Working with Sphero BOLT Using BLE

Overview

This document describes the basics of working with Sphero BOLT using BLE. The sections below outline the three major steps for connection and communication:

1. Getting the Latest Firmware
2. Connecting to BOLT
3. Creating and Sending Commands

Getting the Latest Firmware

Out of the box, BOLT requires a firmware upate for proper use. This is an automatic process in the Sphero apps.

1. Power the robot on by placing it in the charger
2. Open the Sphero EDU or Sphero Play app and connect to BOLT
3. If new firmware is required, it will be updated automatically
4. On completion of the update, disconnect the robot from the app.

Connecting to BOLT

Scanning/Connecting

When scanning for a BOLT, applications should filter on both advertising name and advertised BLE service.

- **Advertisting name:** All BOLTs advertise with the name `SB-XXXX`, where `XXXX` is a four-character, uppercase hex string.
- **Advertised service:** All BOLTs advertise the Sphero API service, with UUID `00010001-574F-4F20-5370-6865726F2121`.

In settings where many BOLTs may be active, Sphero also recommends filtering by RSSI threshold to locate the nearest robot.

Sending Data to BOLT

Once connected, applications may send commands to BOLT. The robot exposes a single BLE characteristic for sending/receiving data. For your current application, receiving data is not required.

```
API Service:            00010001-574F-4F20-5370-6865726F2121
    API Characteristic: 00010002-574F-4F20-5370-6865726F2121
```

For sending data, the API Characteristic supports Write Request and Write Command. In most cases, Write Command should be fine (and much faster). All command examples in this document can be sent as a single write transaction to the API Characteristic.

Waking the Robot

The final step in connection is to wake the robot by sending the following data to the API characteristic:

`8D 38 11 01 13 0D FF 96 D8`

The robot should light up. See the Power Modes section for more information.

Creating and Sending Commands

The wake command referenced above is part of the Sphero binary protocol (referred to as the "API"). Detailed protocol documentation is available online, but for simplicity a reduced explanation is provided here.

At a high level, commands have five parts:

1. Start of Packet (SOP) byte
2. Header
3. Payload
4. Checksum
5. End of Packet (EOP) byte

As an example, a complete packet to drive the robot at speed 141 with heading 90° would be:

`8D` `38 12 01 16 07 FF` `AB 05 00 5A 00` `B1` `D8`

This example will be discussed further in the following sections, which describe the process of creating and sending a packet.

Overview

Creating and sending a packet is a four step process:

1. Set the header and payload
2. Calculate the checksum
3. Encode and frame the packet
4. Transmit the packet to the robot

Header and Payload

The header section identifies a command and ensures it is routed to the correct processor. For all command examples in this document, the header is a fixed set of bytes that can be used without modification.

The payload section contains zero or more fields used to modify the command. All multi-byte fields are sent big-endian, with supported types including:

- Unsigned integer (8, 16, 32, and 64 bits)
- Signed integer (8, 16, 32, and 64 bits)
- Single precision float (32 bits, IEEE-754)
- Boolean (single byte, zero = false, non-zero = true)
- (and other types not used in this doc)

Example

As an example, the Drive command header is:

Header: `38 12 01 16 07 FF`

The Drive command payload has three fields:

- Speed (0 to 255) - uint8
- Heading (0 to 359°)- uint16
- Flags (0 = forward, 1 = reverse) - uint8

To drive at speed 141 (0x8D) with heading 90° (0x005A) and no flags set (0x00), the payload would be:

Payload: `8D 00 5A 00`

Together, the header and payload for this example would be:

Header & Payload: `38 12 01 16 07 FF` `8D 00 5A 00`

Checksum

The checksum byte is added to the header and payload sections to verify valid transmission. The checksum is summed over all bytes in the header and payload sections as follows.

1. Sum all bytes in the header and payload section
2. Take the lower 8 bits
3. Invert the bits

Example

Given the Drive command example above, the checksum would be calculated as follows (note that all values are expressed in hexadecimal):

1. Sum of bytes: `24E` = (`38 + 12 + 01 + 16 + 07 + FF + 8D + 00 + 5A + 00`)
2. Lower 8 bits: `4E` = (`24E & FF`)
3. Inverted: `B1` = (`4E ^ FF`)

Together, the header, payload, and checksum for this example would be:

Header, Payload & Checksum: `38 12 01 16 07 FF` `8D 00 5A 00` `B1`

Encoding and Framing

To ensure packet boundaries are correctly identified, packet data is encoded and framed. Two special bytes are used for framing, the Start of Packet (SOP) and the End of Packet (EOP). These special characters may *only* be used for packet framing, so the rest of the packet is encoded to remove any special characters. If the values of these characters must be sent within the packet boundaries, they are replaced with a two-byte sequence using an additional special character, the Escape (ESC). Replacement encodings are shown in the table below:

Character	Value to Send	Replacement Encoding
Start of Packet (SOP)	`8D`	`AB 05`
End of Packet (EOP)	`D8`	`AB 50`
Escape (ESC)	`AB`	`AB 23`

This framing/encoding is a two step process:

1. Encode the header, payload, and checksum
2. Enclose the packet in SOP and EOP bytes

Example

In the Drive command example above, note that the special SOP value `8D` is part of the payload for driving at speed 141 (`8D`):

Header, Payload & Checksum: `38 12 01 16 07 FF` `8D 00 5A 00` `B1`

To encode the packet, the special value `8D` must be replaced with `AB 05` as shown in the table above:

Encoded Header, Payload & Checksum: `38 12 01 16 07 FF` `AB 05 00 5A 00` `B1`

The packet is then enclosed in SOP and EOP bytes, producing the complete packet:

Full Packet: `8D` `38 12 01 16 07 FF` `AB 05 00 5A 00` `B1` `D8`

Transmitting a Packet

As described in the Sending Data to BOLT section above, packets are sent as a write transaction to the API Characteristic. All command examples in this document can be sent as a single write transaction.

