# Cavway X1 Communication Protocol

**Version 1.0 Mar. 6th 2025**

The Cavway X1 communicates with mobile devices using the BLE (Bluetooth Low Energy) communication protocol. The UUIDs it uses are as follows:

- **Service UUID:** 6e400001-b5a3-f393-e0a9-e50e24dcca9e
- **Write Characteristic UUID:** 6e400002-b5a3-f393-e0a9-e50e24dcca9e
- **Read Characteristic UUID:** 6e400003-b5a3-f393-e0a9-e50e24dcca9e

In communication with Cavway, data is written through the Write Characteristic and received through the Read Characteristic.

The Cavway X1 communicates with computers using a USB virtual serial port. Its USB PID is **0x55D3**, and its VID is **0x1a86**. The serial port number is obtained via the PID and VID, and data is transmitted and received through the serial port connection.

## Shot data and calibration data:

When the storage is not empty, data is automatically transmitted through the BLE read characteristic (notify).

A data packet has a size of 64 bytes, and its format is defined as follows:

**Byte 0:** Packet type.
0x1 normal packet 0x2: calibration packet
**Byte 1:** the format is defined as follow:

| | | |
|---|---|---|
| Byte 2 - 7 | 1 | 0 (LSB) |
| Not Used | LF | CF |

CF: Calibration flag, 0 identify that this is a calibration packet.
LF: Leg flag, 0 identify that this packet's data belongs to a leg
**Byte 2:** distance (16 - 23 bit)
**Byte 3:** distance (0 – 7 bit)
**Byte 4:** distance (8 – 15 bit)
**Byte 5:** azimuth angle (Low byte)[1]
**Byte 6:** azimuth angle (High byte)
**Byte 7:** inclination angle (Low byte)
**Byte 8:** inclination angle (High byte)
**Byte 9:** roll angle (Low byte)
**Byte 10:** roll angle (High byte)
**Byte 11:** Sensor1 absG: the magnitude of gravitational acceleration (Low byte)
**Byte 12:** Sensor1 absG: the magnitude of gravitational acceleration (High byte)
**Byte 13:** Sensor1 absM: the magnitude of magnetic field strength (Low byte)
**Byte 14:** Sensor1 absM: the magnitude of magnetic field strength (High byte)
**Byte 15:** Sensor1 dip angle (Low byte)

**Byte 16:** Sensor1 dip angle (High byte)
**Byte 17:** timestamp (0 – 7 byte)
**Byte 18:** timestamp (8 – 15 byte)
**Byte 19:** timestamp (16 – 23 byte)
**Byte 20:** timestamp (24 – 31 byte)
**Byte 21:** Sensor1 GX (Gravity sensor1's X axis value, 0 -7 bytes)
**Byte 22:** Sensor1 GX (8 – 15 bytes)
**Byte 23:** Sensor1 GY (0 -7 bytes)
**Byte 24:** Sensor1 GY (8 – 15 bytes)
**Byte 25:** Sensor1 GZ (0 -7 bytes)
**Byte 26:** Sensor1 GZ (8 – 15 bytes)
**Byte 27:** Sensor1 MX (Magnetic sensor1's X axis value, 0 -7 bytes)
**Byte 28:** Sensor1 MX (8 – 15 bytes)
**Byte 29:** Sensor1 MY (0 -7 bytes)
**Byte 30:** Sensor1 MY (8 – 15 bytes)
**Byte 31:** Sensor1 MZ (0 -7 bytes)
**Byte 32:** Sensor1 MZ (8 – 15 bytes)
**Byte 33:** Sensor2 GX (Gravity sensor2's X axis value, 0 -7 bytes)
**Byte 34:** Sensor2 GX (8 – 15 bytes)
**Byte 35:** Sensor2 GY (0 -7 bytes)
**Byte 36:** Sensor2 GY (8 – 15 bytes)
**Byte 37:** Sensor2 GZ (0 -7 bytes)
**Byte 38:** Sensor2 GZ (8 – 15 bytes)
**Byte 39:** Sensor2 MX (Magnetic sensor2's X axis value, 0 -7 bytes)
**Byte 40:** Sensor2 MX (8 – 15 bytes)
**Byte 41:** Sensor2 MY (0 -7 bytes)
**Byte 42:** Sensor2 MY (8 – 15 bytes)
**Byte 43:** Sensor2 MZ (0 -7 bytes)
**Byte 44:** Sensor2 MZ (8 – 15 bytes)
**Byte 45 – 55:** Accuracy error information[2]
**Byte 56:** Laser signal Quality (0 – 7 bit)
**Byte 57:** Laser signal Quality (8 – 15 bit)
**Byte 58:** Sensor2 absG: the magnitude of gravitational acceleration (Low byte)
**Byte 59:** Sensor2 absG: the magnitude of gravitational acceleration (High byte)
**Byte 60:** Sensor2 absM: the magnitude of magnetic field strength (Low byte)
**Byte 61:** Sensor2 absM: the magnitude of magnetic field strength (High byte)
**Byte 62:** Sensor2 dip angle (Low byte)
**Byte 63:** Sensor2 dip angle (High byte)

[1] All angle definitions are as follows:
0x0000 – 0xFFFF corresponds to 0 - 360 degrees (in degrees) or 0 – 2π (in radians).

