BLE Devices on-boarding Flow

This document aims to introduce the general flow of setting up both BLE (**B**luetooth **L**ow **E**nergy) gateway and BLE nodes which are controlled by the gateway. The target audiences can be developers who are involving in BLE development, or those who are interested in the setup flow of BLE devices.

1. Set up and register BLE Gateways

Setting up a BLE Gateway device makes no difference than setting up a general Ayla device. Suppose we are going to set up a Raspberry Pi3 BLE gateway with AP mode registration type, the specific steps are:

1) Check the prerequisites that should be met before moving forward. For example, the gateway should at least be powered on.
2) Scan for target gateway. Initiate Wi-Fi scan for nearby gateways that are broadcasting their SSIDs, which typically start with "Ayla-".
3) Select the target gateway to be registered.
4) Connect mobile to the target gateway to exchange information that are needed to set up the gateway. Such as LAN IP, device features set, etc.
5) Initiate gateway scan for access points the gateway can "see" and to be joining in.
6) Select the target AP network the gateway to be connected to.
7) Enter the password of the target AP the gateway to be joining in.
8) Check the result of connecting the gateway to the selected AP with the provided credential.
9) Reconnect mobile to the original network it was in. By far, the mobile is in the gateway's network, it's required to exit from the gateway's network and reconnect to original network to make sure it has network connectivity.
10) Confirm that the gateway has connected to the internet and is online now.
11) Register the gateway to current user account.
12) Personalize gateway. Do some customizations on the registered gateway, such as more human-readable product name.

2. Set up and register BLE nodes

To set up and register a node device, it must not be paired with device other than the target gateway and can be discovered and paired with the gateway. That said, the target gateway must be registered beforehand. In addition, as the BLE node itself is incapability of connecting to the network, which means all traffics to and from the node must be done through the gateway.

The general steps of setting up a BLE node through the gateway are:

1) Mobile asks Gateway to scan nearby BLE devices.
2) Gateway returns all **supported** BLE devices in range.
3) Mobile represents the list of scan results to user.
4) User chooses which devices to be registered.
5) Mobile initiates pairing between gateway and BLE devices. For devices that require passkeys, passkeys should also be provided during the pairing process.
6) Gateway adds the paired nodes to cloud, which from now on are called registration candidates.

7) Mobile fetches the registration candidates from cloud and register them.

2.1 Device Filtering

In each scan-and-response session, gateway defaults to return only the supported BLE devices that are broadcasting services that can be recognized by the gateway.

The supported device types and the corresponding set of service UUIDs that are used to uniquely identify the devices are listed in below table.

Device Type	Device UUIDs
MagicBlue	[0000ffe5-0000-1000-8000-00805f9b34fb, 0000fff0-0000-1000-8000-00805f9b34fb, 0000ffe0-0000-1000-8000-00805f9b34fb]
Grillright	[2899fe00-c277-48a8-91cb-b29ab0f01ac4]
Ayla devices	[0000fe28-0000-1000-8000-00805f9b34fb]

On the mobile side, it should use the type information contained in the scan result to filter out the target type of devices the user intends to scan and register. For example, user wants to register bulbs however gateway scan returns a collection of bulbs and Grillrights, so mobile should discard the returned Grillrights and only represent bulbs to user so that the user won't get confused. Check the definition of bt_scan_results property for more details.

2.2 Communications between Mobile and Gateway

The communications between mobile and gateway are completed by exchanging property values. Writing value to a property to ask the gateway to do something and reading value from a property to check if something happened in the gateway. Note that, the writings and readings can happen in LAN mode or cloud mode, depending on the gateway and mobile are working in LAN mode or cloud mode.

2.2.1 Properties definitions

Definitions of the key properties that aid in the setup procedure are listed below.

