



# *CYBERITH* SDK

Unity – Integration Guideline

**Cyberith GmbH**

Teslastraße 6  
3100 St. Pölten  
Austria  
FN 410899p

**For any questions, please contact**

Cyberith Support  
[support@cyberith.com](mailto:support@cyberith.com)  
+43 1 890 17 13

---

# Table of Contents

Prerequisites .....	3
Setup example Project .....	4
1. Create new empty project .....	4
2. Import the CybSDK Unity package .....	4
3. Open example Scene .....	4
4. Import HMD unity package .....	5
5. Attach the HMD prefab to the CVirtPlayerController prefab .....	5
No Virtualizer Hardware? – No Problem! .....	5
Editor Settings .....	6
CVirtDeviceController .....	6
CVirtPlayerController .....	6
CVirtHapticListener .....	6
CVirtHapticEmitter .....	7
SDK Documentation .....	8
C# SDK Documentation .....	8
CVirtDeviceController .....	8
GetDevice .....	8
IsDecoupled .....	8
IVirtDeviceUnityExtensions .....	8
GetMovementVector .....	8
GetMovementDirectionVector .....	8
GetPlayerOrientationVector .....	8
GetPlayerOrientationQuaternion .....	8
Example usage .....	9

---

## Prerequisites

**DirectX (recommended: June 2010)**

<https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=35>

**Unity Game Engine (recommended: Unity 2017.4.17 or newer – Compatible with all major versions)**

<http://unity3d.com/>

**Visual Studio (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 an Unity project is done by following five simple steps:

## 1. Create new empty project

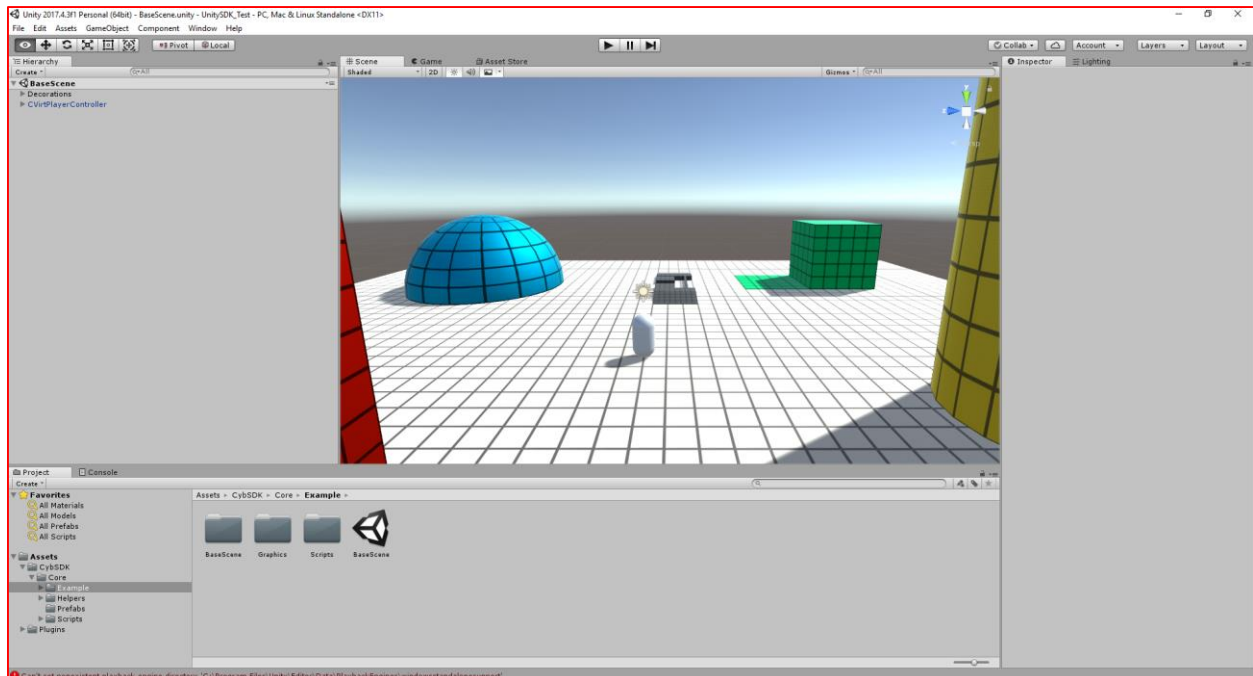
Create a new empty Unity project to first play around with the capabilities of the CybSDK package. You can use any Engine Version you prefer; the Virtualizer SDK has been tested to work with all major versions.

