

HYPEREAL

Lighthouse tracking system-HAL

|  |  |  |
| --- | --- | --- |
| Date | Modified by | Comments |
| 2017-02-10 | Hypereal | Init Version. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

[1 Introduction 3](#_Toc30674)

[2 Data Structure 4](#_Toc23624)

[2.1 HMD【USB Device 1】 4](#_Toc13959)

[2.2 Gamepad & Lighthouse【USB Device 2】 7](#_Toc23765)

[3 Data Packet 13](#_Toc9981)

[3.1 Type definition for HMD 13](#_Toc15544)

[3.2 Type definition for Gamepad 13](#_Toc23709)

[4 Driver Architecture 14](#_Toc24737)

[4.1 Overall view 14](#_Toc21681)

[4.2 Class HVR\_Hmd 15](#_Toc13978)

[4.3 Class HVR\_Gamepad 15](#_Toc23362)

# Introduction

## Introduction

Six degrees of freedom (6DoF) refers to the freedom of movement of a rigid body in three-dimensional space. Specifically, the body is free to change position as forward/backward (surge), up/down (heave), left/right (sway) translation in three perpendicular axes, combined with changes in orientation through rotation about three perpendicular axes, often termed pitch, yaw, and roll.

Position tracking system is used to measure the 6DoF of a rigid body. Position tracking is a very fundamental and important technical field in VR world. It can provide users with a sense of immersion and interactivity. And accurate tracking will further strengthen this kind of experience, even will let users generate "illusion". User thought itself is in the virtual environment, unconsciously interact in the virtual world through tracking system. So precision and low latency is crucial to any VR system.

Common position tracking technology includes GPS, WIFI positioning, optical positioning, lighthouse positioning and so on.

**GPS：**  
The Global Positioning System (GPS), is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions.

The space part of the GPS is composed of 24 GPS working satellites, these GPS working satellites constitute the GPS satellite constellation. 21 satellites are used for navigation and 3 satellites are used for backup.

GPS satellites continuously transmit their current time and position. A GPS receiver monitors multiple satellites and solves equations to determine the precise position of the receiver and its deviation from true time. At a minimum, four satellites must be in view of the receiver for it to compute four unknown quantities.

**WIFI positioning:**

Each AP (wireless Access Points) has an MAC address, and it is assumed that AP does not move over a period of time, so using these features, WIFI can also do a rough positioning.

The most common and widespread localization technique used for positioning with AP is based on measuring the intensity of the received signal (received signal strength indication or RSSI) and the method of "fingerprinting". Typical parameters useful to geolocate the Wi-Fi hotspot or wireless access point include the SSID and the MAC address of the access point.

GPS positioning and WIFI positioning have a common problem, the positioning accuracy is not high. The civil GPS positioning accuracy is about 10m, the accuracy of WIFI is also not enough. So they are not suitable for VR applications.

**Opti Track：**

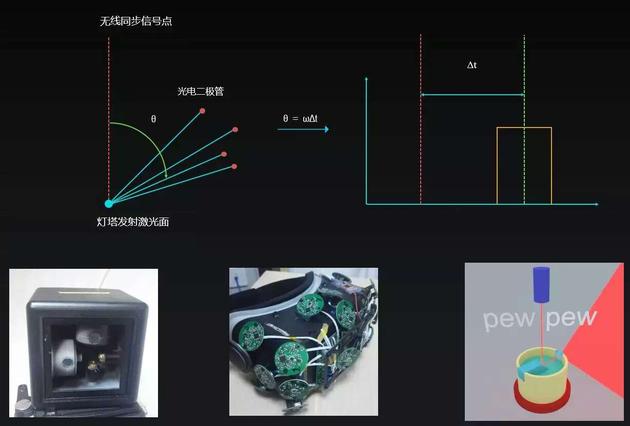
In this scheme, tracking object consist a lot of hidden infrared markers. Infrared camera captures the image and transmit to the computer, fitting three-dimensional model of the equipment through specific algorithm, finally get the location information.

**Lighthouse：**  
Lighthouse is a small box installed with infrared laser transmitter, It can emit laser scanning beam to space along X, Y axes 。The light sensors are covered in a tracking device, it collects the delay time of scanning beam. These information is transmitted to the host, then the position information of device is calculated through a specific algorithm.

