CYBERITH SDK

Unity Core – Integration Guideline

Cyberith GmbH
Teslastraße 6
3100 St. Pölten
Austria
FN 410899p

For any questions, please contact
Cyberith Support
support@cyberith.com
+43 1 890 17 13

Table of Contents

Prerequisites

Window 7 or newer

Unity Game Engine

Recommended: Unity 2017.4.17 or newer – **Compatible:** All major versions
http://unity3d.com/

For compiling C# Code: Visual Studio

This is a requirement of Unity for using it with C# Code. Not a specific requirement of the Cyberith SDK.

Recommended: Visual Studio Community 2017
https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

Setup Example Project

Setting up the Cyberith Virtualizer SDK in a Unity project is done by following five steps:

1. Create new empty project

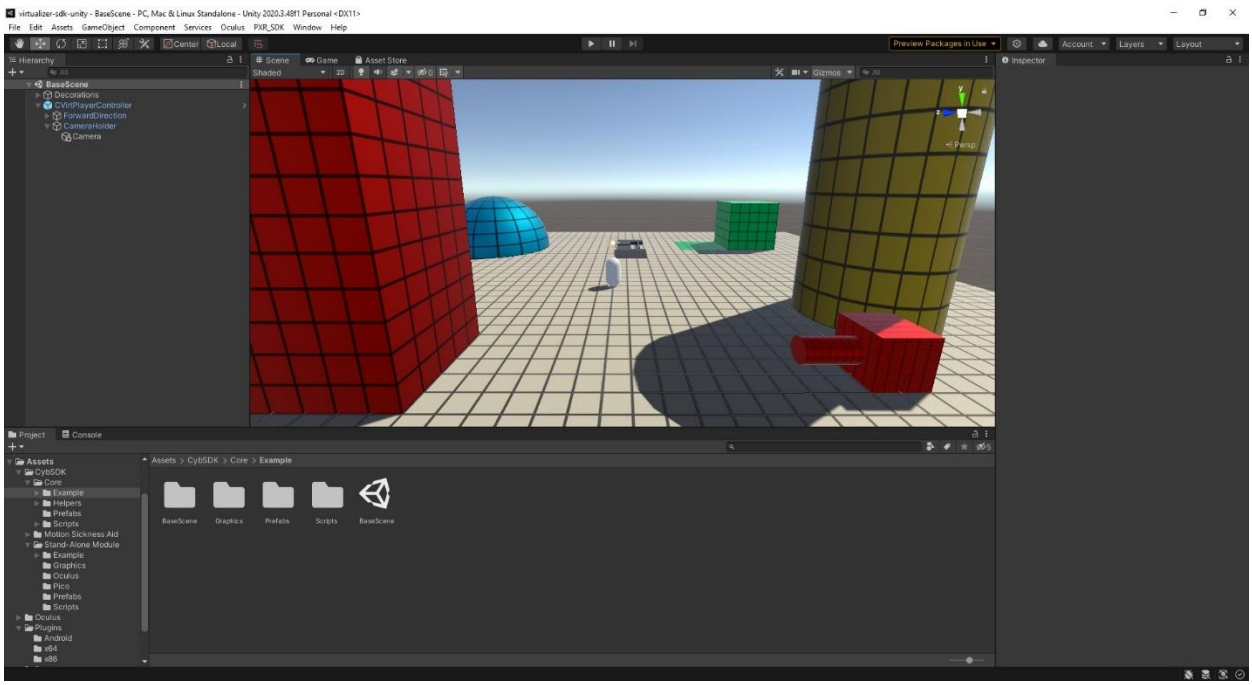
Create a new empty Unity project to first play around with the capabilities of the CybSDK package. You can use any Unity version you prefer; the Virtualizer SDK has been tested to work with all major versions.

