How to Build a Unity Application that Shows Tracked Hands on an XR2 with the Unity Plugin

Status

Version	Comment
Version 1 - Draft	Intended to help with Tools testing as part of L2 testing phase.
Version 1.1	Changed emphasis to focus on the target output being tracked capsule hands. Released with the 4.7.4-partners tooling, intended for L2 testing.
Version 1.2	Updated to add instructions for supporting hand tracking ‘timewarp’ correction. Note, this version introduces some changes to the scene hierarchy and settings from previous versions of this guide.
Version 1.3	Now reflects the new tracking Unity plugin structure.
Version 1.4	Brought up to date with the tracking Unity plugin changes for first release of the plugin. This version will be given to Lynx.
Version 1.5	Updated to work with a the Unity plugin as a package.
Version 1.6	Added info about how to set up the SVR symbol definition used to compile for code on SVR platforms.
Version 1.7	Changed way of importing XR2 experimental package to use package import not the package manager.
Version 1.8	Updated for 4.06 of the SnapDragon XR SDK which now includes asmdef files.

Overview

This guide is intended to enable a user to build a basic VR application using Unity that will show tracked (capsule) hands on an XR2 based headset that uses the Qualcomm Unity Snapdragon XR SDK.

Prerequisites

To be able to build the application you will need:

- The latest Unity 2019 LTS software running on a development PC (e.g. running Windows 10). *Although this version of the plugin adds support for Unity 2020, XR2 support has not been verified on Unity 2020 and there are known issues.*
- The Android development tools, usually presented as option to install as part of the Unity installation process.
- A recent Qualcomm Unity Snapdragon XR SDK (i.e. version >=4.0.6) . This is something you should be able to request access to from Qualcomm, normally using a CreatePoint account.
- The tracking tooling (Unity Plugin) - typically available as a .unitypackage.
- An XR2 based headset running the latest version of the hand tracking service/server which is compatible with the tracking tooling and which uses the SVR Unity plugin - e.g. a Morpheus reference design headset.
- A USB cable to connect the headset to the development PC running Unity

Steps

Create the Project

First, create a new 3D Unity project - e.g. TrackedHandsTest, from the Unity Hub. The target should be the latest Unity 2019 LTS release - e.g. Unity 2019.4.

The project will open in the Unity Editor.

Set up the Android Specific Settings for the XR2

Switch the Build Settings to Android: **File > Build Settings > Android >** then choose **Switch Platform.** Once complete close the **Build Settings** window.

Set the following settings under: **Edit > Project Settings > Player > Other Settings** (for Android):

- Delete the entry for Vulkan under **Graphics APIs**
- **Minimum API Leve**l: Android 10.0 (API level 29)
- **Target API Level**: Android 10.0 (API level 29)
- **Scripting backend**: IL2CPP
- Under **Target Architectures** select ARM64 only
- Select **Allow 'unsafe' Code**

Once done, close the project settings window.

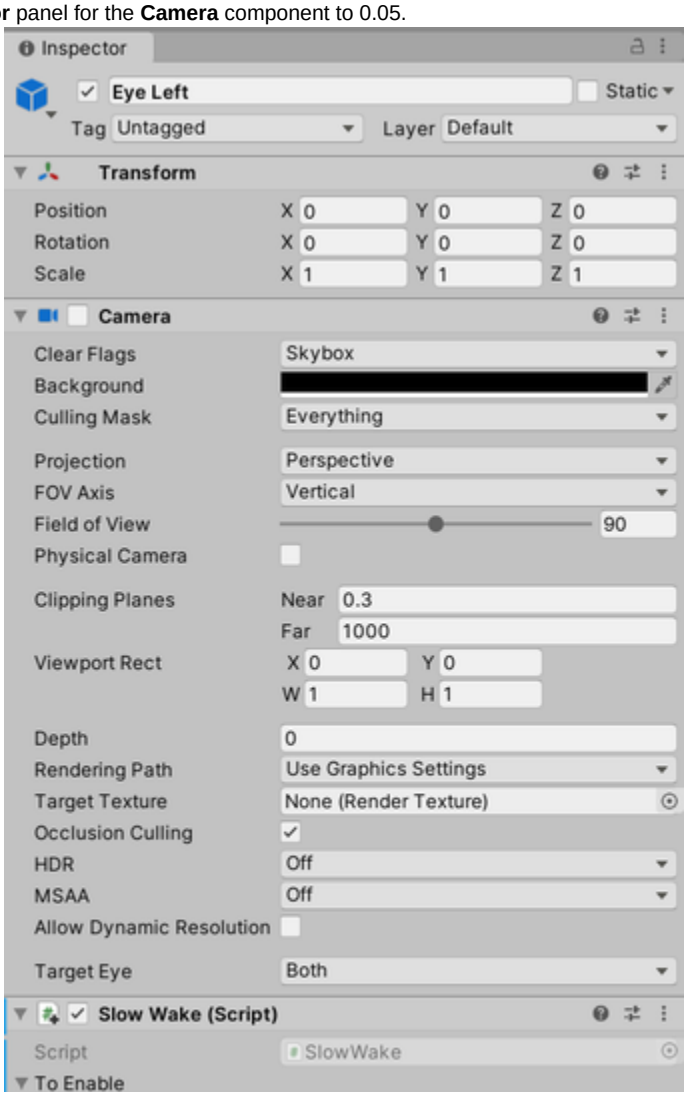
Set up the SVR Components (if used)