## 2. Import the CybSDK Unity package

Cyberith Virtualizer SDK is distributed as custom asset package. To use it in your project you simply 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. 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 our PlayerController.



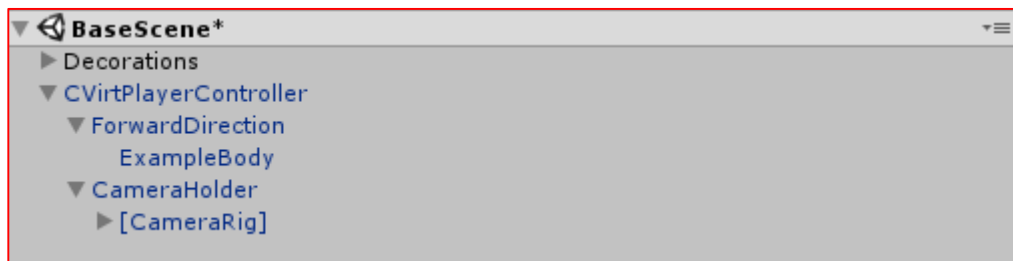
---

## 4. Import HMD unity package

Download and import your preferred HMD package (e.g. SteamVR Plugin) from the Unity Asset Store and import it to your project.

## 5. Attach the HMD prefab to the CVirtPlayerController prefab

The CVirtPlayerController prefab is prepared to work with multiple different setups. To customize it simply drag and drop the HMD prefab into the “CameraHolder”. You also may want to change the player body by replacing the “ExampleBody” with your custom mesh.



At this moment you should be ready and setup to play the BaseScene with your Virtualizer.

## No Virtualizer Hardware? – No Problem!

Many developers want to make applications for the Cyberith Virtualizer while having no access to a real hardware device. For this purpose, we added two kinds of virtual devices emulating a Virtualizer:

- Keyboard
- Controller (Xbox 360 Controller)

These can be configured in the CVirtDeviceController or will be automatically used if no suitable hardware is detected.

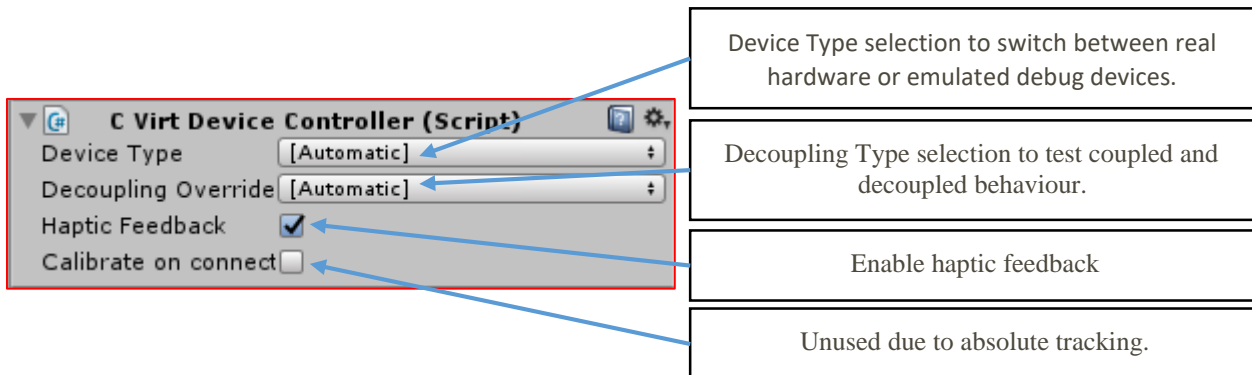
---

## Editor Settings

The CybSDK package consist of four major scripts handling different functionalities of the Virtualizer device.

