Step by step guide mesctl with Zephyr

Introduction

This step-by-step guide is based on the "Tutorial-How-to-set-up-BlueZ_Part2-3" guide [1], but uses the Raspberry Pi 4 and an nRF52-DK specifically. In the following chapters, all steps are given to switch on and off an LED on the nRF52-DK, using bluez meshctl on the Raspberry Pi 4. In addition, pictures are added to visualise the steps. Also, PuTTY will be running on a Windows 10 laptop.

Within all chapters, the left column shows the (terminal/meshctl) commands that should typed in the Raspberry Pi terminal. The right column shows a short description about the left command. Pictures about the steps are given below the specific command(s) for this step.

Note: During long in-active behaviour, the meshctl will disconnect the provisioned device. Step 6b (type: connect) should be re-done before continuing.

Staring meshctl

Step	Raspberry Pi terminal - command	Description
1.	`cd`	Go back to main directory
2.	`cd bluez-5.50/mesh`	Go to directory with `meshctl`
3.	`meshctl`	Launch `meshctl`

Provisioning

First turn on the nRF52-DK and connect its micro-USB input to the USB input of the Windows 10 laptop (can be any other device with a serial monitor, like PuTTY, installed).

Step	`meshctl` - command(s)	Description
4.	Press `Enter`	Reveal `meshctl` prompt
5.	`discover-unprovisioned on`	Discover unprovisioned BLE mesh devices

When a new device appears, make sure it states the correct name after [NEW] (Here: Zephyr) and copy the Device UUID: → continue with step 6a-1.

```
[meshctl]# discover-unprovisioned on
SetDiscoveryFilter success
Discovery started
Adapter property changed
[CHG] Controller DC:A6:32:6F:94:30 Discovering: yes
Services resolved no
                Mesh Provisioning Service (00001827-0000-1000-8000-00805f9b34fb)
                        Device UUID: 34ab739aa2f00000000000000000000c
                        OOB: 0000
[NEW] Device F0:A2:9A:73:AB:34 Zephyr
[meshctl]#
```

When no new device appears, it may already be provisioned: → continue with step 6b.

```
[meshctl]# discover-unprovisioned on
SetDiscoveryFilter success
Discovery started
Adapter property changed
[CHG] Controller DC:A6:32:6F:94:30 Discovering: yes
[meshctl]#
```

Step	`meshctl` - command(s)	Description
6a-1.	`provision 34ab739aa2f0000000000000000000000`	Provision device with its UUID (see step 5). (Note: this is 1 command, do not press enter in between).

When done correctly, this can be seen on the terminal:

```
[meshctl]# provision 34ab739aa2f0000000000000000000000
Trying to connect Device F0:A2:9A:73:AB:34 Zephyr
Adapter property changed
[CHG] Controller DC:A6:32:6F:94:30 Discovering: no
Connection successful
Service added /org/bluez/hci0/dev_F0_A2_9A_73_AB_34/service0010
Char added /org/bluez/hci0/dev_F0_A2_9A_73_AB_34/service0010/char0011:
Char added /org/bluez/hci0/dev_F0_A2_9A_73_AB_34/service0010/char0013:
Services resolved yes
Found matching char: path /org/bluez/hci0/dev_F0_A2_9A_73_AB_34/service0010/char
```

Somewhere in between that, it should now ask for an (ASCII) key:

```
Request ASCII key (max characters 6)
[mesh] Enter key (ascii string):
```

The serial terminal should look like this:

```
COM3 - PuTTY
OOB String 851IR7K
provisioning complete for net idx 0x0000 addr 0x0108
[00:00:31.697,448] <inf> bt_mesh_main: Primary Element: 0x0108
[00:00:31.697,479] <dbg> bt_mesh_main.bt_mesh_provision: net_idx 0x0000 flags 0x
00 iv_index 0x0005
[00:00:31.703,979] <dbg> bt_mesh_main.bt_mesh_provision: Storing network informa
tion persistently
```

Copy the OOB String (here: 851IR7K – for step 6a-2) and the Primary Element without 0x (0108 – for step 8). Continue with step 6a-2.

6a-2.	`851IR7K`	This is the OOB string from the serial terminal (see step 6a-1.) **Note** that this will be different every time, for security purpose. Provisioning is only accepted when the correct key is provided.

Copy the OOB String and paste it here in the Raspberry Pi terminal (meshctl):

```
Request ASCII key (max characters 6)
[mesh] Enter key (ascii string):
```

Something similar to this should appear:

```
Connection successful
Services resolved yes
[Zephyr-Net-0000]#
```

If it fails, try again from step 5 (most likely you must go to step 6b now). If it continuous to fails, restart at step 1. Continue with step 7 if it worked.

6b.	`connect`
	OR:
	`connect <net_idx>`
		Connect to the nRF52-DK, which was previously provisioned and not changed. `<net_idx>` can be found in the serial monitor above the line with the Primary Element (given in step 6a-1 during provisioning a new device).

Something similar should appear. **Note:** This can take some time. If nothing happens or you didn't connect properly (showing the new command request), try again several times.

Step	`meshctl` - command(s)	Description

```
[meshctl]# connect
Looking for mesh network with net index 0000
SetDiscoveryFilter success
Discovery started
Adapter property changed
[CHG] Controller DC:A6:32:6F:94:30 Discovering: yes

                Mesh Proxy Service (00001828-0000-1000-8000-00805f9b34fb)
                Network Beacon for net index 0000
Trying to connect to mesh
Adapter property changed
[CHG] Controller DC:A6:32:6F:94:30 Discovering: no
Connection successful
Services resolved yes
Found matching char: path /org/bluez/hci0/dev_F0_A2_9A_73_AB_34/service0010/char
```

Note the line with “Connection successful”. The new command request look like this:

```
[Zephyr-Net-0000]#
```

Note that it contains the device name (Zephyr) and the net idx (Net-000) between [].

Something similar to this should appear on the serial terminal (PuTTY):

```
COM3 - PuTTY
provisioning complete for net idx 0x0000 addr 0x0108
[00:00:31.697,448] <inf> bt_mesh_main: Primary Element: 0x0108
[00:00:31.697,479] <dbg> bt_mesh_main.bt_mesh_provision: net_idx 0x0000 flags 0x
00 iv_index 0x0005
[00:00:31.703,979] <dbg> bt_mesh_main.bt_mesh_provision: Storing network informa
tion persistently
```

Coy the Primary Element without 0x (0108 - step 8).
If it fails, try again (it can take some time to re-establish the connection). If it continuous to fails, restart at step 1. Continue with step 7 if it worked.