Next set up the SVR components in the scene. To do this, follow the steps outlined in the *SnapDragon Unity XR documentation* (**Qualcomm_Snapdragon_XR_Unity_User_Guide**), which can be found in the 'doc' folder e.g. of the SnapdragonXR-Unity-rel.4.0.6 tooling release. Follow steps 3.1.1 through to 3.1.3. in the **Unity Integration** section. At the end of these steps you should have a scene that contains an SvrCamera rig and the appropriate player settings configured.

Make the following additional changes / corrections to those described in the above guide:

- The Multithreading Rendering option appears under **Other Settings**. Disable this.
- Ensure that under **Project Settings > Player > Resolution and Presentation** the **Default Orientation** is set to 'Landscape Left'.
- On the **SVRCamera** GameObject in the **Hierarchy** tree view, open the **Settings** section in the **Inspector** on the **Svr Manager** script. Change the **Frustum Type** to Camera.

For the **Eye Left** and **Eye Right** game objects in the **Hierarchy** tree view, (under **SvrCamera > Head** in the project hierarchy), change the **Clipping Plane** near value in the **Inspector** panel for the **Camera** component to 0.05.



Delete the default **Main Camera** GameObject from the root of the scene hierarchy.

Finally, add two assembly definition files to the SVR code if using an SDK version that is older than version 4.06. This enables the Leap plugin to reference the SVR scripts. Version 4.06 of the SVR plugin includes these asmdef files.

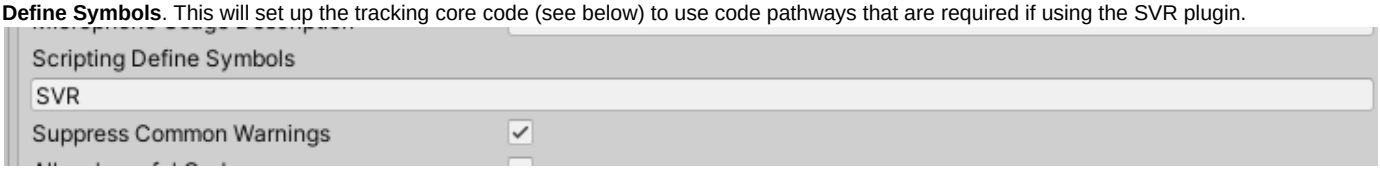
1. Add the **SVR.asmdef** file to the folder under Assets / SVR / Scripts. The file contents are:

```
{
        "name": "SVR"
}
```

2. Add the **SVREditor.asmdef** file to the folder under Assets / SVR / Editor. The file contents are:

```
{
    "name": "SVREditor",
    "references": [],
    "includePlatforms": [
        "Editor"
    ],
    "excludePlatforms": [],
    "allowUnsafeCode": false,
    "overrideReferences": false,
    "precompiledReferences": [],
    "autoReferenced": true,
    "defineConstraints": [],
    "versionDefines": [],
    "noEngineReferences": false
}
```

Go to the player settings - **Edit > Project Settings > Player**. Expand the **Other Settings** section. Add **SVR** in the textbox under **Scripting Define Symbols**. This will set up the tracking core code (see below) to use code pathways that are required if using the SVR plugin.



Set up Hand Tracking Components

Import the Ultraleap Unity Tracking Plugin.unitypackage into the project.

Go to **Assets > Import Package > Custom Package**. Select the supplied Ultraleap Unity Tracking Plugin.unitypackage and **Open** it. Choose **Import**.