### CVirtDeviceController

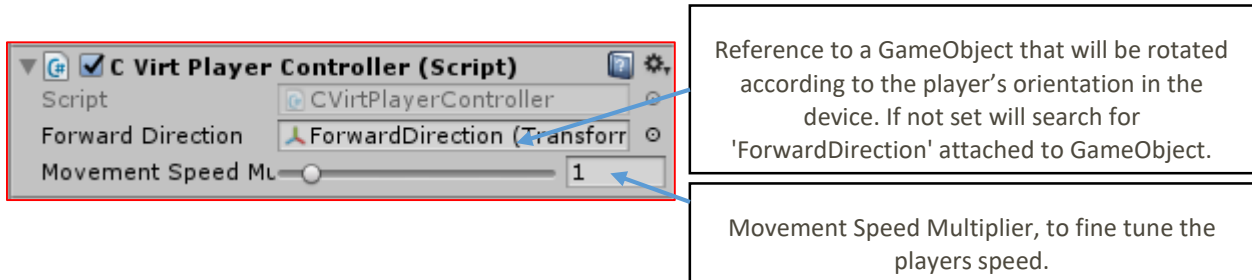
This script has full authority over the Virtualizer device. Here the device is selected, a connection established and managed.



Device Type	[Automatic]	Device Type selection to switch between real hardware or emulated debug devices.
Decoupling Override	[Automatic]	Decoupling Type selection to test coupled and decoupled behaviour.
Haptic Feedback	<input checked="" type="checkbox"/>	Enable haptic feedback
Calibrate on connect	<input type="checkbox"/>	Unused due to absolute tracking.

### CVirtPlayerController

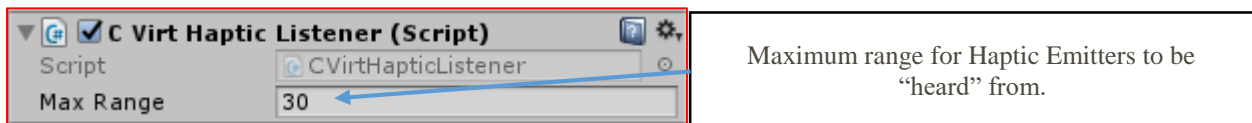
This script moves the pawn according to the Virtualizer input, as described in the Example Usage.



Script	CVirtPlayerController	Reference to a GameObject that will be rotated according to the player's orientation in the device. If not set will search for 'ForwardDirection' attached to GameObject.
Forward Direction	ForwardDirection (Transform)	
Movement Speed Multiplier	1	Movement Speed Multiplier, to fine tune the players speed.

