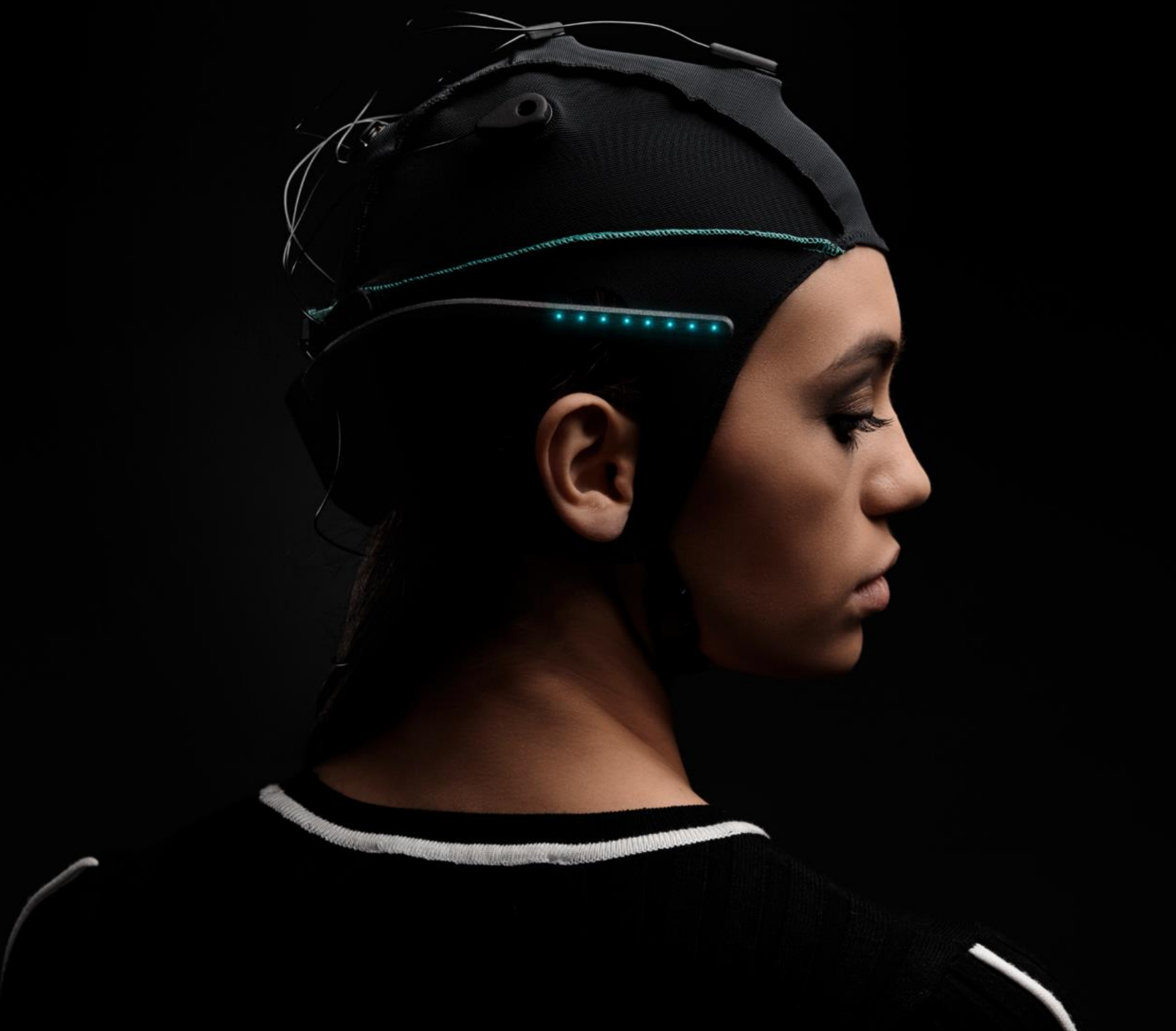
unicorn
Bluetooth Protocol

USER MANUAL



User Manual for Bluetooth Protocol

Version Name: Unicorn Hybrid Black

Version Number: 1.18.00

www.unicorn-bi.com | hello@unicorn-bi.com

CONTENTS

1. UNICORN BLUETOOTH PROTOCOL

The Unicorn Bluetooth Protocol describes the raw Bluetooth data stream between Unicorn and any Bluetooth device.

1.1. REQUIREMENTS

Hardware	Properties
Unicorn	Hybrid Black

1.2. COMMANDSET

Start Acquisition

Sets the Unicorn into Acquisition mode. The Unicorn sends an acknowledge message that the "Start Acquisition" command has been received. After the acknowledge message was sent, the Unicorn starts to send payloads in regular time intervals until the "Stop Acquisition" command is received.

