UWB Networking Package

The UWB Networking Package utilizes the free version of Photon Unity Networking to create synchronized online game rooms that allow many HTC Vives, HoloLenses, Kinects, and Oculus Rifts to interact in the same virtual space. Furthermore, a HoloLens is capable of scanning a room to create a “room mesh”, which can be sent to all of the other connected devices.

The goal of this project is to implement a networking platform that makes multiplayer networking and room sharing relatively straightforward for virtual and augmented reality developers and allow them to focus on content creation.

[**Getting Started**](#_9kwtf8iryef5)[**3**](#_9kwtf8iryef5)

[Installing Unity](#_os3h19si2te6) [3](#_os3h19si2te6)

[Using Unity](#_gzkivhmxyxjr) [3](#_gzkivhmxyxjr)

[Importing the UWB Networking Package](#_xz6yid8iwgcn) [4](#_xz6yid8iwgcn)

[Importing Dependencies](#_mdg20e9b5wfe) [6](#_mdg20e9b5wfe)

[Importing Photon Unity Networking](#_1esxmy8j7pbf) [6](#_1esxmy8j7pbf)

[Importing HoloToolkit](#_sdhkgjhixntk) [7](#_sdhkgjhixntk)

[Specifying Photon Server Settings](#_47mj1uzfln75) [7](#_47mj1uzfln75)

[Adding Network Manager Prefab](#_ll8fi9km338g) [8](#_ll8fi9km338g)

[Specify Device](#_qpf2j1r8lwt8) [9](#_qpf2j1r8lwt8)

[Master Client Setup](#_1vw5c7wxwd6) [10](#_1vw5c7wxwd6)

[Adding Spatial Mapping Prefab](#_3fp5kqoy4dln) [10](#_3fp5kqoy4dln)

[**Usage**](#_7vjb0sk5d428)[**11**](#_7vjb0sk5d428)

[Object Synchronization](#_y05rtd4hebqx) [11](#_y05rtd4hebqx)

[Photon Resources Folder](#_2jc3aptadgnf) [11](#_2jc3aptadgnf)

[Required Scripts](#_x4qn52pck401) [12](#_x4qn52pck401)

[Synchronization Options](#_rxxb8xot0k6o) [15](#_rxxb8xot0k6o)

[Ownership Option](#_tjp5ocecnka6) [15](#_tjp5ocecnka6)

[Observe Option](#_lihc0wdz6p09) [16](#_lihc0wdz6p09)

[Transform Serialization](#_j9d42m2mbvr9) [16](#_j9d42m2mbvr9)

[Object Instantiation](#_nrjrte25aq1g) [16](#_nrjrte25aq1g)

[Object Ownership](#_s261rule4e0a) [17](#_s261rule4e0a)

[Requesting Ownership](#_55ffbhytxcgi) [17](#_55ffbhytxcgi)

[Transferring Ownership](#_vbbkprjfgb9z) [18](#_vbbkprjfgb9z)

[Object Deletion](#_9foossd92q6m) [18](#_9foossd92q6m)

[Further Information](#_wa911rtq3fcs) [18](#_wa911rtq3fcs)

[Room Mesh Networking](#_ld8c43xtvmcu) [18](#_ld8c43xtvmcu)

[Creating a Room Mesh](#_q2m91miobpyu) [18](#_q2m91miobpyu)

[Sending a Room Mesh](#_3uir6pls3sdv) [18](#_3uir6pls3sdv)

[Receiving a Room Mesh](#_8ism4aj1etiy) [19](#_8ism4aj1etiy)

[Displaying a Room Mesh](#_gi5eos7cn0du) [19](#_gi5eos7cn0du)

[**Common Photon Connection Issues**](#_y9869djxf1pn)[**19**](#_y9869djxf1pn)

[**Demo Projects**](#_icoz395vsg73)[**19**](#_icoz395vsg73)

[Master Client Demo](#_fq7bbuiradrq) [20](#_fq7bbuiradrq)

[HoloLens Demo](#_kny1vtrgfcdn) [20](#_kny1vtrgfcdn)

[Vive Demo](#_r8953te41a8q) [20](#_r8953te41a8q)

[Oculus Demo](#_7m9s6yn2pbyf) [20](#_7m9s6yn2pbyf)

