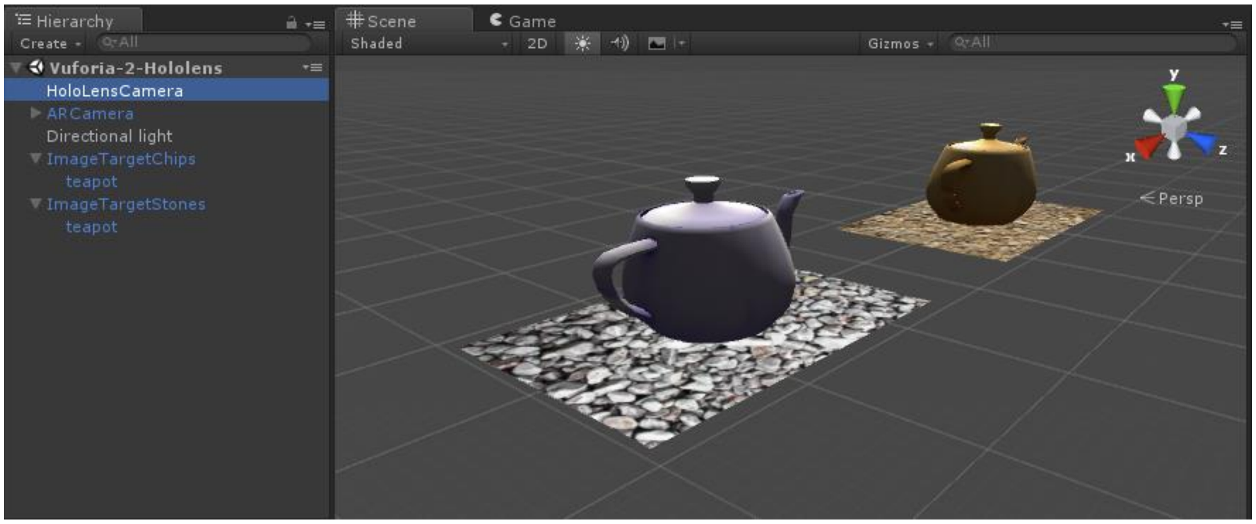
Working with the Hololens Sample in Unity

The Vuforia HoloLens Sample project provides a pre-configured scene in the Unity Editor that you can use as a reference and starting point for your own HoloLens apps that use Vuforia. This document will show you how to author a Vuforia scene for HoloLens, and how to customize event handling to implement custom app behaviors.

Also see: **Getting Started Developing Vuforia Apps for Hololens**



Contents

Supported Versions

The Early Access Vuforia HoloLens sample supports the following operating system and tool versions. These tools will need to be installed on your development system in order to build and run the sample.

Device OS	Development OS	Unity Version	Visual Studio Version
Windows 10	Windows (2) 7+	Windows (1,3) v5.4.0 Beta 14 & 16 (2)	Windows (1) Visual Studio 2015 update 2
1. 32-bit only 2. 32 & 64-bit 3. HoloLens Technical Preview			

Note that the Unity HoloLens Technical Preview does not currently support Vuforia's Play Mode simulator. You'll need to deploy your apps to a device to run your HoloLens scenes. Alternatively you can develop your scene in another version of Unity that supports Play Mode and then migrate it to the HTP edition when you're ready to deploy to HoloLens.

Importing the sample

To import the HoloLens sample into a new Unity project..

1. Either double click on the HoloLens-x-x-x.unitypackage to launch the Import Package dialog or import the same Unity package by selecting it from the Unity Editor menu from Assets > Import Package > Custom Package.
2. Click the Import buttons at the bottom right of the Importing Package window.

To import the HoloLens sample into an existing Vuforia Unity project..

1. Follow the steps in the Vuforia Unity project migration guide

2. Follow the steps to import the HoloLens sample into a Unity project above.

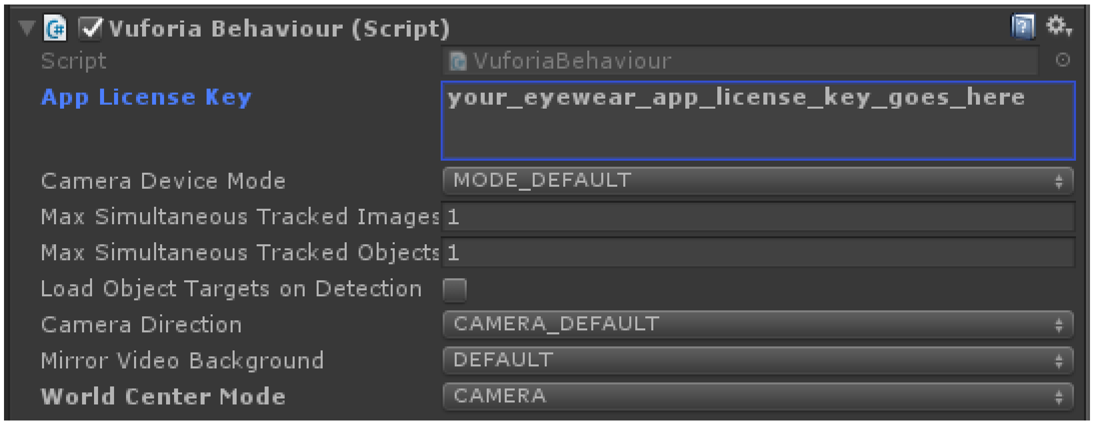
To import the HoloLens sample into an existing Unity project that doesn't already contain the Vuforia extension.