2. Import the CybSDK Unity package

Cyberith Virtualizer SDK is distributed as a custom asset package. To use it in your project you press *"Assets → Import Package → Custom Package …* " and search for the CybSDK file before pressing *"Open"*. In the following "Import Unity Package" dialogue all assets should be selected by default and you accept by pressing *"Import"*

3. a) Open Example Scene

CybSDK comes with a prebuilt example scene to demonstrate the packages capabilities. You can find it under **CybSDK/Core/Example/BaseScene**. The only thing left to do is to add your HMD prefab to the CVirtPlayerController prefab. This prefab is described in more detail in chapter CVirtPlayerController Prefab.



3. b) Integrate Virtualizer in your own scene

If you want to integrate the Virtualizer into your own scene, navigate into the CybSDK/Core/Prefabs folder and Drag & Drop the CVirtPlayerController Prefab into your Scene Hierarchy. This prefab is preconfigured and integrates the Virtualizer in your scene.

4. a) Activate Unity VR (only Unity 2017.2 or newer)

One option is to follow the steps described in the **Unity User Manual/XR/VR overview** to activate Unity VR: https://docs.unity3d.com/Manual/VROverview.html

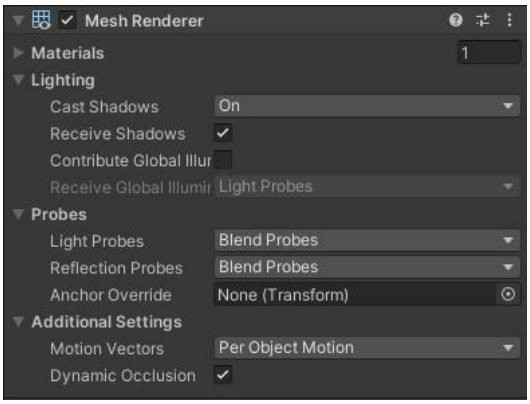
4. b) Import HMD asset package (compatible with all Unity versions)

Alternatively, you can download and import your preferred HMD package (e.g. SteamVR Plugin) from the Unity Asset Store and import it to your project.

The CVirtPlayerController prefab is prepared to work with multiple different setups. For using the Virtualizer with a standard HMD, simply drag and drop the HMD prefab into the *"CameraHolder"*.

You may also want to change the player body by replacing the "ExampleBody" with your custom mesh. Alternatively can also make the "ExampleBody" and its shadow invisible by deactivating its "Mesh Renderer (by unchecking the according checkbox).

5. Start your Virtualizer Experience



Now, you should be ready and set up to try out the BaseScene with your Virtualizer.

Please check two points:

- The Locomotion: Walk on your Virtualizer to see your character moving accordingly.
- The Haptic Feedback: Walk to the green elevator. When the elevator moves up or down, you should feel Vibrations coming from the Virtualizer baseplate.

If you run into any problems with the steps above feel free to contact us: support@cyberith.com

Note: The contents of the plugin folder “Stand-Alone Module” is only meant to be used for projects that involve our product the “Virtualizer Stand-Alone Module” (VirtSAM). The VirtSAM is an extension to the Virtualizer that allows for a wireless connection between the Virtualizer and standalone headsets like the Meta Quest 2 and VIVE Focus 3.

No Virtualizer Hardware? – No Problem!

You may want to create applications for the Cyberith Virtualizer without having access to a real hardware device.
For this purpose, we added two kinds of virtual devices emulating a Virtualizer:

- Keyboard – WASD for movement and QE for rotation
- Controller (Xbox 360 & Xbox One Controller) – left joystick for movement and right joystick for rotation. Please plug the Xbox Controller to your PC per USB cable to ensure proper functionality.

This means:
If you don't have a Virtualizer available you can still test the functionality of your implementation with the help of an Xbox Controller or with the help of your keyboard!

In case no Virtualizer is plugged, the system will automatically use the Xbox Controller. If neither a Virtualizer nor an Xbox Controller are plugged, the system will automatically use the keyboard.
Alternatively to the automatic selection, these inputs methods can be configured in the **CVirtDeviceController** – as described in the next chapter.