Configuration

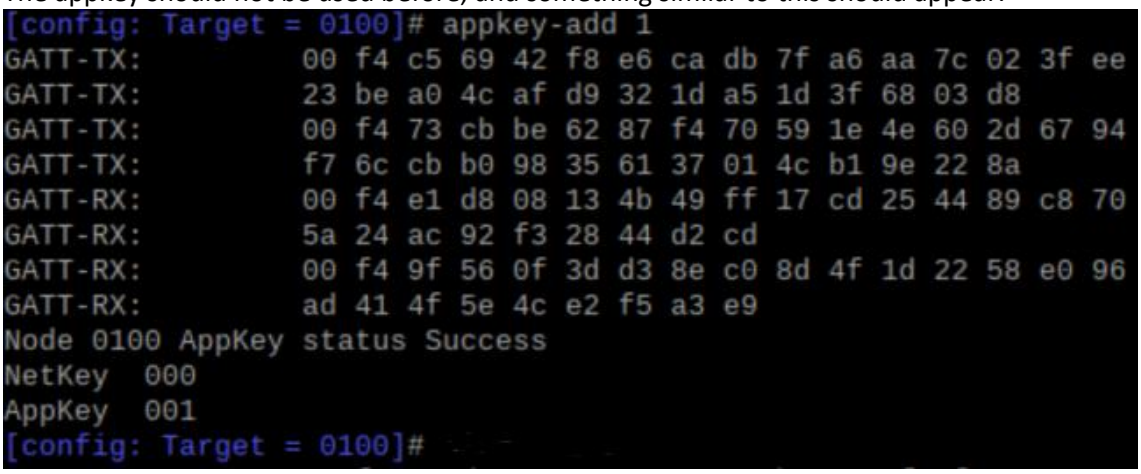
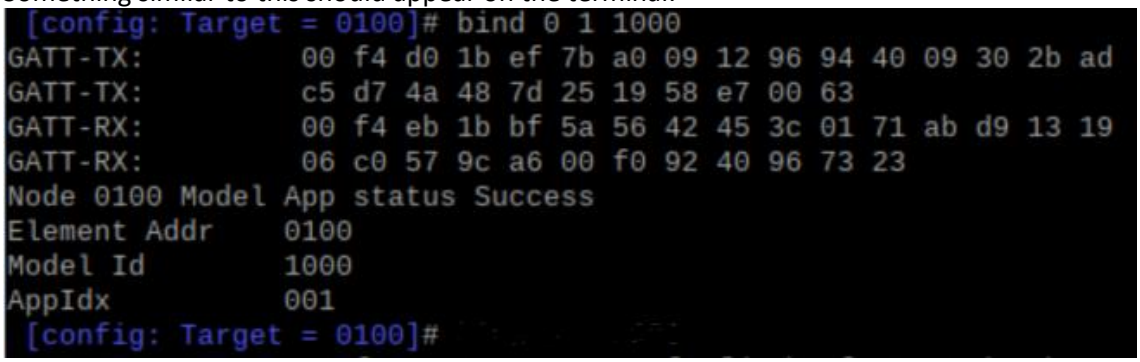
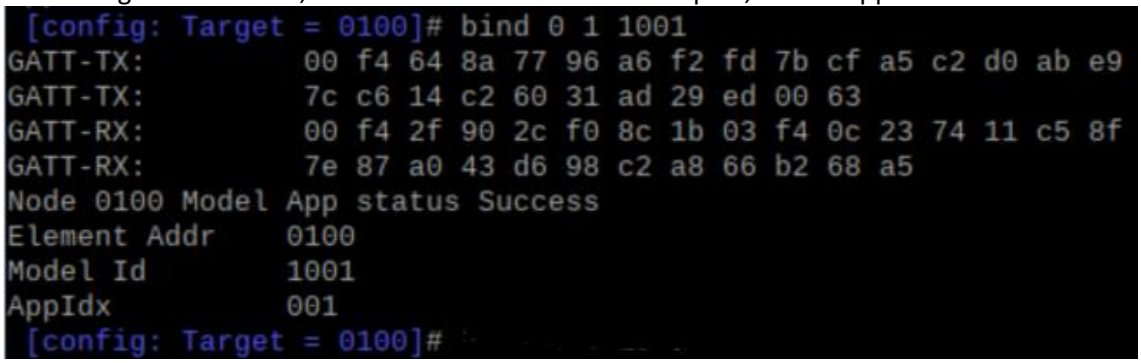
This should only be done ones per LED/like-wise. When you already did this, continue with step 15 in the next chapter.

Step	`meshctl` - command(s)	Description
7.	`menu config`	Entering Configuration Menu of `meshctl`
8.	`target 0108`	This Unicast Address (Primary Element from serial terminal) will from now on be used for any `target` command. (Always refer to the serial monitor, PuTTY, for the correct target. Errors, like “Destination not set” or “Node xxxx not found” – Change this target device when another must be selected).

The name should change to [config: Target = 01008]:

```
[Zephyr-Net-0000]# target 0100
Configuring node 0100
[config: Target = 0100]#
```

Note line 2 indicating configuration on target node 0100: *“Configuring node 0100”*.

Step	`meshctl` - command(s)	Description
Also, note that the Unicast Address in the screenshot is 0100, because another Unicast Address was given via the serial terminal (step 6a-2 or step 6b). All following screenshots will contain Unicast Address 0100, but keep in mind that you should use your own Unicast Address.		
9.	`appkey-add 1`	Load `AppKey` #1 from `prov_db.json`
The appkey should not be used before, and something similar to this should appear:  Note the line *"Node 0100 AppKey status Success"*.		
10.	`bind 0 1 1000`	Expose Generic On/Off server, locked by `AppKey` #1. Syntax: `bind <ele_indx> <app_idx> <mod_id>`. Where: `<ele_index>` = 0, `<app_idx>` = appkey nr from step 9, and `<mod_id>` = 1000
Something similar to this should appear on the terminal:  Note *Success* in the line: *"Node 0100 Model App status Success"*.		
11.	`bind 0 1 1001`	Repeating previous step, but with `<mod_id>` = 1001
Something similar to this, but other in GATT-xx than in step 10, should appear on the terminal:  Note *Success* in the line: *"Node 0100 Model App status Success"*.		