Because both of Opti track and lighthouse can provide low latency and high-accuracy positioning，they are very suitable for VR applications。

The following is a brief introduction to the principle and system composition of the infrared laser positioning (Lighthouse) technology used by Hypereal.

## lighthouse technology and Hypereal lighthouse tracking system components

A base station is a little box that’s foundation of the Lighthouse Tracking System. It uses alternating sweeps of horizontal and vertical lasers to pass over the headset. Controllers which are covered in small sensors that detect that lasers as they go by. The system cleverly integrates all of this data to determine the rotation of the devices and their position in 3D space. High speed on-board IMUs in each device are used to aid in tracking.

Light comes from a whole bunch of stationary LEDs, plus a pair of active laser emitters. Sixty times every second, the LEDs flash, then one of the two spinning lasers sweep a beam of light across the room. Meanwhile, the receiver (read: VR headset or controller) is covered with little light sensors that detect the flashes and the laser beams. When a flash occurs, the headset simply starts counting (like a stopwatch) until it “sees” which one of its light sensor gets hit by a laser beam—and uses the relationship between where that light sensor exists on the headset, and when the beam hit the light sensor, to mathematically calculate its exact position relative to the base stations in the room.

Hit enough of those light sensors with a laser at the same time, and they form a “pose” — a 3D shape that not only lets you know where the headset is, but the direction it’s facing.

Hypereal lighthouse tracking system is composed of hardware and software components.

USB

HDMI

MCU

wireless

USB Dev 2

USB Host

USB Hub

**PC**

**Repeater**

**(Optional)**

**HMD**

USB Dev 1

HDMI to 2 DSI

HDMI IF

IMU sensor

light sensors

MCU

double screens

**Gamepad/handset \* 2**

**Lighthouse \* 2**

wireless

wireless

DSI

MCU

MCU

light sensors

Buttons & Triggers

Battery

Laser Modulation

Motor Control

wireless

(Only Rx)

Hardware components include：wireless repeater， lighthouse X 2, HMD, Gamepad X 2

Hardware components use USB and HDMI connecting to PC. USB is used for tracking data transmission, and HDMI is used for display. There are total two USB devices connecting to PC host 。One USB device represents HMD, IMU Sensor/Light Sensor packets are transfer through this data path. The other USB device represent Repeaters. Repeater is working as a packet proxy here, Gamepad IMU/key/Lightsensor packets are sent to Repeater over wireless channel first, and then packets are sent to PC through USB channel.

# Data Structure

## HMD【USB Device 1】

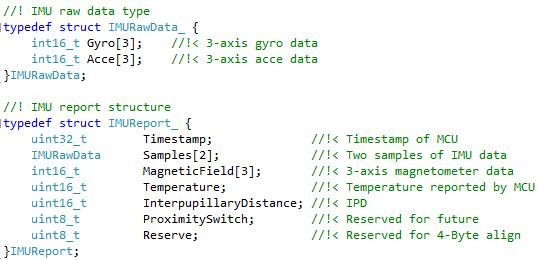
USB HID devices, Bulk endpoint. VID = 0x0484, PID = 0x5751

### Input Report

MCU send IMU data, Light sensor data, and Version data to PC with a specified frequency.

#### IMU report (2ms)

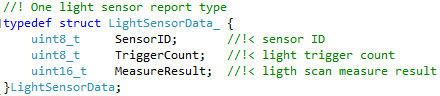
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | | **Type** | **Bytes** | **Description** |
| Timestamp | | uint32\_t | 4 | Timestamp recorded by MCU, accuracy: 1ms/digit |
| Samples[2] | Gyro[3] | int16\_t | 2 \* 3 \* 2 | Two samples of gyroscope, axis x/y/z. FSC was fixed to 2000dps. Unit: 15.0 LSB/dps |
| Acce[3] | int16\_t | 2 \* 3 \* 2 | Two samples of accelerometer, axis x/y/z, FSC was fixed to 16g. Unit: 2.048 LSB/mg |
| MagneticField[3] | | int16\_t | 2 \* 3 | One sample of magnetometer |
| Temperature | | uint16\_t | 2 | Temperature reported by IMU, high 8-bit was integer part, low 8-bit was decimal part, unit: degree Celsius |
| InterpupillaryDistance | | uint16\_t | 2 | Interpupillary distance |
| ProximitySwitch | | uint8\_t | 1 | Proximity switch, reserved |
| Reserve | | uint8\_t | 1 | 4-byte alignment, reserved |
| **Total Bytes:** | | | **40** |  |