[Kinect Demo](#_p04zonphl7bi) [21](#_p04zonphl7bi)

[**Project Background**](#_z6tmudoidtf4)[**21**](#_z6tmudoidtf4)

[Summer 2016 Work](#_6b509mfu9bxx) [21](#_6b509mfu9bxx)

[Why Photon Unity Networking](#_9gzgr5rlcpt7) [21](#_9gzgr5rlcpt7)

[Accomplishments](#_8afku9kn8mkf) [21](#_8afku9kn8mkf)

[Limitations](#_38eglxpjskns) [22](#_38eglxpjskns)

[**Class Design**](#_ouc9ogzk4l)[**22**](#_ouc9ogzk4l)

[Database](#_r6hb0jo94ao1) [22](#_r6hb0jo94ao1)

[NetworkManager](#_idacywvbp2dh) [22](#_idacywvbp2dh)

[Launcher](#_4ajhqrvk6xjy) [22](#_4ajhqrvk6xjy)

[MasterClientLauncher : Launcher](#_a6bo1dfng4es) [22](#_a6bo1dfng4es)

[HololensLauncher : Launcher](#_zbrg1r5a4b7g) [23](#_zbrg1r5a4b7g)

[ReceivingClientLauncher : Launcher](#_iov4ocpqvoec) [23](#_iov4ocpqvoec)

[ViveLauncher : ReceivingClientLauncher](#_uu4enf3i4aps) [23](#_uu4enf3i4aps)

[OculusLauncher : ReceivingClientLauncher](#_mqzk005mibx7) [23](#_mqzk005mibx7)

[KinectLaucher : ReceivingClientLauncher](#_6pfge4rqn5pe) [23](#_6pfge4rqn5pe)

[**Networking Implementation Details**](#_aqbltirgaa94)[**23**](#_aqbltirgaa94)

[Photon Unity Networking (PUN)](#_n5c2iz1w623h) [23](#_n5c2iz1w623h)

[Remote Procedure Calls (RPC)](#_muo31iqv47s2) [24](#_muo31iqv47s2)

[Photon View](#_7ecefx2r12go) [24](#_7ecefx2r12go)

[Rooms and Lobbies](#_d35j7obiuqyt) [24](#_d35j7obiuqyt)

[Master Client](#_y8l9gvo86tms) [24](#_y8l9gvo86tms)

[Room Mesh](#_9wofd099jwhl) [24](#_9wofd099jwhl)

[Directives](#_2oxezop3db5e) [24](#_2oxezop3db5e)

[HoloLens](#_z83qxbb6ic56) [24](#_z83qxbb6ic56)

[Stream Sockets](#_6x7g9m6qzw1y) [24](#_6x7g9m6qzw1y)

[Master Client](#_6hh0jluetbs) [25](#_6hh0jluetbs)

[Tcp Listener](#_njdhvzmi9bla) [25](#_njdhvzmi9bla)

[Receiving Clients](#_h4a77acser9c) [25](#_h4a77acser9c)

[Tcp Client](#_v5wisjd8um6z) [25](#_v5wisjd8um6z)

[**Further Development**](#_xar02v1v7k65)[**25**](#_xar02v1v7k65)

[Feature Backlog](#_rt0xcrk4jg60) [25](#_rt0xcrk4jg60)

[Known Bugs](#_5qkc9r9q3837) [25](#_5qkc9r9q3837)

[**Disclaimer**](#_gy92hkb1zmaw)[**25**](#_gy92hkb1zmaw)

[**Authors**](#_xuh6nns717pr)[**25**](#_xuh6nns717pr)

[Fall 2016 Development](#_zi4o3zr7bb5l) [26](#_zi4o3zr7bb5l)

[Special Thanks](#_kt15puq6mw37) [26](#_kt15puq6mw37)

# Getting Started

Follow the instructions to download and import the UWB Networking Package in your Unity project and setup your initial project structure.

**Important:** A separate project is needed for each device type you intend on including in your application. For example, if you would like to make an application for Vive and HoloLens, then you must create both a Vive project, and a HoloLens project.

Note: A project made for a specific device can be ran by multiple devices. For instance, a Vive project can be ran by multiple separate Vive machines in order to interact within the same game space.