Note:

- Be aware, that these settings will still be active in a built executable and allow to test the finished (built) application without the need of real Virtualizer hardware.
- Currently, you can not walk backwards with an Xbox Controller. If you press the left joystick back, the avatar still walks forwards. That does not mean that you can't walk back with the Virtualizer! It is a problem caused by the Xbox controller implementation.
- The Xbox Controller allows you to basically check on the functionality of the haptic feedback (although it can only rumble with one frequency and not in many different ones like the Virtualizer.)
  The keyboard input does not allow you to check on the haptic feedback functionality.
- For development and testing purposes it can be practical to use an Xbox Controller even if you have a real Virtualizer next to you. Using such a controller, you don't need to stand up from your comfortable chair for every single test ;)

CVirtPlayerController Prefab

The CVirtPlayerController Prefab is a player object that includes all the Virtualizer functionality you need to move around a player in a scene. This prefab handles the Virtualizer movement as well as the Virtualizer haptic feedback. The Prefab is included in our BaseScene example. You can also include it into your own custom projects.

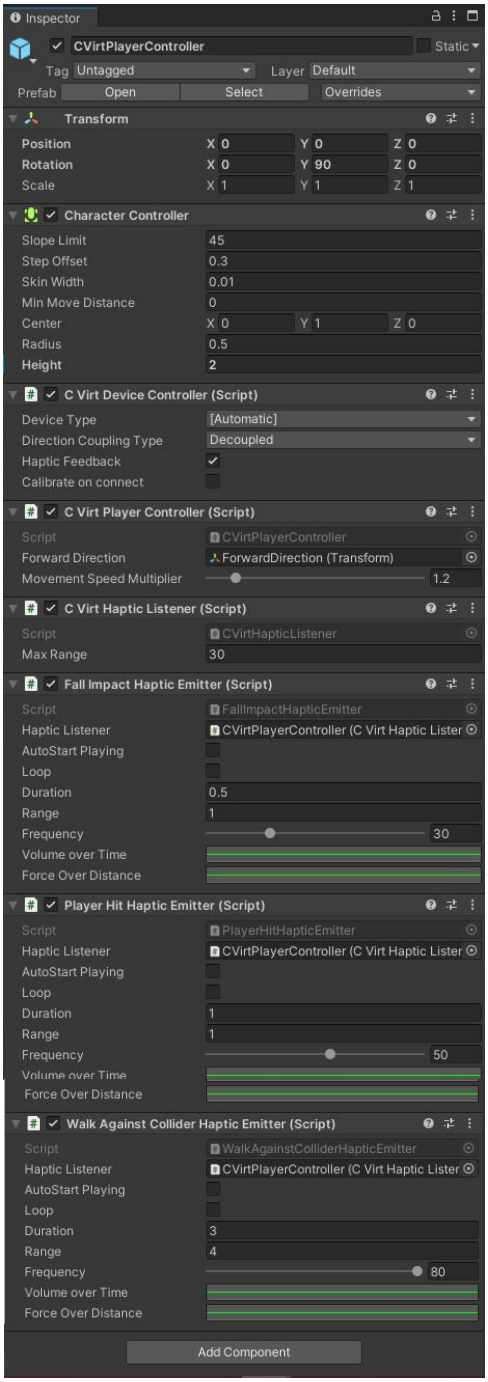
The Character Controller is a Unity component responsible for handling the movement in Unity.

The other scripts are provided by Cyberith and are described in the following chapters in greater detail.

You can see the following types of scripts:

- Device Controller: Handles the connection to the Virtualizer
- Player Controller: Handles the Virtualizer's locomotion functionality
- Haptic:
    - Haptic Listener
    - (multiple) Haptic Emitters

If you want to use this prefab, but not all the scripts, you can deactivate each script by unchecking the specific script.



