[The JSON File (Lighthouse Devices) · ValveSoftware/openvr Wiki · GitHub](https://github.com/ValveSoftware/openvr/wiki/The-JSON-File-(Lighthouse-Devices))

[SteamVR Documentation: How to make a Driver](https://github.com/ValveSoftware/openvr/wiki/Driver-Documentation)

[SteamVR HDK Discussion Board](https://steamcommunity.com/app/507090/discussions/)

In order to get HDK Software, must be a SteamVR Tracking Licensee. We must resort to 3rd-party applications in order to reach our goal.

## [API Documentation:](https://github.com/ValveSoftware/openvr/wiki/API-Documentation)

* [IVRSystem](https://github.com/ValveSoftware/openvr/wiki/IVRSystem_Overview) - Main interface for display, distortion, tracking, controller, and event access.
* [IVRChaperone](https://github.com/ValveSoftware/openvr/wiki/IVRChaperone_Overview) - Provides access to chaperone soft and hard bounds.
* [IVRCompositor](https://github.com/ValveSoftware/openvr/wiki/IVRCompositor_Overview) - Allows an application to render 3D content through the VR compositor.
* [IVROverlay](https://github.com/ValveSoftware/openvr/wiki/IVROverlay_Overview) - Allows an application to render 2D content through the VR compositor.
* [IVRRenderModels](https://github.com/ValveSoftware/openvr/wiki/IVRRenderModels_Overview) - Allows an application access to render models.
* [IVRScreenshots](https://github.com/ValveSoftware/openvr/wiki/IVRScreenshots_Overview) - Allows an application to request and submit screenshots.
* [IVRInput](https://github.com/ValveSoftware/openvr/wiki/SteamVR-Input) - Allows an application to define and query invocable actions (and action sets) so that users may create, edit, and share custom bindings with any supported device.

*“An OpenVR driver is a software that introduces VR devices into the SteamVR system. You would want to write a driver when you manufacture new hardware (or virtual hardware) such as a headset or a controller. To write a driver, you should start with writing a HmdDriverFactory() function and include openvr\_driver.h.”*

Drivers -> objects -> openVR software

## [Making a Driver](https://github.com/ValveSoftware/openvr/wiki/Driver-Documentation):

[Driver Local Registration](https://github.com/ValveSoftware/openvr/wiki/Local-Driver-Registration)

## The JSON File

*The JSON file begins its life with a minimum of sensor positions and orientations, but is augmented throughout the design and integration process to include IMU data, lens distortion data, and other metadata used by SteamVR™. Finally, calibration routines refine the original sensor geometry values to the exact sensor locations on a specific object and rewrite the JSON file, resulting in a unique JSON file for every tracked object.*

**Manufacturer:** *string* that represents manufacturer’s company name

“Manufacturer” : “valve”

**Model\_number:** *string* value representing model number of object assigned by manufacturer

“Model\_number” : “REF-HMD”

**“Device\_class”:** *string* value can be set to “hmd” or “controller”

“Device\_class” : “controller” Setting it to controller tells steamVR to render it in VR

“Device\_class” : “hmd” Setting it to “hmd” tells SteamVR to associate the tracked object into the correct POV into VR

**“Device\_vid”:** the usb vendor identification *int*

“Device\_vid” : 10462

**“Device\_pid”:** the usb product identification *int*; one PID per model

“Device\_pid” : 8960

**“Device\_serial\_number”:** *string* representing unique serial number of device; used to match JSON files to real objects; each object has its own serial number

“Device\_serial\_number” : “LHR-F8DE9EBE”

**“Lighthouse\_config”:** an object containing three member arrays; each array index corresponds to a sensor on tracked object; modelPoints, modelNormals, and channelMap define physical placement of the sensor, its orientation, and its electrical connection (port #) SteamVR uses tracking data from sensor on particular port and uses lighthouse\_config to associate the signal with specific sensor on the tracked object and establish sensor geometry to figure out current position of the object

**“modelNormals”**: an array of [x, y, z] unit vectors that specify direction the sensor faces

At 45o: "modelNormals" : [ [ -0.7071, 0.7071, 0.0 ], [ -0.7071, 0.0, -0.7071 ], [ -0.7071, 0.0, 0.7071 ], [ -0.7071, 0.0, -0.7071 ], [ -0.7071, -0.7071, 0.0 ] ]

**“modelPoints”**: an array of [x, y, z] coordinates that specify location of the center optical sensor’s photosensitive area in the coordinate system (expressed in meters)

**“channelMap”**: array of port numbers; an element for each sensor on the object;

**“Head”** (Heading)**:** Object using members that make the object correlate with the object in the real world

*HMD Use Case:*  When the object is using this property, the aforementioned head variable positions SteamVR’s coordinates system to the object that is being tracked.

The “origin” of steamvr is located between the users pupils with y being up, x pointing to the right, and z pointing out alone the line of sight. Adjustments may be needed if the model has a coordinate system different from SteamVR

*Controller:* When the objects is being used as a controller the head positions the rendered model that is shown in SteamVR to the tracked object. Render model ois created in the Steam VR coordinate system (referred to above). A method for determining the head variable is to export the render model as an STL file, importing the file into the 3D CAD space following up with measuring the plus x and plus

**“Plus\_x”:**

HMD: represents the head’s x axis (unit vector)

Controller: Positive X from IMU (unit vector)

**“Plus\_z”:**

HMD: represents the head’s z axis

Controller: Positive Z from IMU (unit vector)

**“Position”:**

HMD: coordinate located at the midpoint between the average user’s physical pupil location

Controller: coordinate that positions the IMU within the model

**“Render\_model”:** holds string value that specifies name of subfolder within steamVR “rendermodels” folder that holds the default rendermodel for the object

“Render\_model” : “ref\_controller”

**"display\_edid"** : [ "", "" ],

**“direct\_mode\_edid\_vid”:** tells SteamVR™ which display to use when displaying VR to the HMD

"direct\_mode\_edid\_vid" : xxxxx”

**“Direct\_mode\_edid\_pid” :** tells SteamVR™ which display to use when displaying VR to the HMD

"direct\_mode\_edid\_vid" : xxxxx”

**“Tracking\_to\_eye\_transform”:** stores the lens calibration information; lens is circularly symmetric and parallel to display panel; coordinate system is independent for each eye

* **“distortion”, “distortion\_blue”, “distortion\_red”**
* **“Intrinsics”:** 3x3 matrix that describes linear project after distortion is corrected; 5 values based on focal length and center of project
* **“Extrinsics”:** 3x4 matrix that describes the rotation and translation of lens and panel system;
* **"grow\_for\_undistort"** : set to 0.6
* **"undistort\_r2\_cutoff"**: set to 1.5

**“Type”** = always set to “Lighthouse\_HMD,” regardless of controller

Pseudocode:

We start with

**“Lighthouse\_Config”** :

{

“channelMap” int[]

“modelNormals” double[]

“modelPoints” double[]

}

**“Head”** :

{

“plus\_x” int[xyz]

“plus\_z” int[xyz]

“position” [0,0,0]

}

**“Imu”** :

{

"acc\_scale"

"acc\_bias"

"gyro\_scale"

"gyro\_bias"

“plus\_x” int[xyz]

“plus\_z” int[xyz]

“position” [0,0,0]

}

Input: Json sent from controller?

Output: controller object ot steamVR

Figure out a way to prepare before the boards arrive

Mock up controller prototype using the boards

Unity takes data from open vr (from controller) and puts it into the game, steam vr runs game,

Drivers translates controller data to open vr as an object

Unity finds control object and imports it into game software

Data from controllers →openVR →unity →game(steamVR)