## Installing Unity

Click the download link and follow the instructions to install the latest version of [Unity](https://store.unity.com/download?ref=update) (Personal Edition is fine).

Note, that you must be using Unity version 5.5 or above (needed for HoloToolkit).

## Using Unity

Unity is one of the easiest game engines to get started with, if you are unfamiliar with Unity, then you should take the time to explore the [Unity tutorials](https://unity3d.com/learn/tutorials). More specifically, you can look into [Unity VR Development](https://unity3d.com/learn/tutorials/topics/virtual-reality/vr-overview).

It is up to you, the developer, to learn how to develop for the device that you are using. The following links will be helpful if you are unfamiliar with Unity development for the following systems:

* [Microsoft HoloLens](https://developer.microsoft.com/en-us/windows/holographic/unity_development_overview)
* [Oculus Rift](https://docs.unity3d.com/Manual/VRDevices-Oculus.html)
* [HTC Vive](https://vrtoolkit.readme.io/)
* [Microsoft Kinect](https://developer.microsoft.com/en-us/windows/kinect/develop)

## Importing the UWB Networking Package

Once you have created a new Unity project, you are ready to add the UWB Networking Package to your project by following these steps:

1. Go to the [GitHub repository](https://github.com/UWB-ARSandbox/UWBNetworkingPackage/blob/master/UWBNetworkingPackage.unitypackage), and download the UWBNetworkingPackage.unitypackage file.



1. In your Unity project, select “Assets”, “Import Package”, and “Import Custom Package...” and choose to import the UWBNetworkingPackage.unitypackage file that you just downloaded.

 

* 1. If you deselect any of the assets included in the package, then you may have to import the dependencies (Photon Unity Networking and HoloToolkit) separately.
1. After importing the package, you should see the “UWBNetworkingPackage” folder in your Unity project (within the “Assets” folder).



* 1. The dependency folders should also be found in your Unity project: “HoloToolkit”, “Photon Unity Networking”, “PhotonChatApi”, and “Plugins”.

If you did not choose to import HoloToolkit or Photon Unity Networking, or they did not import properly, then you must do so manually.

**Disclaimer:** The UWB Networking Package may not work with newer versions of Photon Unity Networking or HoloToolkit.

##

## Importing Dependencies

Downloading dependencies separately from the UWB Networking Package is an option. Below are steps on how to download and import Photon Unity Networking and HoloToolkit manually. You may skip this section if you imported the dependencies while importing the UWB Networking Package.

### Importing Photon Unity Networking

To download and import Photon Unity Networking, open your Unity project, and select the “Asset Store” tab. In the Asset Store, search for “Photon Unity Networking”, and select the free version:



Click the “Import” button to import Photon Unity Networking into your project. Download everything from Photon Unity Networking (you may exclude the Photon Unity Networking Demos folder).

### Importing HoloToolkit

To download and import HoloToolkit, go to the [GitHub repository](https://github.com/Microsoft/HoloToolkit-Unity/blob/master/GettingStarted.md), and follow the given instructions.

Note: there may be namespace issues between Photon Unity Networking and HoloToolkit. To resolve these issues, you must edit the conflicting scripts manually so that no more errors occur (this is done for you when you import HoloToolkit via the UWB Networking Package). However, as HoloToolkit gets updated, a namespace may be added and this issue may be resolved.

## Specifying Photon Server Settings

To set up Photon, open the “PhotonServerSettings” file by searching for it within your Unity project folder. Your Unity inspector should show the following once the file has been selected:



Ensure that hosting is set to “Photon Cloud”, region is set to whatever is closest to you, and AppId is set to your given AppId.

Your AppId is a unique identifier created for your specific application. You can create and register a new AppId [here](https://doc.photonengine.com/en/realtime/current/getting-started/obtain-your-app-id).

**Important:** All projects that are attempting to connect to the same application must include the same Photon Server Settings (Host type, AppId, Region, etc.); otherwise, there may be connection errors.

Click to learn more about Photon Unity Networking [initial setup](http://doc.photonengine.com/en-us/pun/current/getting-started/initial-setup).

