# The SBrick BLE Protocol

*Version 25*

This document describes the protocol used by SBrick and SBrick Plus. These devices communicate via Bluetooth 4.0 “Low Energy”. This is a description of the services and characteristics in the GATT database, the structure of data read from or written to these and used in advertisement data packets.

# The advertisement data

Advertisement data contains the full device name, and some manufacturer specific data.

Manufacturer specific data contains security, battery voltage, and potentially other pieces of information, and is available in the "advertisement data" and "scan response" packets (in responses to active scanning). Manufacturer specific data may be present in BOTH types of packets.

## Vengit Limited manufacturer specific data fields

SBrick uses manufacturer specific data to advertise product type, battery reading and other data that is not included in the Bluetooth specification.

Manufacturer specific data starts with a length octet describing the whole length of the field. After the length octet, the field type octet and company identifier octets for Vengit follows in little endian order. After these four bytes, the actual data payload follows.

The following table describes the structure of a binary string that contains manufacture specific data as used by VENGIT. Each column describe a part of the string. **The top row contains the description of each field, while the size of the fields are in the bottom row expressed in bits.** We’ll continue to use this format further in the document.

| LENGTH (N) | 0xFF | 0x98 | 0x01 | DATA PAYLOAD |
|---|---|---|---|---|
| 8 | 8 | 8 | 8 | (N-3) * 8 |

The table above describes a string that is N+1 bytes long. The first byte (8 bits) contains the length as an 8 bit unsigned integer. After the length byte the proper number of data bytes follow. The first three bytes are constant, and are 0xFF, 0x98, and 0x01 in order. This part is mandatory as per the Bluetooth specification. The rest of the string is an arbitrary data payload containing N-3 bytes ((N-1)*8 bits). An “N” anywhere in a length description means the value of the first, record length byte.

The DATA PAYLOAD also consists of records. Each record must begin with a single 8-bit unsigned integer length part, followed by the proper number of data bytes in that record. Every record must also contain a record type identifier octet.

## SBrick Data Records

This section describes the record types used in the advertisement data / manufacturer data records, and in the BLE notifications. Since the length byte is mandatory before every record, they are omitted from the description.

### 00 Product type

| 0x00 | Product ID | Hardware version, major | Hardware version, minor | Firmware version, major | Firmware version, minor |
|---|---|---|---|---|---|
| 8 | 8 | 8 | 8 | 8 | 8 |

Product IDs:

| | |
|---|---|
| 00 | SBrick and SBrick Plus |

Hardware versions:

| | |
|---|---|
| 4 | SBrick, first generation |
| 5 | SBrick, second generation |
| 11 | SBrick Plus, first generation |
| 12 | SBrick, third generation |
| 13 | SBrick Plus, second generation |

The examples below, and throughout this document *do* include the length byte. Byte strings are written between quote marks, each byte is written as a two-digit hexadecimal number.

“02 00 00” - Product SBrick

“06 00 00 04 00 04 01” - Product SBrick, HW 4.0, FW 4.1

### 01 BlueGiga ADC sensor raw reading

| 01 | Channel | Raw sensor reading |
|---|---|---|
| 8 | 8 | 16 |

This field was in use with hardware version 4, 5 and 11 where BlueGiga (later Silicon Laboratories) BLE113 module were used. This field is not being used anymore.

Only two channels were in use, 0x00 for the battery reading, and 0x0e for the internal temperature sensor.

See the documentation for the field “06 Voltage measurement” for information on how to read the battery voltage and the temperature of an SBrick or SBrick plus, and any sensors attached to the ports of an SBrick Plus.

**Examples:**

“04 01 00 12 F0” - battery reading '12f0' on SBrick

“04 01 0e 12 F0” - temperature reading '12f0'

### 02 Device Identifier

| 0x02 | Device identifier string |
|---|---|
| 8 | 6 * 8 |

“07 02 0D 23 FC 19 87 63” - SBrick device ID

### 03 Simple Security status

| 0x05 | Status code |
|---|---|
| 8 | 8 |

Security status code

| | |
|---|---|
| 0 | Freely accessible |
| 1 | Authentication needed for some functions |

### 04 Command response

| 0x04 | Return code | Return value |
|---|---|---|
| 8 | 8 | (N-2) * 8 |

The return codes are the following:

| | |
|---|---|
| 00 | Successful operation |
| 01 | Invalid data length |
| 02 | Invalid parameter |
| 03 | No such command |
| 04 | No authentication needed |
| 05 | Authentication error |
| 06 | Authentication needed |
| 07 | Authorization error |
| 08 | Thermal protection is active |
| 09 | The system is in a state where the command does not make sense |

The return value can be zero or more bytes, and contain information related to the execution of the command. The documentation of each command describes what kind of data is returned, if any.