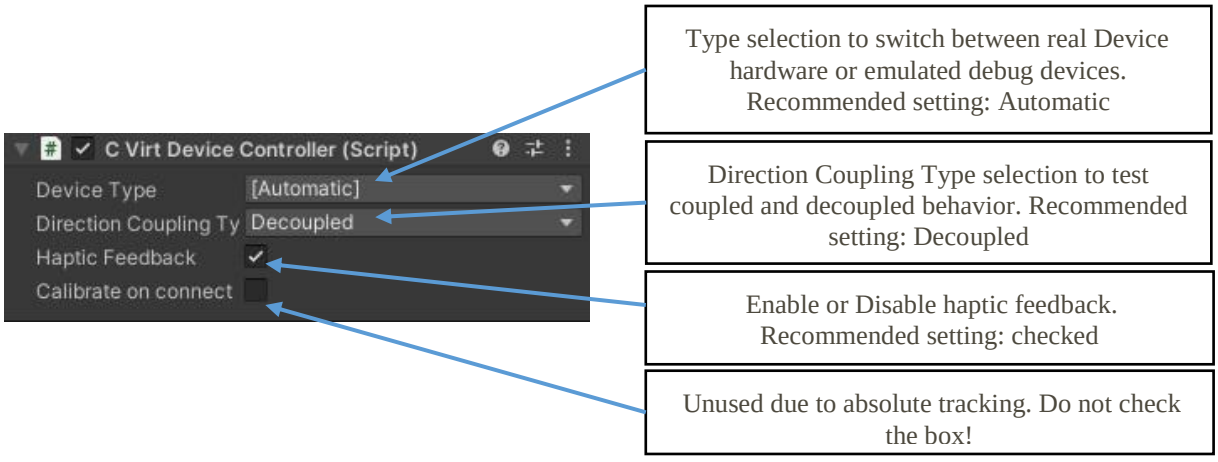
Editor Settings

The CybSDK Unity package consists of four major scripts handling different functionalities of the Virtualizer. Additionally to the major scripts, we added four specific haptic emitter scripts to demonstrate exemplary use cases of the haptic unit.

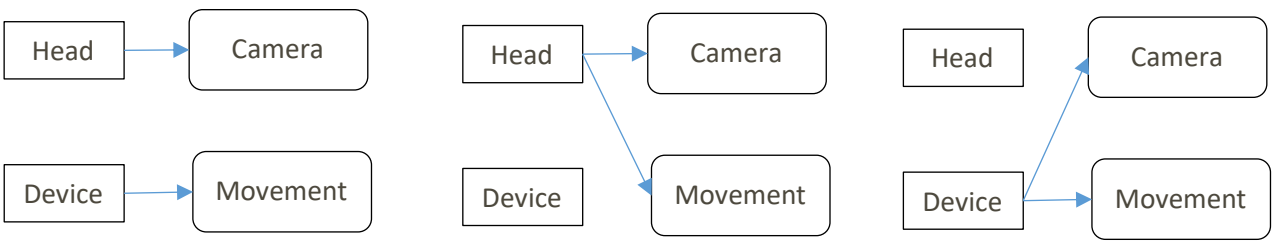
Three of the four major scripts can be found directly in the “CVirtPlayerController”:

CVirtDeviceController (Script)

This script has full authority over the Virtualizer device. In this script, the device is selected and a connection is established and managed.



The Direction Coupling Type defines how the direction of the camera and movement are coupled to the direction of the head or the rotation of the Virtualizer device (player orientation). Decoupled is the standard type and is highly recommended for using with the native Virtualizer device. The coupled types are recommended for using with emulated debug devices (e.g., keyboard or Xbox controller). Device Based Direction is not supported for using with a native Virtualizer device as the ring would then rotate the camera, which could be very uncomfortable for users.