### CVirtHapticListener

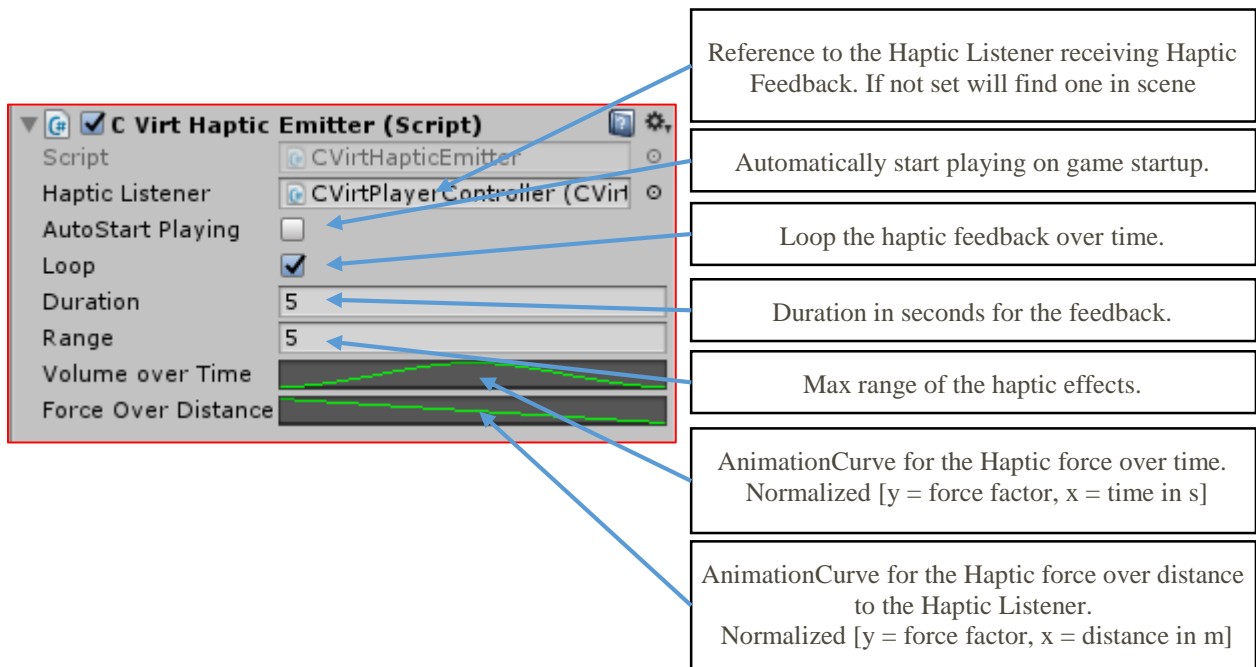
This script receives haptic feedback emitted by all active HapticEmitters in range and activates the Virtualizers haptic vibration unit.



Script	CVirtHapticListener	
Max Range	30	Maximum range for Haptic Emitters to be "heard" from.

## CVirtHapticEmitter

This script emits haptic feedback in a specific radius around it, to be received by a HapticListener.



The image shows the Unity Inspector for the **CVirtHapticEmitter (Script)**. The script is attached to a GameObject. The following properties are visible and explained by callouts:

- Haptic Listener**: Set to **CVirtPlayerController (CVirt...)**. Callout: Reference to the Haptic Listener receiving Haptic Feedback. If not set will find one in scene.
- AutoStart Playing**: ☐ (unchecked). Callout: Automatically start playing on game startup.
- Loop**: ☒ (checked). Callout: Loop the haptic feedback over time.
- Duration**: 5. Callout: Duration in seconds for the feedback.
- Range**: 5. Callout: Max range of the haptic effects.
- Volume over Time**: A graph showing a green curve that starts at 0, rises to a peak, and then falls back to 0. Callout: AnimationCurve for the Haptic force over time. Normalized [y = force factor, x = time in s].
- Force Over Distance**: A graph showing a green curve that starts at 0, rises to a peak, and then falls back to 0. Callout: AnimationCurve for the Haptic force over distance to the Haptic Listener. Normalized [y = force factor, x = distance in m].

---

# 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:** IVirtDevice

Returns the Virtualizer device managed by the CVirtDeviceController.

### IsDecoupled

**Returns:** bool

Returns true if the IVirtDevice supports decoupled movement, otherwise false.

## IVirtDeviceUnityExtensions

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.



---

## Example usage

```
using UnityEngine;
using CybSDK;

public class CVirtPlayerController : MonoBehaviour
{
    // ...

    // Update is called once per frame
    void Update()
    {
        IVirtDevice 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 = localOrientation;

                globalOrientation = forwardDirection.transform.rotation;
            }
            else
            {
                globalOrientation = gameObject.transform.rotation * localOrientation;
            }
        }
        // For coupled movement we rotate the pawn and HMD
        else
        {
            gameObject.transform.rotation = localOrientation;
            globalOrientation = localOrientation;
        }

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