1. Save a separate copy of your project.
2. Either double click on the HoloLens -x.x.x.unitypackage to launch the Import Package dialog or import the same Unity package by selecting it from the Unity Editor menu from Assets > Import Package > Custom Package.
3. Review the folder and asset names used in the sample project to determine if they will clash with named assets that you are using in your project, and if so abort the import, rename your assets, and return to step 2.
4. Once you have ensured that there are no project conflicts, click the Import buttons at the bottom right of the Importing Package window.

Adding a License Key

To build the sample, you'll need to add a Vuforia Eyewear license key to the ARCamera prefab's Vuforia Behaviour component in the Vuforia-3-HoloLens scene.

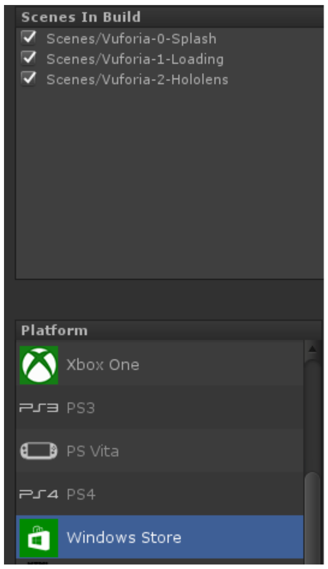
See: [How To add a License Key to your Vuforia App]()


Vuforia Behaviour (Script)

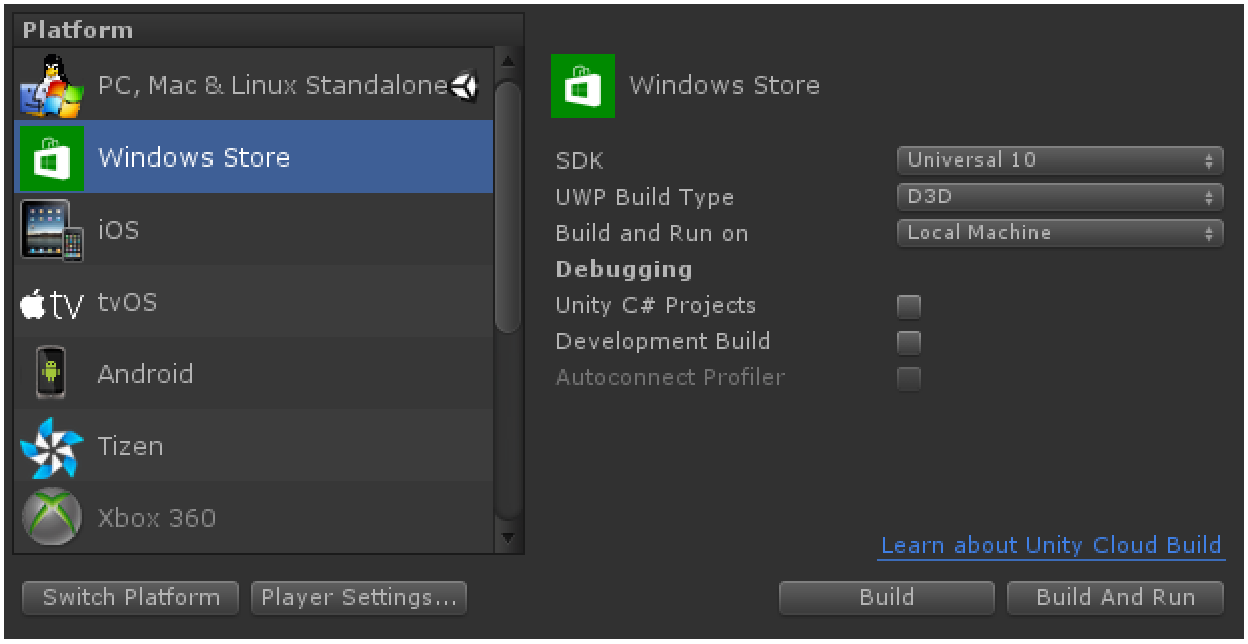
Script	VuforiaBehaviour
App License Key	your_eyewear_app_license_key_goes_here
Camera Device Mode	MODE_DEFAULT
Max Simultaneous Tracked Images	1
Max Simultaneous Tracked Objects	1
Load Object Targets on Detection	☐
Camera Direction	CAMERA_DEFAULT
Mirror Video Background	DEFAULT
World Center Mode	CAMERA

Building and executing the sample

1. [Add an Eyewear App License Key]() in the ARCamera Inspector
2. Apply the recommended [Unity Engine Options for Power and Performance]()
3. Add the sample scenes to *Scenes in Build*



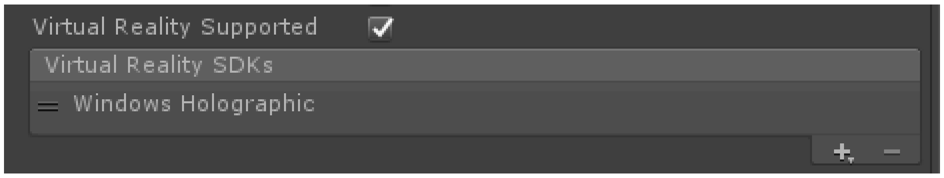
4. Set your platform build target for Windows Store in *File > Build Settings*.
5. Select the following platform build configuration settings
   - *SDK* = Universal10
   - UWP Build Type = D3D



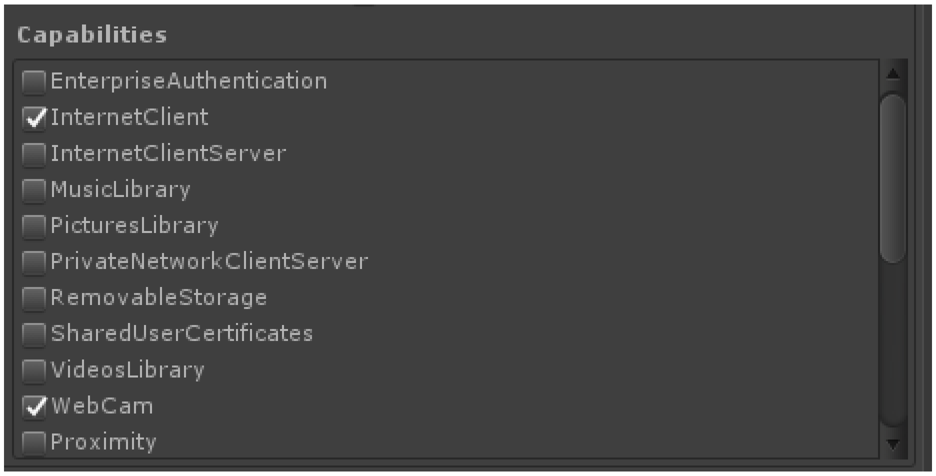
SDK Universal 10
UWP Build Type D3D
Build and Run on Local Machine
Debugging
Unity C# Projects
Development Build
Autoconnect Profiler

6. Define a unique *Product Name*, in *Player Settings*, to serve as the name of the app when installed on the HoloLens.
7. Select *Landscape Left* as the *Default Orientation* in *Player Settings > Resolution and Presentation*

8. Check Virtual Reality Supported + Windows Holographic in Player Settings > Other Settings


Virtual Reality Supported ✓
Virtual Reality SDKs
Windows Holographic

9. Check the following Capabilities in *Player Settings > Publish Settings*
   - InternetClient
   - WebCam
   - SpatialPerception - if you intend to use the Surface Observer API

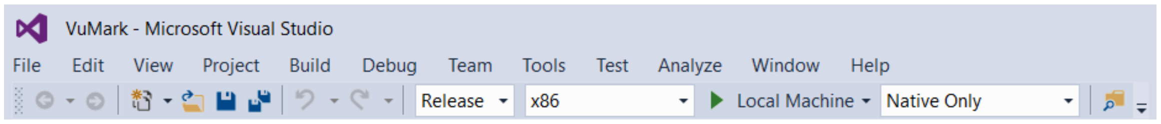

Capabilities

- [ ] EnterpriseAuthentication
- [x] InternetClient
- [ ] InternetClientServer
- [ ] MusicLibrary
- [ ] PicturesLibrary
- [ ] PrivateNetworkClientServer
- [ ] RemovableStorage
- [ ] SharedUserCertificates
- [ ] VideosLibrary
- [x] WebCam
- [ ] Proximity

10. Select *Build* to generate a Visual Studio project
11. Build the executable from Visual Studio and install it on your HoloLens. See: https://developer.microsoft.com/en-us/windows/holographic/exporting_and_building_a_unity_visual_studio_solution