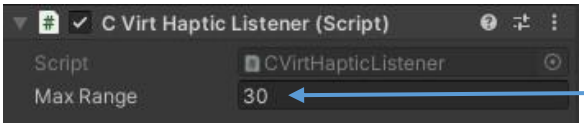
When using Head Based Direction, make sure that your VR camera is tagged as *MainCamera*.

CVirtPlayerController (Script)

This script moves the pawn (virtual character/avatar) according to the Virtualizer input, as described in the chapter “Example Usage”.

CVirtHapticListener (Script)

This script receives signals causing haptic feedback. It activates the Virtualizer's haptic vibration unit accordingly.
The "haptic signals" are emitted by all active HapticEmitters within the defined range. These HapticEmitters are explained in the next point.



Maximum range of the Haptic Listener. All emitters inside this range are considered by the Listener. If a Haptic Emitter is further away, it can not be "heard" from. Recommendation: This number should be at least as high as the highest defined range of all of your Haptic Emitters.

The Haptic Listener sends two parameters for the haptic unit. One of these two parameters is the frequency, the haptic unit vibrates with. The other parameter is the volume (= "strength") the haptic unit vibrates with.

As the resonance frequency of implemented haptic unit is around 40 Hz, the Haptic Listener does not send out frequencies from 35 to 45 Hz. This avoids loud and uncomfortable vibrations. Instead of sending frequencies of 35-39 Hz it sends 34 Hz and instead of sending frequencies of 40-45 Hz it sends the frequency of 46 Hz. You might notice this behavior if you select one of these frequencies in a haptic emitter, which is explained in the next chapter.

In case multiple Haptic Emitters are active and within range, the HapticListener makes a weighted average of all the frequencies and sums up the volume caused by all Emitters. The volume of an emitter is defined by two curves ("Volume Over Time" & "Force over Distance") for each Emitter individually. Note, that the overall strength of the vibration can not exceed the maximum strength, independently of how many emitters you add.

The last one of the four main scripts, the generic Haptic Emitter “CVirtHapticEmitter”, is not inside the “CVirtPlayerController”. Instead, it can be attached to any object you want to cause haptic feedback. In the demo level, a generic Haptic Emitter is attached to the blue Sphere.

CVirtHapticEmitter (Script)