[2] The following code can be used to parse precision error information (from CavwayAssistant, in C#):

Code: C#

```
private const float ABSSCALE = 10000;
private const float ANGLESCALE = (float)(360.0 / 0xFFFF);

private string parseErrInfo(byte[] errbytes)
{
        string res = "";
        //string error_type = "";
        if (errbytes[0] == 0xFF)
        {
                res = "No error";
                return res;
        }
        int err_cnt = 0;
        if (((errbytes[0] >> 7) & 0x1) == 0x00)    //absG error
        {
                res += "absG error:";
                res += " absG1:" + ((errbytes[4 * err_cnt + 2] << 8 | errbytes[4 * err_cnt + 1]) / ABSSCALE).ToString("F4");
                res += " ";
                res += " absG2:" + ((errbytes[4 * err_cnt + 4] << 8 | errbytes[4 * err_cnt + 3]) / ABSSCALE).ToString("F4");
                res += " ";
                err_cnt++;
                if (err_cnt == 2) return res;
        }
        if (((errbytes[0] >> 6) & 0x1) == 0x00)    //absM error
        {
                res += "absM error:";
                res += " absM1:" + ((errbytes[4 * err_cnt + 2] << 8 | errbytes[4 * err_cnt + 1]) / ABSSCALE).ToString("F4");
                res += " ";
                res += " absM2:" + ((errbytes[4 * err_cnt + 4] << 8 | errbytes[4 * err_cnt + 3]) / ABSSCALE).ToString("F4");
                res += " ";
                err_cnt++;
                if (err_cnt == 2) return res;
        }
        if (((errbytes[0] >> 5) & 0x1) == 0x00)    //dip error
        {
                res += "dip error:";
                res += " err1:" + ((Int16)(errbytes[4 * err_cnt + 2] << 8 | errbytes[4 * err_cnt + 1]) * ANGLESCALE).ToString("F2");
                res += " ";
                res += " err2:" + ((Int16)(errbytes[4 * err_cnt + 4] << 8 | errbytes[4 * err_cnt + 3]) * ANGLESCALE).ToString("F2");
                res += " ";
                err_cnt++;
                if (err_cnt == 2) return res;
        }
        if (((errbytes[0] >> 4) & 0x1) == 0x00)    //angle error
        {
                res += "angle error:";
                res += ((errbytes[4 * err_cnt + 2] << 8 | errbytes[4 * err_cnt + 1]) * ANGLESCALE).ToString("F2");
                res += " ";
                err_cnt++;
                if (err_cnt == 2) return res;
        }
        return res;
}
```

After the app receives a data packet via Bluetooth, it needs to send an Acknowledge to the device. The data format for the Acknowledge is defined as follows:

| Packet Header | Acknowledge | Packet Footer |
|---|---|---|
| Only applied for BLE communication | ACK | Only applied for BLE communication |

The value of ACK is fixed as Byte[1] | 0x55, where Byte[1] is the first byte of the received data packet. The values of Header and Footer will be provided in the next section, "Data Query."

# Data Query

The Cavway X1 uses a data query method to retrieve information such as the device's hardware version number, firmware version number, historical data, calibration coefficients, and more.

The query command uses the following format. Note that Packet Header and Packet Footer are only applicable for Bluetooth communication. USB communication does not require a Header or Footer.

| Packet Header | 1st byte | 2nd Byte | 3rd Byte | 4th Byte | Packet Footer |
|---|---|---|---|---|---|
| Only applied for BLE communication | Command : 0x3d | ADDR_L: Low byte of address | ADDR_H: High byte of address | N: number of data to be read | Only applied for BLE communication |

The definition of the Header is as follows (applicable only for Bluetooth communication):

| Packet header Identifier | | | | | Payload Length |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 0x64 | 0x61 | 0x74 | 0x61 | 0x3a | 0x04 |

The definition of the Footer is as follows (applicable only for Bluetooth communication):

| Packet footer | |
|---|---|
| 0x0d | 0x0a |

The address mapping in the query command is as follows:

- 0x0000 – 0x7FFF: Historical data, where 0x0000 represents the most recently recorded data. Requires reading 64 bytes at a time.
- 0x8000: Current device timestamp. Requires reading 4 bytes at a time.
- 0x8008: Serial number, 4 digits. Requires reading 4 bytes at a time.
- 0x9080: Calibration coefficients for two sets of sensors. Requires reading 128 bytes at a time.
- 0xE000: Firmware version. Requires reading 4 bytes at a time.
- 0xE004: Hardware version. Requires reading 4 bytes at a time.

The format of the returned data frame is as follows:

| Command | ADDR_L | ADDR_H | Payload Length | Payload |
|---|---|---|---|---|
| 0x3d | The low byte of the address | The high byte of the address | Number of bytes in payload | Data |

# Send Command

Commands are sent by the following format via write characteristic:

| Packet Header | Length | Command | Packet Footer |
|---|---|---|---|
| Only applied for BLE communication | 0x01 | CMD | Only applied for BLE communication |

CMD:

- 0x30: quit calibration mode
- 0x31: enter calibration mode
- 0x32: convert (enter or quit) calibration mode
- 0x34: power off
- 0x36: Laser on
- 0x37: Laser off
- 0x38: Laser trigger measure