Documenting the LEGO® PoweredUp! System

Christian Treczoks, MBFR – LEGO® Modellbaufans Rheinland e.V.

Version 0.3, December 16th, 2018

Table of Contents

Introduction

This is about documenting the communication from and to the PoweredUp! Handset and the PoweredUp! Hub No.4/Smart Hub. The devices I‘ve currently got are the train motor and the LEDs.

A BOOST color sensor has been ordered and will hopefully arrive soon, and I’m looking for a source to get the “Batmobile Motor” (BL 21980/LEGO 6127110 or 6215354) without paying a fortune for license stuff I don’t need...

This is not about how those devices communicate with each other. The intention is to find how a software running on a PC, SBC, or SOC can communicate with handsets and hubs in order to control a larger train layout in the future.

I have to admit that I am a newbie to BlueTooth Low Energy (BTLE) stack and might still mix up some standard and communication stack related things. Don’t hesitate to correct me!

The chapters start very slowly, listing every move I did to discover things, so anyone can reproduce them and build on top of them.

All this would not have been possible without the information I got from the BOOSTreveng[i] project by Jorge Pereira and his blog page on the Handset[ii]. His works on the Boost and the PoweredUp! system were the keys to understanding the PoweredUp! system.

Big thanks also to the people behind gatttool, which I use on my Ubuntu system to find the inner workings of these devices.

And, last not least, to the PoweredUp! Team at LEGO who developed and built these devices. I seriously hope to meet you one day, and get answers to all the open points from you!

Changelog

From Version 0.2 to Version 0.3

Added tables on supply voltage and readings, and determined the dividers, threshold levels of low-power warnings (blinking LED) and minimum operating voltage for both the Handset and the Smart Hub.

Determined that on the HandSet port 3C readings are neither drawn current nor BT signal quality. True meaning is still unknown.

Added Color and Distance Sensor to the list of recognized devices for the Smart Hub, but did not elaborate further due to inconclusive readings.

The Handset

Finding and Connecting to the Handset

My Handset listens to the device ID 00:81:F9:E4:B2:0D.

There are several ways to find the ID of the devices, one of them is to use the "`hcitool`". It has to be run as superuser, so one either has to use "`sudo`" or a root shell for this. Press the green button on the Handset, enter "`hcitool lescan`" (or "`sudo hcitool lescan`" and enter the root password if requested), and it will reply with "`LE Scan…`" and will list (maybe among other devices) a line like "`00:81:F9:E4:B2:0D Handset`" (well, with the device ID of your Handset, of course…).

To connect with the device from the Linux shell I use the command "`gatttool -b 00:81:F9:E4:B2:0D -I`" which gives me the prompt "`[00:81:F9:E4:B2:0D][LE]>`", at which press the green button on the Handset and enter "`connect`". The gatttool replies to this with "`Attempting to connect to 00:81:F9:E4:B2:0D`" followed by "`Connection successful`". The LED on the Handset changes from blinking white to a continuous white.

Services and Characteristics UUIDs

Finding Services

Once connected, you can ask for the devices service and characteristics UUIDs and work with its handles. For that you can do a Primary Service Discovery with the command "`primary`":

```
[00:81:F9:E4:B2:0D][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-
1000-8000-00805f9b34fb
attr handle: 0x0008, end grp handle: 0x0008 uuid: 00001801-0000-
1000-8000-00805f9b34fb
attr handle: 0x0009, end grp handle: 0xffff uuid: 00001623-1212-
efde-1623-785feabcd123
```

The first UUID line describes "Generic Access" according to the list of BTLE GATT Services[iii] for the handles 0x0001-0x0007, the second UUID line lists as "Generic Attribute" for handle 0x0008, and the last UUID line finally describes a "Vendor Defined Service" for all the other handles from 0x0009 to 0xffff.

Finding Characteristics

If you enter the command "`characteristics`" to do a Characteristics Discovery, you'll get a similar list:

```
[00:81:F9:E4:B2:0D][LE]> characteristics
handle: 0x0002, char properties: 0x02, char value handle: 0x0003,
uuid: 00002a00-0000-1000-8000-00805f9b34fb
```

```
handle: 0x0004, char properties: 0x02, char value handle: 0x0005,
uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007,
uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x000a, char properties: 0x1c, char value handle: 0x000b,
uuid: 00001624-1212-efde-1623-785feabcd123
```

The UUIDs describe (from top to bottom) a “Device Name” characteristic, an “Appearance”, and “Peripheral Preferred Connection Parameters”, according to the BTLE GATT Characterists[iv] list.

The last characteristic is a “Vendor Defined Characteristic”, with the vendor being LEGO.

Note the UUID differences here: the BTLE GATT Services have a “18xx”, the LEGO Services have “1623”, the BTLE GATT Characteristics have “24xx” and the LEGO Characteristics have “1624”.