Visual Studio Build Configuration

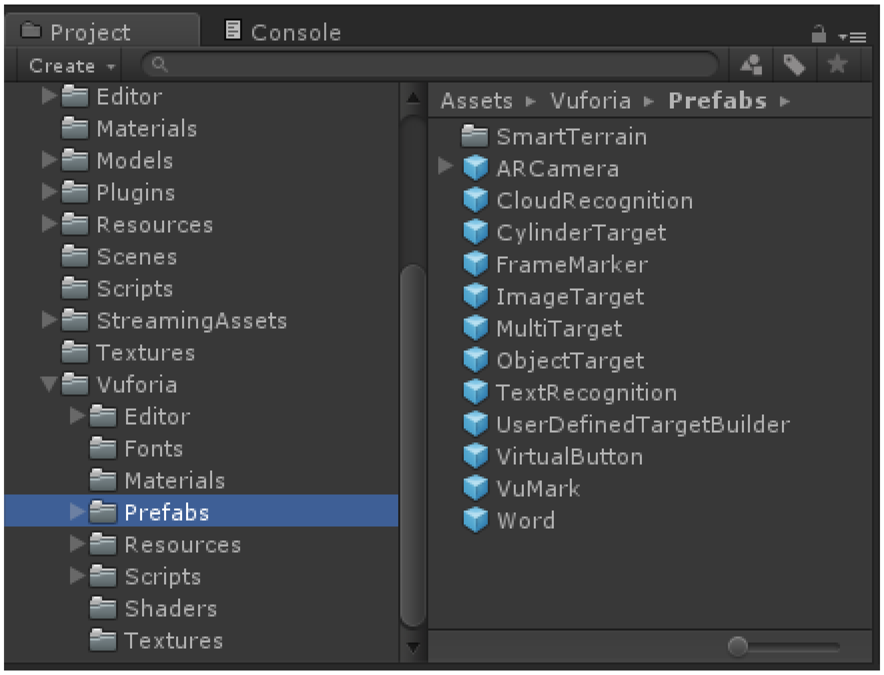
Be sure to set your build target for x86. Note that the EAP release supports only 32bit builds.


VuMark - Microsoft Visual Studio

Release | x86 | Native Only

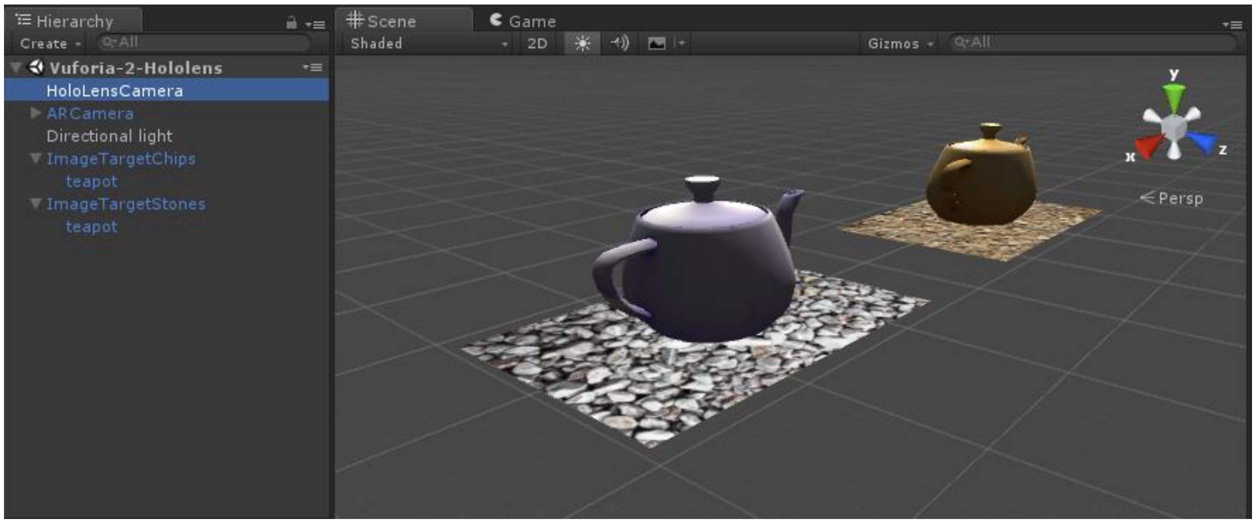
Scene elements & their configuration

The HoloLens sample uses the ARCamera and ImageTarget prefabs. These can be found in the project's Vuforia folder along with the other assets and resources in this sample.



```
Editor                 Assets ► Vuforia ► Prefabs ►
Materials                SmartTerrain
Models                   ARCamera
Plugins                  CloudRecognition
Resources                CylinderTarget
Scenes                   FrameMarker
Scripts                  ImageTarget
StreamingAssets          MultiTarget
Textures                 ObjectTarget
Vuforia                  TextRecognition
  Editor                 UserDefinedTargetBuilder
  Fonts                  VirtualButton
  Materials              VuMark
  Prefabs                Word
  Resources
  Scripts
  Shaders
  Textures
```

The sample's scene Hierarchy demonstrates how to set-up a Vuforia HoloLens scene in Unity.