1) **bt_scan_enable**
   Property name: *bt_scan_enable*
   Description: *initiate gateway to start scanning for BLE devices.*
   Base type: *integer*
   Direction: *to device*
   Value: *writing value greater than 0 to start scanning and writing value 0 to stop scanning. Value greater than 0 also implies a scanning timeout, gateway will automatically stop scanning after that period of time if it was not explicitly asked stop scanning.*

2) **bt_scan_results**
   Property name: *bt_scan_results*

Description: *Gateway scan results in a JSON array format. Each object in the array denotes a specific device and consists of following fields:*

- *name, descriptive name of the BLE device.*
- *bd_addr, MAC address of the device.*
- *rssi, signal strength of the device.*
- *type, gateway supported device types, with a maximum of 16 characters in length. The supported types are MagicBlue, Grillright and AylaPowered.*

Base type: *string*
Direction: *from device*
Value: *A JSON array of the discovered BLE devices. Empty array [] implies no devices were found, otherwise stores the list of discovered devices. Here is an example:*


```
[
  {
   "device": {
    "addr": "CF:66:24:27:E4:B8",
    "rssi": "-66",
    "name": "IDTQ133A",
    "type": "Grillright"
   }
  },
  {
   "device": {
    "addr": "F8:1D:78:60:2F:C1",
    "rssi": "-55",
    "name": "LEDBLE-78602FC1",
    "type": "MagicBlue"
   }
  },
  {
   "device": {
    "addr": "C5:5D:4E:EE:54:21",
    "rssi": "-69",
    "name": "Ayla",
    "type": "AylaPowered"
   }
  }
]
```


3) **bt_connect_id**
Property name: *bt_connect_id*
Description: *Enable pairing between the gateway and the device by writing the target device's unique hardware ID, usually MAC address. The address was returned in the bt_scan_results property.*
Base type: *string*
Direction: *to device*
Value: *BLE device address, such as CF:66:24:27:E4:B8*

4) **bt_connect_passkey_display**

Property name: *bt_connect_passkey_display*
Description: *The Bluetooth pairing code displayed on the gateway. For security considerations, the Bluetooth protocol requires the key on two Bluetooth exactly matches to make sure the pairing process can continue.*
Base type: *integer*
Direction: *from device*
Value: *Bluetooth pairing code, such as 928995.*

5) **bt_connect_passkey**
Property name: *bt_connect_passkey*
Description: *The Bluetooth pairing code displayed on the device, which should be exactly matches and sent to gateway to make sure the pairing process can continue.*
Base type: *integer*
Direction: *to device*
Value: *Bluetooth pairing code, such as 928995.*

6) **bt_connect_result**
Property name: *bt_connect_result*
Description: *Property to indicate the status of the node being registered. Before a node device is ready to be registered, it has to be first connected to the gateway, then added to the cloud to become a cloud candidate, and lastly finishes template association.*
Base type: *string*
Direction: *from device*
Value: *The connection result in JSON format, consisting of following fields:*

{
"bd_addr": string, Mac address of the BLE device node;
"status_code": Integer, status code;
"status_details": string, descriptive info about the status_code.
}

The supported status code and corresponding description:

status_code	status_details	Note
0	connected	device connected to gateway
1	*added*	device was added to cloud
2	*updated*	device template was updated in the cloud
-1	no device found	
-2	unknown error	

Here is an example:

{
"bd_addr": "CF:66:24:27:E4:B8",
"status_code": 0,
"status_details": "connected"
}

7) **bt_disconnect_id**

Property name: *bt_disconnect_id*

Description: *Enable unpairing between the gateway and the device by writing the target device's unique hardware ID, usually MAC address. The address was returned in the bt_scan_results property. If a device remains paired with the gateway, it won't be discovered by other devices. Writing the MAC address of the target device will break pairing between gateway and the device.*

Base type: *string*

Direction: *to device*

Value: *BLE device address, such as CF:66:24:27:E4:B8*

8) **bt_disconnect_result**

Property name: *bt_disconnect_result*

Description: *Result of the last request to disconnect pairing between gateway and the device, returned in JSON format.*

Base type: *string*

Direction: *from device*

Value: *The unpairing result in JSON format, can be disconnected, disconnecting, and error. The JSON result is defined as below:*

{
"bd_addr": string, Mac address of the BLE device node;
"status_code": Integer, status code;
"status_details": string, descriptive info about the status_code.
}

The supported status code and corresponding description:

status_code	status_details
0	disconnected
1	disconnecting
-1	no device found
-2	unknown error

Here is an example:

{
"bd_addr": "CF:66:24:27:E4:B8",
"status_code": 0,
"status_details": "disconnected"
}