What does all that mean?

Honestly, a lot of the BTLE standard things still leave a blank with me. At least one standard characteristic I figured out: The first standard characteristic is called “Device Name”, and if you read the associated “`char value handle`” 0x0003, something wonderful happens:

```
[00:81:F9:E4:B2:0D][LE]> char-read-hnd 0003
Characteristic value/descriptor: 48 61 6e 64 73 65 74
```

This does not look spectacular to you? Well, just translate the hexadecimal ASCII codes and you get the string “Handset” - the BTLE generic name of the device!

Handles

With the command “`char-read-hnd`” I read all the handles from 0x0001 to 0x000d like this (all the others just produce “`Invalid handle`” errors):

```
[00:81:F9:E4:B2:0D][LE]> char-read-hnd 0001
Characteristic value/descriptor: 00 18
```

In tabular form, we'll get:

Handle	Reply
0001	00 18
0002	02 03 00 00 2a
0003	48 61 6e 64 73 65 74 Device Name "Handset"
0004	02 05 00 01 2a
0005	00 00 Appearence: Unknown (Source: Jorge Pereira)
0006	02 07 00 04 2a
0007	50 00 a0 00 00 00 e8 03 Peripheral preferred Connection Parameters: "Connection Interval 100-200ms, Slave latency 0, Supervision timeout multiplier 1000" (Source: Jorge Pereira)
0008	01 18
0009	23 d1 bc ea 5f 78 23 16 de ef 12 12 23 16 00 00
000a	1c 0b 00 23 d1 bc ea 5f 78 23 16 de ef 12 12 24 16 00 00
000b	00 01 02 03 04 05 06 07
000c	00 00
000d	4c 57 50 20 50 72 6f 74 6f 63 6f 6c 43 68 61 72 Translates to "LWP ProtocolChar", whatever this may be.

So handles 0x0001 to 0x0008 are BTLEs own things with 0x0003 being the name, 0x0005 being a bit worthless, and 0x007 describing a bunch of timeout information.

Handles 0x0009 to 0x000d are LEGO related.

So far, the meaning of the handles 0x0009 and 0x000a have eluded me. Reading them repeatedly while pressing or releasing buttons have not shown any change so far.

Handle 0x000d produces the text "LWP ProtocolChar". No idea, maybe LWP is "LEGO Wireless Protocol"?

Jorge Pereira offered the clues how to deal with the remaining handles, and there it starts getting interesting…

Commands and Notifications

With a write to handle 0x000c we can enable notifications:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000c 0100
Characteristic value was written successfully
Notification handle = 0x000b value: 0f 00 04 00 01 37 00 00 00 00
10 00 00 00 10
Notification handle = 0x000b value: 0f 00 04 01 01 37 00 00 00 00
10 00 00 00 10
Notification handle = 0x000b value: 0f 00 04 34 01 17 00 00 00 00
```

```
10 00 00 00 10
Notification handle = 0x000b value: 0f 00 04 3b 01 14 00 02 00 00
00 02 00 00 00
Notification handle = 0x000b value: 0f 00 04 3c 01 38 00 00 00 00
10 00 00 00 10
```

This immediately gives us five messages via handle 0x000b. Jorge Pereia and a guy he calls Nathan (sorry, the link Jorge provided is dead) have figured the following out from this:

Byte Nr.	**Content**	**Description**
1	0f	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	04	Command code for “Notification”
4	00/01/34/3b/3c	Internal port number: 00 and 01: Button pads 34: RGB LED 3B and 3C: Power or battery readings?
5	01	Unknown, always 01
6	37/17/14/38	Device type on this internal port: 37: Button pads 17: RGB LED 14 and 38: Power or battery readings?
7-15	s.a.	Unknown.

Basic Notifications

With notifications enabled, we can already get some information without the need to subscribe anything.

Press green button:

```
Notification handle = 0x000b value: 05 00 08 02 01
```

Release green button:

```
Notification handle = 0x000b value: 05 00 08 02 00
Notification handle = 0x000b value: 04 00 08 03
```

Press and hold green button:

```
Notification handle = 0x000b value: 05 00 08 02 01
```

After about five seconds we get:

```
Notification handle = 0x000b value: 04 00 02 30
```

and the handset switches off.

Requested Notifications

To get more notifications, we need to send subscription commands to the handle 0x000b of the handset to enable them for the specific ports. The subscription command in detail:

Byte Nr.	Content	Description
1	0a	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	41	Command code for “Subscription”
4	00/01/3B/3C	Internal port number: 00: Left or “A” Button pad 01: Right or “B” Button pad 3B: Port related to power 3C: Port related to power
5-9	00 00 00 00 00	Unknown, always 00
10	00 or other value	Subscription active: 00: Subscription disabled 01..ff: Subscription enabled

The handset replies with an identical message except that the command code 41 for “Subscribe” is replaced by a code 47 for “Acknowledge Subscription”.

Notifications on Keys

Enable reporting on left or “A” keys on port 00:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000b 0a004100000000000001
Characteristic value was written successfully
Notification handle = 0x000b value: 0a 00 47 01 00 00 00 00 00 01
```