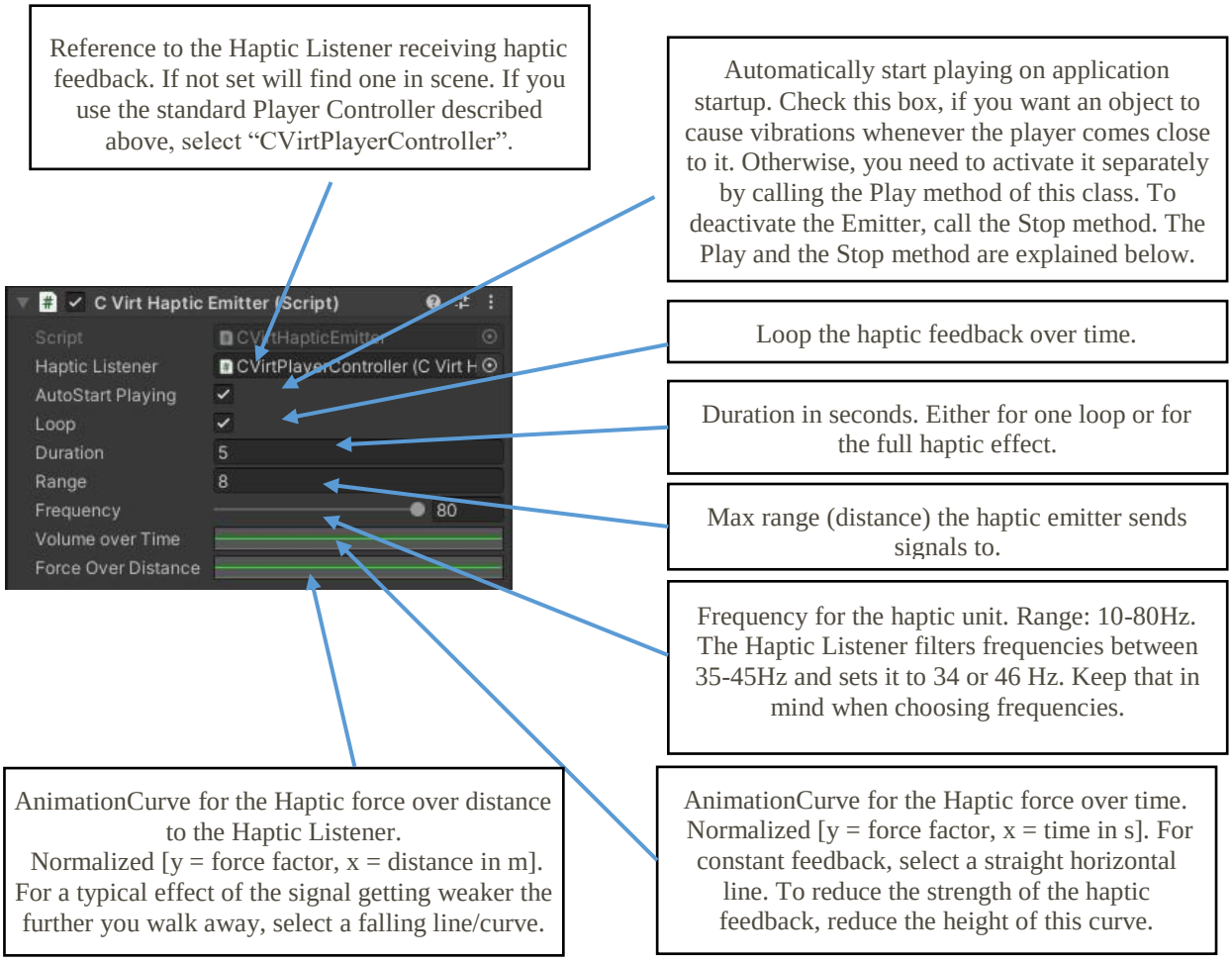
This script emits signals causing haptic feedback in a specific radius around it. These signals are received by a HapticListener, which in turn causes the Virtualizer baseplate to vibrate.

Attach a “CVirtHapticEmitter” to objects, that you want to cause haptic feedback!

Add an HapticEmitter by:

- Select an object you want to cause vibrations
- Click “Add Component”
- Select “Scripts” / “CybSDK” / “C Virt Haptic Emitter”

Once added, you will see this Haptic Emitter Script:



The generic Haptic Emitter “CVirtHapticEmitter” script is the base script for causing haptic feedback. You can either activate the haptic functionality by ticking the “AutoStart Playing” checkbox or call the Play() and Stop() method of the script to manually activate and deactivate the haptic emitter whenever you like to do so.

Additionally to the generic Haptic Emitter script, we added four specific Haptic Emitter scripts for special use cases.
In these specific Haptic Emitter Scripts, you can see the the Play and Stop methods in action. Read the code of these scripts and you can find examples of how these methods are used.

Specific Haptic Emitter Scripts

The following specific Haptic Emitter Scripts are part of Cyberith’s Unity Plugin to demonstrate exemplary use cases of haptic feedback. Feel free to add your own specific Haptic Emitter Scripts for whatever purposes you like.

- FallImpactHapticEmitter: Haptic is triggered when the player lands on the floor after falling/jumping from higher ground. The player has to fall for more than 0.1 seconds to activate the haptic feedback once reaching the ground. This script is attached to the CVirtPlayerController in the BaseScene example.

- MovementTriggeredHapticEmitter: Haptic is triggered when the object this script is attached to moves. This script is attached to the Elevator object in the BaseScene example.

