Step-by-Step Guide

How to Deploy BlueZ v5.50 on Raspberry Pi 3 and Use It

Part 2 — Provisioning

BlueZ is the official Linux Bluetooth® protocol stack. As stated in the BlueZ v5.47 release notes, “this release comes with initial support for it in the form of a new meshctl tool. Using this tool, it’s possible to provision mesh devices through the GATT Provisioning Bearer (PB-GATT), as well as communicate with them (e.g. configure them) using the GATT Proxy protocol.” This tutorial shows you how to build a new (unprovisioned) device, provisioned by meshctl on Raspberry Pi3 (R Pi3) board.

By the end of this step-by-step guide, you will be able to issue a meshctl command in the folder ~/bluez-5.50/mesh/, run the meshctl utility, and know how to use the meshctl utility to provision a new (unprovisioned) device and manage the network.

To learn the steps for installing BlueZ v5.50 on R Pi3, see Part 1 of this guide, Deployment.

Author: Kai Ren
Version: 1.0
Revision Date: 12 October 2018
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2 October 2018</td>
<td>Kai Ren</td>
<td>Initial Version</td>
</tr>
</tbody>
</table>
# Table of Contents

1.0 Prerequisite .................................. 4  
  1.1 Sample Project .............................. 4  

2.0 Install BlueZ v5.49 .......................... 5  
  2.1 Open prov_db.json ......................... 5  
  2.2 Edit prov_db.json ........................... 5  

3.0 Provisioning .................................. 8  
  3.1 Commands Available on Main Menu .... 8  
  3.2 Commands Available on Main Menu .... 8  
  3.3 Provision .................................... 9  
  3.4 Composition ................................ 10  

4.0 Model Configuration ......................... 12  
  4.1 Configuration ............................... 12  

5.0 Verification ................................... 14  

6.0 Summary ..................................... 15
1.0 Prerequisite

Set Up the Development Environment

Before building a new device, you should follow this guide to setup the dev environment.

- **Windows**
- **macOS**
- **Linux**

Sample Project

This [sample project](#) illustrates how to use `meshctl` to provision and model config the device: this project is a sample from the [Zephyr Project](#).

Please follow the [sample project](#) webpage to build and flash the firmware on the target board (nRF52840 DK). Then, open a serial terminal like **Putty** or **Tera Term** to monitor out-of-band (OOB) output from the target board. Select the correct serial port on your host computer, the serial terminal setting is:

- Baud rate: 115200
- Data bit: 8
- Parity: none
- Stop bit: 1

When you press the reset button on the board, the serial terminal should look like the image below.

```
```

For any other project which includes a **GenericOnOff Server** model and/or a **GenericOnOff Client** model, you can use this guide to provision and model config the device as well.
2.0 Provisioner Configuration

`meshctl` is a tool in BlueZ v5.50 that works as a provisioner and distributes provisioning data (unicast address, NetKey, key index, and IV Index) to new, unprovisioned devices; with it, users can configure provisioning data by themselves. For example, change NetKey or IV Index, editing a json file on BlueZ can make it happen.

Open prov_db.json

Access the folder `./bluez-5.50/mesh/` and use the editor you would like to use to open `prov_db.json`. For this tutorial, we used `vi`.

```
pigrraspberrypi:~ $ cd bluez-5.50/mesh/
pigrraspberrypi:~/bluez-5.50/mesh $ vi prov_db.json
```

Edit prov_db.json

After opening `prov_db.json`, you should see:

If you want to change the IV Index setting, edit at line 4 as shown in the image below.

```
"IVindex": 5  #change IV Index, current setting is 5
```
If you want to change the NetKey setting, edit at line 6 as shown in the image below.

```json
"netKeys": [
    {
        "index": 0,  #change netkey index, current setting is 0
        "keyRefresh": 0,
        "key": "18eed9c2a56add85049ffc3c59ad0e12"  #change netkey
    }
],
```

If you want to change the AppKey setting, edit at line 13 as shown in the image below.

```json
"appKeys": [
    {
        "index": 0,  #change AppKey index, current setting is 0
        "boundNetKey": 0,
        "key": "F30AC76210160E03F2D8B4F1CF4510E2"  #change AppKey0
    },
    {
        "index": 1,  #change AppKey index, current setting is 1
        "boundNetKey": 0,
        "key": "F30AC76210160E03F2D8B4F1CF4510E2"  #change AppKey1
    }
],
```
If you want to change the *Unicast Address* pool, edit at line 29 as shown in the image below.

```json
"allocatedUnicastRange": [  
  
  { 
    "lowAddress": "0100",  # head of unicast address pool 
    "highAddress": "7fff"  # tail of unicast address pool 
  }  
]
```
### 3.0 Provisioning

#### Commands Available on Main Menu

After reading Part 1, you should know how to launch `meshctl`. When `meshctl` is foreground, type `help` to list main menu commands as shown below.

