Photon Unity Networking

v1.72

Generated by Doxygen 1.8.7

Wed Jun 22 2016 11:30:14

Contents

Chapter 1

Main Page

Introduction

Photon is a real-time multiplayer game development framework that is fast, lean and flexible. Photon consists of a server and several client SDKs for major platforms.

Photon Unity Network (PUN) is a special client framework which aims to re-implement and enhance the features of Unity's built-in networking. Under the hood, it uses Photon's features to communicate and match players.

As the PhotonNetwork API is very similar to Unity's built-in solution, users with prior networking experience in Unity should feel at home immediately. An automatic converter could assist you porting existing multiplayer projects to the Photon equivalent.

Full source code is available, so you can scale this plugin to support any type of multiplayer game you'd ever need.

This plugin is compatible with the hosted Photon Cloud service, which runs Photon Servers for you. A setup window registers you (for free) in less than a minute.

Most notable features:

- Dead-easy API
- Server available as hosted service (free for development) or as "On Premise"
- Load-balanced workflow scales across servers (with no extra effort)
- Outstanding performance of the Photon Server
- No direct P2P and no NAT punch-through needed
- Offline mode: re-use your multiplayer code in singleplayer game modes

First Steps

If you know how to use Unity's networking, you should feel right at home. You might want to run the converter (start in Wizard: ALT+P) on one of your projects and dive into the code and reference.

General Documentation (API)

Continue reading the General Documentation.

Marco Polo Tutorial

Alternatively, you can first do the Marco Polo Tutorial. And then just refer to the General Documentation.

Chapter 2

General Documentation

Brief overview of Photon, subscriptions, hosting options and how to start.

2.1 Photon

Unlike Unity's built-in networking, PUN always connects to a dedicated server which provides rooms, matchmaking and in-room communication for players. Behind the scenes Photon Unity Networking uses more than one server: Several "Game Servers" run the actual rooms (matches) while a "Master Server" keeps track of rooms and match players.

You have two options for the server side.

Exit Games Cloud

The Exit Games Cloud is a service which provides hosted and load balanced Photon Servers for you, fully managed by Exit Games. Free trials are available and `subscription costs for commercial use` are competitively low.

The service runs a fixed logic, so you can't implement your own server-side game logic. Instead, the clients need to be authoritative.

Clients are separated by "application id", which relates to your game title and a "game version". With that, your players won't clash with those of another developer or older game iterations.

Subscriptions bought in Asset Store

Follow these steps, if you bought a package with Photon Cloud Subscription in the Asset Store:

- Register a Photon Cloud Account: `exitgames.com/en/Account/SignUp`
- Create an App and get your AppID from the `Dashboard`
- Send a Mail to: `developer@exitgames.com`
- With:
  - Your Name and Company (if applicable)
  - Invoice/Purchase ID from the Asset Store
  - Photon Cloud AppID

Photon Server SDK

As alternative to the Photon Cloud service, you can run your own server and develop server side logic on top of our “Load Balancing” C# solution. This gives you full control of the server logic.

The Photon Server SDK can be downloaded on: www.exitgames.com/en/OnPremise/Download

Starting the Server: doc.exitgames.com/en/onpremise/current/getting-started/photon-server-in-5mi

Photon Unity Networking - First steps

When you import PUN, the "Wizard" window will popup. Either enter your email address to register for the cloud, skip this step to enter the AppId of an existing account or switch to "self hosted" Photon to enter your server's address.

This creates a configuration for either the cloud service or your own Photon server in the project: PhotonServer↩ Settings.

PUN consists of quite a few files, however there's only one that truly matters: **PhotonNetwork**. This class contains all functions and variables needed. If you ever have custom requirements, you can always modify the source files - this plugin is just an implementation of Photon after all.

To use PUN from UnityScript, move both folders "PhotonNetwork" and "UtilityScripts" to the Assets\ folder.

To show you how this API works, here are a few examples right away.

2.1.1 Master Server And Lobby

PUN always uses a master server and one or more game servers. The master server manages currently running games on the various game servers and will provide a game server address when you join or create a room. PUN (the client) automatically switches to that game server.

Individual matches are known as Rooms. They are independent of each other and identified by name. Rooms are grouped into one or multiple lobbies. Lobbies are an optional part in matchmaking. If you don't use custom lobbies explicitly, PUN will use a single lobby for all rooms.

By default, PUN will join the default lobby after connecting. This lobby sends a list of existing rooms to the client, so the player can pick a room (by name or some properties listed). Access the current list by using PhotonNetwork.↩ GetRoomList(). The lists is updated in intervals to keep traffic low.

Clients don't have to join a lobby to join or create rooms. If you don't want to show a list of rooms in your client, set PhotonNetwork.autoJoinLobby = false before you connect and your clients will skip the lobby.

You can use more than one lobby to organize room-lists as needed for your game. PhotonNetwork.JoinLobby is the method to join a specific lobby. You can make them up on the client side - the server will keep track of them. As long as name and type are the same, the TypedLobby will be the same for all clients, too.

A client is always just in one lobby and while being in a lobby, creating a room will relate to this lobby, too. Multiple lobbies mean the clients get shorter rooms lists, which is good. There is no limit to the rooms lists.

A parameter in JoinRoom, JoinRandomRoom and CreateRoom enables you to select a lobby without joining it.

Players won't notice each other in the Lobby and can't send data (to prevent issues when it's getting crowded).

The servers are all run on dedicated machines - there is no such thing as player-hosted ‘servers’. You don't have to bother remembering about the server organization though, as the API all hides this for you.

```
PhotonNetwork.ConnectUsingSettings("v1.0");
```

The code above is required to make use of any PhotonNetwork features. It sets your client's game version and uses the setup-wizard's config (stored in: PhotonServerSettings). The wizard can also be used when you host Photon yourself. Alternatively, use Connect() and you can ignore the PhotonServerSettings file.

Versioning

The loadbalancing logic for Photon uses your appID to separate your players from anyone else's. The same is done by game version, which separates players with a new client from those with older clients. As we can't guarantee that different Photon Unity Networking versions are compatible with each other, we add the PUN version to your game's version before sending it (since PUN v1.7).

Creating and Joining Games

Next, you'll want to join or create a room. The following code showcases some required functions:

```
//Join a room
PhotonNetwork.JoinRoom(roomName);

//Create this room.
PhotonNetwork.CreateRoom(roomName);
// Fails if it already exists and calls: OnPhotonCreateGameFailed

//Tries to join any random game:
PhotonNetwork.JoinRandomRoom();
//Fails if there are no matching games: OnPhotonRandomJoinFailed
```

A list of currently running games is provided by the master server's lobby. It can be joined like other rooms but only provides and updates the list of rooms. The PhotonNetwork plugin will automatically join the lobby after connecting. When you're joining a room, the list will no longer update.

To display the list of rooms (in a lobby):

```
foreach (RoomInfo game in PhotonNetwork.GetRoomList())
{
    GUILayout.Label(game.name + " " + game.playerCount + "/" + game.maxPlayers);
}
```

Alternatively, the game can use random matchmaking: It will try to join any room and fail if none has room for another player. In that case: Create a room without name and wait until other players join it randomly.

Advanced Matchmaking & Room Properties

Fully random matchmaking is not always something players enjoy. Sometimes you just want to play a certain map or just two versus two.

In Photon Cloud and Loadbalancing, you can set arbitrary room properties and filter for those in JoinRandom.

Room Properties and the Lobby

Room properties are synced to all players in the room and can be useful to keep track of the current map, round, starttime, etc. They are handled as Hashtable with string keys. Preferably short keys.

You can forward selected properties to the lobby, too. This makes them available for listing them and for random matchmaking, too. Not all room properties are interesting in the lobby, so you define the set of properties for the lobby on room creation.

```
Hashtable roomProps = new Hashtable() { { "map", 1 } };
string[] roomPropsInLobby = { "map", "ai" };

RoomOptions roomOptions = new RoomOptions() { customRoomProperties = roomProps,
      customRoomPropertiesForLobby = roomPropsInLobby }
CreateRoom(roomName, roomOptions, TypedLobby.Default)
```

Note that "ai" is not a key in the room-properties yet. It won't show up in the lobby until it's set in the game via Room.SetCustomProperties(). When you change the values for "map" or "ai", they will be updated in the lobby with a short delay, too.

Keep the list short to make sure performance doesn't suffer from loading the list.

Filtering Room Properties in Join Random

In JoinRandom, you could pass a Hashtable with expected room properties and max player value. These work as filters when the server selects a "fitting" room for you.

```
Hashtable expectedCustomRoomProperties = new Hashtable() { { "map", 1 } };
JoinRandomRoom(expectedCustomRoomProperties, 4);
```

If you pass more filter properties, chances are lower that a room matches them. Better limit the options.

Make sure you never filter for properties that are not known to the lobby (see above).

MonoBehaviour Callbacks

PUN uses several callbacks to let your game know about state changes like “connected” or “joined a game”. All you have to do is implement the fitting method in any MonoBehaviour and it gets called when the event happens.

To get a good overview of available callbacks, take a look at the class Photon.PunBehaviour. If you make your script a `PunBehaviour` (instead of a MonoBehaviour), you can override individual callbacks easily. If you begin to type "override", your coding IDE should provide you a list of callbacks, so they are easy to find while coding, too.

This covers the basics of setting up game rooms. Next up is actual communication in games.

Sending messages in rooms

Inside a room you are able to send network messages to other connected players. Furthermore you are able to send buffered messages that will also be sent to players that connect in the future (for spawning your player for instance).

Sending messages can be done using two methods. Either RPCs or by using the PhotonView property OnSerializePhotonView. There is more network interaction though. You can listen for callbacks for certain network events (e.g. OnPhotonInstantiate, OnPhotonPlayerConnected) and you can trigger some of these events (PhotonNetwork.Instantiate). Don’t worry if you’re confused by the last paragraph, next up we’ll explain for each of these subjects.

Using Groups in PUN

Groups are not synchronized when they are changed on any PhotonView. It’s up to the developer to keep photonviews in the same groups on all clients, if that’s needed. Using different group numbers for the same photonview on several clients will cause some inconsistent behaviour.

Some network messages are checked for their receiver group at the receiver side only, namely:

- RPCS that are targeted to a single player (or MasterClient)
- RPCS that are buffered (AllBuffered/OthersBuffered).
- This includes PhotonNetwork.Instantiate (as it is buffered).

Technical reason for this: the photon server only supports interestgroups for messages that are not cached and are not targetted at sepcific actor(s). This might change in the future.

PhotonView

PhotonView is a script component that is used to send messages (RPCs and OnSerializePhotonView). You need to attach the PhotonView to your games gameobjects. 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.

To add a PhotonView to a gameobject, simply select a gameobject and use: “Components/Miscellaneous/Photon View”.

Observe Transform

If you attach a Transform to a PhotonView’s Observe property, you can choose to sync Position, Rotation and Scale or a combination of those across the players. This can be a great help for prototyping or smaller games. Note: A change to any observed value will send out all observed values - not just the single value that’s changed. Also, updates are not smoothed or interpolated.

Observe MonoBehaviour

A PhotonView can be set to observe a MonoBehaviour. In this case, the script’s OnPhotonSerializeView method will be called. This method is called for writing an object’s state and for reading it, depending on whether the script is controlled by the local player.

The simple code below shows how to add character state synchronization with just a few lines of code more:

```
void OnPhotonSerializeView(PhotonStream stream,
      PhotonMessageInfo info)
{
   if (stream.isWriting)
   {
       //We own this player: send the others our data
       stream.SendNext((int)controllerScript._characterState);
       stream.SendNext(transform.position);
       stream.SendNext(transform.rotation);
   }
   else
   {
       //Network player, receive data
       controllerScript._characterState = (CharacterState)(int)stream.ReceiveNext();
       correctPlayerPos = (Vector3)stream.ReceiveNext();
       correctPlayerRot = (Quaternion)stream.ReceiveNext();
   }
}
```

If you send something “ReliableDeltaCompressed”, make sure to always write data to the stream in the same order. If you write no data to the PhotonStream, the update is not sent. This can be useful in pauses. Now on, to yet another way to communicate: RPCs.

2.1.2 Remote Procedure Calls

Remote Procedure Calls (RPCs) are exactly what the name implies: methods that can be called on remote clients in the same room. To enable remote calls for a method of a MonoBehaviour, you must apply the attribute: [PunRPC]. A PhotonView instance is needed on the same GameObject, to call the marked functions.

```
[PunRPC]
void ChatMessage(string a, string b)
{
    Debug.Log("ChatMessage " + a + " " + b);
}
```

To call the method from any script, you need access to a PhotonView object. If your script derives from Photon.↩ MonoBehaviour, it has a photonView field. Any regular MonoBehaviour or GameObject can use: PhotonView.↩ Get(this) to get access to its PhotonView component and then call RPCs on it.

```
PhotonView photonView = PhotonView.Get(this);
photonView.RPC("ChatMessage", PhotonTargets.All, "jup", "and jup!");
```

So, instead of directly calling the target method, you call RPC() on a PhotonView. Provide the name of the method to call, which players should call the method and then provide a list of parameters.

Careful: The parameters list used in RPC() has to match the number of expected parameters! If the receiving client can't find a matching method, it will log an error. There is one exception to this rule: The last parameter of a RPC method can be of type PhotonMessageInfo, which will provide some context for each call.

```
[PunRPC]
void ChatMessage(string a, string b, PhotonMessageInfo info)
{
    Debug.Log(String.Format("Info: {0} {1} {2}", info.sender, info.photonView, info.timestamp));
}
```

Timing for RPCs and Loading Levels

RPCs are called on specific PhotonViews and always target the matching one on the remote client. If the remote client does not know the fitting PhotonView, the RPC is lost.

A typical cause for lost RPCs is when clients load and set up levels. One client is faster or in the room for a longer time and sends important RPCs for objects that are not yet loaded on the other clients. The same happens when RPCs are buffered.

The solution is to pause the message queue, during scene loading. This code shows how how you can do it:

```
private IEnumerator MoveToGameScene()
{
    // Temporary disable processing of futher network messages
    PhotonNetwork.isMessageQueueRunning = false;
    Application.LoadLevel(levelName);
}
```

Alternatively you can use PhotonNetwork.LoadLevel. It temporarily disables the message queue as well.

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.

RPCs that belonged to the previously loaded scene but still arrived will now be discarded. But you should be able to define a break between both scenes by RPC.

Various topics

Differences to Unity Networking

1. Host model
    - Unity networking is server-client based (NOT P2P!). Servers are run via a Unity client (so via one of the players)
    - Photon is server-client based as well, but has a dedicated server; No more dropped connections due to hosts leaving.
2. Connectivity
    - Unity networking works with NAT punchthrough to try to improve connectivity: since players host the network servers, the connection often fails due to firewalls/routers etc. Connectivity can never be guaranteed, there is a low success rate.
    - Photon has a dedicated server, there is no need for NAT punchthrough or other concepts. Connectivity is a guaranteed 100%. If, in the rare case, a connection fails it must be due to a very strict client side network (a business VPN for example).
3. Performance
    - Photon beats Unity networking performance wise. We do not have the figures to prove this yet but the library has been optimized for years now. Furthermore, since the Unity servers are player hosted latency/ping is usually worse; you rely on the connection of the player acting as server. These connections are never any better then the connection of your dedicated Photon server.

4. Price
    - Like the Unity Networking solution, the Photon Unity Networking plugin is free as well. You can subscribe to use Photon Cloud hosting service for your game. Alternatively, you can rent your own servers and run Photon on them. The free license enables up to 100 concurrent players. Other licenses cost a one-time fee (as you do the hosting) and lift the concurrent user limits.
5. Features & maintenance
    - Unity does not seem to give much priority to their Networking implementation. There are rarely feature improvements and bugfixes are as seldom. The Photon solution is actively maintained and parts of it are available with source code. Furthermore, Photon already offers more features than Unity, such as the built-in load balancing and offline mode.
6. Master Server
    - The Master Server for Photon is a bit different from the Master Server for plain Unity Networking: In our case, it's a Photon Server that lists room-names of currently played games in so called "lobbies". Like Unity's Master, it will forward clients to the Game Server(s), where the actual gameplay is done.