## Adding Network Manager Prefab

The next step is to add NetworkManager prefab into your Unity project’s hierarchy. To do so, go to /UWBNetworkPackage/Prefabs/ and select the “NetworkManager” prefab, and simply drag it into your hierarchy.





### Specify Device

Now that the NetworkManager prefab is in your hierarchy, select the prefab, to view it in the Unity inspector. You will see the following device options:



Now select whichever device will be running on your project.

**Important:** A Master Client device (project) is needed for each application.

## Master Client Setup

A Master Client project is needed to create a room for your application; luckily, a Master Client is easy to setup.

Simply specify that your project is a Master Client device in the NetworkManager prefab inspector options.



**Important:** Your Master Client should be the first project to run when starting your application. If there is no Master Client running, then your other device projects will be unable to connect to each other.

## Adding Spatial Mapping Prefab

The SpatialMapping prefab, which is provided by HoloToolkit, is needed for all projects that wish to send and receive a room mesh. You may skip this step if your application does not need to send a room mesh to all connected devices.

To add the SpatialMapping prefab into your Unity project’s hierarchy, go to /HoloToolkit/SpatialMapping/Prefabs/ and select the “SpatialMapping” prefab, and simply drag it into your hierarchy.





# Usage

Learn how to use the UWB Networking Package within your project.

## Object Synchronization

Learn how to synchronize objects within your application’s game space. Object synchronization allows for objects to be created, moved, and deleted across the network - these actions can be seen and done by all connected devices.

Object synchronization is up to the you, the developer, to implement since the needs of object synchronization will vary based on project requirements (it is content dependent).

### Photon Resources Folder

All objects that you want synchronized must be stored in the

/Photon Unity Networking/Resources/ folder.



**Important:** The Photon Resources folder must contain the same contents throughout all projects (HoloLens, Vive, MasterClient, etc.) if you want to have full synchronization of all instantiated objects. For example, an instantiated object will not appear in a scene if the project does not contain the instantiated object in its Photon Resources folder.

For instance, in each demo project (including MasterClientDemo) the Photon Resources folder contains the following elements:



Thus each object instance can be synchronized between all projects.

### Required Scripts

A synchronized object must have a PhotonView.cs, and a PhotonTransformView.cs script added to its components.

Both scripts are found within the “/Photon Unity Networking/Plugins/PhotonNetwork/” folder.

The easiest way to add them to an object is to select the object you wish to synchronize, in the Inspector for the object select “Add Component”, use the search bar to search for “Photon Transform View”, and select the Photon Transform View script (Photon View will automatically be added as a component as well).

 



**Important:** You must add the synchronized object into the “Observed Components” area of the Photon View that you just added. For example:



As you can see in the example, the Cube object’s transform has been added to its Photon View’s Observed Components list.

### Synchronization Options

The Photon View and Photon Transform View components allow for you to choose the object’s synchronization options.



Choose the options that best fit the needs of your synchronized object.

#### Ownership Option

The ownership option can either be set to “Fixed”, “Request”, or “Takeover”.

* Fixed - only allows one owner per object (based on instantiation).
* Request - allows other users to request object ownership from the current owner.
* Takeover - allows other users to acquire object ownership without permissions from the current owner.

Object ownership is explained in further detail in the [Object Ownership](#_m6eeefgedx3p) section.

#### Observe Option

The field Observe Option lets you select how and when updates are sent. There are three options: “Off”, “Unreliable”, “Unreliable on Change”, and “Reliable Delta Compressed”.

* Off - updates are never sent.
* Unreliable - updates are sent "as is" but could become lost. Will send updates even if the object stops moving.
* Unreliable on Change - will check each update for changes. Works like “Unreliable”, but is better suited for objects that stay in one place for longer periods of time.
* Reliable Delta Compressed - will compare each value of an update to its predecessor. Updates are more reliable, but overhead is higher.

For further detail please visit Photon’s [Photon View - Observe Option](https://doc.photonengine.com/en/pun/current/getting-started/feature-overview#_viewpun) documentation.

#### Transform Serialization

Allows you to select which transform attributes to synchronize:

* Only Position - only synchronize the object's position.
* Only Rotation - only synchronize the object’s rotation.
* Only Scale - only synchronize the object’s scale.
* Position And Rotation - synchronize everything but scale.
* All - synchronize all transform attributes.