### 05 Thermal protection status

| 0x05 | Status |
|---|---|
| 8 | 8 |

Status may be:

| 1 | temperature is over the limit |
|---|---|
| 0 | temperature is below the limit |

### 06 Voltage measurement

| 0x06 | Measurement data |
|---|---|
| 8 | (N-1) * 8 |

Measurement data may contain measurements over multiple channels. Each measurement is described over 2 bytes. The 3 upper nibbles contain the 12 bit raw ADC data. The low nibble contains the channel number.

The channels are the following:

| 0 | Port 0 (A), C1 |
|---|---|
| 1 | Port 0 (A), C2 |
| 2 | Port 1 (C), C1 |
| 3 | Port 1 (C), C2 |

| | |
|---|---|
| 4 | Port 2 (B), C1 |
| 5 | Port 2 (B), C2 |
| 6 | Port 3 (D), C1 |
| 7 | Port 3 (D), C2 |
| 8 | Battery voltage |
| 9 | Internal temperature |

**Example:** manufacturer specific data for SBrick, containing device ID with hw/sw revision, with battery readings and device ID:

“1A FF 98 01

06 00 00 04 00 04 02

04 01 0E 12 f0

07 02 0D 23 FC 19 87 63

02 03 00”

**Example:** data sent in a notification on the Remote Control Commands characteristic. It contains a command acknowledgement with no data returned, and raw ADC reading of “12F0” on channel 00:

“02 04 00 04 01 00 12 F0”

### 07 Signal Completed

| |
|---|
| 0x07 |
| 8 |

A notification with a “signal completed” record will be sent whenever a “33 Send Signal” command is completed.

# The GATT database

The following services and characteristics are in the database:

1. Generic GAP service
2. Device information service
3. OTA service
4. Remote control service

The following sections describe each service and characteristic. The hexadecimal string in the title is the service or characteristic UUID.

## Generic Attribute - 1801

This service only presents in hardware version 12 and 13. It contains a single “service changed” characteristic. It should be handled according to the Bluetooth specification.

## Generic Access - 1800

Only contains the device name and appearance characteristics. The device name (2a00), and the appearance (2a01) is always 0384 a.k.a. "generic remote control", according to the Bluetooth specification.

The device name is "SBrick" out of the box, but can be changed either by issuing the appropriate remote control command, or by writing this characteristic.

## Device information - 180a

Contains mandatory device information fields.

- Model number string - 00: "SBrick". Same as the "Product type" above.
- Firmware revision string
- Hardware revision string
- Software revision string
    - These are version information strings. The "firmware" and "software" revision string are always the same.
    - The revision string consist of a major and a minor revision, separated by a dot. (Example: 4.1 - Major is 4, minor is 1.)
    - A firmware is ONLY compatible with a hardware, if their MAJOR REVISION NUMBERS ARE EXACTLY THE SAME.
- Manufacturer string - "Vengit Ltd."

## OTA service - 1d14d6ee-fd63-4fa1-bfa4-8f47b42119f0

**WARNING. THE SERVICE AND CHARACTERISTICS DESCRIBED HERE MUST BE USED ONE OF TWO DIFFERENT WAYS WITH HARDWARE VERSIONS 4, 5, AND 11, AND THE OTHER WAY WITH VERSIONS 12, AND 13. THE DESCRIPTION BELOW DESCRIBES THE DIFFERENCES.**

The OTA service is compatible with BlueGiga and/or Silicon Laboratories OTA solutions. See the application notes “AN984: BLUETOOTH SMART SOFTWARE Implementing Over-the-Air Firmware Upgrade” https://www.silabs.com/documents/login/application-notes/AN984.pdf and “AN1045: Bluetooth ® Over-the-Air Device Firmware Update for EFR32xG1 and BGM11x Series Products” https://www.silabs.com/documents/login/application-notes/an1045-bt-ota-dfu.pdf respectively.