![Image of meshctl commands]

From the image above, you can see that every command has a description, and this guide shows you a few examples.

<table>
<thead>
<tr>
<th><strong>Main Menu Commands</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>discover-unprovisioned &lt;on/off&gt;</code></td>
<td>Look for devices to provision</td>
</tr>
<tr>
<td><code>provision &lt;uuid&gt;</code></td>
<td>Initiate provisioning</td>
</tr>
<tr>
<td><code>disconnect &lt;dev&gt;</code></td>
<td>Disconnect device</td>
</tr>
</tbody>
</table>

#### Discover an Unprovisioned Device

To discover nearby unprovisioned devices, type in the command below.

```
discover-unprovisioned on
```

```bash
[meshctl]$ discover-unprovisioned on
SetDiscoveryFilter success
Discovery started
Adapter property changed
[CHG] Controller 00:1B:DC:80:38:29 Discovering: yes
Mesh Provisioning Service: 829e6a00-0000-1000-0000-0000000000005f9b34fb
Device UID: 0000
0000:6b8b-0000:6b8b-0000:6b8b-0000:6b8b-0000:6b8b
[meshctl]$`
```
The device UUID is `6bebf114000000000000000000000000`. The endian is LSB to MSB.

**Provision**

Copy the device UUID and paste it after the provision command, as shown below, to initiate the provision process.

```
provision 6bebf114000000000000000000000000   #paste the target device UUID on your own
```

After hitting the `Enter` key on your keyboard, you will see that the provisioning process is initiating, as shown in the image below.

```
[meshctl]$ provision 6bebf114000000000000000000000000
Trying to connect Device ED:E4:14:F1:18:60 Zephyr
Adapter property changed
Connection successful
Service added /org/bluez/hci0/dev ED:E4:14:F1:18:60/service0000
Char added /org/bluez/hci0/dev ED:E4:14:F1:18:60/char0000
Char added /org/bluez/hci0/dev ED:E4:14:F1:18:60/char0001
Services resolved yes
```

Next, `meshctl` will ask you to type the OOB output value. This value can be obtained on the serial terminal. Please type the OOB output value as shown in the example below. You can see that the current OOB output value on serial terminal is `FSQR3G`.

```
Request ASCII key (max characters 6)
[mesh]$ Enter key (ascii string): FSQR3G
```

If the OOB output value is correct, `meshctl` will move forward and reach the final step: Composition Data (details about Composition Data refer to *Mesh Profile Specification v1.0*, Section 4.2.1) as shown in the image below. This means the provisioning process was successful and the Provisioner got Composition Data from a just-provisioned node.
Composition

After interpreting the Composition Data, the information means:

- cid, company identifier of the node: 0x05f1
- pid, vendor-assigned product identifier of the node: 0x0000
- vid, vendor-assigned product version identifier of the node: 0x0000
- crpl, the minimum number of replay protection list entries in a device: 0x000a
- features:
  - relay: enabled;
  - proxy: enabled;
  - friend: disable;
  - low power node: disable
<table>
<thead>
<tr>
<th>element</th>
<th>model</th>
<th>model id</th>
<th>unicast address (element address)</th>
<th>element index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary element</strong></td>
<td>Configuration server</td>
<td>0x0000</td>
<td>0x011b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Configuration client</td>
<td>0x0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Health server</td>
<td>0x0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GenericOnOff Server</td>
<td>0x1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GenericOnOff Client</td>
<td>0x1001</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Secondary element</strong></td>
<td>GenericOnOff Server</td>
<td>0x1000</td>
<td>0x011c</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>GenericOnOff Client</td>
<td>0x1001</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Secondary element</strong></td>
<td>GenericOnOff Server</td>
<td>0x1000</td>
<td>0x011d</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GenericOnOff Client</td>
<td>0x1001</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Secondary element</strong></td>
<td>GenericOnOff Server</td>
<td>0x1000</td>
<td>0x011e</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>GenericOnOff Client</td>
<td>0x1001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With the information above, you can start the model configuration.
### 4.0 Model Configuration

**Configuration**

Now that provisioning is complete, it is time to perform model configuration. Type in the command below on main menu.

```
menu config
```

`meshctl` will go to the `menu config` submenu as shown in the image below.