Command:

Index	0	1	2
Value	0x62	0x4C	0xE4

Response:

Index	0	1	2
Value	0x00	0x00	0x00

Stop Acquisition

Stops a running data acquisition. The unicorn stops sending payloads. An acknowledge message is sent after the last payload.

Command:

Index	0	1	2
Value	0x63	0x5C	0xC5

Response:

Index	0	1	2
Value	0x00	0x00	0x00

1.3. PAYLOAD

After the "Start Acquisition Command" has been received the Unicorn Hybrid Black will respond with payloads sent with a sampling rate of 250Hz until the "Stop Acquisition Command" is received. The payload is a byte array structured as following:

Index	0…1	2	3…26	27…32	33…38	39…42	43…44
Value	Start	Battery	Acquisition Data	Accelerometer	Gyroscope	Counter	Stop

Start

Index	0	1
Value	0xC0	0x00

Start sequence

Battery

Index	2
Value	BatteryValue

The bits [3:0] contain the battery voltage coded in 4 Bit integral value (MSB first).

$$Battery\ Voltage\ [\%] = \frac{100}{1.3} \cdot \frac{1.3 * (BatteryValue\ \&\ 0x0F)}{15}$$

Acquisition Data

Index	3…5	6…8	9…11	12…14	15…17	18…20	21…23	24…26
Value	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8

3 bytes (24 bit) per channel (two's complement, Big-Endian)

Convert the 3 bytes array of each channel's value to a 32-bit signed integral value:

$$channelValue = CH_n[0] \ll 16 \mid CH_n[1] \ll 8 \mid CH_n[2]$$

Apply two's complement to the 32-bit signed integral value if the sign bit is set:

$$if\,(channelValue\ \&\ 0x00800000)$$

$$channelValue = channelValue \mid 0xFF000000$$

Convert the 32-bit signed integral value to a 32 bits floating point value representing microvolts:

$$Channel\ [\mu V] = channelValue \cdot \frac{4500000}{50331642}$$

Accelerometer

Index	27…28	29…30	31…32
Value	X	Y	Z

Acceleration data for x, y and z axis (twos complement, Little-Endian)

Convert the 2 bytes array of each channel's value to a 16-bit signed integral value:

$$accelerationValue = ACC_{x,y,z}[0] \mid ACC_{x,y,z}[1] \ll 8$$

Convert the 16-bit signed integral value to a 32 bits floating point value:

$$Acceleration\ [g] = \frac{accelerationValue}{4096}$$

Gyroscope

Index	33…34	35…36	37…38
Value	X	Y	Z

Gyroscope data for x, y and z axis (twos complement, Little-Endian)

Convert the 2 bytes array of each channel's value to a 16-bit signed integral value:

$$gyroscopeValue = GYR_{x,y,z}[0] \mid GYR_{x,y,z}[1] \ll 8$$

Convert the 16-bit signed integral value to a 32 bits floating point value:

$$Angular\ Velocity\ [°/s] = \frac{gyroscopeValue}{32.8}$$

Counter

Index	39…42
Value	Counter

Sample counter (Little-Endian)

Convert the 4 bytes array of each channel's value to a 32-bit signed integral value:

$$counterValue = Counter[0] \mid Counter[1] \ll 8 \mid Counter[1] \ll 16 \mid Counter[1] \ll 24$$

Stop

Index	43	44
Value	0x0D	0x0A

Stop sequence

1.4. DATA ACQUISITION EXAMPLE

1. Send "Start Acquisition" command
2. Receive "Start Acquisition" acknowledge message
3. Receive payload
4. Convert binary data to physical values
5. Repeat step 3 and 4 until the data acquisition should be terminated
6. Send "Stop Acquisition" command
7. Receive "Start Acquisition" acknowledge message

1.5. PAYLOAD CONVERSION EXAMPLE

Payload Example:

Index	0	1	2	3	4	5	6	7	8	9	10
Value	0xC0	0x00	0x0F	0x00	0x9F	0xAF	0x00	0x9F	0xD4	0x00	0xA0

11	12	13	14	15	16	17	18	19	20	21	22
0x40	0x00	0x9F	0x43	0x00	0x9F	0x9A	0x00	0x9F	0xE3	0x00	0x9F

23	24	25	26	27	28	29	30	31	32	33	34
0x85	0x00	0x9F	0xBB	0x2E	0xF6	0xE9	0x02	0x8D	0xF2	0xF3	0xFF