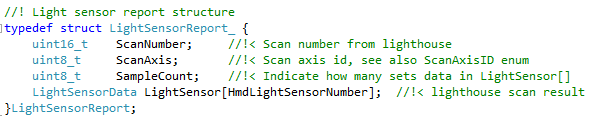


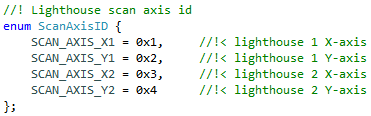
#### Light sensor report (≈8.3ms)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | | **Type** | **Bytes** | **Description** |
| ScanNumber | | uint16\_t | 2 | Indicate the scan sequence for alignment data |
| ScanAxis | | uint8\_t | 1 | Indicate scan axis  1 - X1  2 - Y1  3 - X2  4 - Y2 |
| SampleCount | | uint8\_t | 1 | Indicate length of valid data |
| LightSensor  [32] | SensorID | uint8\_t | 1 \* 32 | Indicate sensor id |
| TriggerCount | uint8\_t | 1 \* 32 | Indicate scan quality. (hardcode == 1) |
| MeasureResult | uint16\_t | 2 \* 32 | Indicate time parameter |
| **Total Bytes:** | | | **132** |  |

The max length is 136, minimum 8（Variable length，depending on SampleCount，SampleCount may == 0）

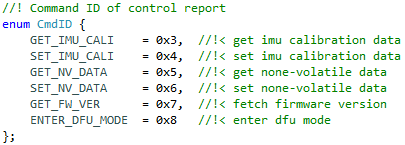






### Control Report

For both Send packet and Receive packet, the first byte of Control Report was fixed to Command ID. The corresponding value of different Command ID was shown as below.



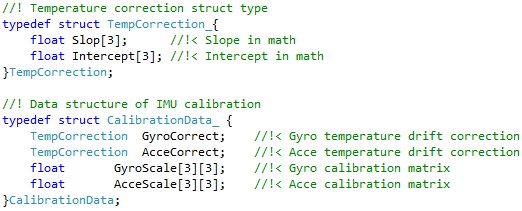
#### Get IMU calibration (Have reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte ，fix to 0x3 |
| **Total Bytes:** | | **1** |  |

* Receive packet:

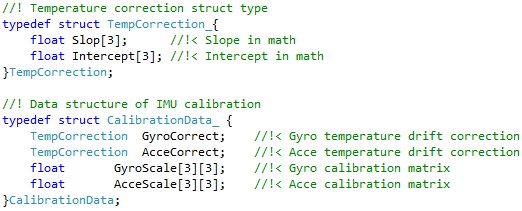
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | | **Type** | **Bytes** | **Description** |
| Cmd | | uint8\_t | 1 | Cmd byte，fix to 0x3 |
| GyroCorrect | Slop[3] | float | 4 \* 3 | Gryo temperature drift slope |
| Intercept[3] | float | 4 \* 3 | Gyro temperature drift intercept |
| AcceCorrect | Slop[3] | float | 4 \* 3 | Accelerometer temperature drift slope |
| Intercept[3] | float | 4 \* 3 | Accelerometer temperature intercept |
| GyroScale[3][3] | | float | 4 \* 3 \* 3 | Gyro scale and cross axis sensitivity |
| AcceScale[3][3] | | float | 4 \* 3 \* 3 | Acce scale and cross axis sensitivity |
| **Total Bytes:** | | | **121** |  |



#### Set IMU calibration (No reply)

* Send packet:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | | **Type** | **Bytes** | **Description** |
| Cmd | | uint8\_t | 1 | Cmd byte，fix to 0x4 |
| GyroCorrect | Slop[3] | float | 4 \* 3 | Gyro temperature drift slope |
| Intercept[3] | float | 4 \* 3 | Gyro temperature drift intercept |
| AcceCorrect | Slop[3] | float | 4 \* 3 | Accelerometer temperature drift slope |
| Intercept[3] | float | 4 \* 3 | Accelerometer temperature intercept |
| GyroScale[3][3] | | float | 4 \* 3 \* 3 | Gyro scale and cross axis sensitivity |
| AcceScale[3][3] | | float | 4 \* 3 \* 3 | Acce scale and cross axis sensitivity |
| **Total Bytes:** | | | **121** |  |