### OTA control - f7bf3564-fb6d-4e53-88a4-5e37e0326063

This characteristic can be used to send OTA-specific commands. The only command that is usable for SBrick and SBrick plus from (including) hardware version 4 to 11 is “03 Reboot into DFU mode” (as per AN984). With hardware version 12 and 13, any write on this characteristic will result in the BLE module being rebooted into DFU mode. **With hardwares 12 and 13, booting into DFU mode will change the GATT database. The Service Changed characteristic must be used with these hardver versions for GATT caching not to be an issue.**

The “03” command is also used by the new hardware versions. It has to be used in OTA DFU mode after transferring the update file. This essentially does the same thing as in the old versions: it applies the update and reboots the device with the new firmware (or apploader).

**Versions 4, 5, and 11:** *After* successfully transmitting the firmware image, the device can be booted into DFU mode with this command. In this mode, the device checks the flash memory for a firmware image, and calculates the checksum. If the checksum is correct, the firmware image is transferred into the program memory. This procedure takes approximately five seconds. After this, the firmware erases the user flash. During this procedure the ID LED blinks quickly for about one and a half seconds. For every command a notification is sent on the Quick Drive characteristic as an acknowledgement.

**Versions 12 and 13:** These devices must be booted into DFU mode *before* transmitting the firmware image. The device then will change the GATT database, and issue a notification on the Service Changed characteristic. The new GATT database, the one that is present only in OTA mode, will contain the OTA data characteristic. Writing any data to this characteristic will result in the device being booted into OTA DFU mode.

Rebooting the device into OTA DFU mode will change the device name to “OTA_xx” (where “xx” is the hardware version), but will leave the hardware address the same.

### OTA data - 984227f3-34fc-4045-a5d0-2c581f81a153

**With versions 4, 5, and 11:** This characteristic can be used to transfer the firmware image in 20 byte packages. After successfully uploading the firmware, the application MUST issue a command "0x03 Reboot into DFU mode" on the control characteristic to restart the device into DFU mode.

This characteristic can also be read to check the written firmware, or blank-check the flash. One must reset the DFU pointer with command 0x02 on the control characteristic before

attempting  a readback. This feature is probably not that useful in normal circumstances. It was developed to aid development / debugging.

For every command a notification is sent on the Quick Drive characteristic as an acknowledgement.

**With versions 12 and 13:** This characteristic is present **ONLY** in OTA DFU mode.

Earlier versions used extra flash memory to load the firmware image over the air, and then overwriting the firmware during a reboot. The new hardware loads the firmware in-place.

This means that newer SBricks and SBrick Plus-es can be “bricked” - rendered unusable if a firmware update fails. If this happens, the device will keep starting in OTA DFU mode. Any client application must be able to recognize that the device is in OTA DFU mode, and retry loading the new firmware.

After booting into OTA DFU mode, the application may start the data transfer by first writing a zero (0x00) byte to the OTA control characteristic, and then transfer the firmware image by writing the ota data characteristic. Both acknowledged and unacknowledged writes can be used.

A write size of 55 bytes can be used in newer hardware versions.

## OTA procedure

**With versions 4, 5, and 11:**

1. Connect to the device
2. Start writing data to the OTA data characteristic. Both acknowledged and unacknowledged writes may be used. Additional feedback can be gained by subscribing to notifications on the quick drive characteristic.
3. Issue ota control command 0x03 “Reboot into DFU”
4. Wait for the device to come online
5. Verify the firmware version number

**With versions 12 and 13:**

1. Connect to the device.
2. Issue ota control command 0x00.
3. If the device is NOT in OTA DFU mode, it closes the connection, and reboots into that mode. If it is already in OTA DFU mode, the control command 0x00 will initiate the data transfer. **The 0x00 OTA control command MUST be issued before starting data transfer.** Leaving out this step will result in receiving the user-defined BLE error 0x0481 “WRONG_STATE”.
4. In OTA DFU mode, after writing the control command 0x00, write the first, “apploader” firmware image.
5. Issue ota control command 0x03, and disconnect from the device.
6. The device reboots. Wait for it to come online again in either OTA mode.

7. Issue the 0x00 control command again, and write the second, “application” firmware image.
8. Issue ota control command 0x03, and close the connection.
9. Wait for the device to come online again in normal mode.

If the device comes online in OTA mode, the whole procedure can be retried any time, even after accidental power loss, restart, or transfer failure.

The Bluetooth & host controller module of hardware 12 and 13 is manufactured by Silicon Laboratories. Their official OTA firmware update documentation can be read here: https://www.silabs.com/documents/login/application-notes/an1045-bt-ota-dfu.pdf

A demo application with firmware update capability is available for Android: https://play.google.com/store/apps/details?id=com.siliconlabs.bledemo

## Remote control service - 4dc591b0-857c-41de-b5f1-15abda665b0c

This service contains two characteristics:

- Quick Drive: allows remote control with small data packets. Very limited functionality.
- Remote control commands: allows full control, more verbose and slower than quick drive.

The Quick Drive characteristic should be used in situations where multiple channels must be updated quickly. It uses less bandwidth than issuing commands at the Remote control commands characteristic.

The Remote control commands characteristic allows full control over SBrick.

The Remote control commands characteristic supports notifications. Subscribing to this characteristic will enable notifications on various events, including the acknowledgement of every write on the OTA, Quick Drive, and Remote control commands characteristics.

### Notification payload

SBrick can acknowledge every remote control and quick drive data packet (and OTA command or data packets in hardware versions 4, 5, and 11). Besides acknowledgements, SBrick can also send information periodically and autonomously.

To reduce bandwidth, latency, and jitter, a single notification might include several pieces of information, including:

- A single byte indicating boolean properties, such as whether the notification acknowledges a write to the SBrick or not.
- The result of the last command if there were any, including any error codes.
- A/D data sent autonomously or after a read command.

The data included in the notification is formatted according to the "SBrick Data Records" section.

### Remote control commands - 2b8cbcc-0e25-4bda-8790-a15f53e6010f

Commands can be issued by writing data to this characteristic. A command always starts with the command identifier byte, after which parameters may follow. A single BLE write operation can only send a single command.

Certain commands can return a value. The central can also subscribe to notifications on this characteristic, so acknowledgements with return values and other information can be sent by the SBrick.

The "OWNER" note means that the operation requires owner privileges.

The possible commands are following.

**00 Brake**

| 0x00 | Channel 1 | Channel 2 | ... |
|---|---|---|---|
| 8 | 8 | 8 | ... |

At least one, at most four channels can be given

(The default is all channels are freewheeling)

Returns: -

**01 Drive**

| 0x01 | Channel 1 | Direction 1 | Power 1 | ... |
|---|---|---|---|---|
| 8 | 8 | 8 | 8 | ... |

(The default is all channels are freewheeling)

Returns: -

**02 Need authentication?**

| 0x02 |
|---|
| 8 |

If owner password is set, this will return true.

This exact information is reflected in the "simple security" field in manufacturer specific data.

Returns:

| | |
|---|---|
| 0x00 | Authentication is not needed |
| 0x01 | Authentication is needed |

**03 Is authenticated?**

| |
|---|
| 0x03 |
| 8 |

Returns whether the current session is authenticated. This will always return true, if there's no owner password set.

Returns:

| | |
|---|---|
| 0x00 | Not authenticated |
| 0x01 | Authenticated |

**04 Get user ID**

| |
|---|
| 0x04 |
| 8 |

Returns the authenticated user ID. If the user is not authenticated, then a BLE error is returned.

Returns: User ID if authenticated.

**05 Authenticate**

| | | |
|---|---|---|
| 0x05 | User ID | Password |
| 8 | 8 | 8*8 |

New sessions are unauthenticated if a password is set.

New sessions are authenticated if a password is not set.

Returns: -

**06 Clear password (OWNER)**

| 0x06 | Type (0/1) |
|---|---|
| 8 | 8 |

Examples:

06 00: clears owner password. This will "open" SBrick, anyone connecting will get owner rights. Guest password will also be cleared.

06 01: clear only guest password, rendering guests unable to authenticate.