Enable reporting on right or “B” keys on port 01:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000b 0a004101000000000001
Characteristic value was written successfully
Notification handle = 0x000b value: 0a 00 47 01 00 00 00 00 00 01
```

The notification we get for pressed and released buttons is structured as follows:

Byte Nr.	Content	Description
1	05	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	45	Command code for “Key Notification”
4	00/01	Internal port number: 00: Left or “A” Button pad 01: Right or “B” Button pad
5	00/01/7F/FF	Key code: 00: Button Released 01: Button “+” pressed 7f: Button “Red” pressed ff: Button “-” pressed

Notifications on Battery Voltage

Enable notifications for port 3b:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000b 0a00413b000000000001
Characteristic value was written successfully
Notification handle = 0x000b value: 0a 00 47 3b 00 00 00 00 00 01
```

Now we get regular (about once per second) notifications about port 3b:

```
Notification handle = 0x000b value: 06 00 45 3b 50 0b
Notification handle = 0x000b value: 06 00 45 3b 50 0b
Notification handle = 0x000b value: 06 00 45 3b 4f 0b
Notification handle = 0x000b value: 06 00 45 3b 50 0b
Notification handle = 0x000b value: 06 00 45 3b 4f 0b
Notification handle = 0x000b value: 06 00 45 3b 4e 0b
Notification handle = 0x000b value: 06 00 45 3b 50 0b
Notification handle = 0x000b value: 06 00 45 3b 4f 0b
```

The notification we get for Port 3b is structured as follows:

Byte Nr.	Content	Description
1	06	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	45	Command code for “Key Notification”
4	3b	Internal port number: 3B: Port related to power
5-6	varies	My guess is that byte 5 is the low and byte 6 is the high byte of a 16-bit value. 0X0b50=2896. ~~Took my voltage-guess-o-meter and got 5.98V with that reading of 0x0b50, which would give a factor of 484.~~ See below, the factor here is 500.

New Research Results as of December 16th, 2018

I connected the Handset to our labs’ power supply and measured the readings from Port 3B:

Supply Voltage (V)	Low Reading	High Reading	Decimal Average	Average/Voltage
6.0	0x0ba5	0x0ba7	2982.0	497.00
5.9	0x0ba4	0x0ba5	2980.5	505.16
5.8	0x0b40	0x0b41	2880.5	496.64
5.7	0x0b16	0x0b17	2838.5	497.98
5.6	0x0ad2	0x0ad4	2771.0	494.82
5.5	0x0a99	0x0a9a	2713.5	493.36
5.4	0x0a6a	0x0a6b	2666.5	493.80
5.3	0x0a4e	0x0a4f	2638.5	497.83
5.2	0x0a19	0x0a1a	2585.5	497.21

Supply Voltage (V)	Low Reading	High Reading	Decimal Average	Average/Voltage
5.1	0x09e7	0x09e8	2535.5	497.16
5.0	0x09b0	0x09b1	2480.5	496.10
4.9	0x098b	0x098c	2443.5	498.67
4.8	0x0947	0x0947	2375.0	494.79
4.7	0x092a	0x092b	2346.5	499.26
4.6	0x08f4	0x08f5	2292.5	498.37
4.5	0x08b5	0x08b6	2229.5	495.44
4.4	0x0889	0x088a	2185.5	496.70
4.3	0x085c	0x085e	2141.0	497.91
4.2	0x082f	0x0830	2095.5	498.93
4.1	0x07f5	0x07f6	2037.5	496.95
4.0	0x07d2	0x07d3	2002.5	500.63
3.9	0x079c	0x079d	1948.5	499.62
3.8	0x0760	0x0761	1888.5	496.97
3.7	0x0738	0x0739	1848.5	499.59
3.6	0x0701	0x0701	1793.0	498.06
3.5	0x06de	0x06df	1758.5	502.43
3.4	0x069b	0x069c	1691.5	497.50
3.3	0x0669	0x0669	1641.0	497.27
3.2	0x0651	0x0652	1617.5	505.47
3.1	0x0601	0x0602	1537.5	495.97
3.0	0x05d3	0x05d3	1491.0	497.00
2.9	0x05a2	0x05a2	1442.0	497.24
2.8	0x0586	0x0587	1414.5	505.18
2.7	0x054b	0x054c	1355.5	502.04
2.6	0x0517	0x0517	1303.0	501.16
2.5	0x04e1	0x04e1	1249.0	499.60
2.4	0x04b3	0x04b4	1203.5	501.46
2.3	0x0488	0x0488	1160.0	504.35
2.2	0x0461	0x0462	1121.5	509.77
2,1	0x041d	0x041d	1053.0	501.43

The Handset LED starts blinking at ~3.8V, probably at reading 0x780, and returns to normal operation at ~4.0V
The Handset turns off at ~2.0V, probably at reading 0x400.
The average of column 5 is 498.92, so I assume the intended divider here is 500.