Command Examples

Waking the Robot

- Header: `38 11 01 13 0D FF`
- Payload: (none)
- Example: `8D 38 11 01 13 0D FF` `96` `D8`

Powering Off the Robot

- Header: `38 11 01 13 00 FF`
- Payload: (none)
- Example: `8D 38 11 01 13 00 FF` `A3` `D8`

Putting the Robot in Soft Sleep

- Header: `38 11 01 13 01 FF`
- Payload: (none)
- Example: `8D 38 11 01 13 01 FF` `A2` `D8`

Driving

- Header: `38 12 01 16 07 FF`
- Payload:
  - Speed (0 to 255) - uint8
  - Heading (0 to 359°)- uint16
  - Flags (0 = forward, 1 = reverse) - uint8
- Examples:
  - Forward at speed 127, heading 0°: `8D 38 12 01 16 07 FF` `7F 00 00 00` `19` `D8`
  - Reverse at speed 127, heading 0°: `8D 38 12 01 16 07 FF` `7F 00 00 01` `18` `D8`
  - Stop with heading 0° (or turn to 0°): `8D 38 12 01 16 07 FF` `00 00 00 00` `98` `D8`
  - Stop with heading 90° (or turn to 90°): `8D 38 12 01 16 07 FF` `00 00 5A 00` `3E` `D8`
  - Stop with heading 270° (or turn to 270°): `8D 38 12 01 16 07 FF` `00 01 0E 00` `89` `D8`
- Examples demonstrating encoding:
  - Drive at speed 141 (`8D`), heading 0°: `8D 38 12 01 16 07 FF` `AB 05 00 00 00` `0B` `D8`
  - Drive at speed 216 (`D8`), heading 0°: `8D 38 12 01 16 07 FF` `AB 50 00 00 00` `C0` `D8`
  - Drive at speed 171 (`AB`), heading 0°: `8D 38 12 01 16 07 FF` `AB 23 00 00 00` `ED` `D8`
  - Drive at speed 11, heading 0° (checksum `8D`): `8D 38 12 01 16 07 FF` `0B 00 00 00` `AB` `05` `D8`

Setting LEDs

- Header: `38 12 01 1A 1C FF`
- Payload:
  - LED selection mask - uint8 (for BOLT, this is always `3F`)
  - Front LED red (0 to 255) - uint8
  - Front LED green (0 to 255) - uint8
  - Front LED blue (0 to 255) - uint8
  - Back LED red (0 to 255) - uint8
  - Back LED green (0 to 255) - uint8
  - Back LED blue (0 to 255) - uint8
- Examples:
  - Set LEDs to 100% red: `8D 38 12 01 1A 1C FF` `3F FF 00 00 FF 00 00` `42` `D8`
  - Set LEDs to 100% green: `8D 38 12 01 1A 1C FF` `3F 00 FF 00 00 FF 00` `42` `D8`
  - Set LEDs to 100% blue: `8D 38 12 01 1A 1C FF` `3F 00 00 FF 00 00 FF` `42` `D8`
  - Set LEDs to 100% cyan: `8D 38 12 01 1A 1C FF` `3F 00 FF FF 00 FF FF` `44` `D8`
  - Set LEDs to 100% magenta: `8D 38 12 01 1A 1C FF` `3F FF 00 FF FF 00 FF` `44` `D8`
  - Set LEDs to 100% yellow: `8D 38 12 01 1A 1C FF` `3F FF FF 00 FF FF 00` `44` `D8`
  - Set LEDs to 100% white: `8D 38 12 01 1A 1C FF` `3F FF FF FF FF FF FF` `46` `D8`
  - Set LEDs to black: `8D 38 12 01 1A 1C FF` `3F 00 00 00 00 00 00` `40` `D8`

Setting the LED Matrix to a Single Color

- Header: `38 12 01 1A 2F FF`
- Payload:
  - Red (0 to 255) - uint8
  - Green (0 to 255) - uint8
  - Blue (0 to 255) - uint8
- Examples:
  - Set LED matrix to 100% red: `8D 38 12 01 1A 2F FF` `FF 00 00` `6D` `D8`
  - Set LED matrix to 100% green: `8D 38 12 01 1A 2F FF` `00 FF 00` `6D` `D8`
  - Set LED matrix to 100% blue: `8D 38 12 01 1A 2F FF` `00 00 FF` `6D` `D8`

- Set LED matrix to 100% cyan: `8D 38 12 01 1A 2F FF 00 FF FF 6E D8`
- Set LED matrix to 100% magenta: `8D 38 12 01 1A 2F FF FF 00 FF 6E D8`
- Set LED matrix to 100% yellow: `8D 38 12 01 1A 2F FF FF FF 00 6E D8`
- Set LED matrix to 100% white: `8D 38 12 01 1A 2F FF FF FF FF 6F D8`
- Set LED matrix to black: `8D 38 12 01 1A 2F FF 00 00 00 6C D8`

Power Modes

BOLT has three distinct power modes:

Mode	BLE Connectable	Accepts Drive/LED Commands
Off	No	No
Awake	Yes	No
Soft Sleep	Yes	Yes

To use the robot for driving, etc., BOLT must be in the Awake mode. Applications can transition between these modes using the Power Off, Soft Sleep, and Wake commands as shown: