PROJECT 1 – BIKE IOT – WAHOO DEVICES INTERFACING

Contents

1. Background

The Raspberry Pi communicates with the bike through Bluetooth Low Energy (BLE), specifically by pairing with the KICKR smart trainer (the other devices automatically pair with the KICKR smart trainer). The KICKR smart trainer runs an ATT server on it which the Raspberry Pi can then read and write values from/to. This document covers how the above works regarding the bike.

1.1. Bluetooth Low Energy (BLE)

The type of Bluetooth used is BLE instead of classic Bluetooth. The differences between classic and BLE Bluetooth does not really matter beyond researching future changes and additions. It should be noted the GATT protocol only applies to BLE Bluetooth.

1.2. Generic Attribute Profile (GATT)

GATT is a BLE protocol and an extension to the base Attribute (ATT) protocol. It uses a client-server relationship, where only a single client may be connected to the server at any time. Values are stored in a lookup table on the server side, which the client may request to read or write to depending on the value's properties. These values are called characteristics and are covered in more detail in another part of this document. GATT establishes a large set of generic services and characteristics for common devices and uses of ATT, meaning that we can expect devices built for GATT to behave in certain ways.

1.3. Services & Characteristics

<table>
<tr><td colspan="6">Profile</td></tr>
<tr><td colspan="3">Service</td><td colspan="3">Service</td></tr>
<tr><td>Characteristic</td><td>Characteristic</td><td>Characteristic</td><td>Characteristic</td><td>Characteristic</td><td>Characteristic</td></tr>
</table>

Profiles are standardised generic collections of services and characteristics. They are standardised by the Bluetooth standardisation organisation - Bluetooth SIG - and cover a wide range of use cases including fitness machines.

Services are groupings of characteristics under a common context. A service has a 16-bit or 128-bit UUID which identifies it and its characteristics.

Characteristics are the values which can be read or written to and have their own 16-bit or 128-bit UUIDs. They can also have descriptors which provide more context of the characteristic's use.

In addition to the generic services and characteristics, manufacturers may also include custom services and characteristics. These custom ones use the 128-bit UUIDs whereas the generic ones use the 16-bit UUIDs.

1.4. Indications & Notifications

As GATT is a slave-master relationship (where the client is master) response and acknowledgement from the server must be explicitly authorised by the client. So, if we want to be updated when a characteristic on the server is updated, we must enable notification on that

characteristic. Likewise, to receive acknowledgements of success or error on a write, we need to enable indications for that characteristic. The library we use seems to automatically enable indications but this needs to be further investigated.

~~1.5. Standardised Services & Characteristics~~

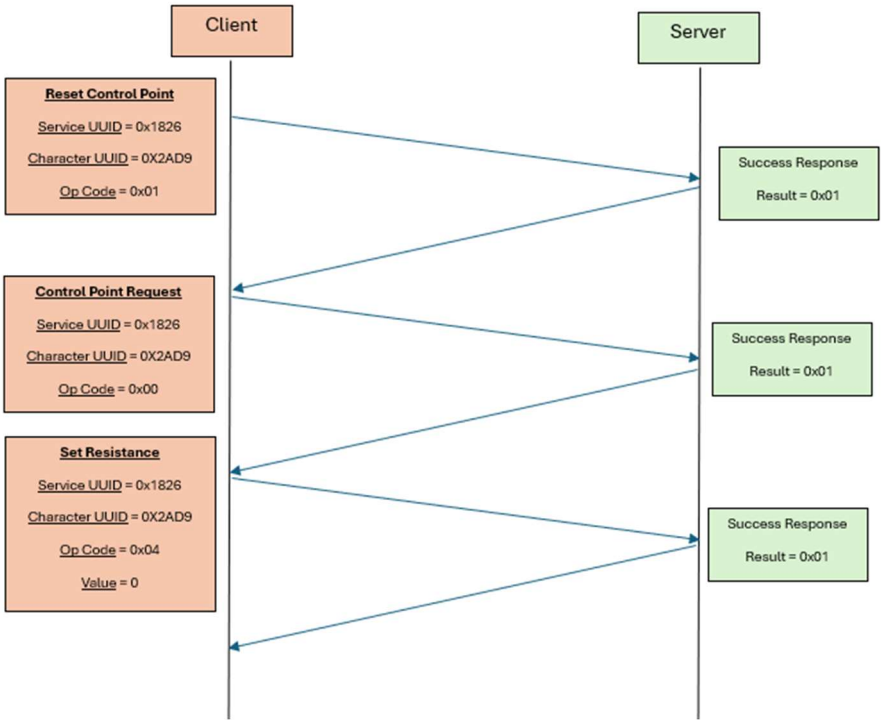
~~UUIDs can be 16-bit or 128-bit depending on if they are standardised service or characteristics or custom ones. GATT is an official protocol and as such it has many established services and characteristics intended to standardise the implementations of its protocol on specific devices. Services and characteristics from this standardisation use 16-bit UUIDs. Likewise, the manufacturer may need to implement custom services and characteristics, and as such use the custom 128-bit UUIDs.~~

~~As for finding the relevant UUIDs, standardised UUIDs for FTMS can be found here and custom ones must be found using the manufacturers documentation (if such a document exists) or can *maybe* be found using a BLE scanner (as was presumably done in this implementation). Last second find of this document which may detail how to discover custom services here.~~

1.6. Fitness Machine Service (FTMS)

The FTMS generic profile deals specifically with fitness machines like the Wahoo devices and the details of their official generic profile can be found here. As these machines may also need to be controlled by a client to execute some sort of functions, for instance increasing the incline on the climber, a control point characteristic is exposed for this purpose.

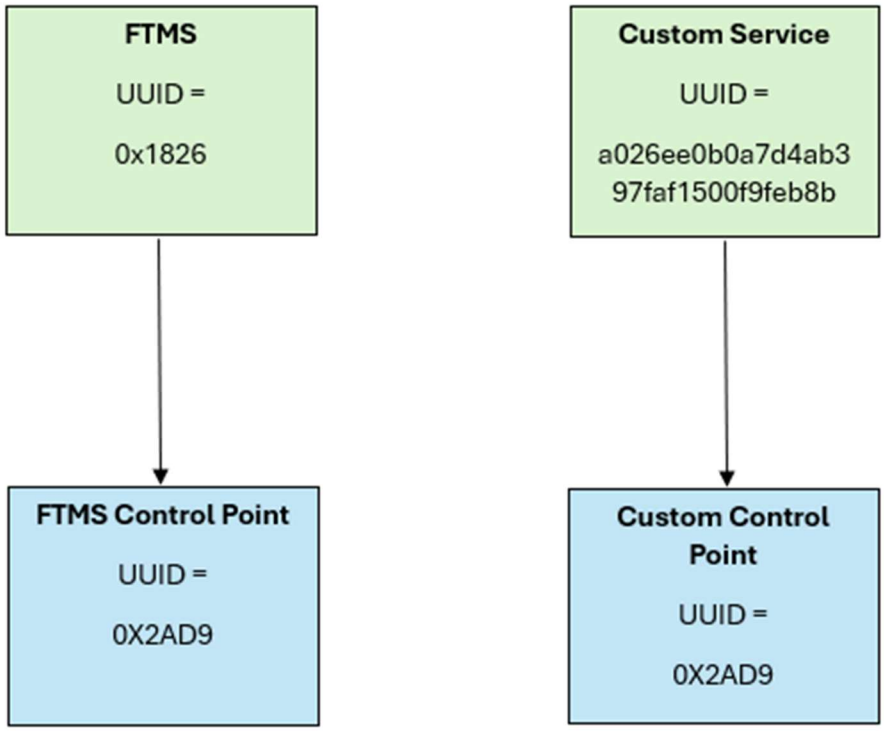
1.7. Control Point



The FTMS control point characteristic enables control over the FTMS functions. Functions have unique operation codes (Op Codes) which are used to request they be executed. To send commands to the control point, the client must first subscribe to indications for the control point and then request control over the control point by using the Request Control Op Code 0x00. If control is given, then other functions can be executed by sending their Op Code and parameter values to the control point. Control over the control point can also be reset by sending the Op Code 0x01.

2. Relevant Services & Characteristics

2.1. Climber Device



Relevant Services:

Service	UUID	Description
FTMS	0x1826	Standard
Custom Incline Control Service	a026ee0b0a7d4ab397faf1500f9feb8b	Service which holds the Custom Control Point Characteristic

Relevant Characteristics:

Characteristic	UUID	Description
FTMS Control Point	0X2AD9	Standard
Custom Control Point	a026e0370a7d4ab397faf1500f9feb8b	Enable control of the incline value