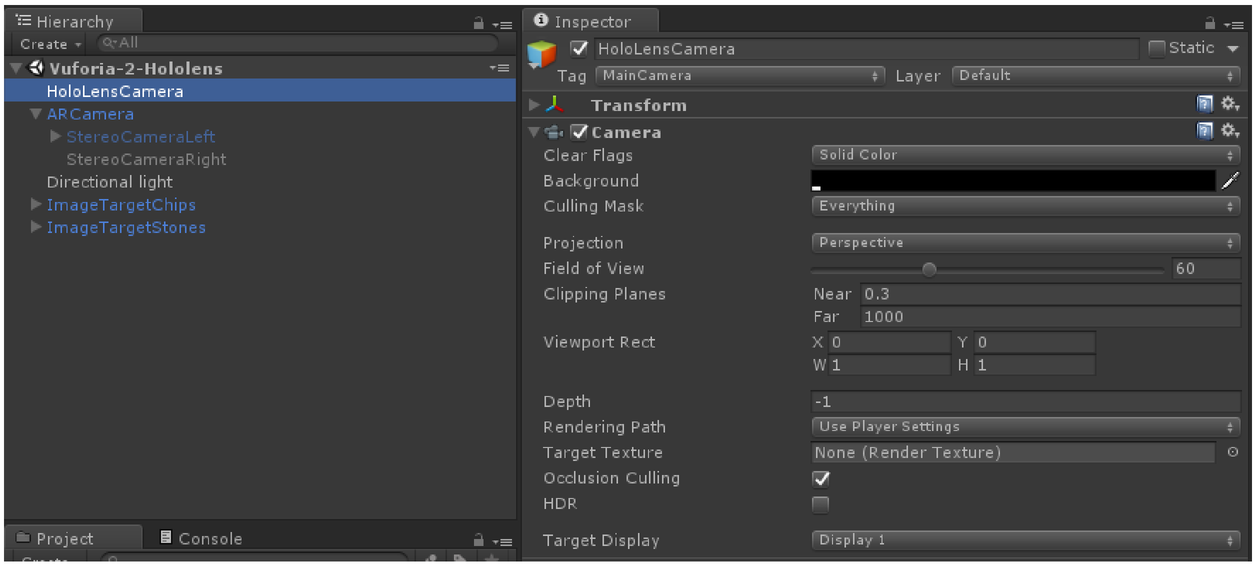


```
Vuforia-2-Hololens
  HoloLensCamera
  ARCamera
  Directional light
  ImageTargetChips
    teapot
  ImageTargetStones
    teapot
```

Scene Hierarchy

HoloLensCamera

The HoloLensCamera object is a standard Unity scene camera that has been configured the properties recommended for HoloLens apps. The sample shows how to bind this camera to the ARCamera to enable Vuforia to transform target poses into the HoloLens spatial coordinate system.



Vuforia-2-Hololens
- HoloLensCamera
- ARCamera
  - StereoCameraLeft
  - StereoCameraRight
- Directional light
- ImageTargetChips
- ImageTargetStones

HoloLensCamera ✓ — Static ☐
Tag: MainCamera — Layer: Default

Transform

Camera ✓

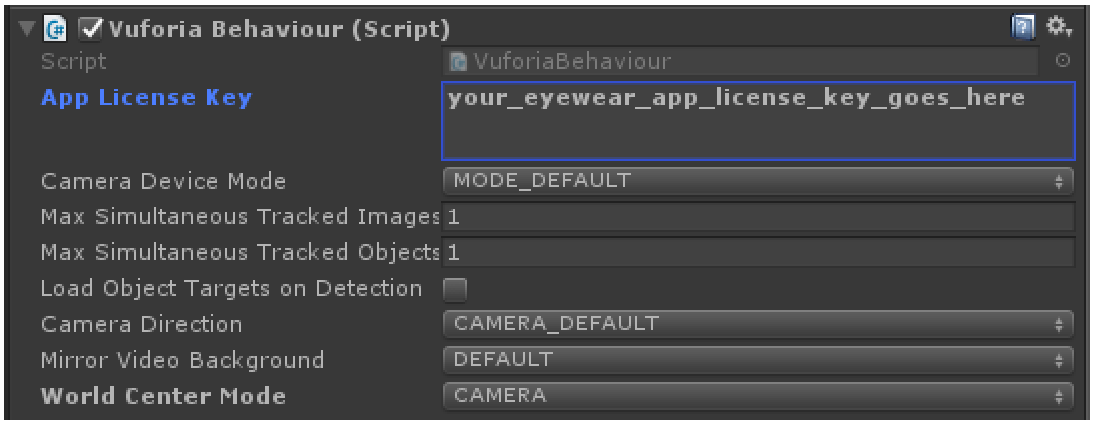
Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Perspective
Field of View	60
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	-1
Rendering Path	Use Player Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	☐
Target Display	Display 1

ARCamera

The ARCamera is the scene camera rig for Vuforia apps in Unity. It defines the properties of both of its child scene cameras, as well as the device camera and rendering behavior of the scene.

