the opcode is a 1- or 2-byte value).

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

On the client side, the following lines display two instances of the same *smart*BASIC event (EVBLEMESH_SIMPLEONOFF) being handled:

```
EVBLEMSG_OPC_MSG elem:0 hndl:434C5400 opc:C40059 (STATUS) data:01
EVBLEMSG_OPC_MSG elem:0 hndl:434C5400 opc:C40059 (STATUS) data:01
```

Although both lines contain the same information/message, the first message arrived because the server sent a **response** using BleMeshReply(). The second instance is because the state was **published** using the function BleMeshPublish(); this allows for all subscribers of that model to be informed of the new state. In this case, the provisioner (when configuring the client node) set the clients node address as a subscriber of that server.

When the STATUS message is received, LED1 on the devkit is also updated as per the data in the message.

To demonstrate this subtle response and subscribe behavior, do the following:

1. At the client side, enter the **mc setunrel 0 1 1** command.
2. Observe that only one EVBLEMSG_OPC_MSG event is thrown at the client side. This corresponds to the publish given that the unreliable set message was sent (on the server side the opcode is now C30059 which, according to Figure 1 is SIMPLE_ON_OFF_OPCODE_SET_UNRELIABLE).

   Also note that the data at the server side is 0101 where the second 01 is the transaction number and is 01 because that was the second message sent to it.

Send the command 'mc get 0' from the client results in the message opcode C20059 at the server and the event EVBLEMESH_OPC_MSG as a result of the response BleMeshReply() from the server and there was no publish message.

Now let us imagine that at the server side there is a local on/off physical switch which can also result in the state changing and since the behaviour is that if there is a state change all subscribers need to be informed then we will expect a message to be sent to the client.

Hence on the UwTerminalX window attached to 'server 0' enter the command 'ms on'. You will see that the *smart*BASIC app will confirm that BleMeshPublish() was called and then see that at the client side it has received the event EVBLEMESH_OPC_MSG.

Try entering the command 'ms off' and you will see another appropriate print statement at the client side.

If you have also provisioned and configured the other 2 optional servers then appropriate messages on the client side will allow you to set the state of the corresponding server, for example command 'mc set 1 1' or 'mc set 2 1' which are messages sent to server 1 and server 2 respectively.

Consult Table 1 to try all the commands that the *smart*BASIC application responds to and you are free to add and or modify the application as you please.

Finally remember that should you wish to understand the provisioning process better and make that happen many times to fully understand the pattern of behaviour, all you have to do is erase the mesh state information from the flash as described in Step 2.

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

MESH RELATED *SMART*BASIC FUNCTIONS AND EVENTS

This section describes the functions and events included with this experimental and engineering firmware release. Considering the early alpha release of the Nordic mesh SDK on which it's based, Laird reserves the right to change or delete any of functions and events listed below.

Mesh Related AT Commands

AT&F 0x100000

This AT command is used to delete all mesh-related flash sectors to ensure that all state information is deleted. This action results in the device reverting to the unprovisioned state at which point it starts sending unprovisioned beacons.

Mesh Related Result Codes

Many of the new functions return result codes and there is a lookup feature in UwTerminalX that describes what each failure result code mean. The following are new mesh-related result codes for this alpha release:

```
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
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
```

Mesh Related Functions

BleMeshSchemaNew

When a mesh is started, it must know the number of elements the device will expose as well as the models and opcodes each of those elements will host. The element/mesh/opcode information can be viewed as a tree structure of information; this function is used to create a container with a single empty element with the index 0. It takes a single integer argument (the *location* value) as defined in the specification. That value is conveyed to a provisioner during provisioning; it can provide the user with context about the element as part of the composition data.

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

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

BleMeshSchemaNew(nLocation)

Returns	INTEGER : resultCode 0x0000 : Success 0x0607 : Location value not in range 0x0000 to 0xFFFF
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 For this alpha release, provide any negative value when registering the Simple On_Off clients after adding the provisioning configuration foundation model client. (See sample example and look for the function **mesh_start_client**)

BleMeshAddSigModel

A BLE Mesh device contains at least one Element (default one added with **BleMeshSchemaNew()**, or when subsequent ones added with **BleMeshAddElement()**).

Each Element in turn contains at least one Model which in turn contains at least one Opcode.