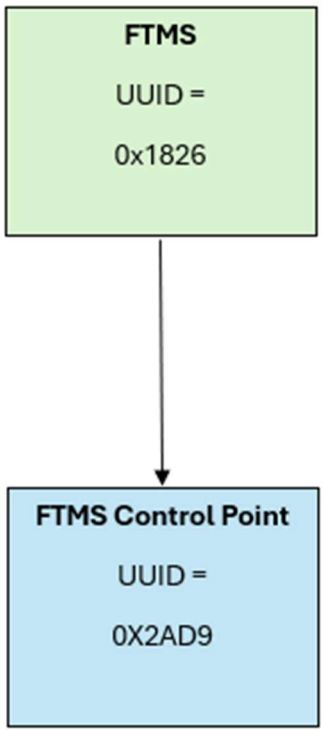
Relevant Functions:

Function	Op Code	Description
Request Control	0x00	Request control of FTMS control point
Reset Control	0x01	Reset control of FTMS control point
Request Custom Control	0x67	Request control of custom control point
Incline Function	0x66	Change the physical incline to the new value

2.2. Indoor Bike Device

The resistance and data aspects use the same devices and therefore the same FTMS service. So in reality the two Service-Characteristics maps should be combined with both characteristics coming off the one service, but for testing purposes these can be treated as separate devices.

2.2.1. Resistant Specific



Relevant Services:

Service	UUID	Description
FTMS	0x1826	Standard FTMS Service

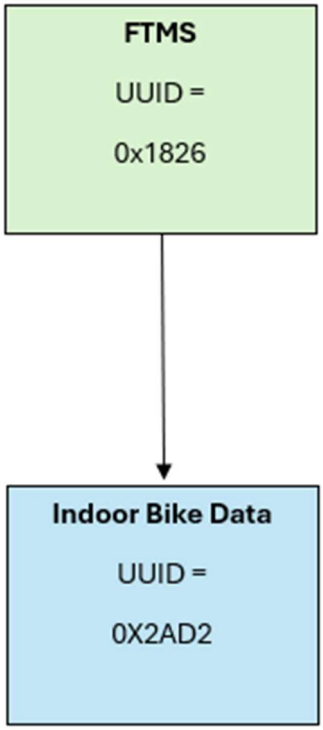
Relevant Characteristics:

Characteristic	UUID	Description
FTMS Control Point	0X2AD9	Standard

Relevant Functions:

Function	Op Code	Description
Request Control	0x00	Request control of FTMS control point
Reset Control	0x01	Reset control of FTMS control point
Set Resistance Level	0x04	Changes the physical resistance to the new value

2.2.2. Bike Data Specific



Relevant Services:

Service	UUID	Description
FTMS	0x1826	Standard FTMS Service

Relevant Characteristics:

Characteristic	UUID	Description
Indoor Bike Data	0X2AD2	More info below

The indoor bike data characteristic returns a bit flag indicating what data it has returned and a payload of bytes holding the corresponding data. To find out more see section 4.9 of the GATT and FTMS document.

2.3. Other Relevant Characteristics & Op Codes

For acquiring the range of values which can be written to set the resistance and incline, resistance range characteristic and incline range characteristic exist to get these values. As they were only required to get once they have been omitted from most of this document.

3. Implementation

3.1. Dependant Libraries

Library	Use	Required
gattctl	Handle GATT protocol	Yes
bluez	Requirement of `gattctl`	Yes
systemctl	Requirement of `gattctl`	Yes
bluetoothctl	Requirement of `gattctl`	Yes
python3-dbus	-	Not sure
paho-mqtt	For MQTT services	Yes for MQTT

3.2. MQTT Client

`*lib/mqtt_client.py*`

This file contains several methods for handling MQTT connecting, publishing, subscribing, and logging.

3.3. BLE Helper

`*lib/ble_helper.py*`

This file contains several helper methods for identifying services & characteristics UUIDs, reading/writing HEX codes, reading long strings of bytes, and converting an incline value into an Op Code value.

Method	Parameters	Description
service_or_characteristic_found	target_uuid, full_uuid	Assume the first four bytes of the UUID are 0 and return True if the target_uuid is in the full_uuid.
service_or_characteristic_found_full_match	target_uuid, full_uuid	Return True if the target_uuid is in the full_uuid.
decode_int_bytes	value	Convert byte `value` into a list of integers.
decode_string_bytes	value	Convert byte `value` into a string.
covert_negative_value_to_valid_bytes	negative_int	Convert integer `negative_int` into signed byte.
convert_incline_to_op_value	incline	Return a pair of incline value op codes equivalent to the passed incline value op code.
covert_hex_values_to_readable_string	array	Convert an array of HEX values into text.