Components:

Vuforia Behaviour



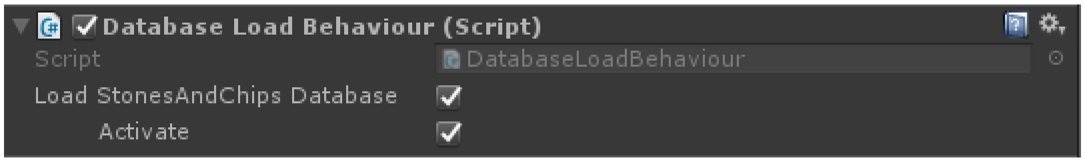
Vuforia Behaviour (Script)	
Script	VuforiaBehaviour
App License Key	your_eyewear_app_license_key_goes_here
Camera Device Mode	MODE_DEFAULT
Max Simultaneous Tracked Images	1
Max Simultaneous Tracked Objects	1
Load Object Targets on Detection	☐
Camera Direction	CAMERA_DEFAULT
Mirror Video Background	DEFAULT
World Center Mode	CAMERA

Camera Device Mode setting enables you to prioritize render quality vs frame rate for your app. Selecting MODE_DEFAULT will typically prioritize rendering except on devices with lower performance characteristics.

Max Simultaneous Tracked Images this value defines the maximum number of targets that can be tracked simultaneously.

World Center Mode defines which object in the scene hierarchy will serve as the origin (0,0,0) of the scene's world space. When CAMERA is chosen the scene is rendered within the camera coordinate frame. This is the recommended World Center for Vuforia when developing a HoloLens app.

Database Load Behaviour



Database Load Behaviour (Script)	
Script	DatabaseLoadBehaviour
Load StonesAndChips Database	✓
Activate	✓

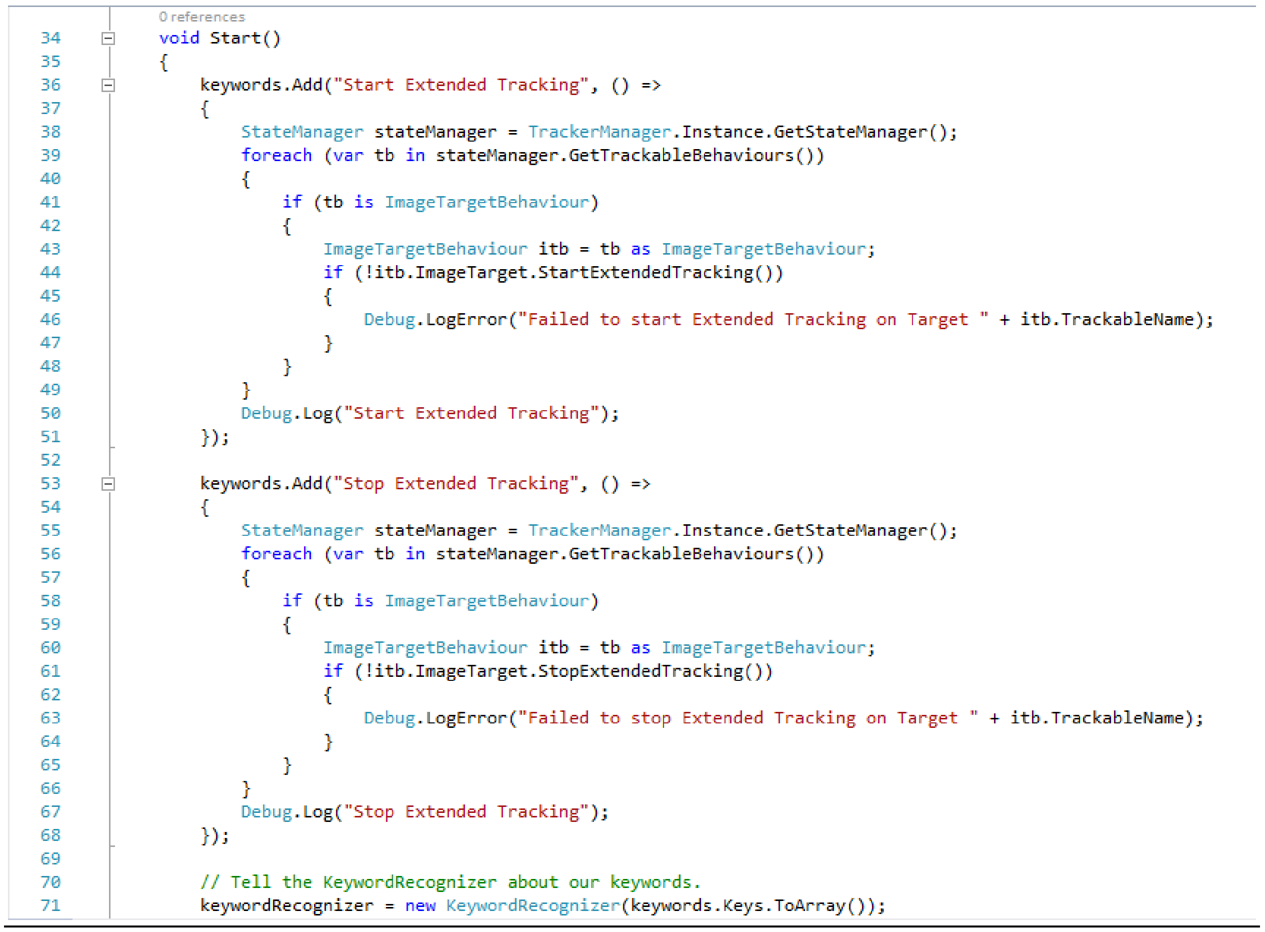
Load Data Set automatically loads the associated dataset from Streaming Assets / QCAR when the app initializes.