#### Get NV data (Have reply)

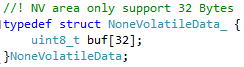
NV(No-Volatile)，is 32 bytes flash space in MCU，which is used to keep data when power off。

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte，fix to 0x5 |
| **Total Bytes:** | | **1** |  |

* Receive packet:

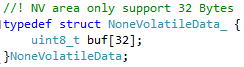
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte, fix to 0x5 |
| Data[32] | uint8\_t | 1 \* 32 | 32 bytes data |
| **Total Bytes:** | | **33** |  |



#### Set NV data (No reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte，fix to 0x6 |
| Data[32] | uint8\_t | 1 \* 32 | 32 bytes data |
| **Total Bytes:** | | **33** |  |



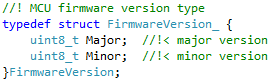
#### Get MCU version (Have reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte，fix to 0x7 |
| **Total Bytes:** | | **1** |  |

* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte，fix to 0x7 |
| FirmwareVersion | uint16\_t | 2 | MCU firmware version |
| **Total Bytes:** | | **3** |  |



#### Enter DFU mode (No reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | Cmd byte，fix to 0x8 |
| **Total Bytes:** | | **1** |  |

DFU is a special mode，which is used for firmware upgrade。

## Repeater【USB Device 2】(OPTIONAL)

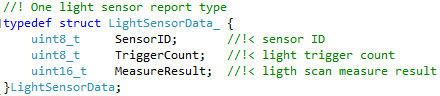
USB HID Device，Bulk transmission，VID：0x0485，PID：0x5752。

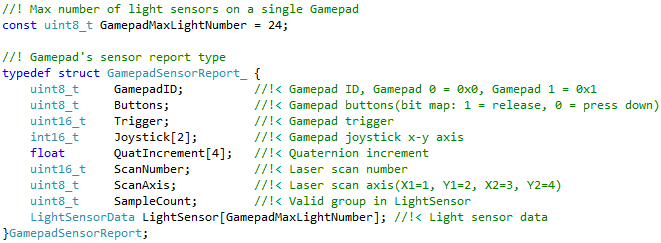
### Input Report

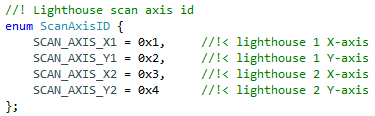
Gamepad and Lighthouse will send data to PC with a specified frequency.

#### Gamepad sensor report (≈8.3ms)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | | **Type** | **Bytes** | **Description** |
| GamepadID | | uint8\_t | 1 | Gamepad ID（0、1） |
| Buttons | | uint8\_t | 1 | 6 Keys |
| Trigger | | uint16\_t | 2 | Trigger analog quantity 0~ 255 |
| Joystick[2] | | int16\_t | 2 \* 2 | Joystick X + Y analog quantity 0~255 |
| QuatIncrement[4] | | float | 4 \* 4 | quaternion increment |
| ScanNumber | | uint16\_t | 2 | Indicate the scan sequence. |
| ScanAxis | | uint8\_t | 1 | Indicate the scan axis  1 - X1  2 - Y1  3 - X2  4 - Y2 |
| SampleCount | | uint8\_t | 1 | Indicate valid sample count |
| LightSensor  [24] | SensorID | uint8\_t | 1 \* 24 | Sensor id |
| TriggerCount | uint8\_t | 1 \* 24 | Scan quality |
| MeasureResult | uint16\_t | 2 \* 24 | Time parameter |
| **Total Bytes:** | | | **124** |  |







The gamepad contains 5 keys and 3 axis analog quantity：

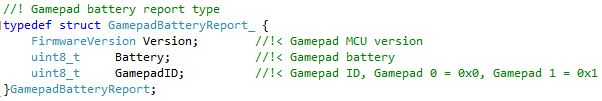
* Joystick ：mapping Buttons bit 0
* Home ：mapping Buttons bit 1
* Assist ：mapping Buttons bit 2
* Trigger ：mapping Buttons bit 3
* Side ：mapping Buttons bit 4
* X + Y analog quantity：mapping Joystick[0] and Joystick[1]
* Trigger analog quantity：mapping Trigger

Remarks：The bit value of the button indicates the button status. 0 for un-press and 1 for press.

