# Get Post

## NetworkManager

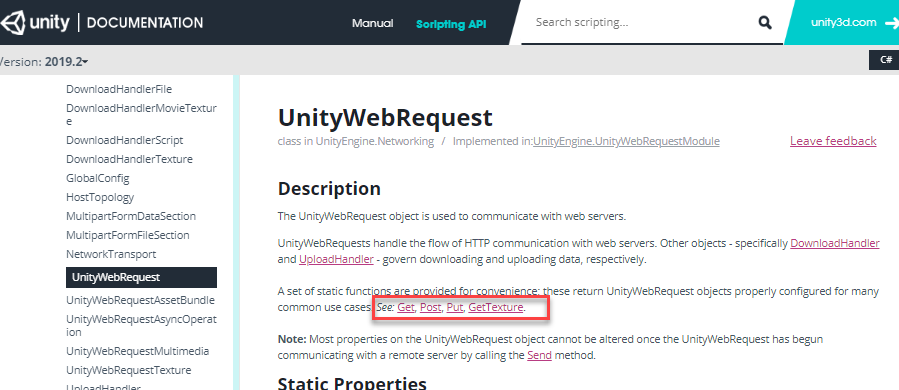
Okay to connect to the server I need some sort of manager

Okay so we need to get the post requests so We need somebody to manager all of them.

And I need someone who will get it all organized.

## Unity web requests

<https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html> - so there is all types of request put get and e.t.c



So this is basically what I need now I need to run some sort of test.

# Mltiplayer for unity

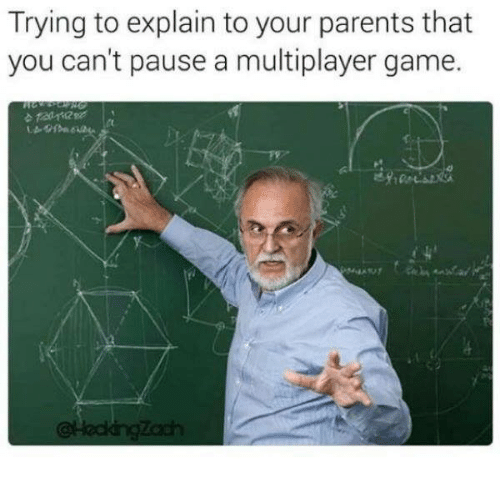
## Refrence

[Azure Back End](https://channel9.msdn.com/Series/Microsoft-Azure-Back-End-for-Gaming/05)

[Unity normal without unity Mutliplayer](https://stackoverflow.com/questions/36091976/can-i-use-the-unity-networking-hlapi-without-paying-for-the-unity-multiplayer-se/36092168)

[Legacy networing](https://docs.unity3d.com/Manual/net-HighLevelOverview.html)

[Valve Networing](https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking)



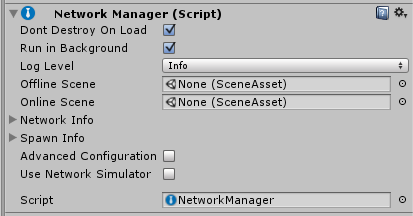
## High Level API

### Refrences

* [Networking overview Unity](https://docs.unity3d.com/Manual/UNetOverview.html?_ga=2.254329867.22273773.1498263144-2120895595.1497649407)
* [HLAPI Unity](https://docs.unity3d.com/Manual/UNetUsingHLAPI.html?_ga=2.254329867.22273773.1498263144-2120895595.1497649407)
* [Using Network Manager](https://docs.unity3d.com/Manual/UNetManager.html?_ga=2.142811574.22273773.1498263144-2120895595.1497649407)
* [Remote Actions](https://docs.unity3d.com/Manual/UNetActions.html?_ga=2.185141546.22273773.1498263144-2120895595.1497649407)
* [State Synchronization](https://docs.unity3d.com/Manual/UNetStateSync.html?_ga=2.11392884.1512550543.1498284585-554096225.1498284585)

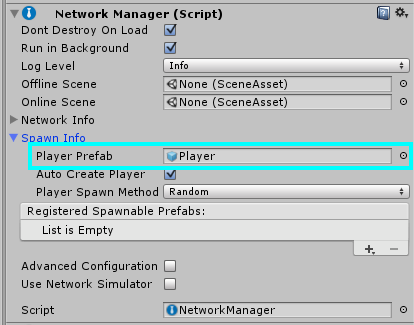
### Components

#### Network Manager

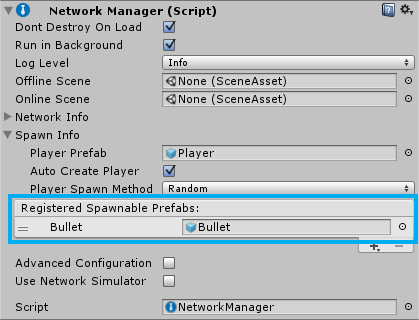


The NetworkManager component manages the network state of the game.

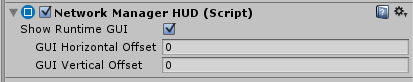
##### Register Player Prefab



##### Register Spawnable Prefabs

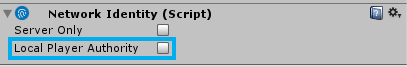


#### NetworkManagerHUD

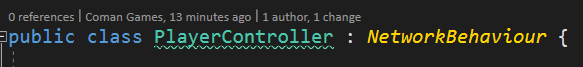
The NetworkManagerHUD component works with the NetworkManager and provides a simple user interface to control the network state of the game when the game is running.

#### NetworkIdentity



The NetworkIdentity component is used to identify the object on the Network and to make the Networking system aware of it.

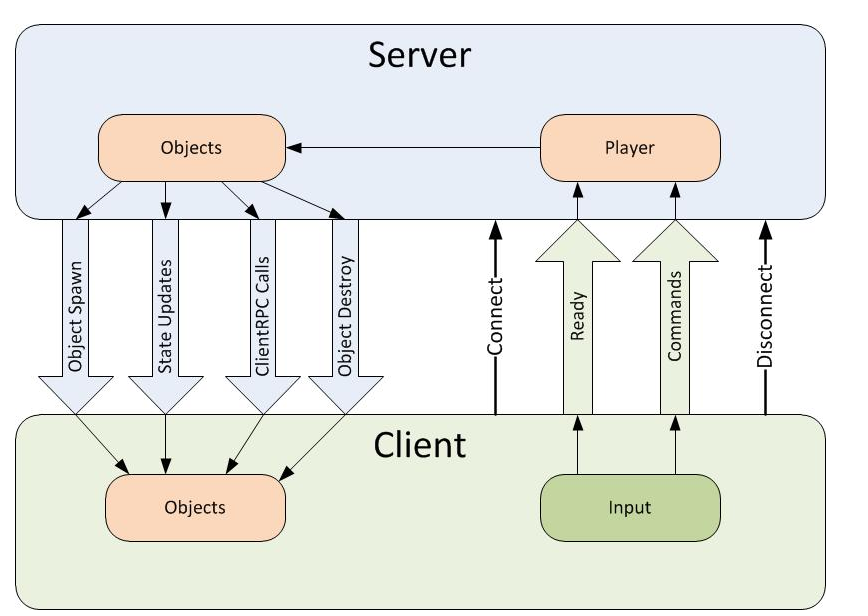
#### NetworkBehaviour

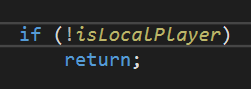


You inherit from it instead of MonoBehaviour.

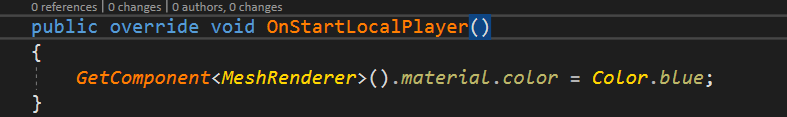
[Manual](https://docs.unity3d.com/Manual/class-NetworkBehaviour.html?_ga=2.142262321.22273773.1498263144-2120895595.1497649407)

[Documentation](https://docs.unity3d.com/ScriptReference/Networking.NetworkBehaviour.html?_ga=2.153758779.22273773.1498263144-2120895595.1497649407)

 j

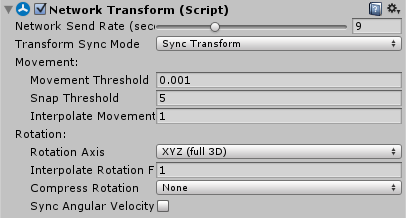


##### OnStartLocalPlayer



To do actions with local player not with all game in general.

#### NetworkTransform



Syncronize all transforms of the object across the network

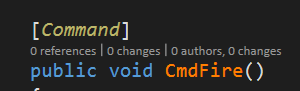
##### Network Send Rate



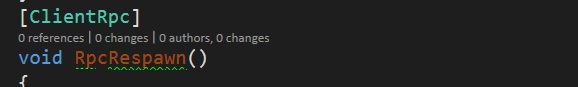
Count of updates on player

#### Network Server

##### Command

 - this attribute indicates that the following function will be called by the Client, but will be run on the Server. when making a networked command, the function name must begin with “Cmd”.

##### ClientRpc

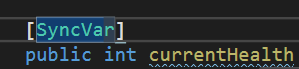


ClientRpc's are the opposite of Commands. Commands are called on the Client, but executed on the Server. ClientRpc's are called on the Server, but executed on the Client.

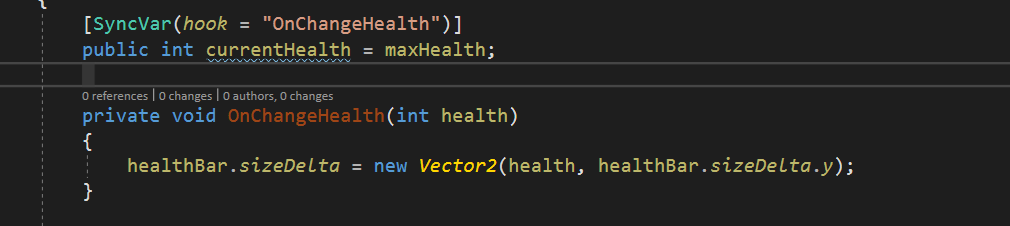
##### Spawn

 - for spawning object on the server

##### SyncVar

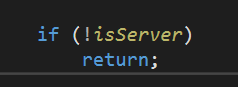
 - so the variables we wan’t to be synchronized throw network we use SyncVar attribute.

##### SyncVar hook

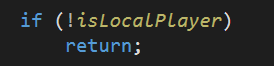


So when some var is changing we wan’t be able to see the changes.

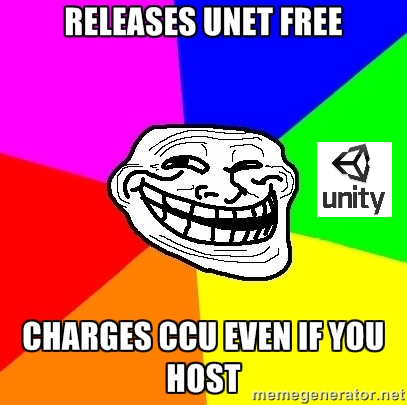
##### isServer

 - to check is action happening on server

##### isLocalPlayer

 Checking is from local Player side

## Photon

\

### Client



Unity Photon in total

#### References

1. [Class specification documentation](https://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_behaviour.html)
2. [Step by step tutorial](https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/pun-basics-tutorial/intro)

#### Basics

##### Rooms



The place for game on server with limited players count let say 10. So players can communicate only inside of the room.

##### Lobby



The place on server where you can get list of rooms.

###### Auto-join Lobby

This is off by default, as the better option is to use Random Matchmaking, as used in all demos.

If "Auto-Join Lobby" is unchecked, OnConnectedToMaster callback will be called. Otherwise, OnConnectedToMaster will be skipped and only OnJoinedLobby will be called.

##### Protocol

UDP is default and most likely .

TCP is also supported but as you know.

##### Game Version

A simple way to siparate player with old and new version.

Is neccessery only when you work with photon cloud dosen’t matter on self hosted.

Console and PC players e.t.c

##### Connect Manually

As alternative to PhotonNetwork.ConnectUsingSettings() you can connect to your own Photon Servers by using PhotonNetwork.ConnectToMaster(). This is useful when you host Photon On-Premises.

For ConnectToMaster(), you need to provide a masterServerAddress and a port. The address is either your On-Premises DNS name or an IP. It can include the port after a colon (then pass 0 as port) or you can pass the port separately.

ConnectToMaster() has two more parameters: "appID" and "gameVersion". Both are only relevant for the Photon Cloud and can be set to any value when you host Photon yourself.

For the Photon Cloud, use ConnectUsingSettings(). It involves our Name Server to find the Master Server of a region automatically.

##### Unity implementation

##### Connecting to server

###### Using settings file



###### For manual connection use

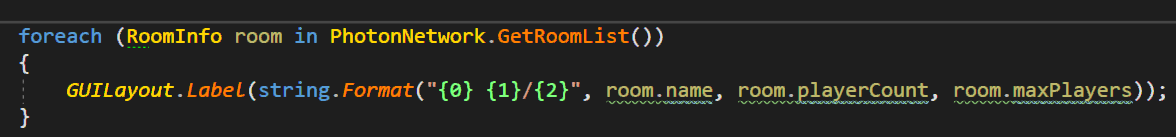
Connect() to ignore the PhotonServerSettings file.

##### Rooms

##### Joining and creating

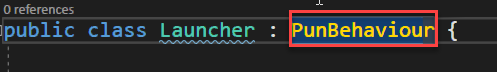


###### Getting room list



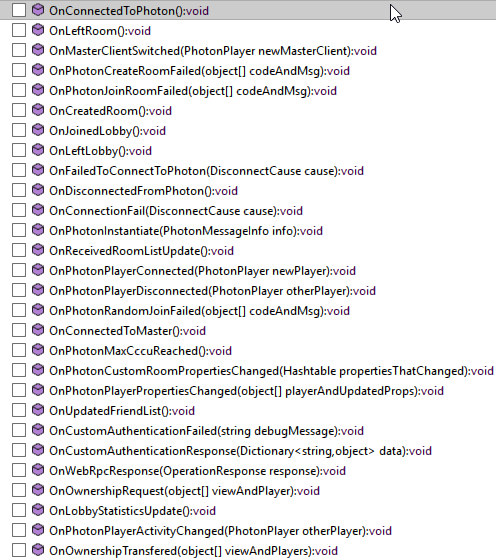
###### PunBehavoiour

Instead of usuall Mono Behaviour now is PunBehaviour



Methods To overwrite

[Link to full documentation and specialization](https://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_behaviour.html)



##### Communication

Two methods to communicate

1. RPC (Remote Protocol Calls)
2. By implementing OnSerializePhotonView in a script that is observed by a PhotonView.

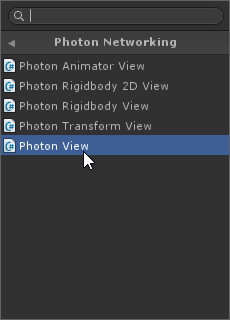
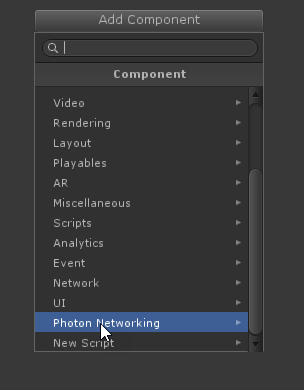
##### Photon View

You select what to observe or you need to observe just postion rotation and scale or you need to observer something more complicated like monobehaviour.

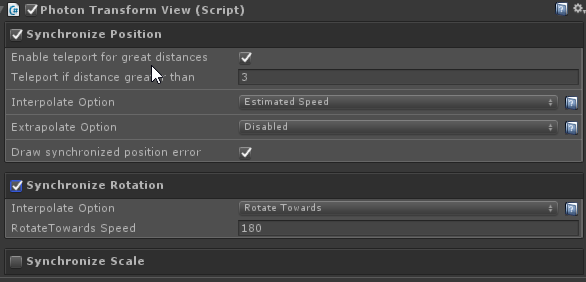
PhotonView is a script component that is used to send messages (RPCs and OnSerializePhotonView). You need to attach the PhotonView to gameobjects or prefabs. Note that the PhotonView is very similar to Unity's NetworkView.

At all times, you need at least one PhotonView in your game in order to send messages and optionally instantiate/allocate other PhotonViews.

###### Add Photon View



###### Photon Transform Observer

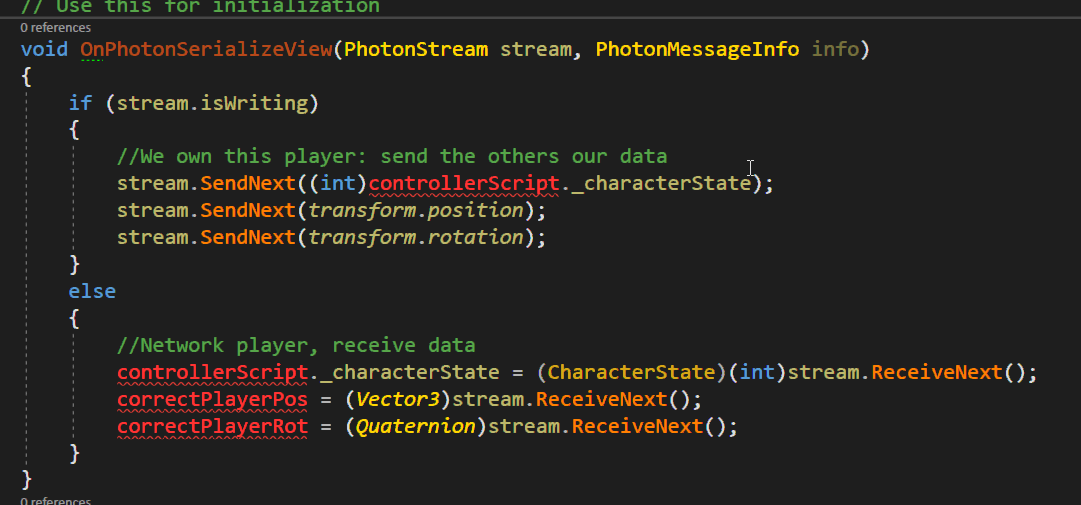


For Synchronization position rotation and scale

###### Observe Mono Behavior

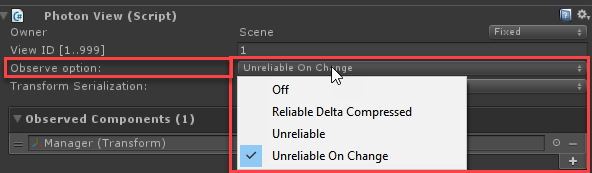
A PhotonView can be set to observe a MonoBehaviour. In this case, the script's OnPhotonSerializeView method will be called.

Example of character state synchronization



###### Observer Options

The field Observe Option lets you select how and when updates are sent. It also affects how often OnPhotonSerializeViewgets called.



**Off** explains itself. Can be useful if a PhotonView is reserved for RPCs only.

**Unreliable** updates are sent "as is" but could become lost. The idea is that the next update arrives soon and provides the correct/absolute values needed. This is good for position and other absolute data but not good for triggers like switching weapons. When used to sync a GameObject's position, it will always send updates, even if the GameObject stops moving (which is bad).

**Unreliable on Change** will check each update for changes. If all values stay the same as previously sent, one update will be sent as reliable and then the owner stops sending updates unless things change again. This is good for GameObjects that might stop moving and that don't create further updates for a while. Like Boxes that are no longer moved after finding their place.

**Reliable Delta Compressed** will compare each value of an update to its predecessor. Values that did not change are skipped to keep traffic low. The receiving side simply fills-in the values of previous updates. Anything you write via OnPhotonSerializeViewatomatically gets checked and compressed this way. If nothing changed, OnPhotonSerializeView doesn't get called on the receiving clients. The "reliable" part here takes some toll, so for small updates, the overhead should be considered.

##### RPC

[More about RPC](https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent)

"Remote Procedure Calls" enable you to call a method on all clients in a room. PUN keeps a list of those methods in the PhotonServerSettings. For the initial setup, this is not relevant however. See [Remote Procedure Calls](https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent).

Methods that should be called as RPC must be on a GameObject with a PhotonView component. The method itself must be marked by the attribute: [PunRPC].

###### Call RPC

To call the method, access to the target object's PhotonView. Instead of directly calling the target method, call PhotonView.RPC() and provide the name of the method to call:



##### Timing for loading Levels(scenes)

[Link](https://doc.photonengine.com/en-us/pun/current/getting-started/feature-overview#_messageQ)

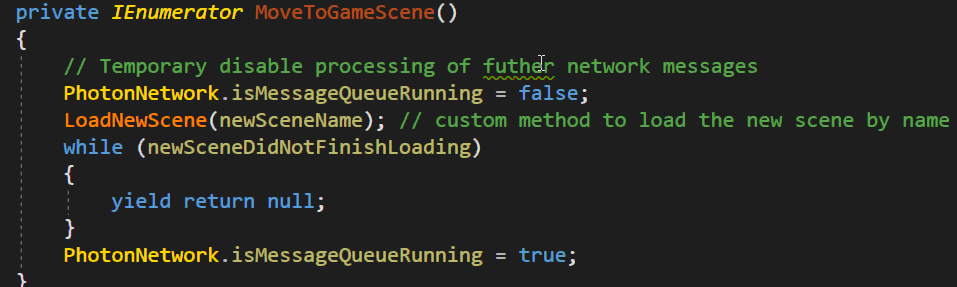
RPCs are called on specific PhotonViews and always target the matching one on receiving clients. If a remote client did not load or create the matching PhotonView yet, a RPC gets lost!

Due to that, a typical cause for lost RPCs is when clients load new scenes. It just needs one client which already loaded a scene with new GameObjects and the other clients can't understand this one (until they also loaded the same scene).

PUN can take care of that. Just set PhotonNetwork.automaticallySyncScene = true before you connect and use PhotonNetwork.LoadLevel() on the Master Client of a room. This way, one client defines which level all clients have to load in the room / game.

To prevent losing RPCs a client can stop executing incoming messages (this is what LoadLevel does for you). When you get an RPC to load some scene, immediately set isMessageQueueRunning = false until the content is initialized. Disabling the message queue will delay incoming and outgoing messages until the queue is unlocked. Obviously, **it's very important to unlock the queue** when you're ready to go on.

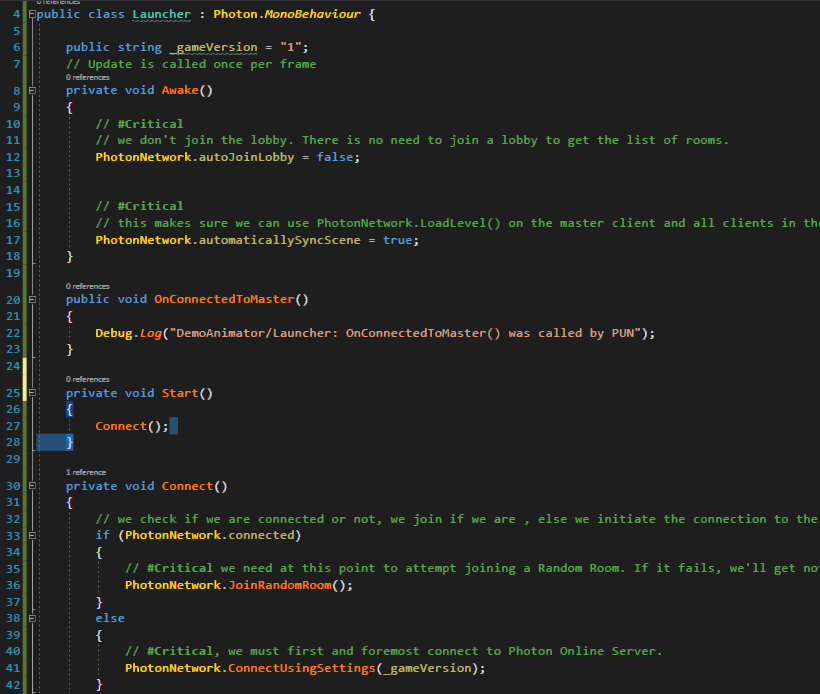
###### Example



#### Implementation

##### Connecting to server

Basic script example



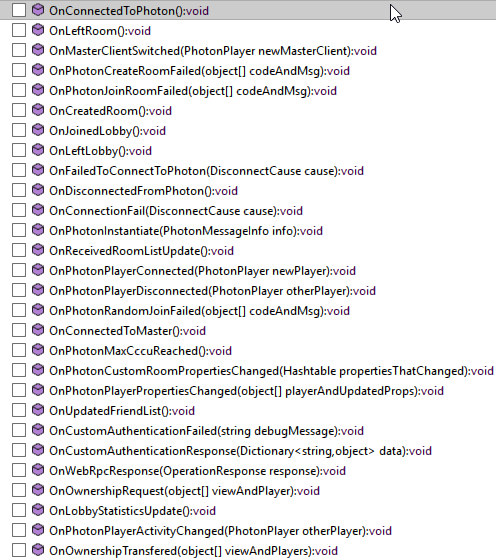
##### Pun Call bacsk

Getting information from the Pun about current game state.

###### Overriding methods in PunBehaviour

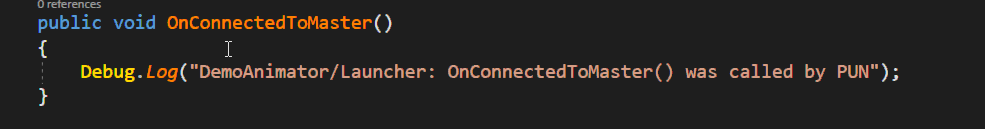
It is main and best way just create your custom class and override methods.





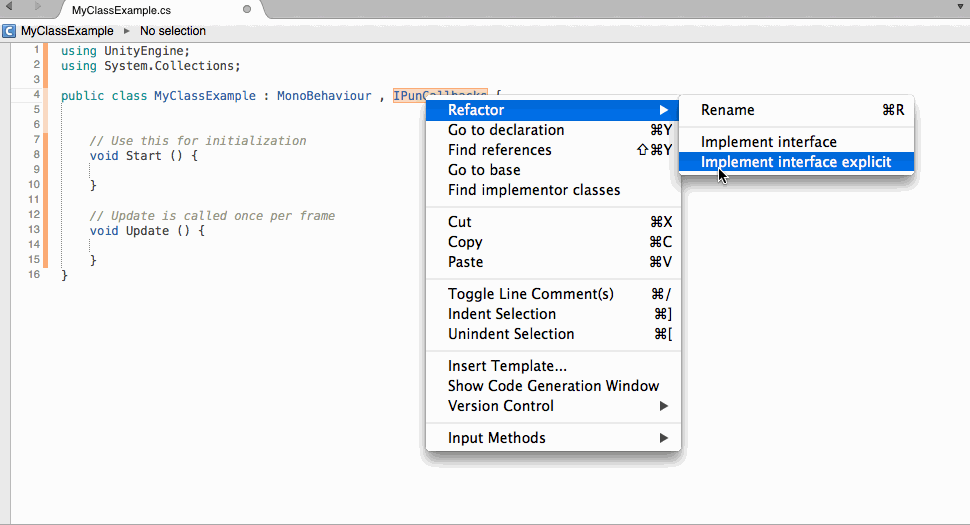
###### “Magical” methods on MonoBehaviour

On any monobehaviour object you can get event when we are connected to the server.



###### Implementing interface

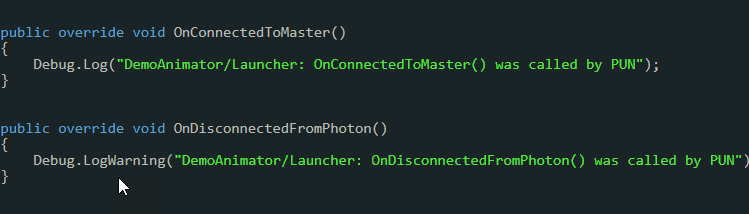
PUN is providing two c# [Interfaces](https://msdn.microsoft.com/en-us/library/ms173156.aspx) that you can implement in your class: IPunObservable and IPunCallbacks.



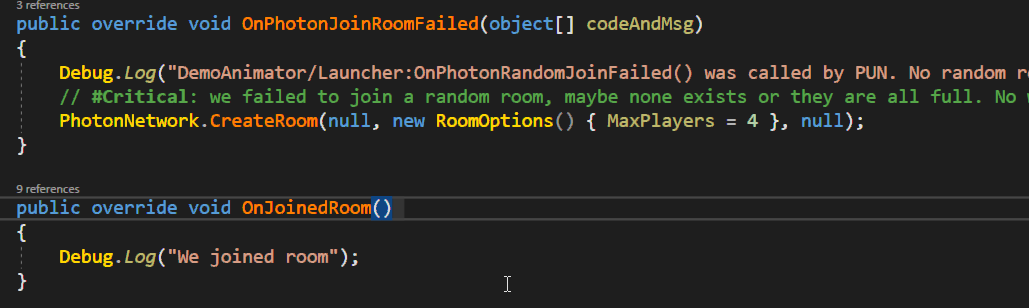
#### PunBehavour

[Official Documenation](http://doc-api.photonengine.com/en/pun/current/class_photon_1_1_pun_behaviour.html)

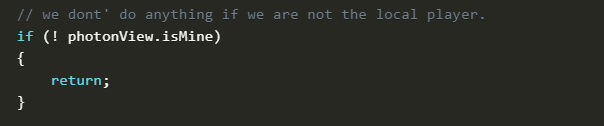
###### On connected on disconnected



###### Joining the room and on joining to room failed

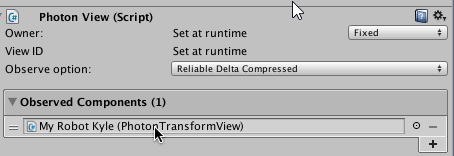


###### Check is local player

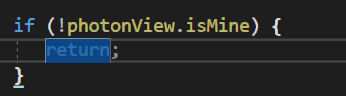


#### PhotonView

Photon view is simply class who updated all throw the network.



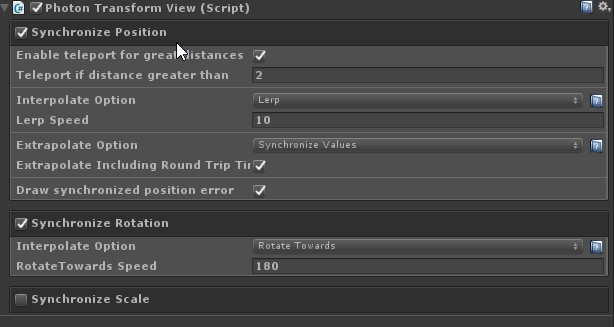
##### Is Local player



##### Extra

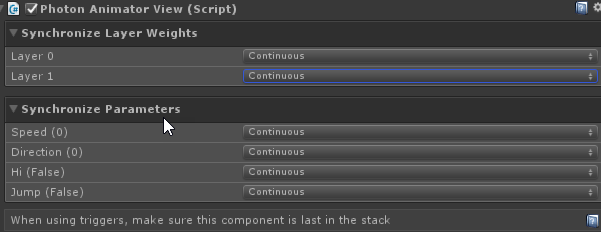
###### Photon Transform Veiw

For Transform synchronization



###### Photon Animator View

For animator synchronization.

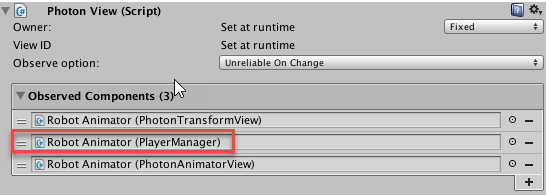


Discrete synchronization not so smooth but less work for network packet.

Continuous  - means it more smooth but more hard for network

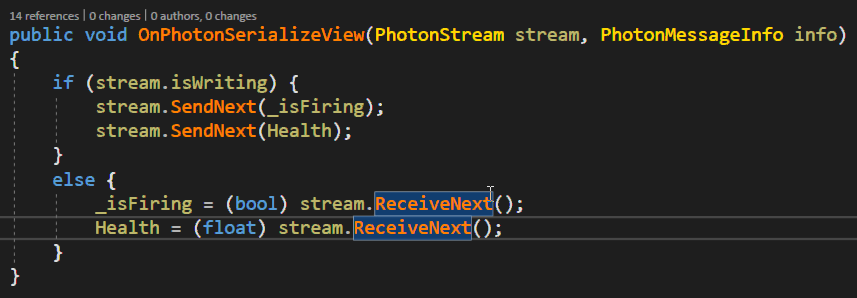
##### IPunObservable(sync data)

Is Interface that we implement to be able observe with PhotonView any class



###### Example

Of getting and putting data in stream



#### PhotonNetwork

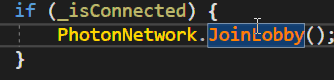
##### Player name

PhotonNetwork.playerName = value + " ";

##### Master Client

PhotonNetwork.isMasterClient

##### Join Lobby



##### LoadLevel



##### Instantiate prefabs

**WARNING:** Always make sure Prefabs that are supposed to be instantiated over the network are within a [Resources](http://docs.unity3d.com/ScriptReference/Resources.html) folder, this is a Photon Requirement.



### Server



#### Basics

##### Application

An Application is the server side logic for a game. All features of a game (e.g. remote procedure calls, data storing, etc.) are implemented in a Photon application.

##### Game Logic

The game logic defines how a client can interact with the server. It implements operations, events, and anything that the server does by itself.

A good basis for room based games is provided as ["Lite" Application](https://doc.photonengine.com/en-us/onpremise/v3/reference/lite/lite-concepts), found in the SDK folder src-server/Lite. It does not know (your) game's logic (yet), but provides rooms where players can communicate and interact easily. The application ["Lite Lobby"](https://doc.photonengine.com/en-us/onpremise/v3/reference/lite/lite-lobby-concepts) extends Lite with lists of rooms if you want your users to select a room manually.

##### Operations

An operation is Photon's equivalent to a remote procedure call. Photon clients use operation calls for anything they want to get done.

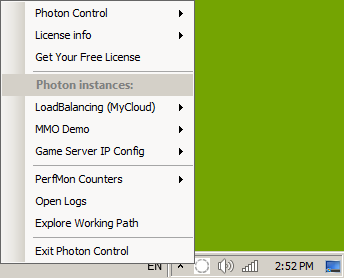
##### Events

Photon Events are messages for clients. Each event is typed by a byte code and carries game updates. The Lite Application defines several events but it's possible to define custom events purely on the client side.

#### Phton Control

Refrence

<https://doc.photonengine.com/en-us/onpremise/current/applications/photon-control-application>

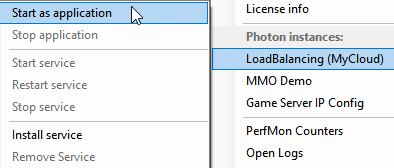


#### Master (Loading balancing,MyCloud)

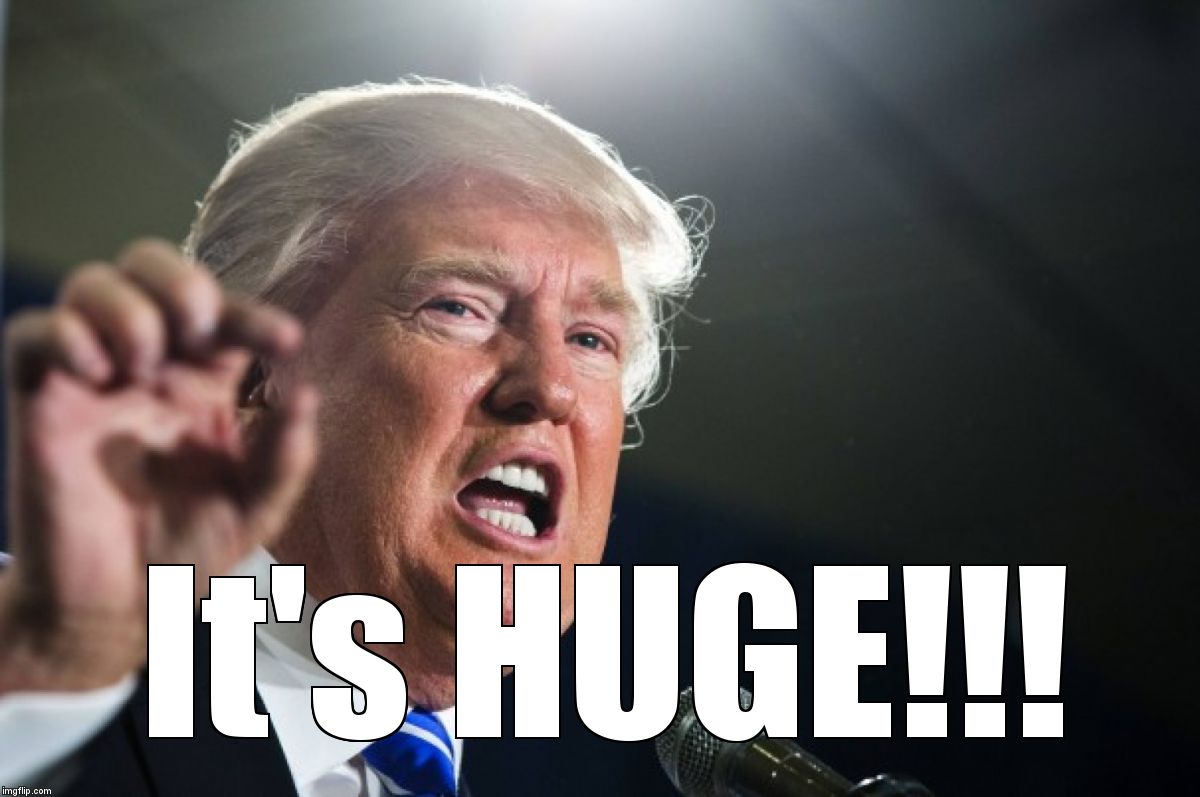
If you done game wiht PUN



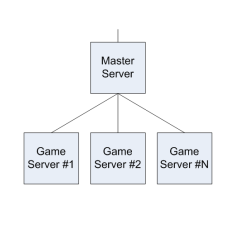
You need Loading blacne app to start

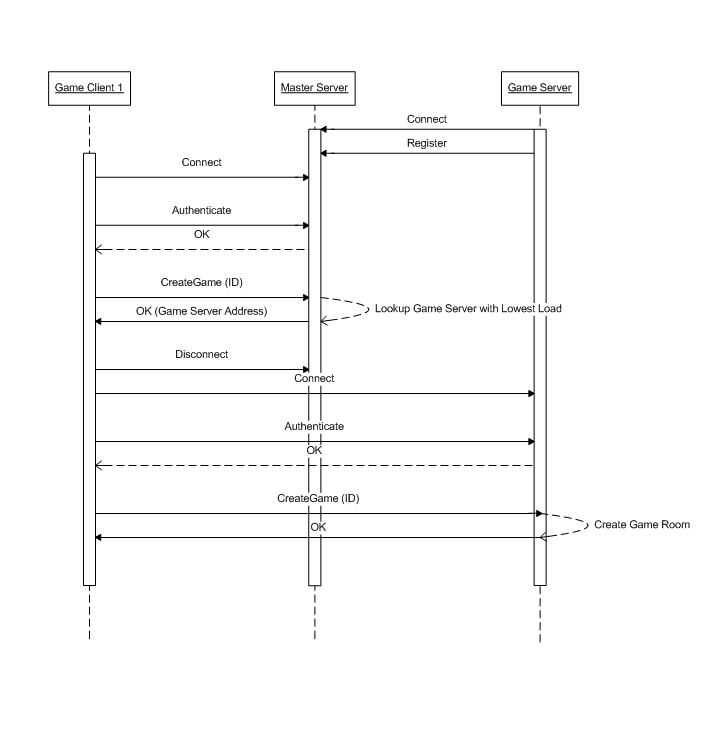


#### Load Balancing



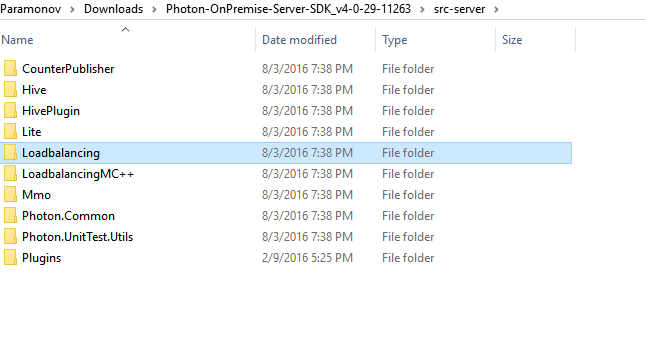
So in simple word it is 2 things master server plus n count of game servers



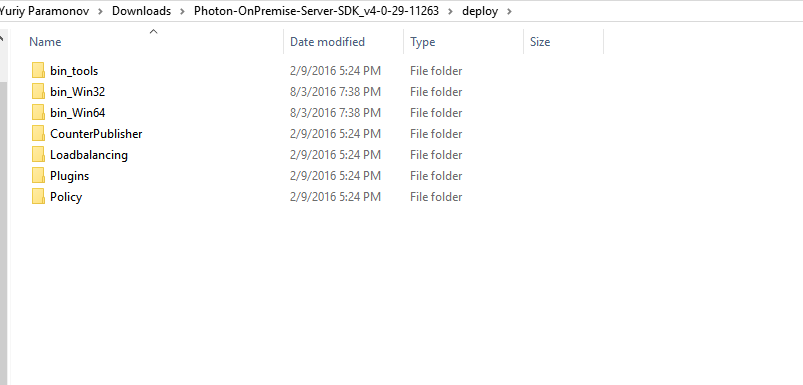


##### Sources and instances

If you need to change the sorces or just see them



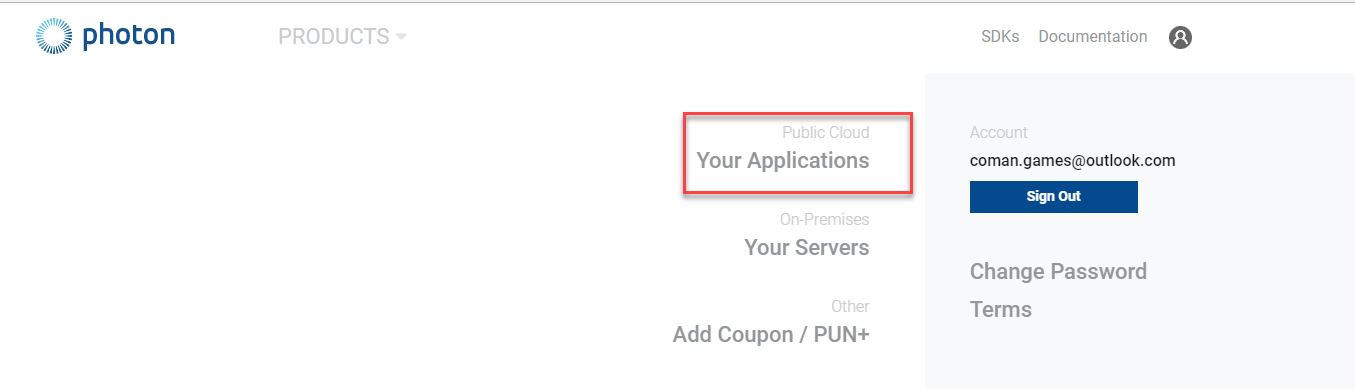
Or if you need to change the deploy

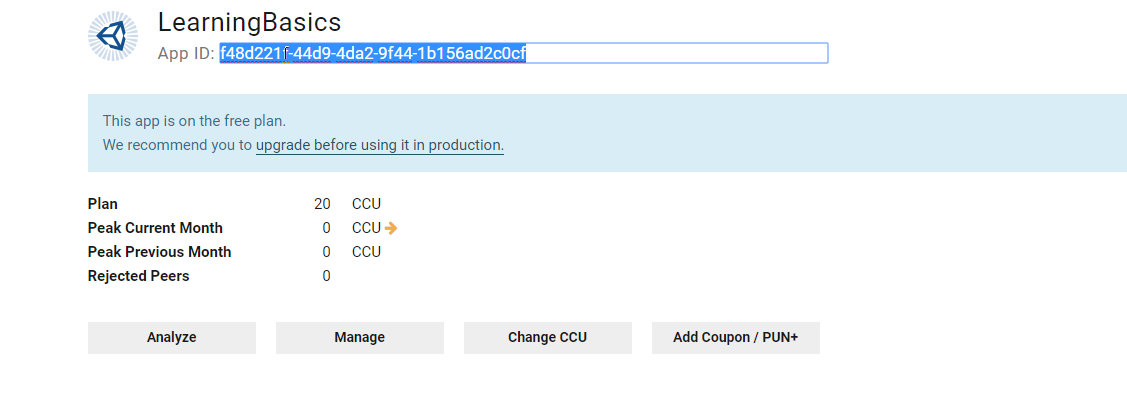


### Quick Start

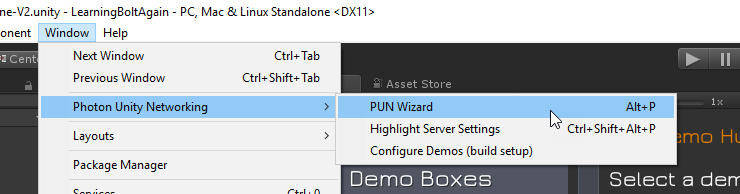
#### Step one

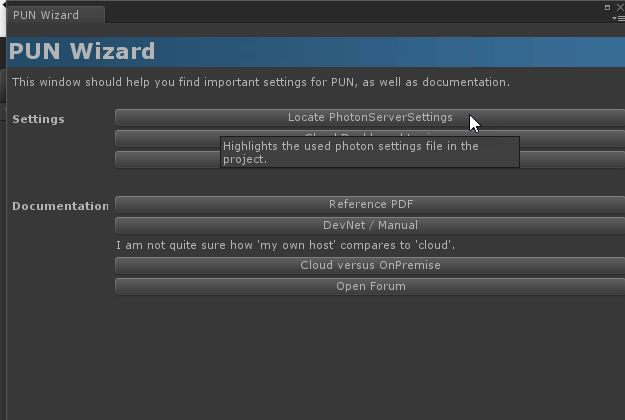
So step one is go to the photon page and open it and find your app ID