Activate automatically activates the dataset after it is loaded.

Note: *if you don't load and activate datasets through the Editor, you'll need to do so using the Vuforia API, See: How To Load and Activate Multiple Device Databases at Runtime*

Voice Commands

The *Voice Commands* script on the ARCamera defines voice commands for starting and stopping Extended Tracking on the Image Targets, and registers these with the HoloLens Keyword Recognizer, it then starts the recognizer. This occurs when the app is first started.



```
0 references
void Start()
{
    keywords.Add("Start Extended Tracking", () =>
    {
        StateManager stateManager = TrackerManager.Instance.GetStateManager();
        foreach (var tb in stateManager.GetTrackableBehaviours())
        {
            if (tb is ImageTargetBehaviour)
            {
                ImageTargetBehaviour itb = tb as ImageTargetBehaviour;
                if (!itb.ImageTarget.StartExtendedTracking())
                {
                    Debug.LogError("Failed to start Extended Tracking on Target " + itb.TrackableName);
                }
            }
        }
        Debug.Log("Start Extended Tracking");
    });

    keywords.Add("Stop Extended Tracking", () =>
    {
        StateManager stateManager = TrackerManager.Instance.GetStateManager();
        foreach (var tb in stateManager.GetTrackableBehaviours())
        {
            if (tb is ImageTargetBehaviour)
            {
                ImageTargetBehaviour itb = tb as ImageTargetBehaviour;
                if (!itb.ImageTarget.StopExtendedTracking())
                {
                    Debug.LogError("Failed to stop Extended Tracking on Target " + itb.TrackableName);
                }
            }
        }
        Debug.Log("Stop Extended Tracking");
    });

    // Tell the KeywordRecognizer about our keywords.
    keywordRecognizer = new KeywordRecognizer(keywords.Keys.ToArray());
```

ImageTargetChips & ImageTargetStones

The ImageTarget prefabs encapsulate the ImageTargetBehavior and Default Trackable Event Handler. These are the primary script components that you will use to customize an app that uses HoloLens.

Components:

Image Target Behaviour

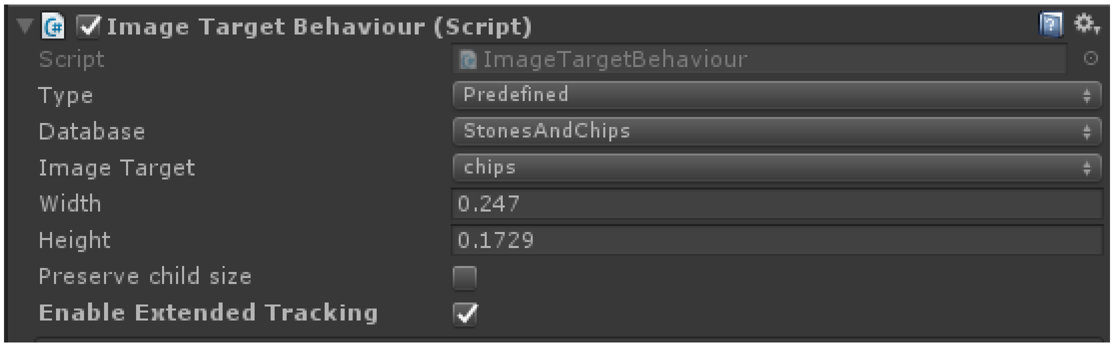


Image Target Behaviour (Script)

Script	ImageTargetBehaviour
Type	Predefined
Database	StonesAndChips
Image Target	chips
Width	0.247
Height	0.1729
Preserve child size	☐
Enable Extended Tracking	✓

Database identifies the database that contains the Image Target that will be assigned to this Image Target Behaviour

Image Target defines which target in the database to assign to this Image Target Behaviour