。

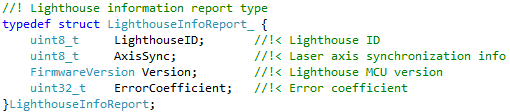
#### Gamepad battery report (6s)

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| FirmwareVersion | uint16\_t | 2 | MCU firmware version |
| Battery | uint8\_t | 1 | Battery of gamepad, 0~252 (bit0和bit1固定为 0) |
| GamepadID | uint8\_t | 1 | Gamepad ID, 0 or 1 |
| **Total Bytes:** | | **4** |  |



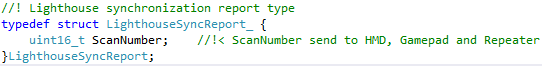
#### Lighthouse version report (6s)

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| LighthouseID | uint8\_t | 1 | Lighthouse ID, 0 or 1 |
| AxisSync | uint8\_t | 1 | Axis synchronization |
| FirmwareVersion | uint16\_t | 2 | MCU firmware version |
| ErrorCoefficient | uint32\_t | 4 | Error coefficient |
| **Total Bytes:** | | **8** |  |



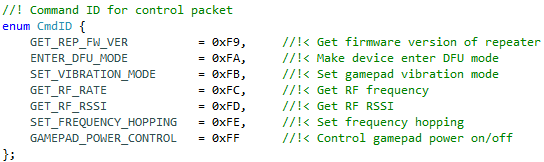
#### Synchronization report(≈33.3ms)

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| ScanNumber | uint16\_t | 2 | Sync sequence. Increase by 1 every 33 milliseconds |



### Control Report

For both Send packet and Receive packet, the first byte of Control Report was fixed to Command ID. The corresponding value of different Command ID was shown as below.



#### Set frequency hopping (Have reply)(optional)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfe |
| Freq | uint8\_t | 1 | frequency to be setted |
| **Total Bytes:** | | **2** |  |

* Receive packet:

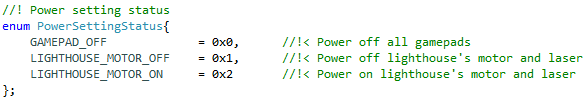
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfe |
| Status | int8\_t | 1 | Acknowledge by MCU |
| **Total Bytes:** | | **2** |  |

Status = 0 means success, other value on error

#### Power control (Have reply)(optional)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xff |
| Power setting | uint8\_t | 1 | See the definition below |
| **Total Bytes:** | | **2** |  |



* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xff |
| Status | int8\_t | 1 | Acknowledge by MCU |
| **Total Bytes:** | | **2** |  |

Status = 0 means success, other value on error

#### Query channel’s RSSI (Have reply)(optional)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfd |
| **Total Bytes:** | | **1** |  |

* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfd |
| RSSI | uint8\_t | 1 | RSSI returned |
| **Total Bytes:** | | **2** |  |

#### Get current frequency (Have reply)(optional)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfc |
| **Total Bytes:** | | **1** |  |

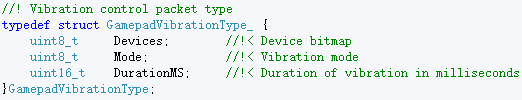
* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfc |
| Freq | uint8\_t | 1 | Current frequency |
| **Total Bytes:** | | **2** |  |

#### Set vibration mode (No reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfb |
| Devices | uint8\_t | 1 | Which device vibrate, each bit correspond to one device,  e.g. bit 0 correspond to gamepad device 0. |
| Mode | uint8\_t | 1 | Vibration mode |
| Duration | uint16\_t | 2 | Vibration duration, unit: millisecond |
| **Total Bytes:** | | **5** |  |



* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfb |
| Status | int8\_t | 1 | Acknowledge by MCU, success = 0x0, error = 0x1 |
| **Total Bytes:** | | **2** |  |

Due to the transmission interval restriction of the wireless channel（Each 33.3ms can only be transmitted one control packet）：