2.1.3 Instantiating Networked Objects

In about every game you need to instantiate one or more player objects for every player. There are various options to do so which are listed below.

PhotonNetwork.Instantiate

PUN can automatically take care of spawning an object by passing a starting position, rotation and a prefab name to the PhotonNetwork.Instantiate method. Requirement: The prefab should be available directly under a Resources/ folder so that the prefab can be loaded at run time. Watch out with webplayers: Everything in the resources folder will be streamed at the very first scene per default. Under the webplayer settings you can specify the first level that uses assets from the Resources folder by using the "First streamed level". If you set this to your first game scene, your preloader and mainmenu will not be slowed down if they don't use the Resources folder assets.

```
void SpawnMyPlayerEverywhere()
{
    PhotonNetwork.Instantiate("MyPrefabName", new Vector3(0,0,0), Quaternion.identity, 0);
    //The last argument is an optional group number, feel free to ignore it for now.
}
```

Gain more control: Manually instantiate

If don't want to rely on the Resources folders to instantiate objects over the network you'll have to manually Instantiate objects as shown in the example at the end of this section.

The main reason for wanting to instantiate manually is gaining control over what is downloaded when for streaming webplayers. The details about streaming and the Resources folder in Unity can be found here.

If you spawn manually, you will have to assign a PhotonViewID yourself, these viewID's are the key to routing network messages to the correct gameobject/scripts. The player who wants to own and spawn a new object should allocate a new viewID using PhotonNetwork.AllocateViewID();. This PhotonViewID should then be send to all other players using a PhotonView that has already been set up (for example an existing scene PhotonView). You will have to keep in mind that this RPC needs to be buffered so that any clients that connect later will also receive the spawn instructions. Then the RPC message that is used to spawn the object will need a reference to your desired prefab and instantiate this using Unity's GameObject.Instantiate. Finally you will need to set setup the PhotonViews attached to this prefab by assigning all PhotonViews a PhotonViewID.

```
void SpawnMyPlayerEverywhere()
{
    //Manually allocate PhotonViewID
    PhotonViewID id1 = PhotonNetwork.AllocateViewID();
```

```
    photonView.RPC("SpawnOnNetwork", PhotonTargets.AllBuffered, transform.position
      ,
        transform.rotation, id1, PhotonNetwork.player);
}

public Transform playerPrefab; //set this in the inspector

[PunRPC]
void SpawnOnNetwork(Vector3 pos, Quaternion rot, PhotonViewID id1, PhotonPlayer np)
{
    Transform newPlayer = Instantiate(playerPrefab, pos, rot) as Transform;

    //Set the PhotonView
    PhotonView[] nViews = go.GetComponentsInChildren<PhotonView>();
    nViews[0].viewID = id1;
}
```

If you want to use asset bundles to load your network objects from, all you have to do is add your own assetbundle loading code and replace the “playerPrefab” from the example with the prefab from your asset bundle.

Offline mode

Offline mode is a feature to be able to re-use your multiplayer code in singleplayer game modes as well.

Mike Hergaarden: At M2H we had to rebuild our games several times as game portals usually require you to remove multiplayer functionality completely. Furthermore, being able to use the same code for single and multiplayer saves a lot of work on itself.

The most common features that you’ll want to be able to use in singleplayer are sending RPCs and using PhotonNetwork.Instantiate. The main goal of offline mode is to disable nullreferences and other errors when using PhotonNetwork functionality while not connected. You would still need to keep track of the fact that you’re running a singleplayer game, to set up the game etc. However, while running the game, all code should be reusable.

You need to manually enable offline mode, as PhotonNetwork needs to be able to distinguish erroneous from intended behaviour. Enabling this feature is very easy:

```
PhotonNetwork.offlineMode = true;
```

You can now reuse certain multiplayer methods without generating any connections and errors. Furthermore there is no noticeable overhead. Below follows a list of PhotonNetwork functions and variables and their results during offline mode:

PhotonNetwork.player The player ID is always -1 PhotonNetwork.playerName Works as expected. PhotonNetwork.playerList Contains only the local player PhotonNetwork.otherPlayers Always empty PhotonNetwork.time returns Time.time; PhotonNetwork.isMasterClient Always true PhotonNetwork.AllocateViewID() Works as expected. PhotonNetwork.Instantiate Works as expected PhotonNetwork.Destroy Works as expected. PhotonNetwork.RemoveRPCs/RemoveRPCsInGroup/SetReceivingEnabled/SetSendingEnabled/SetLevelPrefix While these make no sense in Singleplayer, they will not hurt either. PhotonView.RPC Works as expected.

Note that using other methods than the ones above can yield unexpected results and some will simply do nothing. E.g. PhotonNetwork.room will, obviously, return null. If you intend on starting a game in singleplayer, but move it to multiplayer at a later stage, you might want to consider hosting a 1 player game instead; this will preserve buffered RPCs and Instantiation calls, whereas offline mode Instantiations will not automatically carry over after Connecting.

Either set PhotonNetwork.offlineMode = false; or Simply call Connect() to stop offline mode.

Limitations

Views and players

For performance reasons, the PhotonNetwork API supports up to 1000 PhotonViews per player and a maximum of 2,147,483 players (note that this is WAY higher than your hardware can support!). You can easily allow for more PhotonViews per player, at the cost of maximum players. This works as follows: PhotonViews send out a viewID for every network message. This viewID is an integer and it is composed of the player ID and the player’s view ID.

The maximum size of an int is 2,147,483,647, divided by our MAX_VIEW_IDS(1000) that allows for over 2 million players, each having 1000 view IDs. As you can see, you can easily increase the player count by reducing the MAX_VIEW_IDS. The other way around, you can give all players more VIEW_IDS at the cost of less maximum players. It is important to note that most games will never need more than a few view ID's per player (one or two for the character..and that's usually it). If you need much more then you might be doing something wrong! It is extremely inefficient to assign a PhotonView and ID for every bullet that your weapon fires, instead keep track of your fire bullets via the player or weapon's PhotonView.

There is room for improving your bandwidth performance by reducing the int to a short (value range: 32,768 to 32,768). By setting MAX_VIEW_IDS to 32 you can then still support 1023 players Search for "//LIMITS NETWO↩ RKVIEWS&PLAYERS" for all occurrences of the int viewID. Furthermore, currently the API is not using uint/ushort but only the positive range of the numbers. This is done for simplicity and the usage of viewIDs is not a crucial performance issue for most situations.

Groups and Scoping

The PhotonNetwork plugin does not support network groups fully. See above: "Using Groups in PUN".

Unity's "scope" feature is not implemented.

Feedback

We are interested in your feedback, as this solution is an ongoing project for us. Let us know if something was too hidden, missing or not working. To let us know, post in our Forum: forum.exitgames.com

F.A.Q.

Can I use multiple PhotonViews per GameObject? Why?

Yes this is perfectly fine. You will need multiple PhotonViews if you need to observe 2 or more targets; You can only observe one per PhotonView. For your RPC's you'll only ever need one PhotonView and this can be the same PhotonView that is already observing something. RPC's never clash with an observed target.

Can I use UnityScript / Javascript?

To use PUN from UnityScript, move both folders "PhotonNetwork" and "UtilityScripts" to the Assets\ folder. Now PUN compiles before UnityScript and that makes it available from regular UnityScript code.

Converting your Unity networking project to Photon

Converting your Unity networking project to Photon can be done in one day. Just to be sure, make a backup of your project, as our automated converter will change your scripts. After this is done, run the converter from the Photon editor window (Window -> Photon Unity Networking -> Converter -> Start). The automatic conversion takes between 30 seconds to 10 minutes, depending on the size of your project and your computers performance. This automatic conversion takes care of the following:

- All NetworkViews are replaced with PhotonViews and the exact same settings. This is applied for all scenes and all prefabs.
- All scripts (JS/BOO/C#) are scanned for Network API calls, and they are replaced with PhotonNetwork calls.

There are some minor differences, therefore you will need to manually fix a few script conversion bugs. After conversion, you will most likely see some compile errors. You'll have to fix these first. Most common compile errors:

PhotonNetwork.RemoveRPCs(player); PhotonNetwork.DestroyPlayerObjects(player); These do not exist, and can be safely removed. Photon automatically cleans up players when they leave (even though you can disable this and take care of cleanup yourself if you want to) ..CloseConnection takes '2' arguments... Remove the second, boolean, argument from this call. PhotonNetwork.GetPing(player); GetPing does not take any arguments, you can only request the ping to the photon server, not ping to other players. myPlayerClass.transform.photonView.XX↩ X error You will need to convert code like this to: myPlayerClass.transform.GetComponent<PhotonView>().XXX Inside of scripts, you can use photonView to get the attached PhotonView component. However, you cannot call this on an external transform directly. RegisterServer There's no more need to register your games to a masterserver, Photon does this automatically.

You should be able to fix all compile errors in 5-30 minutes. Most errors will originate from main menu/GUI code, related to IPs/Ports/Lobby GUI.

This is where Photon differs most from Unity's solution:

There is only one Photon server and you connect using the room names. Therefore all references to IPs/ports can be removed from your code (usually GUI code). PhotonNetwork.JoinRoom(string room) only takes a room argument, you'll need to remove your old IP/port/NAT arguments. If you have been using the "Ultimate Unity networking project" by M2H, you should remove the MultiplayerFunctions class.

Lastly, all old MasterServer calls can be removed. You never need to register servers, and fetching the room list is as easy as calling PhotonNetwork.GetRoomList(). This list is always up to date (no need to fetch/poll etc). Rewriting the room listing can be most work, if your GUI needs to be redone, it might be simpler to write the GUI from scratch.

Chapter 3

Network Simulation GUI

Simple GUI element to control the built-in network condition simulation.

The Photon client library can simulate network conditions for lag (message delay) and loss, which can be a good tool for developer when testing with a local server or on near perfect network conditions.

To use it, add the component PhotonNetSimSettingsGui to an enabled GameObject in your scene. At runtime, the top left of the screen shows the current roundtrip time (RTT) and the controls for network simulation:

- RTT: The roundtrip time is the average of milliseconds until a message was acknowledged by the server. The variance value (behind the +/-) shows how stable the rtt is (a lower value being better).
- "Sim" toggle: Enables and disables the simulation. A sudden, big change of network conditions might result in disconnects.
- "Lag" slider: Adds a fixed delay to all outgoing and incoming messages. In milliseconds.
- "Jit" slider: Adds a random delay of "up to X milliseconds" per message.
- "Loss" slider: Drops the set percentage of messages. You can expect less than 2% drop in the internet today.

Chapter 4

Network Statistics GUI

The PhotonStatsGui is a simple GUI component to track and show network-metrics at runtime.

Usage

Just add the PhotonStatsGui component to any active GameObject in the hierarchy. A window appears (at runtime) and shows the message count.

A few toggles let you configure the window:

- buttons: Show buttons for "stats on", "reset stats" and "to log"
- traffic: Show lower level network traffic (bytes per direction)
- health: Show timing of sending, dispatches and their longest gaps

Message Statistics

The top most values showns are counter for "messages". Any operation, response and event are counted. Shown are the total count of outgoing, incoming and the sum of those messages as total and as average for the timespan that is tracked.

Traffic Statistics

These are the byte and packet counters. Anything that leaves or arrives via network is counted here. Even if there are few messages, they could be huge by accident and still cause less powerful clients to drop connection. You also see that there are packages sent when you don't send messages. They keeps the connection alive.

Health Statistics

The block beginning with "longest delta between" is about the performance of your client. We measure how much time passed between consecutive calls of send and dispatch. Usually they should be called ten times per second. If these values go beyond one second, you should check why Update() calls are delayed.

Button "Reset"

This resets the stats but keeps tracking them. This is useful to track message counts for different situations.

Button "To Log"

Pressing this simply logs the current stat values. This can be useful to have a overview how things evolved or just as reference.

Button "Stats On" (Enabling Traffic Stats)

The Photon library can track various network statistics but usually this feature is turned off. The PhotonStatsGui will enable the tracking and show those values.

The "stats on" toggle in the Gui controls if traffic stats are collected at all. The "Traffic Stats On" checkbox in the Inspector is the same value.

Chapter 5

Public API Module

The Public API module rounds up the most commonly used classes of PUN.

These classes are grouped into a "module" to make it easier to find the important stuff in PUN. Classes like Photon↩ Network and Photon.PunBehaviour are good entry points to learn how to code with PUN.

Opposed to that, there are several classes that are for internal use by the PUN framework. Even some of the internally used classes are public. This is for ease of use and in parts a result of how Unity works.

Open the Public API module

Chapter 6

Module Documentation

6.1 Public API

Groups the most important classes that you need to understand early on.

Classes

- interface IPunObservable

  Defines the OnPhotonSerializeView method to make it easy to implement correctly for observable scripts.
- interface IPunCallbacks

  This interface is used as definition of all callback methods of PUN, except OnPhotonSerializeView. Preferably, implement them individually.
- class Photon.PunBehaviour

  This class provides a .photonView and all callbacks/events that PUN can call. Override the events/methods you want to use.
- class PhotonMessageInfo

  Container class for info about a particular message, RPC or update.
- class PhotonStream

  This container is used in OnPhotonSerializeView() to either provide incoming data of a PhotonView or for you to provide it.
- class PhotonNetwork

  The main class to use the PhotonNetwork plugin. This class is static.
- class PhotonPlayer

  Summarizes a "player" within a room, identified (in that room) by actorID.
- class PhotonView

  PUN's NetworkView replacement class for networking. Use it like a NetworkView.
- class Room

  This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a RoomInfo and you can close or hide "your" room.
- class RoomInfo

  A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc).

Enumerations

- enum PhotonNetworkingMessage {
  PhotonNetworkingMessage.OnConnectedToPhoton, PhotonNetworkingMessage.OnLeftRoom, PhotonNetworkingMessage.OnMasterClientSwitched, PhotonNetworkingMessage.OnPhotonCreateRoomFailed, PhotonNetworkingMessage.OnPhotonJoinRoomFailed, PhotonNetworkingMessage.OnCreatedRoom,

PhotonNetworkingMessage.OnJoinedLobby, PhotonNetworkingMessage.OnLeftLobby, PhotonNetworkingMessage.OnDisconnectedFromPhoton, PhotonNetworkingMessage.OnConnectionFail, PhotonNetworkingMessage.OnFailedToConnectToPhoton, PhotonNetworkingMessage.OnReceivedRoomListUpdate, PhotonNetworkingMessage.OnJoinedRoom, PhotonNetworkingMessage.OnPhotonPlayerConnected, PhotonNetworkingMessage.OnPhotonPlayerDisconnected, PhotonNetworkingMessage.OnPhotonRandomJoinFailed, PhotonNetworkingMessage.OnConnectedToMaster, PhotonNetworkingMessage.OnPhotonSerializeView, PhotonNetworkingMessage.OnPhotonInstantiate, PhotonNetworkingMessage.OnPhotonMaxCccuReached, PhotonNetworkingMessage.OnPhotonCustomRoomPropertiesChanged, PhotonNetworkingMessage.OnPhotonPlayerPropertiesChanged, PhotonNetworkingMessage.OnUpdatedFriendList, PhotonNetworkingMessage.OnCustomAuthenticationFailed, PhotonNetworkingMessage.OnCustomAuthenticationResponse, PhotonNetworkingMessage.OnWebRpcResponse, PhotonNetworkingMessage.OnOwnershipRequest, PhotonNetworkingMessage.OnLobbyStatisticsUpdate }

This enum defines the set of MonoMessages Photon Unity Networking is using as callbacks. Implemented by PunBehaviour.

- enum PhotonLogLevel { PhotonLogLevel.ErrorsOnly, PhotonLogLevel.Informational, PhotonLogLevel.Full }

  Used to define the level of logging output created by the PUN classes. Either log errors, info (some more) or full.

- enum PhotonTargets { PhotonTargets.All, PhotonTargets.Others, PhotonTargets.MasterClient, PhotonTargets.AllBuffered, PhotonTargets.OthersBuffered, PhotonTargets.AllViaServer, PhotonTargets.AllBufferedViaServer }

  Enum of "target" options for RPCs. These define which remote clients get your RPC call.