*Note, the tracking package now contains Core, Interaction Engine and Hands. It also contains hidden folders (in **Assets / Plugins / unityplugin***) for experimental features and examples. The UI Input module and some early higher-level XR2 based support is located inside the experimental folder. To enable Unity to show the content, remove the trailing ~ from the folder.*

The sub folder path 'Assets / Plugins / unityplugin' is expected to change before release.

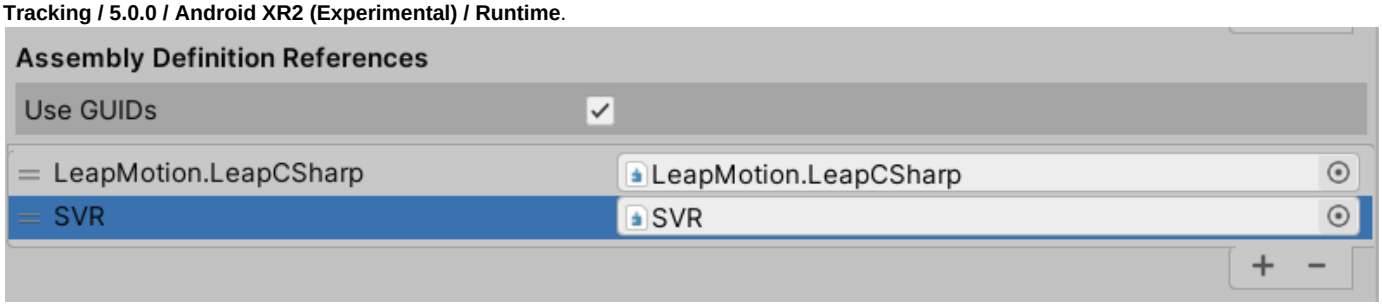
Import the Experimental Android-XR2 content

Go to **Assets > Import Package > Custom Package.** Select the supplied Experimental package (**Ultraleap Unity Tracking Plugin Experimental**.**unitypackage**) and **Open** it. Choose **Import**.

Add References

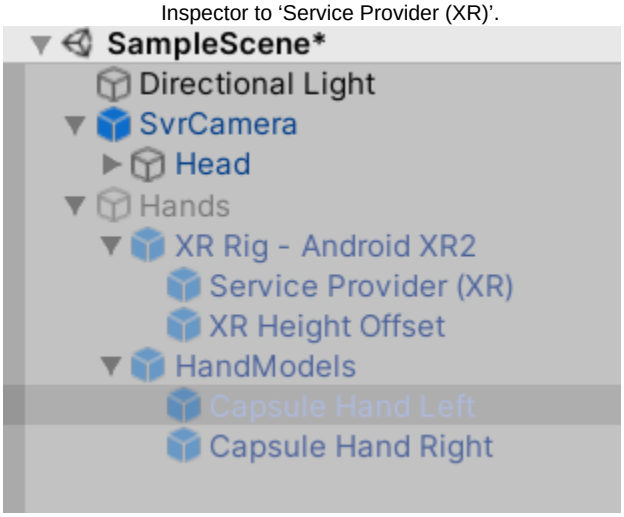
If not already set, add a reference to the SVR script assembly. Select the Ultraleap.Tracking.Core.asmdef file under **Assets / Plugins / unityplugin / Core / Runtime.** In the **Inspector** tab, add an entry to '**SVR**' in the list of **Assembly Definition References.** Also select 'Allow unsafe Code', if not set, and apply the changes when prompted.

Repeat this process to add a refence to SVR to the Android_XR2 project. Select the **Android_XR2.asmdef** under **Assets / Samples / Ultraleap Tracking / 5.0.0 / Android XR2 (Experimental) / Runtime**.



Set up the Scene

- Create a new empty GameObject at the root of the scene. Rename it to **Hands** and disable it.
- Add the **XR Rig - Android XR2** prefab (under **Assets / Plugins / unityplugin / Experimental / Android_XR2 / Runtimes / Prefabs**) as a child of the **Hands** GameObject in the scene Hierarchy.
- Add a **HandModels** prefab (from **Assets / Plugins / unityplugin / Core / Runtime / Prefabs**) as a child of the **Hands** GameObject.
    - Select both the **Capsule Hands Left** and **Capsule Hands Right** GameObjects and set the **Leap Provider** value in the Inspector to 'Service Provider (XR)'.



Add the '**SlowWake**' script to the **SvrCamera** game object using the **Add Component** button in the **Inspector.** This script can be found in the **An droid_XR2** scripts folder in the **Experimental** section and is reproduced here.