- PlayerHitHapticEmitter: Haptic is triggered when the player is hit by an object. The object hitting the player has to call the HitImpact method of this script. This script is attached to the CVirtPlayerController in the BaseScene example. Furthermore, bullets (looking like big white balls) are spawned during runtime from the red canon (consisting of a cube and a cylinder). The bullets call the HitImpact method to cause haptic feedback upon hitting the player controller.

- WalkAgainstColliderHapticEmitter: Haptic is triggered when you walk against an obstacle. The haptic is activated when you walk against the obstacle for more than 0.5 seconds. This script is attached to the CVirtPlayerController in the BaseScene example.

If you add one of these scripts to an object in your scene, you don’t have to attach the generic Haptic Emitter script (“CVirtHapticEmitter”). These specific Haptic Emitters work autonomously.

The “AutoStart Playing” checkbox is not available on any of these scripts. These emitters use the Play and Stop methods of the emitter script.

SDK Documentation

C# SDK Documentation

For full documentation of the C# SDK take a look into the official online Documentation.
All classes and functions are documented via the XML documentation file **CybSDK.xml** and should show up in your Visual Studio IntelliSense.

CVirtDeviceController

For multiplayer games make sure to activate the preprocessor define:
CVirtDeviceController_Networking

GetDevice

Returns: UVirtDevice
Returns the Virtualizer device managed by the CVirtDeviceController.

UVirtDeviceUnityExtensions

This extension class holds multiple Unity specific methods.

GetMovementVector

Returns: Vector3
Returns the movement direction as a speed scaled vector relative to the current player orientation.

GetMovementDirectionVector

Returns: Vector3
Returns the movement direction as vector relative to the current player orientation.

GetPlayerOrientationVector

Returns: Vector3
Returns the orientation of the player as vector.

GetPlayerOrientationQuaternion

Returns: Quaternion
Returns the orientation of the player as quaternion.

CVirtHapticEmitter

Play

Start playing the Haptic Emitter by adding it to the Haptic Listener

Stop

Stop playing the Haptic Emitter by removing it from the Haptic Listener

Example Usage

Locomotion

```
using UnityEngine;
using CybSDK;

public class CVirtPlayerController : MonoBehaviour
{
	// ...

	// Update is called once per frame
	void Update()
	{
		UVirtDevice device = deviceController.GetDevice();

		if (device == null || !device.IsOpen()) return;

		// MOVE
		////////////
		Vector3 movement = device.GetMovementVector() * movementSpeedMultiplier;

		// ROTATION
		////////////
		Quaternion localOrientation = device.GetPlayerOrientationQuaternion();

		// Determine global orientation for characterController Movement
		Quaternion globalOrientation;

		// For decoupled movement we do not rotate the pawn --> HMD does that
		if (deviceController.IsDecoupled())
		{
			if (forwardDirection != null)
			{
				forwardDirection.transform.localRotation = localOrienta-
tion;
				globalOrientation = forwardDirection.transform.rotation;
			}
			else
			{
				globalOrientation = gameObject.transform.rotation * localO-
rientation;
			}
		}
		// For coupled movement we rotate the pawn and HMD
		else
		{
			gameObject.transform.rotation = localOrientation;
			globalOrientation = localOrientation;
		}

		Vector3 motionVector = globalOrientation * movement;
		characterController.SimpleMove(motionVector);
	}
}
```

Haptic Emitter

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


namespace CybSDK
{
    public class MovementTriggeredHapticEmitter : CVirtHapticEmitter
    {
        private Vector3 oldPosition;

        // Use this for initialization
        protected override void Start()
        {
            base.Start();
            autoStart = false;

            oldPosition = new Vector3(transform.position.x, transform.position.y,
transform.position.z);
        }

        // Update is called once per frame
        void FixedUpdate()
        {
            if (oldPosition != transform.position)
            {
                Play();
            }
            else
            {
                Stop();
            }

            oldPosition = transform.position;
        }

        void OnDisable()
        {
            Stop();
        }
    }
}
```