### Object Instantiation

To instantiate a synchronized object, means to create an instance of the object within your current project and all connected projects.

An instantiated object MUST exist within the Photon Resources folder (otherwise instantiation will fail).

To instantiate an object, call the function “PhotonNetwork.Instantiate(string objectName, Vector3 position, Quaternion rotation, int group)”. To have access to the method, any script that calls PhotonNetwork.Instantiate() from must extend “PunBehavior”.

For an example, refer to “ViveHeadTracking.cs”, which is used in the [Vive Demo](#_r8953te41a8q) to instantiate a red cube to represent a Vive player’s head.



In the example, a “ViveHeadObject” instance is instantiated when the current project joins the room (game space). It is up to you to decide when objects are instantiated, for instance, you could create an event (such as a button click) to trigger object instantiation. More examples can be found in the [Demo Projects](#_icoz395vsg73).

To learn more about object instantiation, visit Photon’s [Instantiation](https://doc.photonengine.com/en-us/pun/current/manuals-and-demos/instantiation) documentation.

### Object Ownership

The owner of an object is the player (or user) that instantiated the object (unless PhotonNetwork.InstantiateSceneObject() was used - in that case the Master Client becomes the owner of the object).

When a GameObject's current owner leaves, the Master Client becomes the new owner. The player who created a GameObject has to request ownership to get it back.

**Important**: A project (or user) must have ownership over a synchronized object before updating the object’s transform; otherwise, the updates to the object’s transform will not be reflected across the network.

For example, let’s say a cube was instantiated by a HoloLens. If a Vive user wanted to move the cube, it should first acquire ownership of it. If the Vive user acquires ownership of the cube, then the transform of the cube would be updated across all projects connected to that room (game space). If the Vive user did not acquire ownership of the cube, then the object’s transform will only be updated within the Vive project, and inconsistencies within the game space would occur.

#### Requesting Ownership

To request for the ownership of an instantiated object, you must have reference to that object instance’s Photon View. From there you can call obj.photonView.RequestOwnership().

An example of requesting object ownership can be found in “ObjectOwnership.cs”, which is used in the [Vive Demo](#_r8953te41a8q).

In the example script, the player requests object ownership whenever the object is touched by the Vive player.

 

#### Transferring Ownership

To transfer ownership of an instantiated object, you must already have ownership of the instance of the object you want to transfer, reference to that object’s Photon View, and the Player ID of the player your wish to transfer to. From there you can call obj.photonView.TransferOwnership(requestingPlayer.ID).

Examples of transferring ownership are not in the demo projects, but you can refer to Photon’s [Ownership Transfer](https://doc.photonengine.com/en-us/pun/current/manuals-and-demos/ownership-transfer) documentation to learn more.

To learn about the different types of object ownership refer to [Ownership Option](#_tjp5ocecnka6).

### Object Deletion

To delete a synchronized object across the network, call PhotonNetwork.Destroy(GameObject). Any script that calls this method must extend “PunBehavior”.



### Further Information

For further information about using Photon Unity Networking, refer to the [Photon Documentation](https://doc.photonengine.com/en-us/pun/current/tutorials/pun-basics-tutorial/intro).

## Room Mesh Networking

### Creating a Room Mesh

A project with the device specified as “HoloLens”, will start observing the physical room around the HoloLens user when the project is first started. The HoloLens user should take the time to scan (look at) objects in the room that they want incorporated in the room mesh.

### Sending a Room Mesh

The method, “HoloSendMesh()”, which exists in the NetworkManager.cs script (/UWBNetworkingPackage/Scripts/NetworkManager.cs) and is a component of the NetworkManager prefab, sends the mesh that the HoloLens has been constructing since runtime.

Calling the HoloSendMesh() function in your HoloLens project will send the room mesh to the Master Client. The Master Client will send a room mesh to all other devices (except other HoloLens devices) including devices that connect to the game space after the HoloLens sent the mesh.

Your project may execute HoloSendMesh() however you see fit. You may refer to our [HoloLens Demo](#_kny1vtrgfcdn) project to see how we call HoloSendMesh() via HoloLens voice commands.