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

namespace Ultraleap
{
    public class SlowWake : MonoBehaviour
```

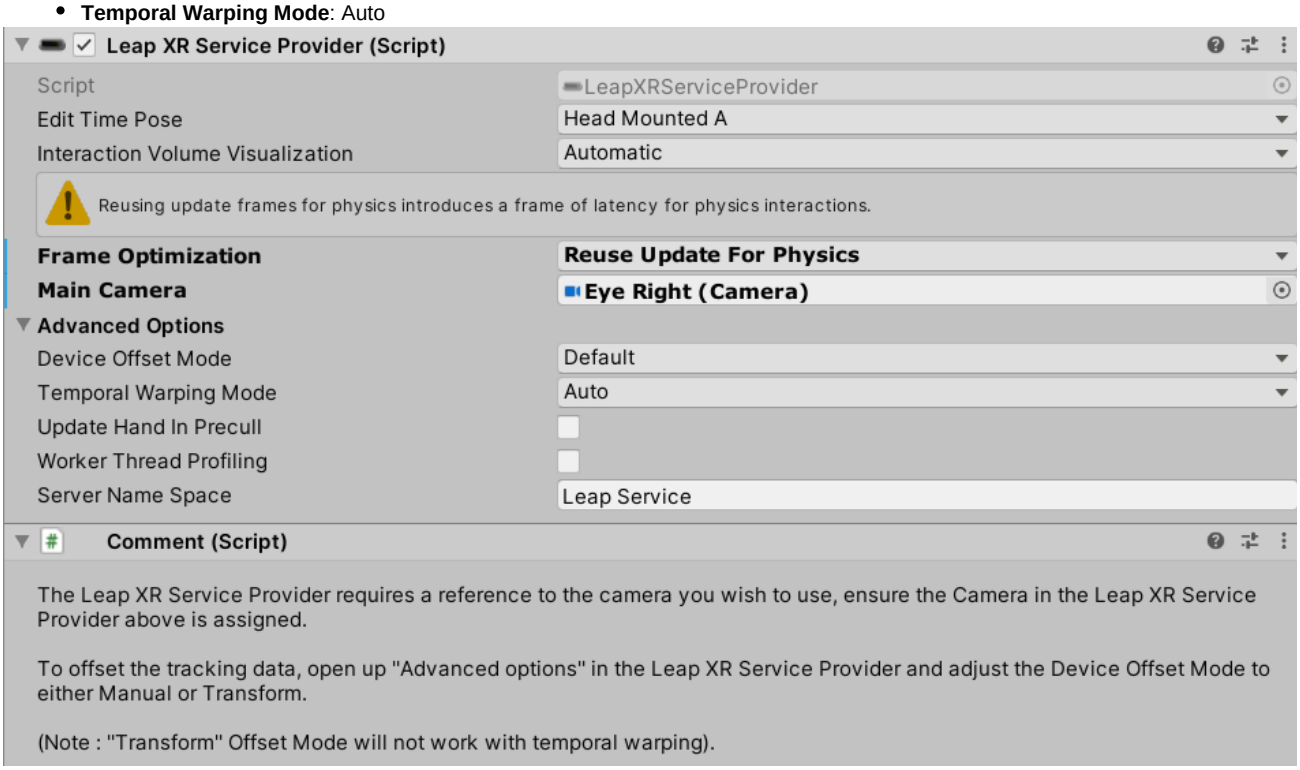
```
    {
        public GameObject[] toEnable; //List in chronological order (E.
G. Leap, scene, etc)

        public void Awake()
        {
            Debug.Log($"SlowWake : Awake");
        }

        public void Start()
        {
            Debug.Log($"SlowWake : Start");
            for (int i = 0; i < toEnable.Length; i++) toEnable[i].
SetActive(false);
            StartCoroutine(SlowWakeCoroutine());
        }

        public void OnEnable()
        {
            Debug.Log($"SlowWake : OnEnable");
        }

        public void OnDisable()
        {
            Debug.Log($"SlowWake : OnDisable");
        }