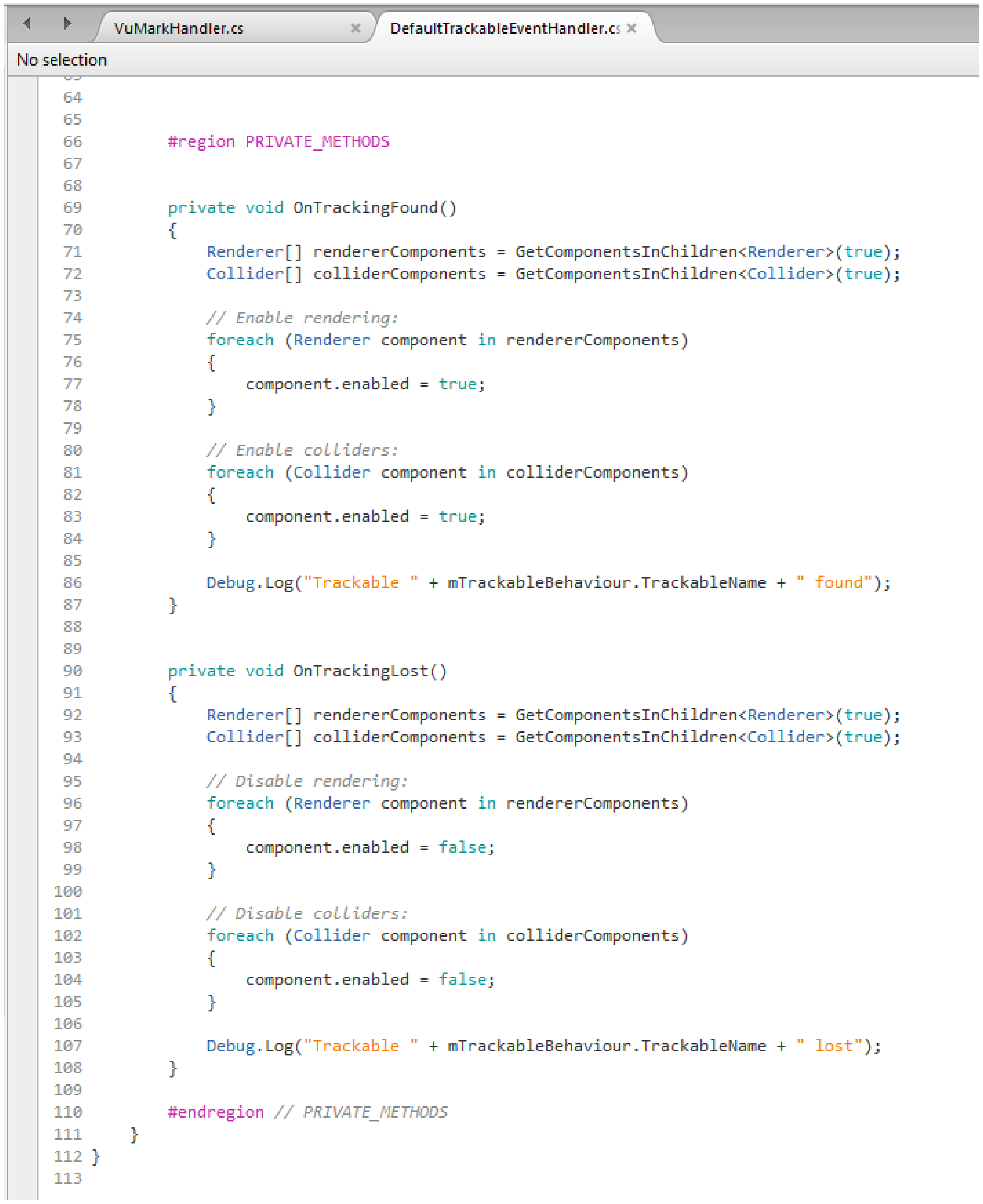
Enable Extended Tracking activates the extended tracking feature to support content registration when the target is not in view. This feature **must** be activated to enable Vuforia to work with the HoloLens spatial mapping engine.

Default Trackable Event Handler

The Default Trackable Event Handler component is responsible for handling callbacks to the Image Target Behaviour arising from changes in the state of the Image Target, such as when it has been detected and is then being tracked.

This script is used to enable and disable rendering and collision detection on digital content that is a child of the target.

Extend this script's OnTrackingFound() and OnTrackingLost() methods to implement custom event handling for your app.


VuMarkHandler.cs | DefaultTrackableEventHandler.cs

No selection

```csharp


        #region PRIVATE_METHODS


        private void OnTrackingFound()
        {
            Renderer[] rendererComponents = GetComponentsInChildren<Renderer>(true);
            Collider[] colliderComponents = GetComponentsInChildren<Collider>(true);

            // Enable rendering:
            foreach (Renderer component in rendererComponents)
            {
                component.enabled = true;
            }

            // Enable colliders:
            foreach (Collider component in colliderComponents)
            {
                component.enabled = true;
            }

            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " found");
        }


        private void OnTrackingLost()
        {
            Renderer[] rendererComponents = GetComponentsInChildren<Renderer>(true);
            Collider[] colliderComponents = GetComponentsInChildren<Collider>(true);

            // Disable rendering:
            foreach (Renderer component in rendererComponents)
            {
                component.enabled = false;
            }

            // Disable colliders:
            foreach (Collider component in colliderComponents)
            {
                component.enabled = false;
            }

            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " lost");
        }

        #endregion // PRIVATE_METHODS
    }
}

```