### Receiving a Room Mesh

All non-HoloLens devices receive the sent room mesh. If the device was connected when a HoloLens sends a room mesh, then the device will receive the room mesh right away. If the device joins the game space after the mesh was sent, then the device will receive the room mesh when it joins.

### Displaying a Room Mesh

Once a room mesh is received, it is displayed in the project’s scene (true for every device except HoloLens - HoloLens only sends a room mesh, but never receives one).

# Common Photon Connection Issues

* Different versions of Photon Unity Networking.
	+ Solution: Update all clients’ Photon Unity Networking to the newest version.
* Different Room Name.
	+ Solution: In NetworkManager prefab, modify the Room Name property value to be the same across all projects.
* Different Photon Region.
	+ Solution: Search “PhotonServerSettings” in Assets folder. Double click it to modify the region so all clients are the same (the chosen region should be specific to your location).
* Different App ID.
	+ Solution: Search “PhotonServerSettings” in Assets folder. Double click it to modify the App ID so all projects are contain the same one.
* Different Scene Names.
	+ Solution: Rename your scenes so all projects contain the same scene names.
* Different RPC methods
	+ Solution: Search “PhotonServerSettings” in Assets folder. Double click it to verify that the RPC lists are the same across the clients. If they are not, you can:
1. Manually modify the RPC List.
2. Clear the RPC List by clicking “Clear RPCs”, then refresh the RPC List by clicking Refresh RPCs.
* Different Photon Resource folder contents
	+ Solution: Make sure resource folder are the same across all projects.
* Cannot display Room Mesh.
	+ Solution: Add SpatialMapping prefab from HoloToolkit to hierarchy.

# Demo Projects

Each demo project utilizes the UWB Networking Package to demonstrate how each device can connect and interact within the game space.

## Master Client Demo

A demo project that utilizes the UWB Networking Package to create a Master Client.

The Master Client requires no special scripts, just select the "Master Client" option in the NetworkManager prefab.

Visit the Master Client Demo [GitHub repository](https://github.com/UWB-ARSandbox/MasterClientDemo).

## HoloLens Demo

A demo HoloLens project that utilizes the UWB Networking Package.

In this simple demo, a HoloLens user is able to create synchronized network objects via the voice commands "Create Cube" and "Create Sphere", move synchronized network objects via the voice commands "Move Cube" and "Move Sphere", and send a Room Mesh via the voice command "Send Mesh".

Visit the HoloLens Demo [GitHub repository](https://github.com/UWB-ARSandbox/HoloLensDemo) and view the HoloLens specific scripts located in the /Assets/HoloLensDemo/ folder to learn more.

## Vive Demo

A demo Vive project that utilizes the UWB Networking Package.

In this simple demo, a Vive user is able to create synchronized network objects by pressing the touchpad button, and move synchronized network objects by touching them and holding the trigger button.

Visit the Vive Demo [GitHub repository](https://github.com/UWB-ARSandbox/ViveDemo) and view the Vive specific scripts located in the /Assets/ViveDemo/ folder to learn more.

## Oculus Demo

A demo Oculus project that utilizes the UWB Networking Package.

In this simple demo, a Oculus user is able to create, view and move graphic object in virtual shared space. To create a new cube, pointing gaze pointer to a empty place and press select button on Oculus Remote. To create select a graphic object, pointing gaze pointer to the object then press select button. To move object horizontally, select the object then use left and right button to move. To move object vertically or along Z axis, select the object then use select button to change mode; use up and down button to move. To finalize the position, use back button.

Visit the Oculus Demo [GitHub repository](https://github.com/UWB-ARSandbox/OculusDemo.git) and view the Oculus specific scrips located in /Assets/OculusDemo/ folder to learn more

## Kinect Demo

A demo Kinect project that utilize the UWB Networking Package.

In this simple demo, a Kinect user can create and view graphic object in virtual space created by networking package. To create a sphere, press "s" once connected to the room. To create a cube, press "c" once connected to the room.

Visit the Kinect Demo [GitHub repository](https://github.com/UWB-ARSandbox/KinectDemo.git) and view the Kinect specific scripts located in /Assets/KinectDemo/ folder to learn more.