The menu config screenshot above shows you how it supports the operations of Bluetooth mesh configuration and management like NetKey, AppKey, subscribe, publish, etc. Next, you will learn how to use some of these operations.

```
target 011b     # set primary element, 0x011b, to config
appkey-add 1    # add AppKey
bind 0 1 1000   # bind AppKey with certain model in certain element
sub-add 011b c000 1000   # add subscribe group address for certain model in certain element
bind 0 1 1001   # bind AppKey 1 to button 2 on element 1 (unicast 0101)
pub-set 010f c000 1 0 1001  # publish button 2 to group address c000
```
The table below explains the commands that were used.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target 011b</td>
<td>target &lt;unicast&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;unicast&gt; -- unicast address</td>
</tr>
<tr>
<td></td>
<td>Set the target node to configure, 011b is the unicast address of the primary element.</td>
</tr>
<tr>
<td>appkey-add 1</td>
<td>appkey-add &lt;app_idx&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;app_idx&gt; -- AppKey index which points the key stored in prov_db.json, please refer to Edit prov_db.json.</td>
</tr>
<tr>
<td>bind 0 1 1000</td>
<td>bind &lt;ele_idx&gt; &lt;app_idx&gt; &lt;mod_id&gt; [cid]</td>
</tr>
<tr>
<td></td>
<td>&lt;ele_idx&gt; -- element index, please refer to section Composition</td>
</tr>
<tr>
<td></td>
<td>&lt;app_idx&gt; -- AppKey index which points the key stored in prov_db.json</td>
</tr>
<tr>
<td></td>
<td>&lt;mod_id&gt; -- model id, please refer to section Composition and</td>
</tr>
<tr>
<td></td>
<td>Mesh Model Specification v1.0</td>
</tr>
<tr>
<td></td>
<td>&lt;cid&gt; -- optional</td>
</tr>
<tr>
<td>sub-add 010d c000 1000</td>
<td>sub-add &lt;ele_addr&gt; &lt;sub_addr&gt; &lt;mod id&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;ele_addr&gt; -- element address a.k.a unicast address, please refer to section Composition</td>
</tr>
<tr>
<td></td>
<td>&lt;sub_addr&gt; -- group address to subscribe, range: 0xc000 ~ 0xffff</td>
</tr>
<tr>
<td></td>
<td>&lt;mod_id&gt; -- model id, please refer to section Composition and</td>
</tr>
<tr>
<td></td>
<td>Mesh Model Specification v1.0</td>
</tr>
<tr>
<td>pus-set 011b c000 1 0 0 1001</td>
<td>sub-add &lt;ele_addr&gt; &lt;sub_addr&gt; &lt;mod id&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;ele_addr&gt; -- element address a.k.a unicast address, please refer to section Composition</td>
</tr>
<tr>
<td></td>
<td>&lt;sub_addr&gt; -- group address to subscribe, range: 0xc000 ~ 0xffff</td>
</tr>
<tr>
<td></td>
<td>&lt;mod_id&gt; -- model id, please refer to section Composition and</td>
</tr>
<tr>
<td></td>
<td>Mesh Model Specification v1.0</td>
</tr>
</tbody>
</table>
5.0 Verification

This section shows you how to use the command below to verify that the GenericOnOffServer model and GenericOnOffClient model are working.

Please type below commands on meshctl.

```
back
menu onoff
target 011b
onoff 1  # turn LED1 on
onoff 0  # turn LED1 off
get  # get LED1's status
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>Back to main menu.</td>
</tr>
<tr>
<td>menu onoff</td>
<td>Go to “menu onoff” menu.</td>
</tr>
<tr>
<td>target 011b</td>
<td>target &lt;unicast&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;unicast&gt; -- unicast address</td>
</tr>
<tr>
<td></td>
<td>Set the target node to configure, 011b is the unicast address of the primary</td>
</tr>
<tr>
<td></td>
<td>element</td>
</tr>
<tr>
<td>onoff x</td>
<td>onoff &lt;0/1&gt; -- send “SET ON/OFF” command.</td>
</tr>
<tr>
<td>get</td>
<td>get – Get ON/OFF status.</td>
</tr>
</tbody>
</table>

Meanwhile, you also can use Button 1 on the board to control LED1 on or off: single click to turn.
6.0 Summary

After following these steps, you may find that there are 4 LEDs and 4 buttons on the board, as well as 4 elements which were assigned the unicast addresses from 0x011b to 0x011e, but here we just configure primary element, 0x011b. You should be able to configure the last three elements and models on your own.