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 contains opcodes and the function **BleMeshAddOpcode()** 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 opcodes 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 a previously-added model then it shall be the handle of that model obtained when BleMeshAddSigModel() or BleMeshAddVendorModel() is called, **otherwise it shall contain 0**. On exit, this is an opaque handle value that the *smart*BASIC app uses to describe a model when an API interacts with a model or when a message arrives, this value is presented to enable the developer to channel the behavior accordingly. We recommend that it is stored in a global *smart*BASIC variable.

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

BleMeshAddVendorModel

A BLE Mesh device contains at least one Element (default one added with **BleMeshSchemaNew()**, or when subsequent ones added with **BleMeshAddElement()**).

Each Element in turn contains at least one Model which in turn contains at least one Opcode.

Use this function to add a model using a 32-bit vendor identifier in the form of two 16 bit values (nCompanyId and nModelId) to the mesh schema. It is added to the most recently-added element. A model contains opcodes and the function **BleMeshAddOpcode()** 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 opcodes 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 note that if you want to interact with a Nordic defined model, it is 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 previously-added model then it shall be the handle of that model obtained when BleMeshAddSigModel() or BleMeshAddVendorModel() was called, **otherwise it shall contain 0**. 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 Laird recommends that it be stored in a global *smart*BASIC variable.

BleMeshAddOpcode

A BLE Mesh device contains at least one Element (default one added with **BleMeshSchemaNew()**, or when subsequent ones added with **BleMeshAddElement()**).

Each Element in turn contains at least one Model (added using the function **BleMeshAddSigModel()** or **BleMeshAddVendorModel()**) which in turn contains at least one Opcode so that incoming messages containing those opcodes can be processed

Use this function to add a **packed** opcode which is a value in up to 3-bytes long.

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

Note: If this function fails with BLE_MESH_DUPLICATE_OPCODEID (0x60CB), it implies that your mesh structure is faulty. If you need a duplicate opcode, 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.

BleMeshAddOpcode(nPackedOpcode)

Returns	INTEGER : resultCode 0x0000 : Success 0x06C3 : No models have been added to the current element 0x60C2 : Too many opcodes have been added. Limit will be exceeded 0x60CE : nPackedOpcode is invalid 0x60CB : Current element already has this opcode 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.

BleMeshAddElement

Use this function to add another element to the container started with **BleMeshSchemaNew()** to which more instances of models and op-codes are added.

As mentioned in the description for **BleMeshAddOpcode()** a new element is needed if a device will end up with multiple instances of opcodes. The Mesh specification mandates that an element SHALL have only one instance of an opcode.

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

BleMeshStart

Once an Element/Model/Opcode tree has been defined using the functions described above, it must be registered with the Mesh stack and started. This function consistently does this 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 and knows how to fork from there. If the non-volatile data is missing or does not match, 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.

BleMeshStart(nflags, nCompanyId, nProductId, nVersionId, nFeatures, nDefaultTTL)

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:	
nFlags	**byVAL *nFlags* AS INTEGER.** Bit 0 is set if a Provisioning Config Client is to be added to the primary element. Bits 1 to 31 are for future use and should be set to 0
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 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 desired value; 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 desired value.
nFeatures	**byVAL *nFeatures* AS INTEGER.** The following define which mesh features each bit mask specifies: 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 this release always set this value to 1. The other features have not yet been implemented in the underlying stack.
nDefaultTTL	**byVAL *nDefaultTTL* AS INTEGER.** The default time to live for all mesh network messages sent from this node. It can be overridden in the publication state

BleMeshPublish

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

BleMeshPublish(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
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 is a value in the range 0x0000 to 0xFFFF. For a vendor-defined opcode, the value is 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 is sent as payload for the message. The specification allows this to be from 0 to 380-bytes. It is appropriately lower if the opcode is three-bytes long

BleMeshPublishReliable

This function is used to publish a reliable message with all the parameters as described for the function BleMeshPublish(). It requires two additional 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.

BleMeshPublishReliable(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 is a value in the range 0x0000 to 0xFFFF. For a vendor-defined opcode, this is the value 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.

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

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 is a value in the range 0x0000 to 0xFFFF. For a vendor-defined opcode, the value is 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.
sData$	**byREF *sData$* AS STRING.** This contains the data that is sent as payload for the message. The specification allows this to be from 0 to 380-bytes. It is appropriately lower if the opcode is three-bytes long

BleMeshReply

This function is used to send a response to an incoming message with the opcode and data specified using the destination details embedded in the opaque parameter sReplyData$ which was supplied when the incoming message arrived via the event EVBLEMESH_OPC_MSG. This is described later.

The sReplyData$ also contains the appkey that was used by the incoming message; the response needs to use the same one.

Note: handleModel and nPackedOpcode were also supplied in the EVBLEMESH_OPC_MSG event when the incoming message arrived.

BleMeshReply(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 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 is a value in the range 0x0000 to 0xFFFF. For a vendor-defined opcode, the value is 0xPPVVVV where PP is a value in the range 0xC0 to 0xFF and VVVV is the companyID.
sData$	**byREF *sData$* AS STRING.** This contains the data that is sent as payload for the message. The specification allows it to be from 0 to 380-bytes. It is appropriately lower if the opcode is three-bytes long
sReplyInfo$	**byREF *sReplyInfo$* AS STRING.** This is supplied in the EVBLEMSG_OPC_MSG event and MUST be supplied unmodified from there. It is an opaque object and is checked for modification. If modifications exist, it results in a failure to send a response.

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

Mesh Related Events

EVBLEMESH_STATE

This event occurs when the mesh state of the device changes.

Parameters:	
nNewState	**byVAL *nNewState* AS INTEGER.** This contains the new state
sContext$	**byREF *sContext$* AS STRING.** This contains context data for the new state

The values for nNewState and associated context strings are described in the following table (). If the context column is 'none' then the string will be empty.

Table 3: nNewState and associated context strings

Value	Description	Context Data
100	WAIT_FOR_PROVISIONING	None
110	PROVISIONING_START	None
120	PROVISIONING_OUTPUT_REQ	None A usage example will be provided in a future release rather than this firmware release.
130	PROVISIONING_INPUT_REQ	None A usage example will be provided in a future release rather than this firmware release.
140	PROVISIONING_STATIC_REQ	None
150	PROVISIONING_OOB_PUBKEY_REQ	None A usage example will be provided in a future release rather than this firmware release.
190	PROVISIONING_FAIL	None
200	PROVISIONED	First two bytes – First element node address Second two bytes – Number of elements **Note:** Two-bytes entities are little endian.
210	CONFIGURATION_START	None
280	CONFIGURATION_DONE	None
290	CONFIGURATION_FAIL	None
300	KEY_REFRESH_START	None
390	KEY_REFRESH_END	None
400	IV_UPDATE_NOTIFICATION	None
500	UNPROVISIONED_DEVICE	16-byte device UUID

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

EVBLEMESH_OPC_MSG

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

Parameters:	
nElementIndex	**byVAL *nElementIndex* AS INTEGER.** This contains the element index 0 to N and corresponds to the elements that are added using BleMeshElementAdd()
handleModel	**byVAL *handleModel* AS INTEGER.** This contains the handle returned by BleMeshAddSigModel() or BleMeshAddVendorModel()
nPackedOpcode	**byVAL *nPackedOpcode* AS INTEGER.** This contains the packed opcode. For a SIG-defined opcode, this is a value in the range 0x0000 to 0xFFFF. For a vendor-defined opcode, the value is **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 is used if BleMeshReply() is called and should be supplied unmodified to that function.

Typically, the *smart*BASIC handler switch on the nPackedOpcode value (using the Select compound statement) and then calls an appropriate function to handle the data

*SMART*BASIC APP CODE WALKTHROUGH

This section describes code fragments from the following *smart*BASIC application:
$autorun$.mesh.light.switch.example.sb

The application waits for characters to arrive over the UART. When a carriage return (0x0D) character is received, the application passes all characters accumulated since the last carriage return character to the **OnUartCmd()** function for processing.

When a character arrives, the EVUARTRX event handler – **HandlerUartRxCmd()** – is invoked**.** The handler for that UARTRX event is registered using the **OnEvent** statement. Search for that statement towards the end of the *.sb* file.

Near the same statement are three more OnEvent statements for the two mesh-related events described in this application note. These events are **EVBLEMESH_OPC_MSG** and **EVBLEMESH_STATE** which invoke the handlers **HandlerMeshOpcMsg**() and **HandlerMeshState**() respectively.

- **HandlerMeshState() for event EVBLEMESH_STATE** – Prints the state and context value, if it exists.
- **HandlerMeshOpcMsg() for event EVBLEMESH_OPC_MSG** – Prints all parameters and then, based on the opcode, sends a reply and/or publishes the current on/off state.

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

REFERENCES

The following documents are also accessible from the BL652 product page of the Laird website (Documentation tab):

- BL652 *smart*BASIC extension manual
- BL652 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

REVISION HISTORY

Version	Date	Notes	Contributor(s)	Approver
0.10.0/rel10	20 Dec 17	Initial Release	Mahendra Tailor	Jonathan Kaye