#

# Project Background

## Summer 2016 Work

During Summer 2016, a previous group of students created a network infrastructure for the HoloLens, Kinect, and Vive. Although, the infrastructure worked, it was difficult to set up, non-reliable, hard to maintain and refactor, difficult to expand (add new devices), and used hard coded IP Addresses. That led to need of creating a reliable, expandable, and easy to use networking package, that can allow mixed reality developers to focus on content creation rather than network implementation.

## Why Photon Unity Networking

During the initial research and design phase of the project, we researched and tested a wide range of implementation solutions, from Windows Communication Foundation (WCF) to a Network File System (NFS). It was especially difficult to find a solution that works across all devices, especially the HoloLens.

Luckily, while at a Seattle mixed reality convention, Immerse Summit, we discussed our issues with Pete Moss, one of the head developers for Unity’s Virtual Reality department, and he recommended looking into Photon.

Photon meet our expectations perfectly, as it works on all devices, provides us with object synchronization functionality, and allows us to use RPC calls to avoid hard coded IP addresses.

Photon also includes Photon Server SDKs - and writing our own Photon Server is one of the goals for Winter 2017.

## Accomplishments

* Minimum setup, you just need to specified AppID and room name.
* No hard coded IP adresses.
* Easy to extend to new platform.
* Easy to add extra functionality to each device.

## Limitations

* 500 msg/s with Photon for free version.
* HoloLens cannot connect on campus (internet is required with Photon Servers), but this is an issue specific in our campus.

# Class Design

## Database

Database class is used to store room mesh. It stores the mesh as bytes array but you can pass both IEnumerator<Mesh> or byte[] to the class to store it. You can also retrieve the mesh as IEnumerator<Mesh> or byte[]. This is a static class.

## NetworkManager

NetworkManager will add selected launcher component. You can select from the following devices: Master Client, HoloLens, Vive, Oculus, and Kinect.

## Launcher

Launcher will connect you to PUN using specified AppID, version number, and room name. If a connection is failed, it will disconnection from PUN and terminate the program. Launcher also contain 4 RPC methods for sending and receiving room mesh. We will introduce the detail of these method in the following section. Launcher is an abstract class.

## MasterClientLauncher : Launcher

Master client launcher will be run when you select Master Client as your device in NetworkManager. The launcher will create a room instead of joining one when connecting to PUN. It will then start a TCP listener on a new thread that will wait for connection from ReceivingClient. Once a new connection is established, it will send the room mesh to the client if there is one. SendMesh(int) is a RPC method, that will take a player id as a parameter from the receiving client. It will then send its ip address and port number to this client by calling ReceiveMesh(String) at the receiving client. ReceiveMesh(int) is a RPC method, that will take the player id as a parameter from Hololens. Once this is called, it will then send its ip address and port number to Hololens by calling SendMesh(String) at Hololens.

## HololensLauncher : Launcher

HoloLens launcher will be run when you select Hololens as your device in NetworkManager. The launcher will observe and map your surroundings when you connect to Photon. It also contains methods that will send the room mesh to the master client when you say “send mesh”. SendMesh(String) is a RPC method, that will take the network configuration as a parameter from the Master Client. The network configuration contains an ip address and port number, and the room mesh will be sent to the host specified by these. Calling the other three RPC methods will print out error statements to the log.

## ReceivingClientLauncher : Launcher

Receiving client launcher will ask the master client for the room mesh once it joins a room by calling SendMesh(int) at the master client. ReceiveMesh(String) is a RPC method that will help the receiving client connect to the master client and receive the mesh.

## ViveLauncher : ReceivingClientLauncher

No extra functionality has been implemented, but you can use this script to implement Vive specific functionality during further development.

## OculusLauncher : ReceivingClientLauncher

No extra functionality has been implemented, but you can use this script to implement Oculus specific functionality during further development.

## KinectLaucher : ReceivingClientLauncher

No extra functionality has been implemented, but you can use this script to implement Kinect specific functionality during further development.

# Networking Implementation Details

## Photon Unity Networking (PUN)

Photon Unity Networking (PUN) is used to replace Unity Networking (Built-in) and part of the socket programming we built in the summer. PUN is a good fit because we can synchronize action and game objects across members in same room. PUN is also really easy to set up. To learn more about PUN, please go [here](https://www.photonengine.com/en-US/PUN).