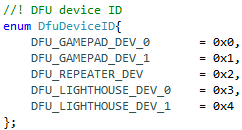
* If uplayer knows the total vibration time，set the duration ms directly.
* If uplayer doesn’t know the total vibration time，the duration can set to 33ms，and continue to send the pkt every 33ms。

Vibration hardware：Alps AFT14A901B：

1. Vibration Mode： 0-no Vib 1-Weak 2-Strong（Alps built-in attribute ）
2. Vibration instructions can be issued two times per 33.3ms. It will be cached if issued too fast, specially It also will dropped if the cache is full.
3. Each instruction is supported on a single device settings. (Both A and B vibrate in one instruction is not supported)

#### Enter DFU mode (No reply)(optional)

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xfa |
| devID | uint8\_t | 1 | Which device enter dfu mode |
| **Total Bytes:** | | **2** |  |



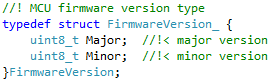
#### Get repeater firmware version(Have reply)

* Send packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | 0xF9 |
| **Total Bytes:** | | **1** |  |

* Receive packet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Bytes** | **Description** |
| Cmd | uint8\_t | 1 | CMD byte，fix to 0xF9 |
| FirmwareVersion | uint16\_t | 2 | MCU firmware version |
| **Total Bytes:** | | **3** |  |



# Data Packet

In order to insure the integrity of data, we encapsulate our data into a packet which have Header, Tail.

Its format was shown in the table below.

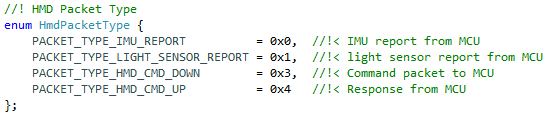
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Header（6Byte）** | | | **Body** | **Tail（3Byte）** | |
| Magic Number  （2 Bytes） | Length  （2 Bytes） | Type  （2 Bytes） | Payload Data  （Length Bytes） | CRC  (1Byte) | Magic Number  （2 Bytes） |
| 0x2B2B | -- | -- | -- | -- | 0xB2B2 |

We stipulate that:

* Magic Number of Header was fixed to 0x2B2B.
* Magic Number of Tail was fixed to 0xB2B2.
* Length indicate how many bytes we have in Body(Payload Data).
* Type definition see sections below.
* Payload Data was the actual data we want to transfer to/from MCU.
* One byte CRC was the accumulation addition of Payload Data, NOT the traditional CRC check.

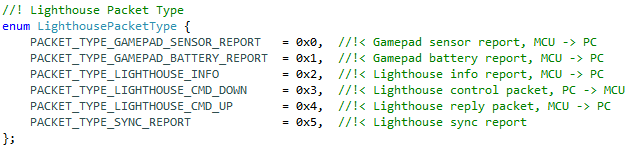
## Type definition for HMD

Type definition:



## Type definition for Repeater(optional)

Type definition:



# Driver Architecture

## Overall view

USB devices

libusb driver

class LIBUSB\_Device

class HVR\_Hmd

class HVR\_Gamepad

libusb-win32 API

Hardware

Kernel mode

User mode

HVR\_Lighthouse Project

## Class HVR\_Hmd

HVR\_Hmd hmd(imuReport, lightReport, gamepadReport);

//Create object, register Callback function

hmd.Open();

//Open the USB device

//start background thread

hmd.Close();

//kill background thread

//Close the USB device

USB bulk read

**Background Thread**

check packet

handle packet

IMU report

Call imuReport()

Call lightReport()

Light report

Gamepad report

Call gamepadReport()

Control report reply

Save the data, then wakeup the thread who was waiting for this reply

## Class HVR\_Repeater (Optional)

HVR\_Repeater repeater(gpdSensor, gpdBattery, lthInfo, lthSync);

//Create object, register Callback function

repeater.Open();

//Open the USB device

//start background thread

repeater.Close();

//kill background thread

//Close the USB device

USB bulk read

**Background Thread**

check packet

handle packet

Sensor report

Call gpdSensor()

Call gpdBattery()

Battery report

Lighthouse Info report

Call lthInfo()

Control report reply

Save the data, then wakeup the thread who was waiting for this reply

Lighthouse Sync report

Call lthSync()