Returns: -

**07 Set password (OWNER)**

| 0x07 | User ID | Password |
|---|---|---|
| 8 | 8 | 8*8 |

User IDs:

| 0x00 | Owner |
|---|---|
| 0x01 | Guest |

Guest password can only be set if there is a password set for the owner too (e.g. "need authentication?" returns 1)

Returns: -

**08 Set authentication timeout (OWNER)**

| 0x08 | Duration |
|---|---|
| 8 | 8 |

Duration is 0.1 seconds x N, minimum 1, maximum 25.5 seconds.

Sets the authentication timeout. This value is saved to the persistent store, and loaded at boot time.

Returns: -

**09 Get authentication timeout (OWNER)**

| 0x09 |
|---|
| 8 |

Returns: <1 byte auth timeout in 0.1 sec. ticks>

**0A Get brick ID**

| 0x0A |
|---|
| 8 |

Returns: < BRICK ID, 6 byte ID >

**0B  Quick Drive Setup**

| 0x0B | Channel 1 | Channel 2 | ... |
|---|---|---|---|
| 8 | 8 | 8 | ... |

At least one, at most five channels can be given.

Default: First five channels in ascending order.

Returns: -

**0C  Read Quick Drive Setup**

| 0x0C |
| --- |
| 8 |

Returns: <5 byte quick drive setup>

**0D Set watchdog timeout**

| 0x0D | Timeout in 0.1 secs. |
| --- | --- |
| 8 | 8 |

The purpose of the watchdog is to stop driving in case of an application failure. Watchdog starts when the first DRIVE command is issued during a connection.

Watchdog is stopped when all channels are set to zero drive.

The value is saved to the persistent store.

The recommended watchdog frequency is 0.2-0.5 seconds, but a smaller and many larger settings are also available.

Writing a zero disables the watchdog.

By default watchdog is set to 5, which means a 0.5 second timeout.

Returns: -

**0E Get watchdog timeout**

| 0x0E |
| --- |
| 8 |

Returns: < 1 byte watchdog timeout >

**0F Query ADC**

**WARNING. THE INFORMATION BELOW IS OUTDATED. HARDWARE VERSIONS 12 AND 13 USE DIFFERENT CALCULATIONS FOR THE BATTERY MEASUREMENT AND THE INTERNAL TEMPERATURE MEASUREMENT.**

| 0x0F | ADC channel ID, 0x00 through 0x09 |
|---|---|
| 8 | 8 |

The ADC channels are read approximately five times a second. These values are stored in variables, and this query simply reads those variables.

Temperature and battery voltage measurements are always taken. Use the command "2C Set up periodic voltage measurement" to measure port pins on SBrick Plus (hardware version 11 and 13) models.

Temperature can be read on channel 0x09, voltage on 0x08.

Returns:

2 byte, little endian, 12 bit resolution ADC reading on given channel.

Value is stored MSB. (Must be divided by 16)

All ADC channels are using the internal 1.24V reference.

The PSU voltage is dropped through a 10:1 voltage divider.

VPSU = (ADC * 0.83875) / 2047.0

Temperature can be calculated as: celsius = ADC / 118.85795 - 160

Where 160 is an offset

**11 Erase user flash on next reboot (compromises OTA!)**

| 0x11 |
|---|
| 8 |

This command only works on hardware version 4, 5, and 11. On version 12 and 13, this command is implemented, but does nothing for compatibility reasons.

Returns: -

**12 Reboot**

| 0x12 |
| --- |
| 8 |

After issuing this command, the remote device will gracefully terminate the connection, and reboot into normal (non-DFU) mode.

To reboot in DFU mode and possibly update firmware, use the OTA service.

Returns: -

**13 Brake with PWM support**

| 0x13 | Channel 1 | Power 1 | ... |
| --- | --- | --- | --- |
| 8 | 8 | 8 | ... |

Multiple channel-power pairs can be given. (The default is all channels are freewheeling.)

Returns: -

**14 Set thermal limit**

| 0x14 | ADC value |
| --- | --- |
| 8 | 16 |

Sets the thermal protection limit.

Returns: -

**15 Read thermal limit**

| 0x15 |
| --- |
| 8 |

Returns: 2 bytes, the raw ADC value set for thermal limit

**1F Set PWM counter top value**

| 0x1F | PWM counter top value |
|---|---|
| 8 | 8 |

Sets the PWM  counter value by writing into the counter hardware control registers. Certain register values are also recalculated to keep the PWM duty cycle as constant across changes as possible.

The 2-byte value is the top value of the counter, and is an little endian unsigned integer.

The default PWM value with different hardware versions (decimal):

| **Hardware version** | **PWM counter top value** | **PWM counter clock frequency (MHz)** | **PWM frequency (Hz)** |
|---|---|---|---|
| 4 | 31874 | 32 | ~1003.9 |
| 5, 11 | 3823 | 4 | ~1046.3 |
| 12, 13 | 4588 | 4.8 | ~1046.2 |

To calculate the PWM counter top value from the Timer clock frequency, use the formula:

PWMTOP = PWM_CLOCK_HZ / PWM_FREQUENCY

**20 Get PWM counter value**

| 0x20 |
|---|
| 8 |

Returns the 2 byte TIMER1 T1CC0H/L value.

Return <2 byte T1CC0H/L value>

**21 Save PWM counter value**

| 0x21 |
| --- |
| 8 |

Saves PWM counter value to flash

**22 Get channel status**

| 0x22 |
| --- |
| 8 |

Returns the current drive level of a channel

Return < brake status bits, 1 byte, 1:brake on, 0: brake off > <1 byte direction flags> <5 byte channel drive values from 0 to 4>

**23 Is guest password set (OWNER)**

| 0x23 |
| --- |
| 8 |

Returns: 1 or 0

**24 Set connection parameters**

| 0x24 | Interval, min. | Interval, max. | Slave latency | Timeout |
| --- | --- | --- | --- | --- |
| 8 | 8 | 8 | 8 | 8 |

Connection interval values are in 1.25ms units. Timeout is in 10ms units.

Returns: 1 byte return value of Silicon Labs BLE stack function "connection_update"

**25 Get connection parameters**

| 0x25 |
| --- |
| 8 |

Returns: < connection interval * 1.25ms, 2 bytes >< slave latency, 2 bytes >< timeout * 10ms, 2 bytes >

**26 Set release on reset**

| 0x26 | Release? (0/1) |
| --- | --- |
| 8 | 8 |

1: Default: the channel drive values are set to zero, non-braking, and default "0" direction (clockwise with LEGO motors)

0: The channels are left in whatever state the controlling application set them. This option itself is preserved throughout connections.

**27 Get release on reset**

| 0x27 |
| --- |
| 8 |

Returns: <1 byte, 0 or 1>

**28 Read power cycle counter**

| 0x28 |
| --- |
| 8 |

Returns: <4 bytes, uint32>

**29 Read uptime counter**

| 0x29 |
| --- |
| 8 |

Returns: <4 bytes, uint32>

**2A Set device name**

| 0x2A | Devicename |
| --- | --- |
| 8 | 8*1 - 8*10 |

Returns: -

**2B Get device name**

| 0x2B |
| --- |
| 8 |

Return < Device name, 1-10 bytes min-max. >

**2C Set up periodic voltage measurement**

| 0x2C | List of channels |
| --- | --- |
| 8 | N*8 |

Each byte in the parameter list defines a channel number to measure. An empty list disables any active periodic measurement.

Returns: -

**2D Get voltage measurement setup**

| 0x2D |
| --- |
| 8 |

Returns: list of measured channels

**2E Set up periodic voltage notifications**

| 0x2E | List of channels |
| --- | --- |
| 8 | N*8 |

Each byte in the parameter list defines a channel of which to send notifications. An empty list disables any active periodic notification.

Returns: -

**2F Get voltage notification setup**

| 0x2F |
| --- |
| 8 |

Returns: list of measured channels

**30 Set ADC Correction Terms**

| 0x30 | Channel (0-7) | Bank | 3*32 bit signed integer |
| --- | --- | --- | --- |
| 8 | 8 | 8 | 3*(4*8) |

This instruction sets coefficients SBrick will use to calculate values returned on ADC channels 0-7.

The default setting is to return the raw, unaltered values returned by the ADC.

With these coefficients it is possible to scale and offset the ADC to compensate for voltage fluctuations normalize values in a particular range.

For each channel there are 6 coefficients, each isa 32-bit signed integer in the 2-s complement format: P0 - P5. The numbers are received and transmitted in little endian order.