- enum PeerState { PeerState.Uninitialized, PeerState.PeerCreated, PeerState.Queued, PeerState.Authenticated, PeerState.JoinedLobby, PeerState.DisconnectingFromMasterserver, PeerState.ConnectingToGameserver, PeerState.ConnectedToGameserver, PeerState.Joining, PeerState.Joined, PeerState.Leaving, PeerState.DisconnectingFromGameserver, PeerState.ConnectingToMasterserver, PeerState.QueuedComingFromGameserver, PeerState.Disconnecting, PeerState.Disconnected, PeerState.ConnectedToMaster, PeerState.ConnectingToNameServer, PeerState.ConnectedToNameServer, PeerState.DisconnectingFromNameServer, PeerState.Authenticating }

  Detailed connection / networking peer state. PUN implements a loadbalancing and authentication workflow "behind the scenes", so some states will automatically advance to some follow up state. Those states are commented with "(will-change)".

- enum DisconnectCause { DisconnectCause.ExceptionOnConnect = StatusCode.ExceptionOnConnect, DisconnectCause.SecurityExceptionOnConnect = StatusCode.SecurityExceptionOnConnect, DisconnectCause.TimeoutDisconnect = StatusCode.TimeoutDisconnect, DisconnectCause.DisconnectByClientTimeout = StatusCode.TimeoutDisconnect, DisconnectCause.InternalReceiveException = StatusCode.ExceptionOnReceive, DisconnectCause.DisconnectByServer = StatusCode.DisconnectByServer, DisconnectCause.DisconnectByServerTimeout = StatusCode.DisconnectByServer, DisconnectCause.DisconnectByServerLogic = StatusCode.DisconnectByServerLogic, DisconnectCause.DisconnectByServerUserLimit = StatusCode.DisconnectByServerUserLimit, DisconnectCause.Exception = StatusCode.Exception, DisconnectCause.InvalidRegion = ErrorCode.InvalidRegion, DisconnectCause.MaxCcuReached = ErrorCode.MaxCcuReached, DisconnectCause.InvalidAuthentication = ErrorCode.InvalidAuthentication, DisconnectCause.AuthenticationTicketExpired = 32753 }

  Summarizes the cause for a disconnect. Used in: OnConnectionFail and OnFailedToConnectToPhoton.

Functions

- void IPunObservable.OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

  Called by PUN several times per second, so that your script can write and read synchronization data for the PhotonView.

6.1.1 Detailed Description

Groups the most important classes that you need to understand early on.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum DisconnectCause

Summarizes the cause for a disconnect. Used in: OnConnectionFail and OnFailedToConnectToPhoton.

Extracted from the status codes from ExitGames.Client.Photon.StatusCode.

See also

PhotonNetworkingMessage

Enumerator

ExceptionOnConnect Connection could not be established. Possible cause: Local server not running.

SecurityExceptionOnConnect The security settings for client or server don't allow a connection (see remarks).A common cause for this is that browser clients read a "crossdomain" file from the server. If that file is unavailable or not configured to let the client connect, this exception is thrown. Photon usually provides this crossdomain file for Unity. If it fails, read: http://doc.exitgames.com/photon-server/PolicyApp

TimeoutDisconnect Connection timed out. Possible cause: Remote server not running or required ports blocked (due to router or firewall).

DisconnectByClientTimeout Timeout disconnect by client (which decided an ACK was missing for too long).

InternalReceiveException Exception in the receive-loop. Possible cause: Socket failure.

DisconnectByServer Server actively disconnected this client.

DisconnectByServerTimeout Timeout disconnect by server (which decided an ACK was missing for too long).

DisconnectByServerLogic Server actively disconnected this client. Possible cause: Server's send buffer full (too much data for client).

DisconnectByServerUserLimit Server actively disconnected this client. Possible cause: The server's user limit was hit and client was forced to disconnect (on connect).

Exception Some exception caused the connection to close.

InvalidRegion (32756) Authorization on the Photon Cloud failed because the app's subscription does not allow to use a particular region's server.

MaxCcuReached (32757) Authorization on the Photon Cloud failed because the concurrent users (CCU) limit of the app's subscription is reached.

InvalidAuthentication (32767) The Photon Cloud rejected the sent AppId. Check your Dashboard and make sure the AppId you use is complete and correct.

AuthenticationTicketExpired (32753) The Authentication ticket expired. Handle this by connecting again (which includes an authenticate to get a fresh ticket).

6.1.2.2 enum PeerState

Detailed connection / networking peer state. PUN implements a loadbalancing and authentication workflow "behind the scenes", so some states will automatically advance to some follow up state. Those states are commented with "(will-change)".

Enumerator

Uninitialized Not running. Only set before initialization and first use.

PeerCreated Created and available to connect.

Queued Not used at the moment.

Authenticated The application is authenticated. PUN usually joins the lobby now.(will-change) Unless Auto↩ JoinLobby is false.

JoinedLobby Client is in the lobby of the Master Server and gets room listings.Use Join, Create or Join↩ Random to get into a room to play.

DisconnectingFromMasterserver Disconnecting.(will-change)

ConnectingToGameserver Connecting to game server (to join/create a room and play).(will-change)

ConnectedToGameserver Similar to Connected state but on game server. Still in process to join/create room.(will-change)

Joining In process to join/create room (on game server).(will-change)

Joined Final state of a room join/create sequence. This client can now exchange events / call RPCs with other clients.

Leaving Leaving a room.(will-change)

DisconnectingFromGameserver Workflow is leaving the game server and will re-connect to the master server.(will-change)

ConnectingToMasterserver Workflow is connected to master server and will establish encryption and authenticate your app.(will-change)

QueuedComingFromGameserver Same Queued but coming from game server.(will-change)

Disconnecting PUN is disconnecting. This leads to Disconnected.(will-change)

Disconnected No connection is setup, ready to connect. Similar to PeerCreated.

ConnectedToMaster Final state for connecting to master without joining the lobby (AutoJoinLobby is false).

ConnectingToNameServer Client connects to the NameServer. This process includes low level connecting and setting up encryption. When done, state becomes ConnectedToNameServer.

ConnectedToNameServer Client is connected to the NameServer and established enctryption already. You should call OpGetRegions or ConnectToRegionMaster.

DisconnectingFromNameServer When disconnecting from a Photon NameServer.(will-change)

Authenticating When connecting to a Photon Server, this state is intermediate before you can call any operations.(will-change)

6.1.2.3 enum PhotonLogLevel

Used to define the level of logging output created by the PUN classes. Either log errors, info (some more) or full.

Enumerator

ErrorsOnly Show only errors. Minimal output. Note: Some might be "runtime errors" which you have to expect.

Informational Logs some of the workflow, calls and results.

Full Every available log call gets into the console/log. Only use for debugging.

6.1.2.4 enum PhotonNetworkingMessage

This enum defines the set of MonoMessages Photon Unity Networking is using as callbacks. Implemented by PunBehaviour.

Much like "Update()" in Unity, PUN will call methods in specific situations. Often, these methods are triggered when network operations complete (example: when joining a room).

All those methods are defined and described in this enum and implemented by PunBehaviour (which makes it easy to implement them as override).

Each entry is the name of such a method and the description tells you when it gets used by PUN.

Make sure to read the remarks per entry as some methods have optional parameters.

Enumerator

OnConnectedToPhoton Called when the initial connection got established but before you can use the server. OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready. This callback is only useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnFailedToConnectToPhoton() and OnDisconnectedFromPhoton().

OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.

When this is called, the low level connection is established and PUN will send your AppId, the user, etc in the background. This is not called for transitions from the masterserver to game servers.

Example: void OnConnectedToPhoton() { ... }

OnLeftRoom Called when the local user/client left a room. When leaving a room, PUN brings you back to the Master Server. Before you can use lobbies and join or create rooms, OnJoinedLobby() or OnConnectedToMaster() will get called again.

Example: void OnLeftRoom() { ... }

OnMasterClientSwitched Called after switching to a new MasterClient when the current one leaves. This is not called when this client enters a room. The former MasterClient is still in the player list when this method get called.

Example: void OnMasterClientSwitched(PhotonPlayer newMasterClient) { ... }

OnPhotonCreateRoomFailed Called when a CreateRoom() call failed. Optional parameters provide ErrorCode and message. Most likely because the room name is already in use (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Example: void OnPhotonCreateRoomFailed() { ... }

Example: void OnPhotonCreateRoomFailed(object[] codeAndMsg) { // codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg. }

OnPhotonJoinRoomFailed Called when a JoinRoom() call failed. Optional parameters provide ErrorCode and message. Most likely error is that the room does not exist or the room is full (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Example: void OnPhotonJoinRoomFailed() { ... }

Example: void OnPhotonJoinRoomFailed(object[] codeAndMsg) { // codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg. }

OnCreatedRoom Called when this client created a room and entered it. OnJoinedRoom() will be called as well. This callback is only called on the client which created a room (see PhotonNetwork.CreateRoom).

As any client might close (or drop connection) anytime, there is a chance that the creator of a room does not execute OnCreatedRoom.

If you need specific room properties or a "start signal", it is safer to implement OnMasterClientSwitched() and to make the new MasterClient check the room's state.

Example: void OnCreatedRoom() { ... }

OnJoinedLobby Called on entering a lobby on the Master Server. The actual room-list updates will call OnReceivedRoomListUpdate(). Note: When PhotonNetwork.autoJoinLobby is false, OnConnectedToMaster() will be called and the room list won't become available.

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify). The room list gets available when OnReceivedRoomListUpdate() gets called after OnJoinedLobby().

Example: void OnJoinedLobby() { ... }

OnLeftLobby Called after leaving a lobby. When you leave a lobby, CreateRoom and JoinRandomRoom automatically refer to the default lobby.

Example: void OnLeftLobby() { ... }

OnDisconnectedFromPhoton Called after disconnecting from the Photon server. In some cases, other callbacks are called before OnDisconnectedFromPhoton is called. Examples: OnConnectionFail() and OnFailedToConnectToPhoton().

Example: void OnDisconnectedFromPhoton() { ... }

OnConnectionFail Called when something causes the connection to fail (after it was established), followed by a call to OnDisconnectedFromPhoton(). If the server could not be reached in the first place, OnFailedToConnectToPhoton is called instead. The reason for the error is provided as StatusCode.

Example: void OnConnectionFail(DisconnectCause cause) { ... }

OnFailedToConnectToPhoton Called if a connect call to the Photon server failed before the connection was established, followed by a call to OnDisconnectedFromPhoton(). OnConnectionFail only gets called when a connection to a Photon server was established in the first place.

Example: void OnFailedToConnectToPhoton(DisconnectCause cause) { ... }

OnReceivedRoomListUpdate Called for any update of the room-listing while in a lobby (PhotonNetwork.insideLobby) on the Master Server. PUN provides the list of rooms by PhotonNetwork.GetRoomList(). Each item is a RoomInfo which might include custom properties (provided you defined those as lobby-listed when creating a room).

Not all types of lobbies provide a listing of rooms to the client. Some are silent and specialized for server-side matchmaking.

Example: void OnReceivedRoomListUpdate() { ... }

OnJoinedRoom Called when entering a room (by creating or joining it). Called on all clients (including the Master Client). This method is commonly used to instantiate player characters. If a match has to be started "actively", you can instead call an PunRPC triggered by a user's button-press or a timer.

When this is called, you can usually already access the existing players in the room via PhotonNetwork.playerList. Also, all custom properties should be already available as Room.customProperties. Check Room.playerCount to find out if enough players are in the room to start playing.

Example: void OnJoinedRoom() { ... }

OnPhotonPlayerConnected Called when a remote player entered the room. This PhotonPlayer is already added to the playerlist at this time. If your game starts with a certain number of players, this callback can be useful to check the Room.playerCount and find out if you can start.

Example: void OnPhotonPlayerConnected(PhotonPlayer newPlayer) { ... }

OnPhotonPlayerDisconnected Called when a remote player left the room. This PhotonPlayer is already removed from the playerlist at this time. When your client calls PhotonNetwork.leaveRoom, PUN will call this method on the remaining clients. When a remote client drops connection or gets closed, this callback gets executed. after a timeout of several seconds.

Example: void OnPhotonPlayerDisconnected(PhotonPlayer otherPlayer) { ... }

OnPhotonRandomJoinFailed Called after a JoinRandom() call failed. Optional parameters provide ErrorCode and message. Most likely all rooms are full or no rooms are available. When using multiple lobbies (via JoinLobby or TypedLobby), another lobby might have more/fitting rooms. PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Example: void OnPhotonRandomJoinFailed() { ... }

Example: void OnPhotonRandomJoinFailed(object[] codeAndMsg) { // codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg. }

OnConnectedToMaster Called after the connection to the master is established and authenticated but only when PhotonNetwork.autoJoinLobby is false. If you set PhotonNetwork.autoJoinLobby to true, OnJoinedLobby() will be called instead of this.

You can join rooms and create them even without being in a lobby. The default lobby is used in that case. The list of available rooms won't become available unless you join a lobby via PhotonNetwork.joinLobby.

Example: void OnConnectedToMaster() { ... }

OnPhotonSerializeView Implement to customize the data a PhotonView regularly synchronizes. Called every 'network-update' when observed by PhotonView. This method will be called in scripts that are assigned as Observed component of a PhotonView. PhotonNetwork.sendRateOnSerialize affects how often this method is called. PhotonNetwork.sendRate affects how often packages are sent by this client.

Implementing this method, you can customize which data a PhotonView regularly synchronizes. Your code defines what is being sent (content) and how your data is used by receiving clients.

Unlike other callbacks, *OnPhotonSerializeView only gets called when it is assigned to a PhotonView* as PhotonView.observed script.

To make use of this method, the PhotonStream is essential. It will be in "writing" mode" on the client that controls a PhotonView (PhotonStream.isWriting == true) and in "reading mode" on the remote clients that just receive that the controlling client sends.

If you skip writing any value into the stream, PUN will skip the update. Used carefully, this can conserve bandwidth and messages (which have a limit per room/second).

Note that OnPhotonSerializeView is not called on remote clients when the sender does not send any update. This can't be used as "x-times per second Update()".

Example: void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info) { ... }

OnPhotonInstantiate Called on all scripts on a GameObject (and children) that have been Instantiated using PhotonNetwork.Instantiate. PhotonMessageInfo parameter provides info about who created the object and when (based off PhotonNetworking.time).

Example: void OnPhotonInstantiate(PhotonMessageInfo info) { ... }

OnPhotonMaxCccuReached Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting. When this happens, the user might try again later. You can't create or join rooms in OnPhotonMaxCcuReached(), cause the client will be disconnecting. You can raise the CCU limits with a new license (when you host yourself) or extended subscription (when using the Photon Cloud). The Photon Cloud will mail you when the CCU limit was reached. This is also visible in the Dashboard (webpage).

Example: void OnPhotonMaxCccuReached() { ... }

OnPhotonCustomRoomPropertiesChanged Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.SetCustomProperties. Since v1.25 this method has one parameter: Hashtable propertiesThatChanged. Changing properties must be done by Room.SetCustomProperties, which causes this callback locally, too.

Example: void OnPhotonCustomRoomPropertiesChanged(Hashtable propertiesThatChanged) { ... }

OnPhotonPlayerPropertiesChanged Called when custom player-properties are changed. Player and the changed properties are passed as object[]. Since v1.25 this method has one parameter: object[] playerAndUpdatedProps, which contains two entries.
[0] is the affected PhotonPlayer.
[1] is the Hashtable of properties that changed.

We are using a object[] due to limitations of Unity's GameObject.SendMessage (which has only one optional parameter).

Changing properties must be done by PhotonPlayer.SetCustomProperties, which causes this callback locally, too.

Example:

```
void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) {
    PhotonPlayer player = playerAndUpdatedProps[0] as PhotonPlayer;
    Hashtable props = playerAndUpdatedProps[1] as Hashtable;
    //...
}
```

OnUpdatedFriendList Called when the server sent the response to a FindFriends request and updated PhotonNetwork.Friends. The friends list is available as PhotonNetwork.Friends, listing name, online state and the room a user is in (if any).

Example: void OnUpdatedFriendList() { ... }

OnCustomAuthenticationFailed Called when the custom authentication failed. Followed by disconnect! Custom Authentication can fail due to user-input, bad tokens/secrets. If authentication is successful, this method is not called. Implement OnJoinedLobby() or OnConnectedToMaster() (as usual).

During development of a game, it might also fail due to wrong configuration on the server side. In those cases, logging the debugMessage is very important.

Unless you setup a custom authentication service for your app (in the Dashboard), this won't be called!

Example: void OnCustomAuthenticationFailed(string debugMessage) { ... }

OnCustomAuthenticationResponse Called when your Custom Authentication service responds with additional data. Custom Authentication services can include some custom data in their response. When present, that data is made available in this callback as Dictionary. While the keys of your data have to be strings, the values can be either string or a number (in Json). You need to make extra sure, that the value type is the one you expect. Numbers become (currently) int64.

Example: void OnCustomAuthenticationResponse(Dictionary<string, object> data) { ... }

https://doc.photonengine.com/en/realtime/current/reference/custom-authentication

OnWebRpcResponse Called by PUN when the response to a WebRPC is available. See PhotonNetwork.↩ WebRPC. Important: The response.ReturnCode is 0 if Photon was able to reach your web-service. The content of the response is what your web-service sent. You can create a WebResponse instance from it. Example: WebRpcResponse webResponse = new WebRpcResponse(operationResponse);

Please note: Class OperationResponse is in a namespace which needs to be "used": using ExitGames.↩ Client.Photon; // includes OperationResponse (and other classes)

The OperationResponse.ReturnCode by Photon is: 0 for "OK" -3 for "Web-Service not configured" (see Dashboard / WebHooks) -5 for "Web-Service does now have RPC path/name" (at least for Azure)

Example: void OnWebRpcResponse(OperationResponse response) { ... }

OnOwnershipRequest Called when another player requests ownership of a PhotonView from you (the current owner). The parameter viewAndPlayer contains:

PhotonView view = viewAndPlayer[0] as PhotonView;

PhotonPlayer requestingPlayer = viewAndPlayer[1] as PhotonPlayer;

void OnOwnershipRequest(object[] viewAndPlayer) {} //

OnLobbyStatisticsUpdate Called when the Master Server sent an update for the Lobby Statistics, updating PhotonNetwork.LobbyStatistics. This callback has two preconditions: EnableLobbyStatistics must be set to true, before this client connects. And the client has to be connected to the Master Server, which is providing the info about lobbies.

6.1.2.5 enum PhotonTargets

Enum of "target" options for RPCs. These define which remote clients get your RPC call.

Enumerator

All Sends the RPC to everyone else and executes it immediately on this client. Player who join later will not execute this RPC.

Others Sends the RPC to everyone else. This client does not execute the RPC. Player who join later will not execute this RPC.

MasterClient Sends the RPC to MasterClient only. Careful: The MasterClient might disconnect before it executes the RPC and that might cause dropped RPCs.

AllBuffered Sends the RPC to everyone else and executes it immediately on this client. New players get the RPC when they join as it's buffered (until this client leaves).

OthersBuffered Sends the RPC to everyone. This client does not execute the RPC. New players get the RPC when they join as it's buffered (until this client leaves).

AllViaServer Sends the RPC to everyone (including this client) through the server.This client executes the RPC like any other when it received it from the server. Benefit: The server's order of sending the RPCs is the same on all clients.

AllBufferedViaServer Sends the RPC to everyone (including this client) through the server and buffers it for players joining later.This client executes the RPC like any other when it received it from the server. Benefit: The server's order of sending the RPCs is the same on all clients.

6.1.3 Function Documentation

6.1.3.1 void IPunObservable.OnPhotonSerializeView (PhotonStream *stream,* PhotonMessageInfo *info*)

Called by PUN several times per second, so that your script can write and read synchronization data for the Photon↩ View.

This method will be called in scripts that are assigned as Observed component of a PhotonView.
PhotonNetwork.sendRateOnSerialize affects how often this method is called.
PhotonNetwork.sendRate affects how often packages are sent by this client.

Implementing this method, you can customize which data a PhotonView regularly synchronizes. Your code defines what is being sent (content) and how your data is used by receiving clients.

Unlike other callbacks, *OnPhotonSerializeView only gets called when it is assigned to a PhotonView* as PhotonView.observed script.

To make use of this method, the PhotonStream is essential. It will be in "writing" mode" on the client that controls a PhotonView (PhotonStream.isWriting == true) and in "reading mode" on the remote clients that just receive that the controlling client sends.

If you skip writing any value into the stream, PUN will skip the update. Used carefully, this can conserve bandwidth and messages (which have a limit per room/second).

Note that OnPhotonSerializeView is not called on remote clients when the sender does not send any update. This can't be used as "x-times per second Update()".

Implemented in PhotonTransformView.

6.2 Optional Gui Elements

Useful GUI elements for PUN.

Classes

- class PhotonLagSimulationGui

  This MonoBehaviour is a basic GUI for the Photon client's network-simulation feature. It can modify lag (fixed delay), jitter (random lag) and packet loss.
- class PhotonStatsGui

  Basic GUI to show traffic and health statistics of the connection to Photon, toggled by shift+tab.

6.2.1 Detailed Description

Useful GUI elements for PUN.

Chapter 7

Namespace Documentation

7.1 Package ExitGames

Namespaces

- package Client

7.2 Package ExitGames.Client

Namespaces

- package GUI

7.3 Package ExitGames.Client.GUI

Classes

- class GizmoTypeDrawer

Enumerations

- enum GizmoType { GizmoType.WireSphere, GizmoType.Sphere, GizmoType.WireCube, GizmoType.Cube }

7.3.1 Enumeration Type Documentation

7.3.1.1 enum ExitGames.Client.GUI.GizmoType

Enumerator

WireSphere

Sphere

WireCube

Cube

7.4 Package Photon

Classes

- class MonoBehaviour

  This class adds the property photonView, while logging a warning when your game still uses the networkView.
- class PunBehaviour

  This class provides a .photonView and all callbacks/events that PUN can call. Override the events/methods you want to use.

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable

7.4.1 Typedef Documentation

7.4.1.1 using Photon.Hashtable = typedef ExitGames.Client.Photon.Hashtable

7.5 Package UnityEngine

Namespaces

- package SceneManagement

7.6 Package UnityEngine.SceneManagement

Classes

- class SceneManager

  Minimal implementation of the SceneManager for older Unity, up to v5.2.

Chapter 8

Class Documentation

8.1 ActorProperties Class Reference

Class for constants. These (byte) values define "well known" properties for an Actor / Player. Pun uses these constants internally.

Public Attributes

- const byte PlayerName = 255

  (255) Name of a player/actor.
- const byte IsInactive = 254

  (254) Tells you if the player is currently in this game (getting events live).
- const byte UserId = 253

  (253) UserId of the player. Sent when room gets created with RoomOptions.publishUserId = true.

8.1.1 Detailed Description

Class for constants. These (byte) values define "well known" properties for an Actor / Player. Pun uses these constants internally.

"Custom properties" have to use a string-type as key. They can be assigned at will.

8.1.2 Member Data Documentation

8.1.2.1 const byte ActorProperties.IsInactive = 254

(254) Tells you if the player is currently in this game (getting events live).

A server-set value for async games, where players can leave the game and return later.

8.1.2.2 const byte ActorProperties.PlayerName = 255

(255) Name of a player/actor.

8.1.2.3 const byte ActorProperties.UserId = 253

(253) UserId of the player. Sent when room gets created with RoomOptions.publishUserId = true.

8.2 AuthenticationValues Class Reference

Container for user authentication in Photon. Set AuthValues before you connect - all else is handled.

Public Member Functions

- AuthenticationValues ()

  Creates empty auth values without any info.
- AuthenticationValues (string userId)

  Creates minimal info about the user. If this is authenticated or not, depends on the set AuthType.
- virtual void SetAuthPostData (string stringData)

  Sets the data to be passed-on to the auth service via POST.
- virtual void SetAuthPostData (byte[] byteData)

  Sets the data to be passed-on to the auth service via POST.
- virtual void AddAuthParameter (string key, string value)

  Adds a key-value pair to the get-parameters used for Custom Auth.
- override string ToString ()

Properties

- CustomAuthenticationType AuthType `[get, set]`

  The type of custom authentication provider that should be used. Currently only "Custom" or "None" (turns this off).
- string AuthGetParameters `[get, set]`

  This string must contain any (http get) parameters expected by the used authentication service. By default, username and token.
- object AuthPostData `[get, set]`

  Data to be passed-on to the auth service via POST. Default: null (not sent). Either string or byte[] (see setters).
- string Token `[get, set]`

  After initial authentication, Photon provides a token for this client / user, which is subsequently used as (cached) validation.
- string UserId `[get, set]`

  The UserId should be a unique identifier per user. This is for finding friends, etc..

8.2.1 Detailed Description

Container for user authentication in Photon. Set AuthValues before you connect - all else is handled.

On Photon, user authentication is optional but can be useful in many cases. If you want to FindFriends, a unique ID per user is very practical.

There are basically three options for user authentification: None at all, the client sets some UserId or you can use some account web-service to authenticate a user (and set the UserId server-side).

Custom Authentication lets you verify end-users by some kind of login or token. It sends those values to Photon which will verify them before granting access or disconnecting the client.

The Photon Cloud Dashboard will let you enable this feature and set important server values for it. `https://www.photonengine.com/dashboard`

8.2.2 Constructor & Destructor Documentation

8.2.2.1 AuthenticationValues.AuthenticationValues ( )

Creates empty auth values without any info.

8.2.2.2 AuthenticationValues.AuthenticationValues (string *userId*)

Creates minimal info about the user. If this is authenticated or not, depends on the set AuthType.

Parameters

userId	Some UserId to set in Photon.

8.2.3 Member Function Documentation

8.2.3.1 virtual void AuthenticationValues.AddAuthParameter (string *key*, string *value*) [virtual]

Adds a key-value pair to the get-parameters used for Custom Auth.

This method does uri-encoding for you.

Parameters

key	Key for the value to set.
value	Some value relevant for Custom Authentication.

8.2.3.2 virtual void AuthenticationValues.SetAuthPostData (string *stringData*) [virtual]

Sets the data to be passed-on to the auth service via POST.

Parameters

stringData	String data to be used in the body of the POST request. Null or empty string will set AuthPostData to null.

8.2.3.3 virtual void AuthenticationValues.SetAuthPostData (byte[] *byteData*) [virtual]

Sets the data to be passed-on to the auth service via POST.

Parameters

byteData	Binary token / auth-data to pass on.

8.2.3.4 override string AuthenticationValues.ToString ( )

8.2.4 Property Documentation

8.2.4.1 string AuthenticationValues.AuthGetParameters [get], [set]

This string must contain any (http get) parameters expected by the used authentication service. By default, username and token.

Standard http get parameters are used here and passed on to the service that's defined in the server (Photon Cloud Dashboard).

8.2.4.2 object AuthenticationValues.AuthPostData [get], [set]

Data to be passed-on to the auth service via POST. Default: null (not sent). Either string or byte[] (see setters).

8.2.4.3 CustomAuthenticationType AuthenticationValues.AuthType [get], [set]

The type of custom authentication provider that should be used. Currently only "Custom" or "None" (turns this off).

8.2.4.4 string AuthenticationValues.Token [get], [set]

After initial authentication, Photon provides a token for this client / user, which is subsequently used as (cached) validation.

8.2.4.5 string AuthenticationValues.UserId [get], [set]

The UserId should be a unique identifier per user. This is for finding friends, etc..

8.3 ErrorCode Class Reference

ErrorCode defines the default codes associated with Photon client/server communication.

Public Attributes

- const int Ok = 0

  (0) is always "OK", anything else an error or specific situation.
- const int OperationNotAllowedInCurrentState = -3

  (-3) Operation can't be executed yet (e.g. OpJoin can't be called before being authenticated, RaiseEvent cant be used before getting into a room).
- const int InvalidOperationCode = -2

  (-2) The operation you called is not implemented on the server (application) you connect to. Make sure you run the fitting applications.
- const int InvalidOperation = -2

  (-2) The operation you called could not be executed on the server.
- const int InternalServerError = -1

  (-1) Something went wrong in the server. Try to reproduce and contact Exit Games.
- const int InvalidAuthentication = 0x7FFF

  (32767) Authentication failed. Possible cause: AppId is unknown to Photon (in cloud service).
- const int GameIdAlreadyExists = 0x7FFF - 1

  (32766) GameId (name) already in use (can't create another). Change name.
- const int GameFull = 0x7FFF - 2

  (32765) Game is full. This rarely happens when some player joined the room before your join completed.
- const int GameClosed = 0x7FFF - 3

  (32764) Game is closed and can't be joined. Join another game.
- const int AlreadyMatched = 0x7FFF - 4
- const int ServerFull = 0x7FFF - 5

  (32762) Not in use currently.
- const int UserBlocked = 0x7FFF - 6

  (32761) Not in use currently.
- const int NoRandomMatchFound = 0x7FFF - 7

  (32760) Random matchmaking only succeeds if a room exists thats neither closed nor full. Repeat in a few seconds or create a new room.
- const int GameDoesNotExist = 0x7FFF - 9

  (32758) Join can fail if the room (name) is not existing (anymore). This can happen when players leave while you join.
- const int MaxCcuReached = 0x7FFF - 10

  (32757) Authorization on the Photon Cloud failed becaus the concurrent users (CCU) limit of the app's subscription is reached.
- const int InvalidRegion = 0x7FFF - 11

  (32756) Authorization on the Photon Cloud failed because the app's subscription does not allow to use a particular region's server.

- const int CustomAuthenticationFailed = 0x7FFF - 12

  (32755) Custom Authentication of the user failed due to setup reasons (see Cloud Dashboard) or the provided user data (like username or token). Check error message for details.
- const int AuthenticationTicketExpired = 0x7FF1

  (32753) The Authentication ticket expired. Usually, this is refreshed behind the scenes. Connect (and authorize) again.
- const int PluginReportedError = 0x7FFF - 15

  (32752) A server-side plugin (or webhook) failed to execute and reported an error. Check the OperationResponse.↩ DebugMessage.
- const int PluginMismatch = 0x7FFF - 16

  (32751) CreateGame/JoinGame/Join operation fails if expected plugin does not correspond to loaded one.
- const int JoinFailedPeerAlreadyJoined = 32750

  (32750) for join requests. Indicates the current peer already called join and is joined to the room.
- const int JoinFailedFoundInactiveJoiner = 32749

  (32749) for join requests. Indicates the list of InactiveActors already contains an actor with the requested ActorNr or UserId.
- const int JoinFailedWithRejoinerNotFound = 32748

  (32748) for join requests. Indicates the list of Actors (active and inactive) did not contain an actor with the requested ActorNr or UserId.
- const int JoinFailedFoundExcludedUserId = 32747

  (32747) for join requests. Note: for future use - Indicates the requested UserId was found in the ExcludedList.
- const int JoinFailedFoundActiveJoiner = 32746

  (32746) for join requests. Indicates the list of ActiveActors already contains an actor with the requested ActorNr or UserId.
- const int HttpLimitReached = 32745

  (32745) for SetProerties and Raisevent (if flag HttpForward is true) requests. Indicates the maximum allowd http requests per minute was reached.
- const int ExternalHttpCallFailed = 32744

  (32744) for WebRpc requests. Indicates the the call to the external service failed.
- const int SlotError = 32742

  (32742) Server error during matchmaking with slot reservation. E.g. the reserved slots can not exceed MaxPlayers.

8.3.1 Detailed Description

ErrorCode defines the default codes associated with Photon client/server communication.

8.3.2 Member Data Documentation

8.3.2.1 const int ErrorCode.AlreadyMatched = 0x7FFF - 4

8.3.2.2 const int ErrorCode.AuthenticationTicketExpired = 0x7FF1

(32753) The Authentication ticket expired. Usually, this is refreshed behind the scenes. Connect (and authorize) again.

8.3.2.3 const int ErrorCode.CustomAuthenticationFailed = 0x7FFF - 12

(32755) Custom Authentication of the user failed due to setup reasons (see Cloud Dashboard) or the provided user data (like username or token). Check error message for details.

8.3.2.4 const int ErrorCode.ExternalHttpCallFailed = 32744

(32744) for WebRpc requests. Indicates the the call to the external service failed.

8.3.2.5 const int ErrorCode.GameClosed = 0x7FFF - 3

(32764) Game is closed and can't be joined. Join another game.

8.3.2.6 const int ErrorCode.GameDoesNotExist = 0x7FFF - 9

(32758) Join can fail if the room (name) is not existing (anymore). This can happen when players leave while you join.

8.3.2.7 const int ErrorCode.GameFull = 0x7FFF - 2

(32765) Game is full. This rarely happens when some player joined the room before your join completed.

8.3.2.8 const int ErrorCode.GameIdAlreadyExists = 0x7FFF - 1

(32766) GameId (name) already in use (can't create another). Change name.

8.3.2.9 const int ErrorCode.HttpLimitReached = 32745

(32745) for SetProerties and Raisevent (if flag HttpForward is true) requests. Indicates the maximum allowd http requests per minute was reached.

8.3.2.10 const int ErrorCode.InternalServerError = -1

(-1) Something went wrong in the server. Try to reproduce and contact Exit Games.

8.3.2.11 const int ErrorCode.InvalidAuthentication = 0x7FFF

(32767) Authentication failed. Possible cause: AppId is unknown to Photon (in cloud service).

8.3.2.12 const int ErrorCode.InvalidOperation = -2

(-2) The operation you called could not be executed on the server.

Make sure you are connected to the server you expect.

This code is used in several cases: The arguments/parameters of the operation might be out of range, missing entirely or conflicting. The operation you called is not implemented on the server (application). Server-side plugins affect the available operations.

8.3.2.13 const int ErrorCode.InvalidOperationCode = -2

(-2) The operation you called is not implemented on the server (application) you connect to. Make sure you run the fitting applications.

8.3.2.14 const int ErrorCode.InvalidRegion = 0x7FFF - 11

(32756) Authorization on the Photon Cloud failed because the app's subscription does not allow to use a particular region's server.

Some subscription plans for the Photon Cloud are region-bound. Servers of other regions can't be used then. Check your master server address and compare it with your Photon Cloud Dashboard's info. https://www.photonengine.com/dashboard

OpAuthorize is part of connection workflow but only on the Photon Cloud, this error can happen. Self-hosted Photon servers with a CCU limited license won't let a client connect at all.

8.3.2.15 const int ErrorCode.JoinFailedFoundActiveJoiner = 32746

(32746) for join requests. Indicates the list of ActiveActors already contains an actor with the requested ActorNr or UserId.

8.3.2.16 const int ErrorCode.JoinFailedFoundExcludedUserId = 32747

(32747) for join requests. Note: for future use - Indicates the requested UserId was found in the ExcludedList.

8.3.2.17 const int ErrorCode.JoinFailedFoundInactiveJoiner = 32749

(32749) for join requests. Indicates the list of InactiveActors already contains an actor with the requested ActorNr or UserId.

8.3.2.18 const int ErrorCode.JoinFailedPeerAlreadyJoined = 32750

(32750) for join requests. Indicates the current peer already called join and is joined to the room.

8.3.2.19 const int ErrorCode.JoinFailedWithRejoinerNotFound = 32748

(32748) for join requests. Indicates the list of Actors (active and inactive) did not contain an actor with the requested ActorNr or UserId.

8.3.2.20 const int ErrorCode.MaxCcuReached = 0x7FFF - 10

(32757) Authorization on the Photon Cloud failed becaus the concurrent users (CCU) limit of the app's subscription is reached.

Unless you have a plan with "CCU Burst", clients might fail the authentication step during connect. Affected client are unable to call operations. Please note that players who end a game and return to the master server will disconnect and re-connect, which means that they just played and are rejected in the next minute / re-connect. This is a temporary measure. Once the CCU is below the limit, players will be able to connect an play again.

OpAuthorize is part of connection workflow but only on the Photon Cloud, this error can happen. Self-hosted Photon servers with a CCU limited license won't let a client connect at all.

8.3.2.21 const int ErrorCode.NoRandomMatchFound = 0x7FFF - 7

(32760) Random matchmaking only succeeds if a room exists thats neither closed nor full. Repeat in a few seconds or create a new room.

8.3.2.22 const int ErrorCode.Ok = 0

(0) is always "OK", anything else an error or specific situation.

8.3.2.23 const int ErrorCode.OperationNotAllowedInCurrentState = -3

(-3) Operation can't be executed yet (e.g. OpJoin can't be called before being authenticated, RaiseEvent cant be used before getting into a room).

Before you call any operations on the Cloud servers, the automated client workflow must complete its authorization. In PUN, wait until State is: JoinedLobby (with AutoJoinLobby = true) or ConnectedToMaster (AutoJoinLobby = false)

8.3.2.24 const int ErrorCode.PluginMismatch = 0x7FFF - 16

(32751) CreateGame/JoinGame/Join operation fails if expected plugin does not correspond to loaded one.

8.3.2.25 const int ErrorCode.PluginReportedError = 0x7FFF - 15

(32752) A server-side plugin (or webhook) failed to execute and reported an error. Check the OperationResponse.↩ DebugMessage.

8.3.2.26 const int ErrorCode.ServerFull = 0x7FFF - 5

(32762) Not in use currently.

8.3.2.27 const int ErrorCode.SlotError = 32742

(32742) Server error during matchmaking with slot reservation. E.g. the reserved slots can not exceed MaxPlayers.

8.3.2.28 const int ErrorCode.UserBlocked = 0x7FFF - 6

(32761) Not in use currently.

8.4 EventCode Class Reference

Class for constants. These values are for events defined by Photon Loadbalancing. Pun uses these constants internally.

Public Attributes

- const byte GameList = 230

  (230) Initial list of RoomInfos (in lobby on Master)
- const byte GameListUpdate = 229

  (229) Update of RoomInfos to be merged into "initial" list (in lobby on Master)
- const byte QueueState = 228

  (228) Currently not used. State of queueing in case of server-full
- const byte Match = 227

  (227) Currently not used. Event for matchmaking
- const byte AppStats = 226

  (226) Event with stats about this application (players, rooms, etc)
- const byte LobbyStats = 224

  (224) This event provides a list of lobbies with their player and game counts.
- const byte AzureNodeInfo = 210

  (210) Internally used in case of hosting by Azure
- const byte Join = (byte)255

  (255) Event Join: someone joined the game. The new actorNumber is provided as well as the properties of that actor (if set in OpJoin).

- const byte Leave = (byte)254

  (254) Event Leave: The player who left the game can be identified by the actorNumber.
- const byte PropertiesChanged = (byte)253

  (253) When you call OpSetProperties with the broadcast option "on", this event is fired. It contains the properties being set.
- const byte SetProperties = (byte)253

  (253) When you call OpSetProperties with the broadcast option "on", this event is fired. It contains the properties being set.
- const byte ErrorInfo = 251

  (252) When player left game unexpected and the room has a playerTtl $>$ 0, this event is fired to let everyone know about the timeout.
- const byte CacheSliceChanged = 250

  (250) Sent by Photon whent he event cache slice was changed. Done by OpRaiseEvent.

8.4.1 Detailed Description

Class for constants. These values are for events defined by Photon Loadbalancing. Pun uses these constants internally.

They start at 255 and go DOWN. Your own in-game events can start at 0.

8.4.2 Member Data Documentation

8.4.2.1 const byte EventCode.AppStats = 226

(226) Event with stats about this application (players, rooms, etc)

8.4.2.2 const byte EventCode.AzureNodeInfo = 210

(210) Internally used in case of hosting by Azure

8.4.2.3 const byte EventCode.CacheSliceChanged = 250

(250) Sent by Photon whent he event cache slice was changed. Done by OpRaiseEvent.

8.4.2.4 const byte EventCode.ErrorInfo = 251

(252) When player left game unexpected and the room has a playerTtl $>$ 0, this event is fired to let everyone know about the timeout.

Obsolete. Replaced by Leave. public const byte Disconnect = LiteEventCode.Disconnect;

(251) Sent by Photon Cloud when a plugin-call or webhook-call failed. Usually, the execution on the server continues, despite the issue. Contains: ParameterCode.Info.

See also

https://doc.photonengine.com/en/realtime/current/reference/webhooks::options

8.4.2.5 const byte EventCode.GameList = 230

(230) Initial list of RoomInfos (in lobby on Master)

8.4.2.6 const byte EventCode.GameListUpdate = 229

(229) Update of RoomInfos to be merged into "initial" list (in lobby on Master)

8.4.2.7 const byte EventCode.Join = (byte)255

(255) Event Join: someone joined the game. The new actorNumber is provided as well as the properties of that actor (if set in OpJoin).

8.4.2.8 const byte EventCode.Leave = (byte)254

(254) Event Leave: The player who left the game can be identified by the actorNumber.

8.4.2.9 const byte EventCode.LobbyStats = 224

(224) This event provides a list of lobbies with their player and game counts.

8.4.2.10 const byte EventCode.Match = 227

(227) Currently not used. Event for matchmaking

8.4.2.11 const byte EventCode.PropertiesChanged = (byte)253

(253) When you call OpSetProperties with the broadcast option "on", this event is fired. It contains the properties being set.

8.4.2.12 const byte EventCode.QueueState = 228

(228) Currently not used. State of queueing in case of server-full

8.4.2.13 const byte EventCode.SetProperties = (byte)253

(253) When you call OpSetProperties with the broadcast option "on", this event is fired. It contains the properties being set.

8.5 Extensions Class Reference

This static class defines some useful extension methods for several existing classes (e.g. Vector3, float and others).

Static Public Member Functions

- static ParameterInfo[] GetCachedParemeters (this MethodInfo mo)
- static PhotonView[] GetPhotonViewsInChildren (this UnityEngine.GameObject go)
- static PhotonView GetPhotonView (this UnityEngine.GameObject go)
- static bool AlmostEquals (this Vector3 target, Vector3 second, float sqrMagnitudePrecision)

  compares the squared magnitude of target - second to given float value
- static bool AlmostEquals (this Vector2 target, Vector2 second, float sqrMagnitudePrecision)

  compares the squared magnitude of target - second to given float value
- static bool AlmostEquals (this Quaternion target, Quaternion second, float maxAngle)

compares the angle between target and second to given float value

- static bool AlmostEquals (this float target, float second, float floatDiff)

  compares two floats and returns true of their difference is less than floatDiff
- static void Merge (this IDictionary target, IDictionary addHash)

  Merges all keys from addHash into the target. Adds new keys and updates the values of existing keys in target.
- static void MergeStringKeys (this IDictionary target, IDictionary addHash)

  Merges keys of type string to target Hashtable.
- static string ToStringFull (this IDictionary origin)

  Returns a string-representation of the IDictionary's content, inlcuding type-information. Note: This might turn out a "heavy-duty" call if used frequently but it's usfuly to debug Dictionary or Hashtable content.
- static Hashtable StripToStringKeys (this IDictionary original)

  This method copies all string-typed keys of the original into a new Hashtable.
- static void StripKeysWithNullValues (this IDictionary original)

  This removes all key-value pairs that have a null-reference as value. Photon properties are removed by setting their value to null. Changes the original passed IDictionary!
- static bool Contains (this int[] target, int nr)

  Checks if a particular integer value is in an int-array.

Static Public Attributes

- static Dictionary< MethodInfo, ParameterInfo[]> parametersOfMethods = new Dictionary<MethodInfo, ParameterInfo[]>()

8.5.1 Detailed Description

This static class defines some useful extension methods for several existing classes (e.g. Vector3, float and others).

8.5.2 Member Function Documentation

8.5.2.1 static bool Extensions.AlmostEquals (this Vector3 *target,* Vector3 *second,* float *sqrMagnitudePrecision*) [static]

compares the squared magnitude of target - second to given float value

8.5.2.2 static bool Extensions.AlmostEquals (this Vector2 *target,* Vector2 *second,* float *sqrMagnitudePrecision*) [static]

compares the squared magnitude of target - second to given float value

8.5.2.3 static bool Extensions.AlmostEquals (this Quaternion *target,* Quaternion *second,* float *maxAngle*) [static]

compares the angle between target and second to given float value

8.5.2.4 static bool Extensions.AlmostEquals (this float *target,* float *second,* float *floatDiff*) [static]

compares two floats and returns true of their difference is less than floatDiff

8.5.2.5 static bool Extensions.Contains (this int[] *target,* int *nr*) [static]

Checks if a particular integer value is in an int-array.

This might be useful to look up if a particular actorNumber is in the list of players of a room.

Parameters

target	The array of ints to check.
nr	The number to lookup in target.

Returns

True if nr was found in target.

8.5.2.6 static ParameterInfo [] Extensions.GetCachedParemeters (this MethodInfo *mo*) [static]

8.5.2.7 static PhotonView Extensions.GetPhotonView (this UnityEngine.GameObject *go*) [static]

8.5.2.8 static PhotonView [] Extensions.GetPhotonViewsInChildren (this UnityEngine.GameObject *go*) [static]

8.5.2.9 static void Extensions.Merge (this IDictionary *target,* IDictionary *addHash*) [static]

Merges all keys from addHash into the target. Adds new keys and updates the values of existing keys in target.

Parameters

target	The IDictionary to update.
addHash	The IDictionary containing data to merge into target.

8.5.2.10 static void Extensions.MergeStringKeys (this IDictionary *target,* IDictionary *addHash*) [static]

Merges keys of type string to target Hashtable.

Does not remove keys from target (so non-string keys CAN be in target if they were before).

Parameters

target	The target IDicitionary passed in plus all string-typed keys from the addHash.
addHash	A IDictionary that should be merged partly into target to update it.

8.5.2.11 static void Extensions.StripKeysWithNullValues (this IDictionary *original*) [static]

This removes all key-value pairs that have a null-reference as value. Photon properties are removed by setting their value to null. Changes the original passed IDictionary!

Parameters

original	The IDictionary to strip of keys with null-values.

8.5.2.12 static Hashtable Extensions.StripToStringKeys (this IDictionary *original*) [static]

This method copies all string-typed keys of the original into a new Hashtable.

Does not recurse (!) into hashes that might be values in the root-hash. This does not modify the original.

Parameters

original	The original IDictonary to get string-typed keys from.

Returns

New Hashtable containing only string-typed keys of the original.

8.5.2.13 static string Extensions.ToStringFull (this IDictionary *origin*) [static]

Returns a string-representation of the IDictionary's content, inlcuding type-information. Note: This might turn out a "heavy-duty" call if used frequently but it's usfuly to debug Dictionary or Hashtable content.

Parameters

origin	Some Dictionary or Hashtable.

Returns

String of the content of the IDictionary.

8.5.3 Member Data Documentation

8.5.3.1 Dictionary<MethodInfo, ParameterInfo[]> Extensions.parametersOfMethods = new Dictionary<MethodInfo, ParameterInfo[]>() [static]

8.6 FriendInfo Class Reference

Used to store info about a friend's online state and in which room he/she is.

Public Member Functions

- override string ToString ()

Properties

- string Name [get, set]
- bool IsOnline [get, set]
- string Room [get, set]
- bool IsInRoom [get]

8.6.1 Detailed Description

Used to store info about a friend's online state and in which room he/she is.

8.6.2 Member Function Documentation

8.6.2.1 override string FriendInfo.ToString ( )

8.6.3 Property Documentation

8.6.3.1 bool FriendInfo.IsInRoom [get]

8.6.3.2 bool FriendInfo.IsOnline [get], [set]

8.6.3.3 string FriendInfo.Name [get], [set]

8.6.3.4 string FriendInfo.Room [get], [set]

8.7 GameObjectExtensions Class Reference

Small number of extension methods that make it easier for PUN to work cross-Unity-versions.

Static Public Member Functions

- static bool GetActive (this GameObject target)

  Unity-version-independent replacement for active GO property.

8.7.1 Detailed Description

Small number of extension methods that make it easier for PUN to work cross-Unity-versions.

8.7.2 Member Function Documentation

8.7.2.1 static bool GameObjectExtensions.GetActive (this GameObject *target*) [static]

Unity-version-independent replacement for active GO property.

Returns

Unity 3.5: active. Any newer Unity: activeInHierarchy.

8.8 GamePropertyKey Class Reference

Class for constants. These (byte) values are for "well known" room/game properties used in Photon Loadbalancing. Pun uses these constants internally.

Public Attributes

- const byte MaxPlayers = 255

  (255) Max number of players that "fit" into this room. 0 is for "unlimited".
- const byte IsVisible = 254

  (254) Makes this room listed or not in the lobby on master.
- const byte IsOpen = 253

  (253) Allows more players to join a room (or not).
- const byte PlayerCount = 252

  (252) Current count of players in the room. Used only in the lobby on master.
- const byte Removed = 251

  (251) True if the room is to be removed from room listing (used in update to room list in lobby on master)
- const byte PropsListedInLobby = 250

  (250) A list of the room properties to pass to the RoomInfo list in a lobby. This is used in CreateRoom, which defines this list once per room.
- const byte CleanupCacheOnLeave = 249

  (249) Equivalent of Operation Join parameter CleanupCacheOnLeave.
- const byte MasterClientId = (byte)248

  (248) Code for MasterClientId, which is synced by server. When sent as op-parameter this is (byte)203. As room property this is (byte)248.
- const byte ExpectedUsers = (byte)247

  (247) Code for ExpectedUsers in a room. Matchmaking keeps a slot open for the players with these userIDs.

8.8.1 Detailed Description

Class for constants. These (byte) values are for "well known" room/game properties used in Photon Loadbalancing. Pun uses these constants internally.

"Custom properties" have to use a string-type as key. They can be assigned at will.

8.8.2 Member Data Documentation

8.8.2.1 const byte GamePropertyKey.CleanupCacheOnLeave = 249

(249) Equivalent of Operation Join parameter CleanupCacheOnLeave.

8.8.2.2 const byte GamePropertyKey.ExpectedUsers = (byte)247

(247) Code for ExpectedUsers in a room. Matchmaking keeps a slot open for the players with these userIDs.

8.8.2.3 const byte GamePropertyKey.IsOpen = 253

(253) Allows more players to join a room (or not).

8.8.2.4 const byte GamePropertyKey.IsVisible = 254

(254) Makes this room listed or not in the lobby on master.

8.8.2.5 const byte GamePropertyKey.MasterClientId = (byte)248

(248) Code for MasterClientId, which is synced by server. When sent as op-parameter this is (byte)203. As room property this is (byte)248.

Tightly related to ParameterCode.MasterClientId.

8.8.2.6 const byte GamePropertyKey.MaxPlayers = 255

(255) Max number of players that "fit" into this room. 0 is for "unlimited".

8.8.2.7 const byte GamePropertyKey.PlayerCount = 252

(252) Current count of players in the room. Used only in the lobby on master.

8.8.2.8 const byte GamePropertyKey.PropsListedInLobby = 250

(250) A list of the room properties to pass to the RoomInfo list in a lobby. This is used in CreateRoom, which defines this list once per room.

8.8.2.9 const byte GamePropertyKey.Removed = 251

(251) True if the room is to be removed from room listing (used in update to room list in lobby on master)

8.9 ExitGames.Client.GUI.GizmoTypeDrawer Class Reference

Static Public Member Functions

- static void Draw (Vector3 center, GizmoType type, Color color, float size)

8.9.1 Member Function Documentation

8.9.1.1 static void ExitGames.Client.GUI.GizmoTypeDrawer.Draw (Vector3 *center,* GizmoType *type,* Color *color,* float *size*) [static]

8.10 HelpURL Class Reference

Empty implementation of the upcoming HelpURL of Unity 5.1. This one is only for compatibility of attributes.

Inherits Attribute.

Public Member Functions

- HelpURL (string url)

8.10.1 Detailed Description

Empty implementation of the upcoming HelpURL of Unity 5.1. This one is only for compatibility of attributes.

http://feedback.unity3d.com/suggestions/override-component-documentation-slash-help-link

8.10.2 Constructor & Destructor Documentation

8.10.2.1 HelpURL.HelpURL (string *url*)

8.11 IPunCallbacks Interface Reference

This interface is used as definition of all callback methods of PUN, except OnPhotonSerializeView. Preferably, implement them individually.

Inherited by Photon.PunBehaviour.

Public Member Functions

- void OnConnectedToPhoton ()

  Called when the initial connection got established but before you can use the server. OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.
- void OnLeftRoom ()

  Called when the local user/client left a room.
- void OnMasterClientSwitched (PhotonPlayer newMasterClient)

  Called after switching to a new MasterClient when the current one leaves.
- void OnPhotonCreateRoomFailed (object[] codeAndMsg)

  Called when a CreateRoom() call failed. The parameter provides ErrorCode and message (as array).
- void OnPhotonJoinRoomFailed (object[] codeAndMsg)

  Called when a JoinRoom() call failed. The parameter provides ErrorCode and message (as array).
- void OnCreatedRoom ()

  Called when this client created a room and entered it. OnJoinedRoom() will be called as well.
- void OnJoinedLobby ()

  Called on entering a lobby on the Master Server. The actual room-list updates will call OnReceivedRoomListUpdate().
- void OnLeftLobby ()

  Called after leaving a lobby.
- void OnFailedToConnectToPhoton (DisconnectCause cause)

Called if a connect call to the Photon server failed before the connection was established, followed by a call to OnDisconnectedFromPhoton().

- void OnConnectionFail (DisconnectCause cause)

  Called when something causes the connection to fail (after it was established), followed by a call to OnDisconnectedFromPhoton().
- void OnDisconnectedFromPhoton ()

  Called after disconnecting from the Photon server.
- void OnPhotonInstantiate (PhotonMessageInfo info)

  Called on all scripts on a GameObject (and children) that have been Instantiated using PhotonNetwork.Instantiate.
- void OnReceivedRoomListUpdate ()

  Called for any update of the room-listing while in a lobby (PhotonNetwork.insideLobby) on the Master Server.
- void OnJoinedRoom ()

  Called when entering a room (by creating or joining it). Called on all clients (including the Master Client).
- void OnPhotonPlayerConnected (PhotonPlayer newPlayer)

  Called when a remote player entered the room. This PhotonPlayer is already added to the playerlist at this time.
- void OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer)

  Called when a remote player left the room. This PhotonPlayer is already removed from the playerlist at this time.
- void OnPhotonRandomJoinFailed (object[] codeAndMsg)

  Called when a JoinRandom() call failed. The parameter provides ErrorCode and message.
- void OnConnectedToMaster ()

  Called after the connection to the master is established and authenticated but only when PhotonNetwork.autoJoinLobby is false.
- void OnPhotonMaxCccuReached ()

  Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting.
- void OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)

  Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.SetCustomProperties.
- void OnPhotonPlayerPropertiesChanged (object[] playerAndUpdatedProps)

  Called when custom player-properties are changed. Player and the changed properties are passed as object[].
- void OnUpdatedFriendList ()

  Called when the server sent the response to a FindFriends request and updated PhotonNetwork.Friends.
- void OnCustomAuthenticationFailed (string debugMessage)

  Called when the custom authentication failed. Followed by disconnect!
- void OnCustomAuthenticationResponse (Dictionary< string, object > data)

  Called when your Custom Authentication service responds with additional data.
- void OnWebRpcResponse (OperationResponse response)

  Called by PUN when the response to a WebRPC is available. See PhotonNetwork.WebRPC.
- void OnOwnershipRequest (object[] viewAndPlayer)

  Called when another player requests ownership of a PhotonView from you (the current owner).
- void OnLobbyStatisticsUpdate ()

  Called when the Master Server sent an update for the Lobby Statistics, updating PhotonNetwork.LobbyStatistics.

8.11.1 Detailed Description

This interface is used as definition of all callback methods of PUN, except OnPhotonSerializeView. Preferably, implement them individually.

This interface is available for completeness, more than for actually implementing it in a game. You can implement each method individually in any MonoMehaviour, without implementing IPunCallbacks.

PUN calls all callbacks by name. Don't use implement callbacks with fully qualified name. Example: IPunCallbacks.OnConnectedToPhoton won't get called by Unity's SendMessage().

PUN will call these methods on any script that implements them, analog to Unity's events and callbacks. The situation that triggers the call is described per method.

OnPhotonSerializeView is NOT called like these callbacks! It's usage frequency is much higher and it is implemented in: IPunObservable.

8.11.2 Member Function Documentation

8.11.2.1 void IPunCallbacks.OnConnectedToMaster ( )

Called after the connection to the master is established and authenticated but only when PhotonNetwork.autoJoinLobby is false.

If you set PhotonNetwork.autoJoinLobby to true, OnJoinedLobby() will be called instead of this.

You can join rooms and create them even without being in a lobby. The default lobby is used in that case. The list of available rooms won't become available unless you join a lobby via PhotonNetwork.joinLobby.

Implemented in Photon.PunBehaviour.

8.11.2.2 void IPunCallbacks.OnConnectedToPhoton ( )

Called when the initial connection got established but before you can use the server. OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.

This callback is only useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnFailedToConnectToPhoton() and OnDisconnectedFromPhoton().

OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.

When this is called, the low level connection is established and PUN will send your AppId, the user, etc in the background. This is not called for transitions from the masterserver to game servers.

Implemented in Photon.PunBehaviour.

8.11.2.3 void IPunCallbacks.OnConnectionFail (DisconnectCause *cause*)

Called when something causes the connection to fail (after it was established), followed by a call to OnDisconnectedFromPhoton().

If the server could not be reached in the first place, OnFailedToConnectToPhoton is called instead. The reason for the error is provided as DisconnectCause.

Implemented in Photon.PunBehaviour.

8.11.2.4 void IPunCallbacks.OnCreatedRoom ( )

Called when this client created a room and entered it. OnJoinedRoom() will be called as well.

This callback is only called on the client which created a room (see PhotonNetwork.CreateRoom).

As any client might close (or drop connection) anytime, there is a chance that the creator of a room does not execute OnCreatedRoom.

If you need specific room properties or a "start signal", it is safer to implement OnMasterClientSwitched() and to make the new MasterClient check the room's state.

Implemented in Photon.PunBehaviour.

8.11.2.5 void IPunCallbacks.OnCustomAuthenticationFailed (string *debugMessage*)

Called when the custom authentication failed. Followed by disconnect!

Custom Authentication can fail due to user-input, bad tokens/secrets. If authentication is successful, this method is not called. Implement OnJoinedLobby() or OnConnectedToMaster() (as usual).

During development of a game, it might also fail due to wrong configuration on the server side. In those cases, logging the debugMessage is very important.

Unless you setup a custom authentication service for your app (in the Dashboard), this won't be called!

Parameters

debugMessage	Contains a debug message why authentication failed. This has to be fixed during development time.

Implemented in Photon.PunBehaviour.

8.11.2.6 void IPunCallbacks.OnCustomAuthenticationResponse (Dictionary< string, object > *data*)

Called when your Custom Authentication service responds with additional data.

Custom Authentication services can include some custom data in their response. When present, that data is made available in this callback as Dictionary. While the keys of your data have to be strings, the values can be either string or a number (in Json). You need to make extra sure, that the value type is the one you expect. Numbers become (currently) int64.

Example: void OnCustomAuthenticationResponse(Dictionary<string, object> data) { ... }

https://doc.photonengine.com/en/realtime/current/reference/custom-authentication

Implemented in Photon.PunBehaviour.

8.11.2.7 void IPunCallbacks.OnDisconnectedFromPhoton ( )

Called after disconnecting from the Photon server.

In some cases, other callbacks are called before OnDisconnectedFromPhoton is called. Examples: OnConnectionFail() and OnFailedToConnectToPhoton().

Implemented in Photon.PunBehaviour.

8.11.2.8 void IPunCallbacks.OnFailedToConnectToPhoton (DisconnectCause *cause*)

Called if a connect call to the Photon server failed before the connection was established, followed by a call to OnDisconnectedFromPhoton().

This is called when no connection could be established at all. It differs from OnConnectionFail, which is called when an existing connection fails.

Implemented in Photon.PunBehaviour.

8.11.2.9 void IPunCallbacks.OnJoinedLobby ( )

Called on entering a lobby on the Master Server. The actual room-list updates will call OnReceivedRoomListUpdate().

Note: When PhotonNetwork.autoJoinLobby is false, OnConnectedToMaster() will be called and the room list won't become available.

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify). The room list gets available when OnReceivedRoomListUpdate() gets called after OnJoinedLobby().

Implemented in Photon.PunBehaviour.

8.11.2.10 void IPunCallbacks.OnJoinedRoom ( )

Called when entering a room (by creating or joining it). Called on all clients (including the Master Client).

This method is commonly used to instantiate player characters. If a match has to be started "actively", you can call an PunRPC triggered by a user's button-press or a timer.

When this is called, you can usually already access the existing players in the room via PhotonNetwork.playerList. Also, all custom properties should be already available as Room.customProperties. Check Room.playerCount to find out if enough players are in the room to start playing.

Implemented in Photon.PunBehaviour.

8.11.2.11 void IPunCallbacks.OnLeftLobby ( )

Called after leaving a lobby.

When you leave a lobby, CreateRoom and JoinRandomRoom automatically refer to the default lobby.

Implemented in Photon.PunBehaviour.

8.11.2.12 void IPunCallbacks.OnLeftRoom ( )

Called when the local user/client left a room.

When leaving a room, PUN brings you back to the Master Server. Before you can use lobbies and join or create rooms, OnJoinedLobby() or OnConnectedToMaster() will get called again.

Implemented in Photon.PunBehaviour.

8.11.2.13 void IPunCallbacks.OnLobbyStatisticsUpdate ( )

Called when the Master Server sent an update for the Lobby Statistics, updating PhotonNetwork.LobbyStatistics.

This callback has two preconditions: EnableLobbyStatistics must be set to true, before this client connects. And the client has to be connected to the Master Server, which is providing the info about lobbies.

Implemented in Photon.PunBehaviour.

8.11.2.14 void IPunCallbacks.OnMasterClientSwitched (PhotonPlayer *newMasterClient*)

Called after switching to a new MasterClient when the current one leaves.

This is not called when this client enters a room. The former MasterClient is still in the player list when this method get called.

Implemented in Photon.PunBehaviour.

8.11.2.15 void IPunCallbacks.OnOwnershipRequest (object[] *viewAndPlayer*)

Called when another player requests ownership of a PhotonView from you (the current owner).

The parameter viewAndPlayer contains:

PhotonView view = viewAndPlayer[0] as PhotonView;

PhotonPlayer requestingPlayer = viewAndPlayer[1] as PhotonPlayer;

Parameters

viewAndPlayer	The PhotonView is viewAndPlayer[0] and the requesting player is viewAndPlayer[1].

Implemented in Photon.PunBehaviour.

8.11.2.16 void IPunCallbacks.OnPhotonCreateRoomFailed (object[] *codeAndMsg*)

Called when a CreateRoom() call failed. The parameter provides ErrorCode and message (as array).

Most likely because the room name is already in use (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is short ErrorCode and codeAndMsg[1] is a string debug msg.

Implemented in Photon.PunBehaviour.

8.11.2.17 void IPunCallbacks.OnPhotonCustomRoomPropertiesChanged (Hashtable *propertiesThatChanged*)

Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.SetCustomProperties.

Since v1.25 this method has one parameter: Hashtable propertiesThatChanged.
Changing properties must be done by Room.SetCustomProperties, which causes this callback locally, too.

Parameters

propertiesThat↩ Changed	

Implemented in Photon.PunBehaviour.

8.11.2.18 void IPunCallbacks.OnPhotonInstantiate (PhotonMessageInfo *info*)

Called on all scripts on a GameObject (and children) that have been Instantiated using PhotonNetwork.Instantiate.

PhotonMessageInfo parameter provides info about who created the object and when (based off PhotonNetworking.time).

Implemented in Photon.PunBehaviour.

8.11.2.19 void IPunCallbacks.OnPhotonJoinRoomFailed (object[] *codeAndMsg*)

Called when a JoinRoom() call failed. The parameter provides ErrorCode and message (as array).

Most likely error is that the room does not exist or the room is full (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is short ErrorCode and codeAndMsg[1] is string debug msg.

Implemented in Photon.PunBehaviour.

8.11.2.20 void IPunCallbacks.OnPhotonMaxCccuReached ( )

Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting.