        private IEnumerator SlowWakeCoroutine()
        {
            var svrManager = SvrManager.Instance;
            Debug.Log("SlowWake : LeapC Unity hand tracking delayed
initialisation, have SvrManager.Instance");
            yield return new WaitUntil(() => svrManager.Initialized ==
true);
            Debug.Log("SlowWake : LeapC, svrManager.Initialized ==
true");
            yield return new WaitForSeconds(2.0f);
            Debug.Log("SlowWake : LeapC, post 2 second sleep. About to
activate Unity hand controller object");
            for (int i = 0; i < toEnable.Length; i++)
            {
                toEnable[i].SetActive(true);
                yield return new WaitForSeconds(0.1f);
            }
            Debug.Log("SlowWake : LeapC, slow wake start up sequence
completed");
        }
    }
}
```

This script ensures that the hand tracking does not start until the SVR camera is initialised.

In the Unity editor (**Inspector** panel) set the size of the 'To Enable' field on the **Slow Wake** script to 1. Set Element **0** to be the **Hands** GameObject.

XR Rig Changes

Select the **Service Provider (XR)** GameObject in the project **Hierarchy**. Set the **Main Camera** to **Eye Right**, this will unlock more settings on the script. Ensure that the other options on the script are set correctly, by changing the following fields from their default values, if necessary.

- **Frame Optimisation**: Reuse Update For Physics
- Under **Advanced**
  - **Device Offset Mode**: Default
  - **Temporal Warping Mode**: Auto
- 

Device Offsets for Timewarp

To minimise an effect known as 'hand swim', which occurs when the head is moved rapidly relative to the hands (static hands do not stay static in the virtual world), the device offset settings should be set as accurately as possible. The **LeapXRServiceProvider** script contains a set of values (DEFAULT_DEVICE_OFFSET_Y_AXIS, DEFAULT_DEVICE_OFFSET_Z_AXIS, DEFAULT_DEVICE_TILT_X_AXIS) that are correct for the Morpheus reference design. **These should be updated for the relevant target hardware and checked to make sure they are set/serialized correctly on the script - e.g. a Lynx headset, if timewarp is enabled (recommended).**

```
public class LeapXRServiceProvider : LeapServiceProvider
{

    #region Inspector
    // Manual Device Offset
#if UNITY_ANDROID
    private const float DEFAULT_DEVICE_OFFSET_Y_AXIS = -0.0114f;
    private const float DEFAULT_DEVICE_OFFSET_Z_AXIS = 0.0981f;
    private const float DEFAULT_DEVICE_TILT_X_AXIS = 0f;
#else
    private const float DEFAULT_DEVICE_OFFSET_Y_AXIS = 0f;
```

```
        private const float DEFAULT_DEVICE_OFFSET_Z_AXIS = 0.12f;
        private const float DEFAULT_DEVICE_TILT_X_AXIS = 5f;
#endif
```

- **DEFAULT_DEVICE_OFFSET_Y_AXIS**
  - Adjusts the Leap Motion device's virtual height offset from the tracked headset position. This should match the vertical offset of the physical device with respect to the headset in meters
- **DEFAULT_DEVICE_OFFSET_Z_AXIS**
  - Adjusts the Leap Motion device's virtual depth offset from the tracked headset position. This should match the forward offset of the physical device with respect to the headset in meters.
- **DEFAULT_DEVICE_TILT_X_AXIS**
  - Adjusts the Leap Motion device's virtual X axis tilt. This should match the tilt of the physical device with respect to the headset in degrees.

*Note, currently the **LeapXRServiceProvider** handles XR related hand tracking behaviour for both Android and Windows. This will be split out into subclasses in a future update to the plugin.*

Building and Run

- Select the correct scene to build in the **Scenes in Build** list under **File > Build Settings.** If none are shown the press the **Add Open Scenes** button
- Choose **Build** to build the APK, giving it a suitable name and output folder when prompted.
  - Note, you will need to grant your application the following permissions to be able to connect to the hand tracking service on the XR2. To do this add the following lines to the **AndroidManifest.XML** file, e.g. immediately after the <manifest> opening line.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.
ACCESS_NETWORK_STATE" />
```

    - If the application does not have a manifest file, the one Unity creates during a build can be used as a template. Copy it from the temp folder to the **Assets / Plugins / Android** folder and then modify it, as described in this link. On recent versions of Unity it can be found in the **Temp\gradleOut\unityLibrary\src\main** folder once a build has been completed.
      - Rebuild the project.
- Install and run the APK on your XR2 device - e.g. open the location of the APK in a command window and enter ‘adb install trackedhandstest.apk’ (assuming adb has been set up on the system). Replace trackedhandtest with the name of the built APK, if different.
- The application should launch on the headset. The default Unity grey ground plane / blue sky scene should be visible. Capsule hands should be shown when the hands are in view of the headset and should track hand movements.