For every channel 3 coefficient can be set at any time because of the length limitation on BLE packets. The groups of 3 numbers for a channels are called bank 0 and bank 1. In bank 0 there are coefficients P0 - P2, in bank 1 there are coefficients P3 - P5.

The calculations SBrick does with these numbers is:

(P0 * channel + P1 * battery + P2) / (P3 * channel + P4 * battery + P5)

where P0 - P5 are the coefficients, "channel" is the raw 12-bit channel reading, "battery" is the raw 12-bit battery reading.

The default values are:

P0 = 1
P1 = 0
P2 = 0
P3 = 0
P4 = 0
P5 = 1

The commands to set the default values are:

30 01 00 01000000 00000000 00000000
30 01 01 00000000 00000000 01000000

The default values are written to every channel at the beginning of each connection.

Recommended values for scaling the voltage between 0-1000:

P0 = 1000
P1 = 0
P2 = 0
P3 = 0
P4 = 1
P5 = 0

The commands to set these:

30 01 00 E8030000 00000000 00000000
30 01 01 00000000 01000000 00000000

Recommended values for 5V adapters:

P0 = 9850
P1 = -254

P2 = 60050
P3 = 0
P4 = 0
P5 = 7205

The commands to set these:

30 01 00 7A260000 02FFFFFF 92EA0000
30 01 01 00000000 00000000 251C0000

Recommended values for 3.3V adapters

P0 = 9900
P1 = -239
P2 = 35650
P3 = 0
P4 = 0
P5 = 4735

The commands to set these:

30 01 00 AC260000 11FFFFFF 428B0000
30 01 01 00000000 00000000 7F120000

Returns: -

**31 Get ADC Correction Terms**

| 0x31 | Channel (0-7) | Bank |
|---|---|---|
| 8 | 8 | 8 |

Returns: < 3*32 bit signed integer >

**32 Set ADC correction profile**

32 < Channel number 0-7 > < Profile number >

| 0x32 | Channel (0-7) | Profile |
|---|---|---|
| 8 | 8 | 8 |

Profiles:

0: Default, no correction
1: Scale between 0 and 1000

2: 5v adapter compensation
3: 3v adapter compensation

**33 Send Signal**

| 0x33 | Port (0-3) | Direction | Duty cycle | Duration (*200ms) | 16-bit PWM divider |
|---|---|---|---|---|---|
| 8 | 8 | 8 | 8 | 8 | 16 |

Signals completion with the "07 Signal Completed" field.

Examples:

Send signal to port 0, forward, half power, for 1 second, with the default PWM duty cycle in the case of hardware 5 and 11:

33 00 00 7f 05 ef0e

Send a mode change command to an adapter using an EV3 touch sensor on port 0 (hardware 11):

33 00 00 7f 02 6b16

Send a mode change command to an adapter using an EV3 touch sensor on port 0 (hardware 13):

33 00 00 7f 02 e71a

### Quick Drive - 489a6ae0-c1ab-4c9c-bdb2-11d373c1b7fb

The purpose of this characteristic is to make remote controlling possible using as little bandwidth as possible. A two-channel race car can be controlled by sending only two bytes: one for the accelerator and one for steering.

One can write (no response) 0-5 byte data packets to this characteristic to drive channels.

The characteristic can be thought of as a five byte register. Each byte in the register conrtols one channel. With the 0x0A "Quick Drive Setup" command, one can configure which byte controls which channel. The default is that byte 0 controls channel 0, byte 1 conrtols channel 1, and so on.

Quick Drive is the recommended way to control models where low latency is required, and there might be dozens of models in the same area.

Since each channel is driven with one byte, the direction and the PWM information must be fitted into that single byte. This is done in the following way:

Example: "Drive forward (clockwise) 255"

1 1 1 1 1 1 1 0 <- Direction bit

– – – – – – – –

MSb LSb

Example:

writing the byte string “00 FF FE 00” will turn channel #1 CCW full speed, channel #2 CW full speed, and set channels #0 and #3 braking.

- Braking happens when the value is set to zero (less the direction bit).
- When the value is 2 (less the direction bit), it is modified to 0.
- When the value is 0xFE (less the direction bit), it is modified to 0xFF

The last two modification make full and zero throttle possible.