I have to admit I am impressed. I did not expect the Handset to survive below maybe 3V. Chapeau!

Notifications on Unknown Data from Port C3

Enable notification on port 3c:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000b 0a00413c000000000001
Characteristic value was written successfully
Notification handle = 0x000b value: 0a 00 47 3c 00 00 00 00 00 01
```

Now we get regular (about once per second) notifications about port 3c:

```
Notification handle = 0x000b value: 05 00 45 3c c6
Notification handle = 0x000b value: 05 00 45 3c c9
Notification handle = 0x000b value: 05 00 45 3c c9
Notification handle = 0x000b value: 05 00 45 3c c9
Notification handle = 0x000b value: 05 00 45 3c c9
Notification handle = 0x000b value: 05 00 45 3c ca
Notification handle = 0x000b value: 05 00 45 3c c7
Notification handle = 0x000b value: 05 00 45 3c c6
Notification handle = 0x000b value: 05 00 45 3c c6
```

The notification we get for Port 3c is structured as follows:

Byte Nr.	**Content**	**Description**
1	06	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	45	Command code for “Key Notification”
4	3C	Internal port number: 3C: Unidentified Port
5	BD..CA	Unknown

New Research Results as of December 16th, 2018

Using the power supply on the Handset, I got the following voltage to current relations:

Supply Voltage	**Current drawn (Approximately)**
5.5V to 6.0V	> 80mA
4.9V to 5.5V	> 70mA
4.2V to 4.9V	> 60mA
3.8V to 4.2V	> 50mA
2.0V to 3.8V	Varies due to blinking of the LED, readings in the range of 20-40mA

The readings of Port 3C are all over the place, in the range of 0xBD to 0xD2, without any discernible connection to the voltage, or the drawn current. I switched the Handset on and off a few times during the tests, and the results are strange, to say the least: Sometimes the value read from port 3C is in the 0xD0 to 0xD2 range, disconnect, wait to shut down, switch it on, connect, and get a reading in the 0xC9 to 0xCB range, without any change to the power supply settings

I have to admit that my readings are coarse (the power supply only gave me steps of 10mA here), but nonetheless the readings from port 3C are not related to the current drawn from the battery.

Moving around the Handset in relation to the PCs BT antenna also seems to have no influence on the ports readings.

It still might be a current reading, internally, behind the voltage regulator, or maybe an on-chip temperature reading.

Controlling the RGB LED

To set the LED's color, notifications must be enabled! If you change the color without notifications being enabled, the LED will stay white until after enabling them, and then change to the last-set color.

To set the LED color (in this case to color 1, pink), a value for port 0x34 must be set:

```
[00:81:F9:E4:B2:0D][LE]> char-write-req 000b 0800813400510001
Characteristic value was written successfully
```

The command to set the LED is structured as follows:

Byte Nr.	Content	Description
1	08	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	81	Command code for "Set Port"
4	34	34: LED Port
5	00 or other value	Want a notification? 00: No notification necessary. other value: Notification will be sent.
6	51	Unknown, must be 51 or gives an error (`05 00 05 81 05`).
7	00	Unknown, must be 00 or does not work.
8	00..0A	Color of the LED 00: Off 01: Pink 02: Violet 03: Blue 04: Light Blue 05: Light Green 06: Green 07: Ochre 08: Orange 09: Red 0A: White

If requested, the handset replies with a notification:

Byte Nr.	Content	Description
1	05	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	82	Command code for “Port Set”
4	34	34: LED Port
5	0a	Always 0A – maximum value for port?

If byte 6 is not 51, it replies with an error:

Byte Nr.	Content	Description
1	05	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	05	Command code for “Error Message”
4	81	81: Offending command
5	05	05: internal error code?

Open questions on the Handset

1. What is the meaning of the data delivered by handles 0x0009 and 0x000a?

0009	23 d1 bc ea 5f 78 23 16 de ef 12 12 23 16 00 00
000a	1c 0b 00 23 d1 bc ea 5f 78 23 16 de ef 12 12 24 16 00 00

2. What does “LWP ProtocolChar”, delivered by reading handle 0x000d, mean?

000d	4c 57 50 20 50 72 6f 74 6f 63 6f 6c 43 68 61 72 Translates to “LWP ProtocolChar”, whatever this may be.

3. When enabling notifications by writing 0100 to handle 0x000c, it dumps information on the ports 00, 01, 34, 3b, and 3c. What is the meaning of the data in bytes 5 and 7-15 of the notification messages?

4. In general, what is the 00 in the second byte of a message or notification? Is this a high-byte of the message-length in byte 1?

5. Why does releasing the green button deliver two notifications: `05 00 08 02 00` and `04 00 08 03`?

6. Requesting notification (Command 41): What is up with the bytes 5-9, which are always 00?

~~7. Notifications about port 3b: What is the meaning of bytes 5-6? Low-byte, high-byte, and then divide by 484 (approximately) to get the battery voltage?~~

~~7.1. If the value from port 5-6 is indeed the battery voltage, at which point does it get critical for the operation of the handset?~~

8. Notifications about port 3c: What is the meaning of byte5?

9. Is there a way to read ports 3b and 3c once instead of once per second? Or is there a way to lower the notification frequency for those ports?

10. What is the meaning of bytes 6 and 7 when setting the LED color (command code 81, port 34)? Why is this 51 so important?

11. Why do notifications have to be enabled to set the LED color?

The Hub

Finding and Connecting to the Hub

My Hub listens to the device ID 90:84:2B:05:B1:C6. A "`hcitool lescan`" will give me "`90:84:2B:05:B1:C6 HUB NO.4`". No, it is not the fourth Hub I connected, the NO.4 is part of the Hubs name, probably a kind of revision number.

To connect with the device from the Linux shell I use the command "`gatttool -b 90:84:2B:05:B1:C6 -I`" which gives me the prompt "`[90:84:2B:05:B1:C6][LE]>`", at which press the green button on the Hub and enter "`connect`". The gatttool replies to this with "`Attempting to connect to 90:84:2B:05:B1:C6`" followed by "`Connection successful`". The LED on the Hub changes from blinking white to a continuous white.

Services and Characteristics UUIDs

Finding Services

Once connected, you can ask for the devices service and characteristics UUIDs and work with its Handles. For that you can do a Primary Service Discovery with the command "`primary`":

```
[90:84:2B:05:B1:C6][LE]> primary
attr handle: 0x0001, end grp handle: 0x0004 uuid: 00001801-0000-
1000-8000-00805f9b34fb
attr handle: 0x0005, end grp handle: 0x000b uuid: 00001800-0000-
1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001623-1212-
efde-1623-785feabcd123
```

The first UUID line describes "GenericAttribute" according to the list of BTLE GATT Services for the handles 0x0001 to 0x0004, the second UUID line lists as "Generic Access" for handle 0x0005 to 0x000b, and the last UUID line finally describes a "Vendor Defined Service" for all the other handles from 0x000c to 0x000f.

Finding Characteristics

If you enter the command "`characteristics`" to do a Characteristics Discovery, you'll get a similar list:

```
[90:84:2B:05:B1:C6][LE]> characteristics
handle: 0x0002, char properties: 0x20, char value handle: 0x0003,
uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x4e, char value handle: 0x0007,
uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0008, char properties: 0x4e, char value handle: 0x0009,
```

```
uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x000a, char properties: 0x02, char value handle: 0x000b,
uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x1e, char value handle: 0x000e,
uuid: 00001624-1212-efde-1623-785feabcd123
```

The UUIDs describe (from top to bottom) a “Service Changed” characteristic, a “Device Name”, an “Appearance”, and “Peripheral Preferred Connection Parameters”, according to the BTLE GATT Characterists list.

The last characteristic is a “Vendor Defined Characteristic”, with the vendor being LEGO.

Note the UUID differences here: the BTLE GATT Services have a “18xx”, the LEGO Services hahe “1623”, the BTLE GATT Characteristics have “24xx” and the LEGO Characteritics have “1624”.

Handles

With the command “`char-read-hnd`” I read all the handles from 0x0001 to 0x000f like this (all the others just produce “`Invalid handle`” errors):

```
[90:84:2B:05:B1:C6][LE]> char-read-hnd 0001
Characteristic value/descriptor: 01 18
```

In tabular form, we’ll get:

Handle	**Reply**
0001	01 18
0002	Attribute can’t be read
0003	00 18
0004	00 18
0005	00 18
0006	Gatttool crashes!
0007	48 55 42 20 4e 4f 2e 34 Name of the device: “HUB NO.4” or, after update via PoweredUp! app: 53 6d 61 72 74 20 48 75 62 Name of the device: “Smart Hub”
0008	Gatttool crashes!
0009	00 18
000a	Attribute can’t be read
000b	00 18
000c	23 d1 bc ea 5f 78 23 16 de ef 12 12 23 16 00 00
000d	Gatttool crashes!
000e	05 00 04 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Handle	Reply
	or, after update via PoweredUp! app: 05 00 04 39 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000f	00

So handles 0x0001 to 0x000b are BTLEs own things with 0x0007 being the name.

Handles 0x000c to 0x000f are LEGO related, with 0x000e maybe containing a firmware version?

Somehow this feels bad, really bad. First of all, the gatttool crashes. While this is primarly the fault of the tool itself, I suspect that the Hub somehow delivers mangled data on those handles.

Another suspicious thing is that many handles deliver "`00 18`" as a reply, especially for services like "Service Changed" and "Appearance", where those values don't belong. This feels like it comes from a "default" branch of a switch statement.

Things didn't improve even after the firmware update.

Commands and Notifications

With a write to handle 0x000f we can enable notifications:

```
[90:84:2B:05:B1:C6][LE]> char-write-req 000f 0100
Characteristic value was written successfully
Notification handle = 0x000e value: 0f 00 04 32 01 17 00 00 00 00 10 00 00 00 10
Notification handle = 0x000e value: 0f 00 04 3b 01 15 00 02 00 00 00 02 00 00 00
Notification handle = 0x000e value: 0f 00 04 3c 01 14 00 02 00 00 00 02 00 00 00
```

OK, so we here have a port 32 with type 17 (RGB LED), a port 3B with type 15 (unknown), and a port 3c with type 14 (Battery voltage?).

Subscribing to Port 3b

Subscribing notifications to port 3b gives an endless list of notifications at high speed:

```
[90:84:2B:05:B1:C6][LE]> char-write-req 000e 0a00413b000000000001
Characteristic value was written successfully
Notification handle = 0x000e value: 06 00 45 3b a0 00
```

The only action I found that modifies the notification contents is pressing the green button on the Hub.

When pressing (and holding) the green button, the fifth byte changes first from a0 to a1, a2, and even a3. If you hold the green button long enough (>5sec), the Hub shuts down with the messages

```
Notification handle = 0x000e value: 04 00 02 30
Notification handle = 0x000e value: 06 00 45 3b a2 00
Notification handle = 0x000e value: 04 00 02 31
```

I suspect the strange values come from an analog input instead of a digital one for the green button.

Subscribing to Port 3c

Again, the subscription to the port gives a list of notifications at high speed:

```
[90:84:2B:05:B1:C6][LE]> char-write-req 000e 0a00413c000000000001
Characteristic value was written successfully
Notification handle = 0x000e value: 06 00 45 3c 50 0d
```

Analogue to the port type 14, we get probably 16-bit values. My voltage-guess-o-meter said 8.58V at a reading of 0x0d50, giving a divider of 390. Sadly, this is different from the divider of 848 I got from the Handset. Or they have an offset somewhere – this requires further investigation.

New Research Results as of December 16th, 2018

Using our labs power supply instead of the batteries gives me the following readings for port 3c in relation to supply voltage:

Supply Voltage (V)	Low Reading	High Reading	Decimal Average	Average/Voltage
9.0	0x0e2e	0x0e2f	3630.5	403.39
8.9	0x0e03	0x0e04	3587.5	403.09
8.8	0x0dd7	0x0dd8	3543.5	402.67
8.7	0x0dae	0x0dae	3502.0	402.53
8.6	0x0d7c	0x0d7d	3452.5	401.45
8.5	0x0d61	0x0d62	3425.5	403.00
8.4	0x0d30	0x0d30	3376.0	401.90
8.3	0x0d17	0x0d17	3351.0	403.73
8.2	0x0ceb	0x0cec	3307.5	403.35
8.1	0x0cbd	0x0cbd	3261.0	402.59
8.0	0x0c93	0x0c93	3219.0	402.38
7.9	0x0c6b	0x0c6b	3179.0	402.41
7.8	0x0c53	0x0c54	3155.5	404.55
7.7	0x0c1f	0x0c1f	3103.0	402.99
7.6	0x0bfd	0x0bfe	3069.5	403.88
7.5	0x0bca	0x0bca	3018.0	402.40
7.4	0x0bae	0x0bae	2990.0	404.05
7.3	0x0b75	0x0b75	2933.0	401.78
7.2	0x0b52	0x0b52	2898.0	402.50
7.1	0x0b1e	0x0b1e	2846.0	400.85
7.0	0x0afe	0x0afe	2814.0	402.00
6.9	0x0ad1	0x0ad1	2769.0	401.30
6.8	0x0aab	0x0aab	2731.0	401.62
6.7	0x0a79	0x0a79	2681.0	400.15

Supply Voltage (V)	Low Reading	High Reading	Decimal Average	Average/Voltage
6.6	0x0a63	0x0a63	2659.0	402.88
6.5	0x0a28	0x0a28	2600.0	400.00
6.4	0x0a05	0x0a05	2565.0	400.78
6.3	0x09ec	0x09ed	2540.5	403.25
6.2	0x09c0	0x09c0	2496.0	402.58
6.1	0x09a3	0x09a3	2467.0	404.43
6.0	0x0977	0x0978	2423.5	403.92
5.9	0x0948	0x0948	2376.0	402.71
5.8	0x0927	0x0928	2343.5	404.05
5.7	0x0904	0x0904	2308.0	404.91
5.6	0x08d1	0x08d2	2257.5	403.13
5.5	0x089c	0x089d	2204.5	400.82
5.4	0x0873	0x0873	2163.0	400.56
5.3	0x0848	0x0849	2120.5	400.09
5.2	0x0828	0x0829	2088.5	401.63
5.1	0x0810	0x0811	2064.5	404.80
5.0	0x07e7	0x07e7	2023.0	404.60
4.9	0x07c2	0x07c2	1986.0	405.31
4.8	0x079c	0x079d	1948.5	405.94
4.7	0x0771	0x0771	1905.0	405.32
4.6	0x0748	0x0749	1864.5	405.33
4.5	0x0725	0x0726	1829.5	406.56
4.4	0x06f5	0x06f5	1781.0	404.77
4.3	0x06d3	0x06d3	1747.0	406.28
4.2	0x06a0	0x06a0	1696.0	403.81
4.1	0x0675	0x0676	1653.5	403.29
4.0	0x0649	0x0649	1609.0	402.25
3.9	0x061c	0x061d	1564.5	401.15
3.8	0x05f5	0x05f6	1525.5	401.45
3.7	0x05c3	0x05c3	1475.0	398.65
3.6	0x059e	0x059e	1438.0	399.44
3.5	0x0577	0x0577	1399.0	399.71
3.4	0x0556	0x0556	1366.0	401.76
3.3	0x0531	0x0531	1329.0	402.73
3.2	0x04fd	0x04fd	1277.0	399.06

Supply Voltage (V)	Low Reading	High Reading	Decimal Average	Average/Voltage
3.1	0x04cf	0x04d0	1231.5	397.26
3.0	0x04aa	0x04aa	1194.0	398.00
2.9	0x048b	0x048b	1163.0	401.03
2.8	0x045e	0x045e	1118.0	399.29
2.7	0x0439	0x0439	1081.0	400.37
2.6	0x0422	0x0422	1058.0	406.92
2.5	0x03ed	0x03ed	1005.0	402.00
2.4	0x03c6	0x03c6	966.0	402.50
2.3	0x039f	0x039f	927.0	403.04
2.2	0x0370	0x0370	880.0	400.00
2.1	0x0352	0x0352	850.0	404.76
2.0	0x0330	0x0330	816.0	408.00
1.9	0x0301	0x0301	769.0	404.74
1.8	0x02f9	0x02f9	761.0	422.78

The Smart Hubs LED starts blinking at ~5.8V with a reading of ~0x0934 and stops again at ~6.0V with a reading of ~965.
It drops out at 1.8V, after giving a last reading of 0x02f9.
The fifth columns average is 402,87, so I guess they aimed at a factor of 400.

The relation of supply voltage and drawn current (without any devices attached) is as follows:

Supply Voltage	Current drawn (Approximately)
8.3V to 9.0V	>40mA
8.0V to 8.2V	>39mA
7.4V to 7.9V	>38mA
6.7V to 7.3V	>37mA
6.1V to 6.6V	>36mA
5,8V to 6.0V	>35mA
1.8V to 5.7V	Varies due to blinking LED

Using Port 32

As with the Handset, you cannot subscribe for the LED port. But you can set it. The commands are the same as those for the Handset, except that the port in byte 4 is 32 instead of 34. The same applies for the “Port Set” notification, if it was requested.

Again, the strange need for byte 6 of the command to be “51” is still there, but changing this does not raise an error notification – it just doesn’t work.

Adding and Removing Devices

Plugging a train motor into port A gives the following notification:

```
Notification handle = 0x000e value: 0f 00 04 00 01 02 00 00 00 00
00 00 00 00 00
```

Removing the motor from port a gives:

```
Notification handle = 0x000e value: 05 00 04 00 00
```

Plugging the motor into port B gives:

```
Notification handle = 0x000e value: 0f 00 04 01 01 02 00 00 00 00
00 00 00 00 00
```

And removing it from port B results in:

```
Notification handle = 0x000e value: 05 00 04 01 00
```

Let us look at the LEDs. Plugging them into port A:

```
Notification handle = 0x000e value: 0f 00 04 00 01 08 00 00 00 00
00 00 00 00 00
```

And removing them:

```
Notification handle = 0x000e value: 05 00 04 00 00
```

Plugging them into port B:

```
Notification handle = 0x000e value: 0f 00 04 01 01 08 00 00 00 00
00 00 00 00 00
```

And removing them again:

```
Notification handle = 0x000e value: 05 00 04 01 00
```

From this we can derive the following notification format for device insertation:

Byte Nr.	**Content**	**Description**
1	0f	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	04	Command code for “Notification”
4	00/01	Port: 00: Port A 01: Port B
5	02/08/25	Device type: 02: Train Motor 08: LEDs 25: RGB and distance sensor
6-15	00	All zero. Why?

And the following notification format for device removal:

Byte Nr.	Content	Description
1	05	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	04	Command code for “Notification”
4	00/01	Port: 00: Port A 01: Port B
5	00	Fixed 00

If the devices were already present at the startup of the connection, the initial enabling of requests would already list them:

```
[90:84:2B:05:B1:C6][LE]> char-write-req 0f 0100
Characteristic value was written successfully
Notification handle = 0x000e value: 0f 00 04 00 01 02 00 00 00 00
00 00 00 00 00
Notification handle = 0x000e value: 0f 00 04 01 01 08 00 00 00 00
00 00 00 00 00
Notification handle = 0x000e value: 0f 00 04 32 01 17 00 00 00 00
10 00 00 00 10
Notification handle = 0x000e value: 0f 00 04 3b 01 15 00 02 00 00
00 02 00 00 00
Notification handle = 0x000e value: 0f 00 04 3c 01 14 00 02 00 00
00 02 00 00 00
```

Controlling Motors and LEDs

Based on Jorges work[v] on the PoweredUp! Motor control and some additional research by me, the command to control train motors and LEDs is defined like this:

Byte Nr.	Content	Description
1	0a	Total length of this message
2	00	Unknown, always 00, maybe high byte of message length?
3	81	Command code for “Set Port”
4	00/01	Port: 00: Port A 01: Port B
5	00 or other value	Notification (same as with setting the RGB LED): 00: No notification. Other value: Sent notification.
6	60	Always 60, like the 51 for the RGB LEDs. If you change this, the command will not be executed.
7	00	Seems to be unused, any value will work
8	00..ff	Speed: 00: Stop. 01..7e: forward, with 01..08 having the motor just humming and 7e full speed. 7f: stop – maybe emergency stop by shorting the motor? 80..ff: reverse, with 80 full speed and f8..ff having the motor just humming. The LEDs react better, they give a decent light level even at 01 or ff.
9	00.ff	Time: 00: Run infinite 01..ff: Run Value x 0.01 seconds
10	00 or other value	Pendulum Mode: 00: Run infinite, or once if Time!=00 Other value (with Time!=00): Reverse direction after Time x 0.01 seconds

While one can subscribe for notifications for port 00 or 01, neither the Time or Pendulum Mode generate any notifications except for the “Port Set” notification, even if the notification flag is set on the command.

Time also works on LEDs, while Pendulum Mode makes no sense – at least not with an unmodified LED part.

New Research Results as of December 16th, 2018

With the labs power supply, I could measure the power usage of some of the devices, all at 9V:

Train motor, running in the air at full speed: 85mA

Train motor, stalled at full speed: 820mA

LEDs at “full speed” (0x7E): 30mA

LEDs at “half speed” (0x40): 28mA

Color and distance sensor, regardless of idle or measuring: 3mA

Open Questions on the Hub

1. Browsing through the handles leads to errors, crashes, and invalid results for advertised standard BTLE services. Even after the update! Why?

2. Is the green button on port 3b really connected to an analog input, or why does it give do strange readings?

~~3. What does the value in the notification for port 3c tell us? Some battery voltage like in the Handset, but with a different divider?~~

4. The speed of notifications for port 3b and 3c is way to high for real-worl applications. Can it be slowed down?

5. What other types of devices are supported, and how?

6. Why is there no notification of a port change with Time and/or Pendulum Mode settings? This would be nice for many applications.

7. Is there a way to find a Hubs firmware version? The hub I used for these tests is fresh out of the box. After the update via the PoweredUp! App, the device name and the readings from handle 0x000e changed. Does handle 0x000e contain the firmware version? Like 1.0.0 → 57.0.8?

8. What other changes came with the firmware update?

9. Is there a list of the speed levels given by the Handset or app (-10 to +10) and the matching raw motor speed settings (Message 81, “Set Port”, byte 8)?

i Reverse engineering the LEGO BOOST Hub, https://github.com/JorgePe/BOOSTreveng
ii LEGO Powered Up Remote Control, https://ofalcao.pt/blog/series/lego-powered-up-remote-control
iii BTLE GATT Services, https://www.bluetooth.com/specifications/gatt/services
iv BTLE GATT Characteristics, https://www.bluetooth.com/specifications/gatt/characteristics
v BOOSTreveng/PoweredUp.md, https://github.com/JorgePe/BOOSTreveng/blob/master/PoweredUp.md