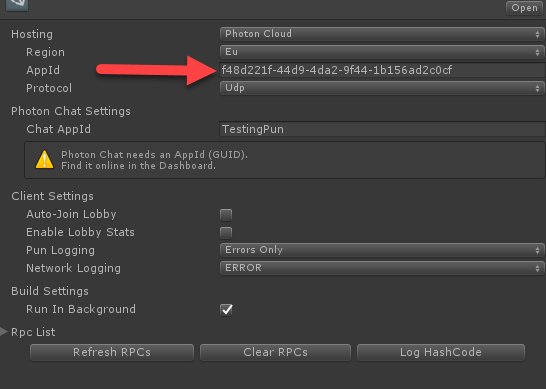


Then go and insert app ID in program settings





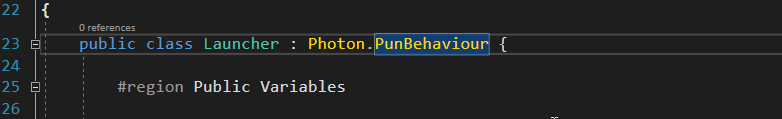
Then go to pun settings.

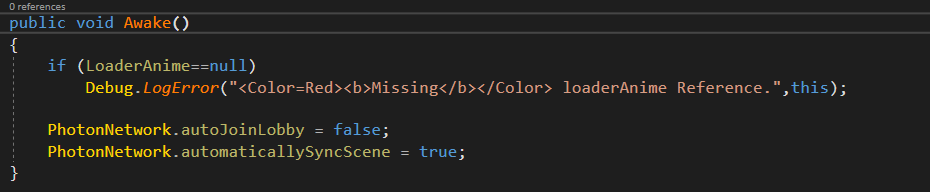


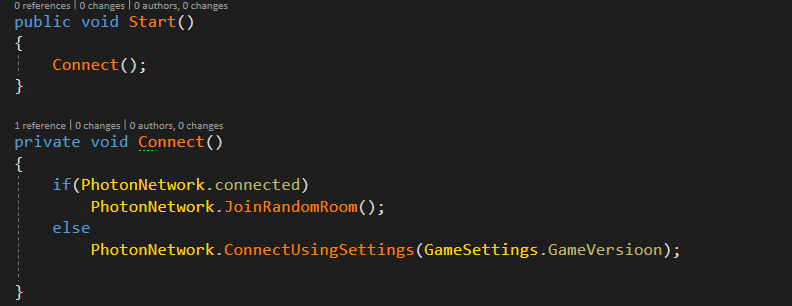
Insert them . And here you go .

#### Step two

The most important in the step 2 is launcher

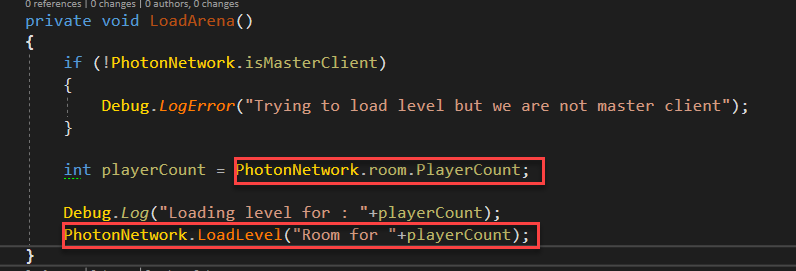




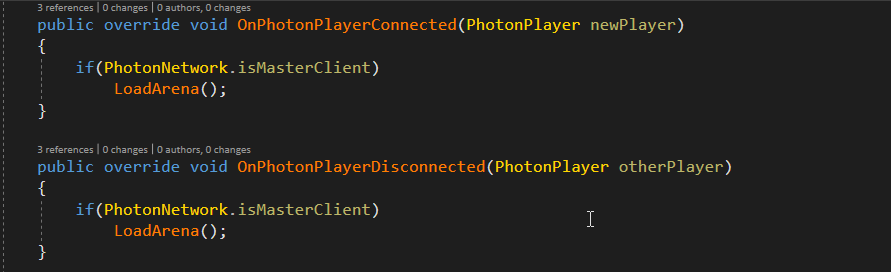


Only if we fail we can create something new

#### Step three

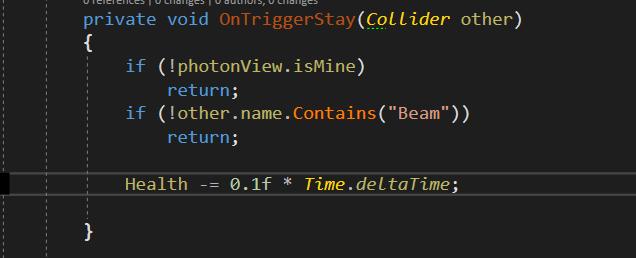


Loading level



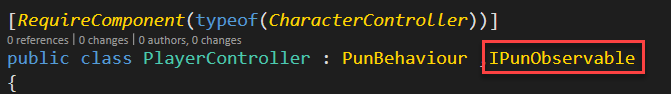
#### Step four

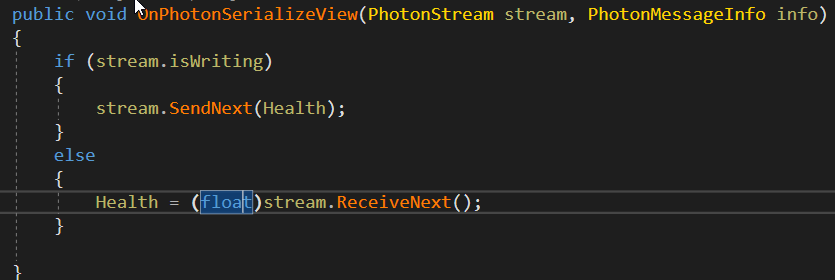
On damage checking is local player



#### Step five

Data synchronization





#### Step six