When this happens, the user might try again later. You can't create or join rooms in OnPhotonMaxCcuReached(), cause the client will be disconnecting. You can raise the CCU limits with a new license (when you host yourself)

or extended subscription (when using the Photon Cloud). The Photon Cloud will mail you when the CCU limit was reached. This is also visible in the Dashboard (webpage).

Implemented in Photon.PunBehaviour.

8.11.2.21 void IPunCallbacks.OnPhotonPlayerConnected (PhotonPlayer *newPlayer*)

Called when a remote player entered the room. This PhotonPlayer is already added to the playerlist at this time.

If your game starts with a certain number of players, this callback can be useful to check the Room.playerCount and find out if you can start.

Implemented in Photon.PunBehaviour.

8.11.2.22 void IPunCallbacks.OnPhotonPlayerDisconnected (PhotonPlayer *otherPlayer*)

Called when a remote player left the room. This PhotonPlayer is already removed from the playerlist at this time.

When your client calls PhotonNetwork.leaveRoom, PUN will call this method on the remaining clients. When a remote client drops connection or gets closed, this callback gets executed. after a timeout of several seconds.

Implemented in Photon.PunBehaviour.

8.11.2.23 void IPunCallbacks.OnPhotonPlayerPropertiesChanged (object[] *playerAndUpdatedProps*)

Called when custom player-properties are changed. Player and the changed properties are passed as object[].

Since v1.25 this method has one parameter: object[] playerAndUpdatedProps, which contains two entries.
[0] is the affected PhotonPlayer.
[1] is the Hashtable of properties that changed.

We are using a object[] due to limitations of Unity's GameObject.SendMessage (which has only one optional parameter).

Changing properties must be done by PhotonPlayer.SetCustomProperties, which causes this callback locally, too.

Example:

```
void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) {
    PhotonPlayer player = playerAndUpdatedProps[0] as PhotonPlayer;
    Hashtable props = playerAndUpdatedProps[1] as Hashtable;
    //...
}
```

Parameters

playerAnd↩ UpdatedProps	Contains PhotonPlayer and the properties that changed See remarks.

Implemented in Photon.PunBehaviour.

8.11.2.24 void IPunCallbacks.OnPhotonRandomJoinFailed (object[] *codeAndMsg*)

Called when a JoinRandom() call failed. The parameter provides ErrorCode and message.

Most likely all rooms are full or no rooms are available.
When using multiple lobbies (via JoinLobby or TypedLobby), another lobby might have more/fitting rooms.
PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg.

Implemented in Photon.PunBehaviour.

8.11.2.25 void IPunCallbacks.OnReceivedRoomListUpdate ( )

Called for any update of the room-listing while in a lobby (PhotonNetwork.insideLobby) on the Master Server.

PUN provides the list of rooms by PhotonNetwork.GetRoomList().
Each item is a RoomInfo which might include custom properties (provided you defined those as lobby-listed when creating a room).

Not all types of lobbies provide a listing of rooms to the client. Some are silent and specialized for server-side matchmaking.

Implemented in Photon.PunBehaviour.

8.11.2.26 void IPunCallbacks.OnUpdatedFriendList ( )

Called when the server sent the response to a FindFriends request and updated PhotonNetwork.Friends.

The friends list is available as PhotonNetwork.Friends, listing name, online state and the room a user is in (if any).

Implemented in Photon.PunBehaviour.

8.11.2.27 void IPunCallbacks.OnWebRpcResponse (OperationResponse *response*)

Called by PUN when the response to a WebRPC is available. See PhotonNetwork.WebRPC.

Important: The response.ReturnCode is 0 if Photon was able to reach your web-service.
The content of the response is what your web-service sent. You can create a WebRpcResponse from it.
Example: WebRpcResponse webResponse = new WebRpcResponse(operationResponse);

Please note: Class OperationResponse is in a namespace which needs to be "used":
using ExitGames.Client.Photon; // includes OperationResponse (and other classes)

The OperationResponse.ReturnCode by Photon is:

```
 0 for "OK"
-3 for "Web-Service not configured" (see Dashboard / WebHooks)
-5 for "Web-Service does now have RPC path/name" (at least for Azure)
```

Implemented in Photon.PunBehaviour.

8.12 IPunObservable Interface Reference

Defines the OnPhotonSerializeView method to make it easy to implement correctly for observable scripts.

Inherited by PhotonTransformView.

Public Member Functions

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

  Called by PUN several times per second, so that your script can write and read synchronization data for the Photon↩ View.

8.12.1 Detailed Description

Defines the OnPhotonSerializeView method to make it easy to implement correctly for observable scripts.

8.13 IPunPrefabPool Interface Reference

Defines all the methods that a Object Pool must implement, so that PUN can use it.

Public Member Functions

- GameObject Instantiate (string prefabId, Vector3 position, Quaternion rotation)

  This is called when PUN wants to create a new instance of an entity prefab. Must return valid GameObject with PhotonView.
- void Destroy (GameObject gameObject)

  This is called when PUN wants to destroy the instance of an entity prefab.

8.13.1 Detailed Description

Defines all the methods that a Object Pool must implement, so that PUN can use it.

To use a Object Pool for instantiation, you can set PhotonNetwork.ObjectPool. That is used for all objects, as long as ObjectPool is not null. The pool has to return a valid non-null GameObject when PUN calls Instantiate. Also, the position and rotation must be applied.

Please note that pooled GameObjects don't get the usual Awake and Start calls. OnEnable will be called (by your pool) but the networking values are not updated yet when that happens. OnEnable will have outdated values for PhotonView (isMine, etc.). You might have to adjust scripts.

PUN will call OnPhotonInstantiate (see IPunCallbacks). This should be used to setup the re-used object with regards to networking values / ownership.

8.13.2 Member Function Documentation

8.13.2.1 void IPunPrefabPool.Destroy (GameObject *gameObject*)

This is called when PUN wants to destroy the instance of an entity prefab.

A pool needs some way to find out which type of GameObject got returned via Destroy(). It could be a tag or name or anything similar.

Parameters

gameObject	The instance to destroy.

8.13.2.2 GameObject IPunPrefabPool.Instantiate (string *prefabId*, Vector3 *position*, Quaternion *rotation*)

This is called when PUN wants to create a new instance of an entity prefab. Must return valid GameObject with PhotonView.

Parameters

prefabId	The id of this prefab.

position	The position we want the instance instantiated at.
rotation	The rotation we want the instance to take.

Returns

The newly instantiated object, or null if a prefab with *prefabId* was not found.

8.14 Photon.MonoBehaviour Class Reference

This class adds the property photonView, while logging a warning when your game still uses the networkView.

Inherits MonoBehaviour.

Inherited by Photon.PunBehaviour, PhotonHandler, and PhotonView.

Properties

- PhotonView photonView `[get]`

  A cached reference to a PhotonView on this GameObject.

- new PhotonView networkView `[get]`

  This property is only here to notify developers when they use the outdated value.

8.14.1 Detailed Description

This class adds the property photonView, while logging a warning when your game still uses the networkView.

8.14.2 Property Documentation

8.14.2.1 new PhotonView Photon.MonoBehaviour.networkView `[get]`

This property is only here to notify developers when they use the outdated value.

If Unity 5.x logs a compiler warning "Use the new keyword if hiding was intended" or "The new keyword is not required", you may suffer from an Editor issue. Try to modify networkView with a if-def condition:

#if UNITY_EDITOR new #endif public PhotonView networkView

8.14.2.2 PhotonView Photon.MonoBehaviour.photonView `[get]`

A cached reference to a PhotonView on this GameObject.

If you intend to work with a PhotonView in a script, it's usually easier to write this.photonView.

If you intend to remove the PhotonView component from the GameObject but keep this Photon.MonoBehaviour, avoid this reference or modify this code to use PhotonView.Get(obj) instead.

8.15 OperationCode Class Reference

Class for constants. Contains operation codes. Pun uses these constants internally.

Public Attributes

- const byte ExchangeKeysForEncryption = 250
- const byte Join = 255

  (255) Code for OpJoin, to get into a room.
- const byte Authenticate = 230

  (230) Authenticates this peer and connects to a virtual application
- const byte JoinLobby = 229

  (229) Joins lobby (on master)
- const byte LeaveLobby = 228

  (228) Leaves lobby (on master)
- const byte CreateGame = 227

  (227) Creates a game (or fails if name exists)
- const byte JoinGame = 226

  (226) Join game (by name)
- const byte JoinRandomGame = 225

  (225) Joins random game (on master)
- const byte Leave = (byte)254

  (254) Code for OpLeave, to get out of a room.
- const byte RaiseEvent = (byte)253

  (253) Raise event (in a room, for other actors/players)
- const byte SetProperties = (byte)252

  (252) Set Properties (of room or actor/player)
- const byte GetProperties = (byte)251

  (251) Get Properties
- const byte ChangeGroups = (byte)248

  (248) Operation code to change interest groups in Rooms (Lite application and extending ones).
- const byte FindFriends = 222

  (222) Request the rooms and online status for a list of friends (by name, which should be unique).
- const byte GetLobbyStats = 221

  (221) Request statistics about a specific list of lobbies (their user and game count).
- const byte GetRegions = 220

  (220) Get list of regional servers from a NameServer.
- const byte WebRpc = 219

  (219) WebRpc Operation.

8.15.1 Detailed Description

Class for constants. Contains operation codes. Pun uses these constants internally.

8.15.2 Member Data Documentation

8.15.2.1 const byte OperationCode.Authenticate = 230

(230) Authenticates this peer and connects to a virtual application

8.15.2.2 const byte OperationCode.ChangeGroups = (byte)248

(248) Operation code to change interest groups in Rooms (Lite application and extending ones).

8.15.2.3 const byte OperationCode.CreateGame = 227

(227) Creates a game (or fails if name exists)

8.15.2.4 const byte OperationCode.ExchangeKeysForEncryption = 250

8.15.2.5 const byte OperationCode.FindFriends = 222

(222) Request the rooms and online status for a list of friends (by name, which should be unique).

8.15.2.6 const byte OperationCode.GetLobbyStats = 221

(221) Request statistics about a specific list of lobbies (their user and game count).

8.15.2.7 const byte OperationCode.GetProperties = (byte)251

(251) Get Properties

8.15.2.8 const byte OperationCode.GetRegions = 220

(220) Get list of regional servers from a NameServer.

8.15.2.9 const byte OperationCode.Join = 255

(255) Code for OpJoin, to get into a room.

8.15.2.10 const byte OperationCode.JoinGame = 226

(226) Join game (by name)

8.15.2.11 const byte OperationCode.JoinLobby = 229

(229) Joins lobby (on master)

8.15.2.12 const byte OperationCode.JoinRandomGame = 225

(225) Joins random game (on master)

8.15.2.13 const byte OperationCode.Leave = (byte)254

(254) Code for OpLeave, to get out of a room.

8.15.2.14 const byte OperationCode.LeaveLobby = 228

(228) Leaves lobby (on master)

8.15.2.15 const byte OperationCode.RaiseEvent = (byte)253

(253) Raise event (in a room, for other actors/players)

8.15.2.16 const byte OperationCode.SetProperties = (byte)252

(252) Set Properties (of room or actor/player)

8.15.2.17 const byte OperationCode.WebRpc = 219

(219) WebRpc Operation.

8.16 ParameterCode Class Reference

Class for constants. Codes for parameters of Operations and Events. Pun uses these constants internally.

Public Attributes

- const byte SuppressRoomEvents = 237

  (237) A bool parameter for creating games. If set to true, no room events are sent to the clients on join and leave. Default: false (and not sent).
- const byte EmptyRoomTTL = 236

  (236) Time To Live (TTL) for a room when the last player leaves. Keeps room in memory for case a player re-joins soon. In milliseconds.
- const byte PlayerTTL = 235

  (235) Time To Live (TTL) for an 'actor' in a room. If a client disconnects, this actor is inactive first and removed after this timeout. In milliseconds.
- const byte EventForward = 234

  (234) Optional parameter of OpRaiseEvent and OpSetCustomProperties to forward the event/operation to a web-service.
- const byte IsComingBack = (byte)233

  (233) Optional parameter of OpLeave in async games. If false, the player does abandons the game (forever). By default players become inactive and can re-join.
- const byte IsInactive = (byte)233

  (233) Used in EvLeave to describe if a user is inactive (and might come back) or not. In rooms with PlayerTTL, becoming inactive is the default case.
- const byte CheckUserOnJoin = (byte)232

  (232) Used when creating rooms to define if any userid can join the room only once.
- const byte ExpectedValues = (byte)231

  (231) Code for "Check And Swap" (CAS) when changing properties.
- const byte Address = 230

  (230) Address of a (game) server to use.
- const byte PeerCount = 229

  (229) Count of players in this application in a rooms (used in stats event)
- const byte GameCount = 228

  (228) Count of games in this application (used in stats event)
- const byte MasterPeerCount = 227

  (227) Count of players on the master server (in this app, looking for rooms)
- const byte UserId = 225

  (225) User's ID
- const byte ApplicationId = 224

  (224) Your application's ID: a name on your own Photon or a GUID on the Photon Cloud
- const byte Position = 223

  (223) Not used currently (as "Position"). If you get queued before connect, this is your position
- const byte MatchMakingType = 223

(223) Modifies the matchmaking algorithm used for OpJoinRandom. Allowed parameter values are defined in enum MatchmakingMode.

- const byte GameList = 222

  (222) List of RoomInfos about open / listed rooms
- const byte Secret = 221

  (221) Internally used to establish encryption
- const byte AppVersion = 220

  (220) Version of your application
- const byte AzureNodeInfo = 210

  (210) Internally used in case of hosting by Azure
- const byte AzureLocalNodeId = 209

  (209) Internally used in case of hosting by Azure
- const byte AzureMasterNodeId = 208

  (208) Internally used in case of hosting by Azure
- const byte RoomName = (byte)255

  (255) Code for the gameId/roomName (a unique name per room). Used in OpJoin and similar.
- const byte Broadcast = (byte)250

  (250) Code for broadcast parameter of OpSetProperties method.
- const byte ActorList = (byte)252

  (252) Code for list of players in a room. Currently not used.
- const byte ActorNr = (byte)254

  (254) Code of the Actor of an operation. Used for property get and set.
- const byte PlayerProperties = (byte)249

  (249) Code for property set (Hashtable).
- const byte CustomEventContent = (byte)245

  (245) Code of data/custom content of an event. Used in OpRaiseEvent.
- const byte Data = (byte)245

  (245) Code of data of an event. Used in OpRaiseEvent.
- const byte Code = (byte)244

  (244) Code used when sending some code-related parameter, like OpRaiseEvent's event-code.
- const byte GameProperties = (byte)248

  (248) Code for property set (Hashtable).
- const byte Properties = (byte)251

  (251) Code for property-set (Hashtable). This key is used when sending only one set of properties. If either ActorProperties or GameProperties are used (or both), check those keys.
- const byte TargetActorNr = (byte)253

  (253) Code of the target Actor of an operation. Used for property set. Is 0 for game
- const byte ReceiverGroup = (byte)246

  (246) Code to select the receivers of events (used in Lite, Operation RaiseEvent).
- const byte Cache = (byte)247

  (247) Code for caching events while raising them.
- const byte CleanupCacheOnLeave = (byte)241

  (241) Bool parameter of CreateGame Operation. If true, server cleans up roomcache of leaving players (their cached events get removed).
- const byte Group = 240

  (240) Code for "group" operation-parameter (as used in Op RaiseEvent).
- const byte Remove = 239

  (239) The "Remove" operation-parameter can be used to remove something from a list. E.g. remove groups from player's interest groups.
- const byte PublishUserId = 239

  (239) Used in Op Join to define if UserIds of the players are broadcast in the room. Useful for FindFriends and reserving slots for expected users.