9) **num_nodes**

Property name: *num_nodes*

Description: *The number of BLE devices associated with the gateway. It's the sum of registered nodes and candidate nodes that are still not registered but paired.*

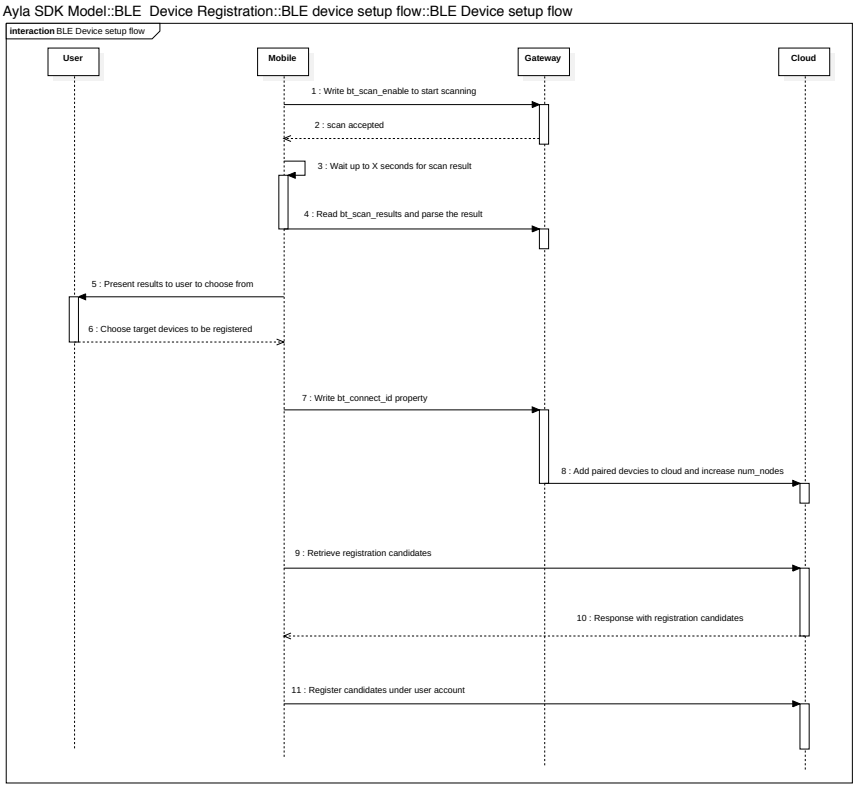
Base type: *integer*

Direction: *from device*

Value: integer value

3. Detailed BLE node setup flow based on properties communications

The property-based nodes setup flow is depicted as follows:



1. Mobile writes bt_scan_enable with value greater than 0 to start scanning device.
2. Gateway accepts the scan request by sending ACK message to bt_scan_enable property.
3. Mobile waits X seconds for scan result. The scan results were reported in a specified interval, 5 seconds e.g., and gateway doesn't guarantee the first report contains all the devices expected to be discovered. So, it's necessary to wait for a reasonable period of time for the expected results.
4. Mobile reads *bt_scan_results* property and parses it for scanned BLE devices.
5. Mobile represents the filtered results to user to determine which devices to be registered. Filtering based on the device type is necessary, otherwise user will see unexpected but supported devices.
6. User chooses the target devices to be set up.
7. Mobile writes *bt_connect_id* property with the chosen devices' address to start pairing between BLE gateway and the device.
8. Gateway adds the successfully paired nodes to cloud. As a result, *num_nodes* property will get increased by the number of added nodes.
9. Mobile reads *num_nodes* property to check if new registration candidates are available.
10. Mobile retrieves the registration candidates from Cloud.

11. Cloud responses with registration candidates.
12. Mobile registers candidates under user account.

Note: If registration failed for some reasons, for example network errors, the device may end up being a registration candidate and remains connected to the gateway, which means it won't be discovered by other devices and becomes unusable. Mobile should write bt_disconnect_id with the corresponding MAC address to disconnect the node from gateway. One reasonable solution for this is to retrieve and represent the cloud candidates, together with local scan results, to user to determine which devices to be registered, and automatically disconnect those cloud candidates that user didn't choose to register by writing bt_disconnect_id property.