*Not sure how the incline value works exactly. Why does it return two values?

3.4. BLE Device Scanner

`ble_devices_scan.py`

Detects BLE devices and displays their MAC Addresses and other information.

3.5. Wahoo Class

`wahoo_device.py`

The core program for connecting and using the Wahoo devices (Bike, Climber, and Head Wind devices). It is a catchall implementation, meaning it can connect to all Wahoo devices, though currently only the Bike and Climber devices are implemented.

It has methods specifically for handling the climber (incline), collecting bike data, resistance, MQTT, and helper methods for the GATT communication – all covered in more detail below.

3.5.1. Incline methods

Method	Description
read_inclination_range	Used to log the range of values which can be set for the incline. Not needed for usual operation.
custom_control_point_enable_notifications	Enable notifications for the custom control point characteristic on the Wahoo Climber.
custom_control_point_set_target_inclination	Write a new incline value to the custom control point characteristic.

3.5.2. Resistance methods

Method	Description
read_resistance_level_range	Used to log the range of values which can be set for the resistance. Not needed for usual operation.
ftms_set_target_resistance_level	Used to set the resistance using the FTMS control point.

3.5.3. Bike data methods

Method	Description
process_indoor_bike_data	Reads the speed, cadence and power values and publishes them. Can also read various other values if available.
characteristic_value_updated	Triggers process_indoor_bike_data when notified about a characteristic update.

3.5.4. MQTT methods

Method	Description
setup_mqtt_connection	Connect to the MQTT server using the supplied credentials.
set_new_inclination	Publishes a new incline value and logs it.
set_new_resistance	Publishes a new resistance value and logs it.
process_indoor_bike_data	Handles the publishing of some values.
mqtt_data_report_payload	-

3.5.5. General GATT methods

Method	Description
services_resolved	The main function looped by the GATT parent. Searches for services & characteristics as well as initialises the control point control requests.
ftms_request_control	Used to request control of the FTMS control point.
ftms_reset_settings	Used to reset control of the FTMS control point and reset resistance and incline values.
set_service_or_characteristic	Identifies and stores services and characteristics.
connect_succeeded	Helper method handled by the GATT parent to log a successful connection.
connect_failed	Helper method handled by the GATT parent to log an error while connecting.
disconnect_succeeded	Helper method handled by the GATT parent to log a successful disconnect.
set_new_inclination_failed	Logs that the inclination write request failed.
set_new_resistance_failed	Logs that the resistance write request failed.
descriptor_read_value_failed	Helper method handled by the GATT parent to log an error while reading a descriptor of a service or characteristic.
characteristic_value_updated	Helper method handled by the GATT parent to log a successful characteristic write request.
characteristic_write_value_failed	-
characteristic_enable_notification_succeeded	Helper method handled by the GATT parent to log successfully enabling notification on a characteristic.
characteristic_enable_notification_failed	-

4. Alternative OOP Implementation

A more stripped-down implementation of `wahoo_device.py` based on the original implementation and having a more OOP based design is also available. This method has all MQTT methods removed as well as most GATT helper logging methods. This implementation is more for testing specific devices and better understanding the original catch all implementations.

It is currently untested and would likely be a good idea to reimplement the GATT logging methods.

4.1. Wahoo Device Class

This class is a parent class meant to be the basis of all Wahoo device implementations. It handles initialising all the GATT connections and requests making the individual device classes more readable and focused on their specific functions.

4.2. Incline Device Class

The Incline Device Class is intended to handle all Wahoo Climber connection and requests. The MAC address is hardcoded, though it may be beneficial to make it passable. It handles both connecting to the custom control point and writing new incline values.

4.3. Bike Device Class

As the bike data and resistance level are part of the same device, these different functions share the same class. With two functional methods (process_indoor_bike_data and ftms_set_target_resistance_level) and two helper methods (set_service_or_characteristic and ftms_reset_settings) this implementation is clearer about what methods are relevant to which devices and functions.

5. Future

5.1. Wind Device

The Wahoo Head Wind devices is the next natural addition to this set of programs. Research and implementation of the head wind device should be a focus for future teams.

5.2. MQTT replaced with BLE or Serialisation

The MQTT service is useful during the prototyping stage but will need to be replaced in the near future due to high latency. As such, whether it is replaced with more focus on BLE or replaced with serialisation, these changes will need to be implemented, at least in part, in these files.

6. Relevant Material/Documents

GATT and FTMS

https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=423422

GATT service discovery

https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf?v=1716040219399