### Remote Procedure Calls (RPC)

PUN also allows us to use RPCs, so we don’t need to hardcode master client’s IP address for any other clients. This makes our project really easy to setup. To learn more about RPCs in Photon, please go [here](https://doc.photonengine.com/en-us/pun/current/manuals-and-demos/rpcsandraiseevent).

### Photon View

For every object that has appearance on network, you need to add PhotonView component. Photon view allow as sync transform of a game object across network. To read more about PhotonView, please go [here](http://doc.photonengine.com/en-us/pun/current/getting-started/feature-overview#_viewpun).

### Rooms and Lobbies

Once you are connected to a game on PUN, you will be put in a lobby. The lobby is shared with others who connect to PUN using the same AppID and Version number. Once you are in a lobby, you will be allowed to join the room. All game objects and RPCs are sync across rooms. To learn more about rooms and lobbies, please go [here](https://doc.photonengine.com/en-us/pun/current/tutorials/pun-basics-tutorial/lobby).

### Master Client

When you create a room, you will automatically become a master client. When a master client leave, the member who has the lowest player id will become the master client. In our network package, the master client will ensure all clients have the latest room mesh.

## Room Mesh

Room mesh is scanned and produced by HoloLens. The mesh will be sent to the master client and the master client will distribute it. The master client can store the latest mesh when it is shut down. To load the stored mesh, press “l” (“L” button) on the keyboard.

Room mesh is the only dependency we have currently with HoloToolKit.

To learn more about room mesh, please go [here](https://github.com/Microsoft/HoloToolkit-Unity/blob/master/Assets/HoloToolkit/SpatialMapping/README.md).

### Directives

We use lots of directives to determine whether the following code is for HoloLens or not. Because HoloLens is using Universal Windows Platform, some of .NET framework functions cannot be applied.

The non-hololens code needs to be put between:

#if UNITY\_EDITOR && !UNITY\_WSA\_10\_0

#endif

### HoloLens

#### Stream Sockets

Stream Socket is used to send mesh from HoloLens to master client. This is the only socket we found that works on HoloLens (in Unity 5.4 HoloLens Version). We haven’t tried on the newest version which is 5.5. On 5.4 HoloLens Version, we tried TCPListener, and Sockets.

### Master Client

#### Tcp Listener

We have a multithreading TCP Listener wait at port 25827 by default (you can change this to anything you want in NetworkManager) for connection. Once a connection is established, it will create another thread to send room mesh to the connected client.

### Receiving Clients

HTC Vive, Oculus Rift, Microsoft Kinect

#### Tcp Client

RPC method ReceiveMesh(String) will create a TcpClient to receive mesh from master client.

# Further Development

## Feature Backlog

* Allow for multiple room meshes to be sent by a single HoloLens (rather than having a HoloLens start observing when it joins the room, have a method that allows the HoloLens to start observing).
* Remove HoloToolkit dependency from non-hololens projects.
* Unity editor scripting for NetworkingManager.cs and object synchronization.
* Self hosted Photon Server.
* Resource folder synchronization.
* Texture mapping for the Room Mesh.

## Known Bugs

* Sending room mesh over internet, this might be a firewall issue.

# Disclaimer

The UWB Networking Package was developed by students and is maintained by students, with new students joining the project and previous students leaving the project after each quarter. Thus the project may not stay up to date and may not work with newer versions of Photon and HoloToolkit.

# Authors

This project has been developed by students of the University of Washington Bothell under the mentorship of Doctor Kelvin Sung.

## Fall 2016 Development

Michael O’Keefe

Shen Li

### Special Thanks

* For attending and helping during development sessions: Destiny Boyer, Riz Ramdhani
* For documentation testing/advice: Thomas Brown
* For work done during Summer 2016 (especially in regards to Room Mesh sending): Duncan MacMichael, Keng Yang, Michael Tanaya, Taran Christensen
* For helping test with multiple HTC Vives: Taran Christensen
* For attending and being involved during weekly meetings: Destiny Boyer, Keng Yang, Michael Tanaya, Rizky Ramdhani, Taran Christensen, Thomas Brown
* For informing us about Photon Unity Networking: Pete Moss

##