35	36	37	38	39	40	41	42	43	44
0xEF	0xFF	0x23	0x00	0xB0	0x00	0x00	0x00	0x0D	0x0A

Converted Payload

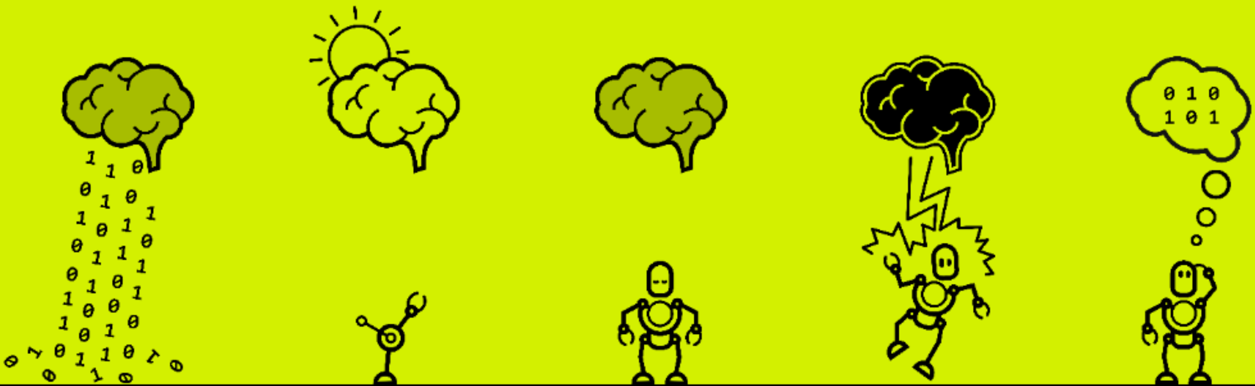
Battery	CH1	CH2	CH3	CH4	CH5	CH6	CH7
100%	3654.87µV	3658.18µV	3667.83µV	3645.21µV	3652.99µV	3659.52µV	3651.11µV

CH8	ACCX	ACCY	ACCZ	GYRX	GYRY	GYRZ	CNT
3655.94µV	-0.614g	0.182g	-0.841g	-0.397°/s	-0.519°/s	1.068°/s	176

1.6. PAYLOAD CONVERSION - C/C++ EXAMPLE

```
#define UNICORN_EEG_CHANNELS_COUNT                      8
#define UNICORN_ACCELEROMETER_CHANNELS_COUNT            3
#define UNICORN_GYROSCOPE_CHANNELS_COUNT                3
#define UNICORN_COUNTER_CHANNELS_COUNT                  1
#define UNICORN_BATTERY_LEVEL_CHANNELS_COUNT            1
#define UNICORN_VALIDATION_INDICATOR_CHANNELS_COUNT     1

#define UNICORN_PAYLOAD_HEADER_LENGTH                   2
#define UNICORN_PAYLOAD_HEADER_OFFSET                   0

#define UNICORN_PAYLOAD_BYTES_PER_BATTERY_LEVEL_CHANNEL 1
#define UNICORN_PAYLOAD_BATTERY_LEVEL_LENGTH     UNICORN_PAYLOAD_BYTES_PER_BATTERY_LEVEL_CHANNEL
#define UNICORN_PAYLOAD_BATTERY_LEVEL_OFFSET     UNICORN_PAYLOAD_HEADER_LENGTH

#define UNICORN_PAYLOAD_BYTES_PER_EEG_CHANNEL    3
#define UNICORN_PAYLOAD_EEG_LENGTH       UNICORN_EEG_CHANNELS_COUNT*UNICORN_PAYLOAD_BYTES_PER_EEG_CHANNEL
#define UNICORN_PAYLOAD_EEG_OFFSET       (UNICORN_PAYLOAD_BATTERY_LEVEL_OFFSET + UNICORN_PAYLOAD_BATTERY_LEVEL_LENGTH)

#define UNICORN_PAYLOAD_BYTES_PER_ACC_CHANNEL    2
#define UNICORN_PAYLOAD_ACC_LENGTH       UNICORN_ACCELEROMETER_CHANNELS_COUNT*UNICORN_PAYLOAD_BYTES_PER_ACC_CHANNEL
#define UNICORN_PAYLOAD_ACC_OFFSET       (UNICORN_PAYLOAD_EEG_OFFSET + UNICORN_PAYLOAD_EEG_LENGTH)

#define UNICORN_PAYLOAD_BYTES_PER_GYR_CHANNEL    2
#define UNICORN_PAYLOAD_GYR_LENGTH       UNICORN_GYROSCOPE_CHANNELS_COUNT*UNICORN_PAYLOAD_BYTES_PER_GYR_CHANNEL
#define UNICORN_PAYLOAD_GYR_OFFSET       (UNICORN_PAYLOAD_ACC_OFFSET + UNICORN_PAYLOAD_ACC_LENGTH)

#define UNICORN_PAYLOAD_BYTES_PER_CNT_CHANNEL    4
#define UNICORN_PAYLOAD_CNT_LENGTH       UNICORN_PAYLOAD_BYTES_PER_CNT_CHANNEL
#define UNICORN_PAYLOAD_CNT_OFFSET       (UNICORN_PAYLOAD_GYR_OFFSET + UNICORN_PAYLOAD_GYR_LENGTH)

#define UNICORN_PAYLOAD_FOOTER_LENGTH    2
#define UNICORN_PAYLOAD_FOOTER_OFFSET    (UNICORN_PAYLOAD_CNT_OFFSET + UNICORN_PAYLOAD_CNT_LENGTH)

int main()
{
        //sample payload
        unsigned char payload[] = { 0xC0, 0x00, 0x0F, 0x00, 0x9F, 0xAF, 0x00, 0x9F, 0xD4, 0x00, 0xA0, 0x40, 0x00, 0x9F,
0x43, 0x00, 0x9F, 0x9A, 0x00, 0x9F, 0xE3, 0x00, 0x9F, 0x85, 0x00, 0x9F, 0xBB, 0x2E, 0xF6, 0xE9, 0x02, 0x8D, 0xF2, 0xF3,
0xFF, 0xEF, 0xFF, 0x23, 0x00, 0xB0, 0x00, 0x00, 0x00, 0x0D, 0x0A };

        //destination buffer
        float data[] = { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };

        //get battery level in percent
        data[0] = (100.0f / 1.3f) *((payload[UNICORN_PAYLOAD_BATTERY_LEVEL_OFFSET] & 0x0F)* 1.3f / 15.0f);

        //get eeg in microvolts
```

```
        for (int i = 0; i < UNICORN_EEG_CHANNELS_COUNT; i++)
        {
                //raw data to int32 value
                int eegTemp = 0;
                eegTemp = (payload[UNICORN_PAYLOAD_EEG_OFFSET + i * UNICORN_PAYLOAD_BYTES_PER_EEG_CHANNEL + 0] << 16 |
payload[UNICORN_PAYLOAD_EEG_OFFSET + i * UNICORN_PAYLOAD_BYTES_PER_EEG_CHANNEL + 1] << 8 |
payload[UNICORN_PAYLOAD_EEG_OFFSET + i * UNICORN_PAYLOAD_BYTES_PER_EEG_CHANNEL + 2]);

                //check if first bit is 1 (2s complement)
                if (eegTemp & 0x00800000)
                        eegTemp |= 0xFF000000;

                //convert to eeg value in microvolts
                data[i + UNICORN_BATTERY_LEVEL_CHANNELS_COUNT] = eegTemp * (4500000.0f) / (50331642.0f);
        }

        //get accelerometer x,y,z in g
        for (int i = 0; i < UNICORN_ACCELEROMETER_CHANNELS_COUNT; i++)
        {
                short accTemp = (payload[UNICORN_PAYLOAD_ACC_OFFSET + i * UNICORN_PAYLOAD_BYTES_PER_ACC_CHANNEL] |
payload[UNICORN_PAYLOAD_ACC_OFFSET + i * UNICORN_PAYLOAD_BYTES_PER_ACC_CHANNEL + 1] << 8);
                data[i + UNICORN_BATTERY_LEVEL_CHANNELS_COUNT + UNICORN_EEG_CHANNELS_COUNT] = accTemp * (1.0f / 4096.0f);
        }

        //get gyroscope x,y,z in deg/s
        for (int i = 0; i < UNICORN_GYROSCOPE_CHANNELS_COUNT; i++)
        {
                short gyrTemp = (payload[UNICORN_PAYLOAD_GYR_OFFSET + i*UNICORN_PAYLOAD_BYTES_PER_GYR_CHANNEL] |
payload[UNICORN_PAYLOAD_GYR_OFFSET + i*UNICORN_PAYLOAD_BYTES_PER_GYR_CHANNEL + 1] << 8);
                data[i + UNICORN_BATTERY_LEVEL_CHANNELS_COUNT + UNICORN_EEG_CHANNELS_COUNT +
UNICORN_ACCELEROMETER_CHANNELS_COUNT] = gyrTemp * (1.0f / 32.8f);
        }

        //get counter
        data[UNICORN_BATTERY_LEVEL_CHANNELS_COUNT + UNICORN_EEG_CHANNELS_COUNT + UNICORN_ACCELEROMETER_CHANNELS_COUNT +
UNICORN_GYROSCOPE_CHANNELS_COUNT] = payload[UNICORN_PAYLOAD_CNT_OFFSET] << 0 | payload[UNICORN_PAYLOAD_CNT_OFFSET + 1] << 8
| payload[UNICORN_PAYLOAD_CNT_OFFSET + 2] << 16 | payload[UNICORN_PAYLOAD_CNT_OFFSET + 3] << 24;

        return 0;
}
```



BR41N.IO

THE BRAIN-COMPUTER INTERFACE
DESIGNERS HACKATHON

WWW.BR41N.IO

unicorn

THE BRAIN INTERFACE