Step	`meshctl` - command(s)	Description
12.	`sub-add` *`0100`* `c000 1000`	Add subscribe to On/Off Updates at Group Address `c000`. Where: `<ele_addr>` = target's Unicast Address, `<sub_addr>` = Group Address = `c000`, and `<model id>` = `<mod_id>` from step 10

Something similar to this, but other in GATT-xx than in step 10 - 11, should appear on the terminal:

```
[config: Target = 0100]# sub-add 0100 c000 1000
GATT-TX:          00 f4 05 b5 f6 25 1c e5 07 88 ee 76 40 5b f8 52
GATT-TX:          ea 68 33 a3 ea 5e 73 5b f3 f8 dc
GATT-RX:          00 f4 01 24 d0 76 11 cc dd fa df 57 6e 0d 9c 14
GATT-RX:          00 56 ee ad a4 65 fc 5c ea e9 c1 3a

Node 0100 Subscription status Success
Element Addr     0100
Model Id         1000
Subscr Addr      c000
[config: Target = 0100]#
```

Note *Success* in the line: *"Node 0100 Subscription status Success"*.

13.	`pub-set` *`0100`* `c000 1 0 5 1001`	Publish On/Off Updates at Group Address `c000`. Where: `<ele-addr>` = target's Unicast Address, `<pub_addr>` = Group Address = `c000`, `<app_id>` = appkey nr from step 9, `<per (setp|res)` = period in ms, `<re-xmt (cnt|per)` = rexmit count, and `<mod id>` = `<mod_id>` from step 11

Something similar to this, but other in GATT-xx than in step 10 - 12, should appear on the terminal:

```
[config: Target = 0100]# pub-set 0100 c000 1 0 5 1001
GATT-TX:          00 f4 77 f6 79 35 fb 83 f2 43 53 3b a1 94 f9 76
GATT-TX:          0e 67 bd 58 6d 4f 23 e4 87 e7 4f b1 6d c7
GATT-TX:          00 f4 df 2a 34 d1 2c 59 28 70 26 fb bd 23 42 7d
GATT-TX:          5d f0 83 8e 9b 16
GATT-RX:          00 f4 f8 b0 e5 92 27 3a bd 90 49 6a 4d 0e 83 78
GATT-RX:          0c 36 d8 3e e0 04 5c 58 6f
GATT-RX:          00 f4 2b f6 44 23 d8 fe b8 e2 d9 27 17 0d 2a a5
GATT-RX:          0d 41 fa 2d 69 60 97 86 d7 ed 0d 59 8f 37
GATT-RX:          00 f4 0c e0 ff 6c 14 87 d4 a2 f0 80 5b e6 8d c3
GATT-RX:          b9 65 a9 34 32 fb e9 97

Node 0100 Publication status Success
Element Addr     0100
Model Id         1001
Pub Addr         c000
Period           0 ms
Rexmit count     0
Rexmit steps     5
GATT-TX:          00 f4 f0 ef 94 27 9d ac 24 c5 c5 8c 4e a5 66 88
GATT-TX:          6a 0f b7 15 94 48 83 ea 27
[config: Target = 0100]#
```

Note *Success* in the line: *"Node 0100 Publication status Success"*.

Step	`meshctl` - command(s)	Description
14.	`back`	Go back to the main menu of `meshctl`

Turning on/off the LED

When all configurations and connections are established, steps 17 and 18 can be repeated to turn on/off the LED several times. Changing between devices (nRF52-DKs) can be done by repeating step 6b and continuing with step 17 and 18 (if all configured, otherwise continue from step 7 after step 6b). Changing between LEDs/like-wise, can be done by targeting a different Unicast Address (target) (stays on the same device) in step 16 (and configurations from step 7 onwards if not done yet).

Step	`meshctl` - command(s)	Description
15.	`menu onoff`	Entering `onoff` Menu of `meshctl`
16.	`target` *`0108`*	Target the new node (use specific Unicast Address).

This should appear on the terminal: (Unicast Address be different, but consistent).

```
[config: Target = 0108]# target 0108
Controlling ON/OFF for node 0108
[on/off: Target = 0108]#
```

Notify the change from [config: Target = 0108] to [on/off: Target = 0108]. Now the node 0108 can be controlled on/off instead of changing configurations for this node.

17.	`onoff 0`	Set on/off client to 0 (= off). LED1 on nRF52-DK should turn off.

The Raspberry Pi terminal should show this:

```
[on/off: Target = 0108]# onoff 0
GATT-TX:          00 f4 06 2f 74 ab f6 dc 1b d0 9f 80 ff 4c 24 2c
GATT-TX:          38 e2 db 5f 3e 19 db
GATT-RX:          00 f4 c4 45 9a a5 3d cc e6 65 47 a9 c7 48 ed 65
GATT-RX:          58 50 79 43 4e e9
On Off Model Message received (1) opcode 8204
        00

[on/off: Target = 0108]#
```

Note the last line copies the `onoff` statement 0.

The serial monitor, PuTTY, shows this:

```
COM3 - PuTTY
gen_onoff_set
addr 0x108 state 0x00
addr 0x0108 onoff 0x00
```

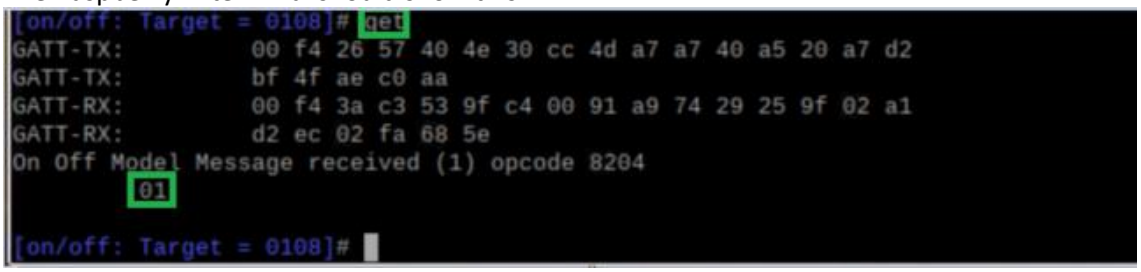
Note the last line showing *onoff 0x00*, which indicates a `onoff` of 0 was commanded.
Also note the Unicast Address, before *onoff 0x00*, corresponds with the targeted Unicast Address.

18.	`onoff 1`	Set on/off client 1 (= on). LED1 on nRF52-DK should turn on.

The Raspberry Pi terminal should show this:

```
[on/off: Target = 0108]# onoff 1
GATT-TX:          00 f4 f3 be 5b 4b 93 31 98 c3 bf dd a7 50 4c 1b
GATT-TX:          a3 f9 db 6a d5 2d 90
GATT-RX:          00 f4 b1 15 21 01 9d f1 84 26 d6 c9 86 6d 5c 6a
GATT-RX:          5d e0 3a ba 58 77
On Off Model Message received (1) opcode 8204
        01

[on/off: Target = 0108]#
```

Step	`meshctl` - command(s)	Description
Note the last line copies the `onoff` statement 1. The serial monitor, PuTTY, shows this:  Note the last line showing *onoff 0x01*, which indicates a `onoff` of 1 was commanded. Also note the Unicast Address, before *onoff 0x01*, corresponds with the targeted Unicast Address. Other commands, like 0xA0 or 0x08, can also be typed, but do not effect the LED other than using 0x01.		
19.	`get`	Show on/off Client status. Can be done at any time, after configuration. Shows the LED status (0 or 1).
The Raspberry Pi terminal should show this:  Note the last shows 01, which means that the LED is on (this was previously commanded). The serial monitor, PuTTY, shows this:  Note this line shows *onoff 0x01*, which means that the LED is on (this was previously commanded). Also note the Unicast Address, before *onoff 0x01*, corresponds with the targeted Unicast Address.		
20.	`exit`	Exit `meshctl`. Mesh details have been automatically updated into `prov_db.json`

References

[1] K. Ren, “Step-by-Step Guide How to deplow BlueZ v5.50 on Raspberry Pi 3 and Use it,” 12 October 2018. [Online]. Available: https://3pl46c46ctx02p7rzdsvsg21-wpengine.netdna-ssl.com/wp-content/uploads/2019/03/Tutorial-How-to-set-up-BlueZ_Part2-3.pdf. [Accessed 9 July 2020].