- const byte Add = 238

  (238) The "Add" operation-parameter can be used to add something to some list or set. E.g. add groups to player's interest groups.
- const byte Info = 218

  (218) Content for EventCode.ErrorInfo and internal debug operations.
- const byte ClientAuthenticationType = 217

  (217) This key's (byte) value defines the target custom authentication type/service the client connects with. Used in OpAuthenticate
- const byte ClientAuthenticationParams = 216

  (216) This key's (string) value provides parameters sent to the custom authentication type/service the client connects with. Used in OpAuthenticate
- const byte JoinMode = 215

  (215) Makes the server create a room if it doesn't exist. OpJoin uses this to always enter a room, unless it exists and is full/closed.
- const byte ClientAuthenticationData = 214

  (214) This key's (string or byte[]) value provides parameters sent to the custom authentication service setup in Photon Dashboard. Used in OpAuthenticate
- const byte MasterClientId = (byte)203

  (203) Code for MasterClientId, which is synced by server. When sent as op-parameter this is code 203.
- const byte FindFriendsRequestList = (byte)1

  (1) Used in Op FindFriends request. Value must be string[] of friends to look up.
- const byte FindFriendsResponseOnlineList = (byte)1

  (1) Used in Op FindFriends response. Contains bool[] list of online states (false if not online).
- const byte FindFriendsResponseRoomIdList = (byte)2

  (2) Used in Op FindFriends response. Contains string[] of room names ("" where not known or no room joined).
- const byte LobbyName = (byte)213

  (213) Used in matchmaking-related methods and when creating a room to name a lobby (to join or to attach a room to).
- const byte LobbyType = (byte)212

  (212) Used in matchmaking-related methods and when creating a room to define the type of a lobby. Combined with the lobby name this identifies the lobby.
- const byte LobbyStats = (byte)211

  (211) This (optional) parameter can be sent in Op Authenticate to turn on Lobby Stats (info about lobby names and their user- and game-counts). See: PhotonNetwork.Lobbies
- const byte Region = (byte)210

  (210) Used for region values in OpAuth and OpGetRegions.
- const byte UriPath = 209

  (209) Path of the WebRPC that got called. Also known as "WebRpc Name". Type: string.
- const byte WebRpcParameters = 208

  (208) Parameters for a WebRPC as: Dictionary<string, object>. This will get serialized to JSon.
- const byte WebRpcReturnCode = 207

  (207) ReturnCode for the WebRPC, as sent by the web service (not by Photon, which uses ErrorCode). Type: byte.
- const byte WebRpcReturnMessage = 206

  (206) Message returned by WebRPC server. Analog to Photon's debug message. Type: string.
- const byte CacheSliceIndex = 205

  (205) Used to define a "slice" for cached events. Slices can easily be removed from cache. Type: int.
- const byte Plugins = 204

  Informs the server of the expected plugin setup. The operation will fail in case of a plugin mismatch returning error code PluginMismatch 32751(0x7FFF - 16). Setting string[]{} means the client expects no plugin to be setup. Note: for backwards compatibility null omits any check.
- const byte NickName = 202

  (202) Used by the server in Operation Responses, when it sends the nickname of the client (the user's nickname).
- const byte PluginName = 201

(201) Informs user about name of plugin load to game

- const byte PluginVersion = 200

  (200) Informs user about version of plugin load to game

8.16.1 Detailed Description

Class for constants. Codes for parameters of Operations and Events. Pun uses these constants internally.

8.16.2 Member Data Documentation

8.16.2.1 const byte ParameterCode.ActorList = (byte)252

(252) Code for list of players in a room. Currently not used.

8.16.2.2 const byte ParameterCode.ActorNr = (byte)254

(254) Code of the Actor of an operation. Used for property get and set.

8.16.2.3 const byte ParameterCode.Add = 238

(238) The "Add" operation-parameter can be used to add something to some list or set. E.g. add groups to player's interest groups.

8.16.2.4 const byte ParameterCode.Address = 230

(230) Address of a (game) server to use.

8.16.2.5 const byte ParameterCode.ApplicationId = 224

(224) Your application's ID: a name on your own Photon or a GUID on the Photon Cloud

8.16.2.6 const byte ParameterCode.AppVersion = 220

(220) Version of your application

8.16.2.7 const byte ParameterCode.AzureLocalNodeId = 209

(209) Internally used in case of hosting by Azure

8.16.2.8 const byte ParameterCode.AzureMasterNodeId = 208

(208) Internally used in case of hosting by Azure

8.16.2.9 const byte ParameterCode.AzureNodeInfo = 210

(210) Internally used in case of hosting by Azure

8.16.2.10 const byte ParameterCode.Broadcast = (byte)250

(250) Code for broadcast parameter of OpSetProperties method.

8.16.2.11 const byte ParameterCode.Cache = (byte)247

(247) Code for caching events while raising them.

8.16.2.12 const byte ParameterCode.CacheSliceIndex = 205

(205) Used to define a "slice" for cached events. Slices can easily be removed from cache. Type: int.

8.16.2.13 const byte ParameterCode.CheckUserOnJoin = (byte)232

(232) Used when creating rooms to define if any userid can join the room only once.

8.16.2.14 const byte ParameterCode.CleanupCacheOnLeave = (byte)241

(241) Bool parameter of CreateGame Operation. If true, server cleans up roomcache of leaving players (their cached events get removed).

8.16.2.15 const byte ParameterCode.ClientAuthenticationData = 214

(214) This key's (string or byte[]) value provides parameters sent to the custom authentication service setup in Photon Dashboard. Used in OpAuthenticate

8.16.2.16 const byte ParameterCode.ClientAuthenticationParams = 216

(216) This key's (string) value provides parameters sent to the custom authentication type/service the client connects with. Used in OpAuthenticate

8.16.2.17 const byte ParameterCode.ClientAuthenticationType = 217

(217) This key's (byte) value defines the target custom authentication type/service the client connects with. Used in OpAuthenticate

8.16.2.18 const byte ParameterCode.Code = (byte)244

(244) Code used when sending some code-related parameter, like OpRaiseEvent's event-code.

This is not the same as the Operation's code, which is no longer sent as part of the parameter Dictionary in Photon 3.

8.16.2.19 const byte ParameterCode.CustomEventContent = (byte)245

(245) Code of data/custom content of an event. Used in OpRaiseEvent.

8.16.2.20 const byte ParameterCode.Data = (byte)245

(245) Code of data of an event. Used in OpRaiseEvent.

8.16.2.21 const byte ParameterCode.EmptyRoomTTL = 236

(236) Time To Live (TTL) for a room when the last player leaves. Keeps room in memory for case a player re-joins soon. In milliseconds.

8.16.2.22 const byte ParameterCode.EventForward = 234

(234) Optional parameter of OpRaiseEvent and OpSetCustomProperties to forward the event/operation to a web-service.

8.16.2.23 const byte ParameterCode.ExpectedValues = (byte)231

(231) Code for "Check And Swap" (CAS) when changing properties.

8.16.2.24 const byte ParameterCode.FindFriendsRequestList = (byte)1

(1) Used in Op FindFriends request. Value must be string[] of friends to look up.

8.16.2.25 const byte ParameterCode.FindFriendsResponseOnlineList = (byte)1

(1) Used in Op FindFriends response. Contains bool[] list of online states (false if not online).

8.16.2.26 const byte ParameterCode.FindFriendsResponseRoomIdList = (byte)2

(2) Used in Op FindFriends response. Contains string[] of room names ("" where not known or no room joined).

8.16.2.27 const byte ParameterCode.GameCount = 228

(228) Count of games in this application (used in stats event)

8.16.2.28 const byte ParameterCode.GameList = 222

(222) List of RoomInfos about open / listed rooms

8.16.2.29 const byte ParameterCode.GameProperties = (byte)248

(248) Code for property set (Hashtable).

8.16.2.30 const byte ParameterCode.Group = 240

(240) Code for "group" operation-parameter (as used in Op RaiseEvent).

8.16.2.31 const byte ParameterCode.Info = 218

(218) Content for EventCode.ErrorInfo and internal debug operations.

8.16.2.32 const byte ParameterCode.IsComingBack = (byte)233

(233) Optional parameter of OpLeave in async games. If false, the player does abandons the game (forever). By default players become inactive and can re-join.

8.16.2.33 const byte ParameterCode.IsInactive = (byte)233

(233) Used in EvLeave to describe if a user is inactive (and might come back) or not. In rooms with PlayerTTL, becoming inactive is the default case.

8.16.2.34 const byte ParameterCode.JoinMode = 215

(215) Makes the server create a room if it doesn't exist. OpJoin uses this to always enter a room, unless it exists and is full/closed.

(215) The JoinMode enum defines which variant of joining a room will be executed: Join only if available, create if not exists or re-join.

Replaces CreateIfNotExists which was only a bool-value.

8.16.2.35 const byte ParameterCode.LobbyName = (byte)213

(213) Used in matchmaking-related methods and when creating a room to name a lobby (to join or to attach a room to).

8.16.2.36 const byte ParameterCode.LobbyStats = (byte)211

(211) This (optional) parameter can be sent in Op Authenticate to turn on Lobby Stats (info about lobby names and their user- and game-counts). See: PhotonNetwork.Lobbies

8.16.2.37 const byte ParameterCode.LobbyType = (byte)212

(212) Used in matchmaking-related methods and when creating a room to define the type of a lobby. Combined with the lobby name this identifies the lobby.

8.16.2.38 const byte ParameterCode.MasterClientId = (byte)203

(203) Code for MasterClientId, which is synced by server. When sent as op-parameter this is code 203.

Tightly related to GamePropertyKey.MasterClientId.

8.16.2.39 const byte ParameterCode.MasterPeerCount = 227

(227) Count of players on the master server (in this app, looking for rooms)

8.16.2.40 const byte ParameterCode.MatchMakingType = 223

(223) Modifies the matchmaking algorithm used for OpJoinRandom. Allowed parameter values are defined in enum MatchmakingMode.

8.16.2.41 const byte ParameterCode.NickName = 202

(202) Used by the server in Operation Responses, when it sends the nickname of the client (the user's nickname).

8.16.2.42 const byte ParameterCode.PeerCount = 229

(229) Count of players in this application in a rooms (used in stats event)

8.16.2.43 const byte ParameterCode.PlayerProperties = (byte)249

(249) Code for property set (Hashtable).

8.16.2.44 const byte ParameterCode.PlayerTTL = 235

(235) Time To Live (TTL) for an 'actor' in a room. If a client disconnects, this actor is inactive first and removed after this timeout. In milliseconds.

8.16.2.45 const byte ParameterCode.PluginName = 201

(201) Informs user about name of plugin load to game

8.16.2.46 const byte ParameterCode.Plugins = 204

Informs the server of the expected plugin setup. The operation will fail in case of a plugin mismatch returning error code PluginMismatch 32751(0x7FFF - 16). Setting string[]{} means the client expects no plugin to be setup. Note: for backwards compatibility null omits any check.

8.16.2.47 const byte ParameterCode.PluginVersion = 200

(200) Informs user about version of plugin load to game

8.16.2.48 const byte ParameterCode.Position = 223

(223) Not used currently (as "Position"). If you get queued before connect, this is your position

8.16.2.49 const byte ParameterCode.Properties = (byte)251

(251) Code for property-set (Hashtable). This key is used when sending only one set of properties. If either ActorProperties or GameProperties are used (or both), check those keys.

8.16.2.50 const byte ParameterCode.PublishUserId = 239

(239) Used in Op Join to define if UserIds of the players are broadcast in the room. Useful for FindFriends and reserving slots for expected users.

8.16.2.51 const byte ParameterCode.ReceiverGroup = (byte)246

(246) Code to select the receivers of events (used in Lite, Operation RaiseEvent).

8.16.2.52 const byte ParameterCode.Region = (byte)210

(210) Used for region values in OpAuth and OpGetRegions.

8.16.2.53 const byte ParameterCode.Remove = 239

(239) The "Remove" operation-parameter can be used to remove something from a list. E.g. remove groups from player's interest groups.

8.16.2.54 const byte ParameterCode.RoomName = (byte)255

(255) Code for the gameId/roomName (a unique name per room). Used in OpJoin and similar.

8.16.2.55 const byte ParameterCode.Secret = 221

(221) Internally used to establish encryption

8.16.2.56 const byte ParameterCode.SuppressRoomEvents = 237

(237) A bool parameter for creating games. If set to true, no room events are sent to the clients on join and leave. Default: false (and not sent).

8.16.2.57 const byte ParameterCode.TargetActorNr = (byte)253

(253) Code of the target Actor of an operation. Used for property set. Is 0 for game

8.16.2.58 const byte ParameterCode.UriPath = 209

(209) Path of the WebRPC that got called. Also known as "WebRpc Name". Type: string.

8.16.2.59 const byte ParameterCode.UserId = 225

(225) User's ID

8.16.2.60 const byte ParameterCode.WebRpcParameters = 208

(208) Parameters for a WebRPC as: Dictionary<string, object>. This will get serialized to JSon.

8.16.2.61 const byte ParameterCode.WebRpcReturnCode = 207

(207) ReturnCode for the WebRPC, as sent by the web service (not by Photon, which uses ErrorCode). Type: byte.

8.16.2.62 const byte ParameterCode.WebRpcReturnMessage = 206

(206) Message returned by WebRPC server. Analog to Photon's debug message. Type: string.

8.17 PhotonAnimatorView Class Reference

This class helps you to synchronize Mecanim animations Simply add the component to your GameObject and make sure that the PhotonAnimatorView is added to the list of observed components

Inherits MonoBehaviour.

Classes

- class SynchronizedLayer
- class SynchronizedParameter

Public Types

- enum ParameterType { ParameterType.Float = 1, ParameterType.Int = 3, ParameterType.Bool = 4, ParameterType.Trigger = 9 }
- enum SynchronizeType { SynchronizeType.Disabled = 0, SynchronizeType.Discrete = 1, SynchronizeType.Continuous = 2 }

Public Member Functions

- void CacheDiscreteTriggers ()

  Caches the discrete triggers values for keeping track of raised triggers, and will be reseted after the sync routine got performed
- bool DoesLayerSynchronizeTypeExist (int layerIndex)

  Check if a specific layer is configured to be synchronize
- bool DoesParameterSynchronizeTypeExist (string name)

  Check if the specified parameter is configured to be synchronized
- List< SynchronizedLayer > GetSynchronizedLayers ()

  Get a list of all synchronized layers
- List< SynchronizedParameter > GetSynchronizedParameters ()

  Get a list of all synchronized parameters
- SynchronizeType GetLayerSynchronizeType (int layerIndex)

  Gets the type how the layer is synchronized
- SynchronizeType GetParameterSynchronizeType (string name)

  Gets the type how the parameter is synchronized
- void SetLayerSynchronized (int layerIndex, SynchronizeType synchronizeType)

  Sets the how a layer should be synchronized
- void SetParameterSynchronized (string name, ParameterType type, SynchronizeType synchronizeType)

  Sets the how a parameter should be synchronized

8.17.1 Detailed Description

This class helps you to synchronize Mecanim animations Simply add the component to your GameObject and make sure that the PhotonAnimatorView is added to the list of observed components

When Using Trigger Parameters, make sure the component that sets the trigger is higher in the stack of Components on the GameObject than 'PhotonAnimatorView' Triggers are raised true during one frame only.

8.17.2 Member Enumeration Documentation

8.17.2.1 enum PhotonAnimatorView.ParameterType

Enumerator

Float

Int

Bool

Trigger

8.17.2.2 enum PhotonAnimatorView.SynchronizeType

Enumerator

Disabled

Discrete

Continuous

8.17.3 Member Function Documentation

8.17.3.1 void PhotonAnimatorView.CacheDiscreteTriggers ( )

Caches the discrete triggers values for keeping track of raised triggers, and will be reseted after the sync routine got performed

8.17.3.2 bool PhotonAnimatorView.DoesLayerSynchronizeTypeExist (int *layerIndex*)

Check if a specific layer is configured to be synchronize

Parameters

layerIndex	Index of the layer.

Returns

True if the layer is synchronized

8.17.3.3 bool PhotonAnimatorView.DoesParameterSynchronizeTypeExist (string *name*)

Check if the specified parameter is configured to be synchronized

Parameters

name	The name of the parameter.

Returns

True if the parameter is synchronized

8.17.3.4 SynchronizeType PhotonAnimatorView.GetLayerSynchronizeType (int *layerIndex*)

Gets the type how the layer is synchronized

Parameters

layerIndex	Index of the layer.

Returns

Disabled/Discrete/Continuous

8.17.3.5 SynchronizeType PhotonAnimatorView.GetParameterSynchronizeType (string *name*)

Gets the type how the parameter is synchronized

Parameters

name	The name of the parameter.

Returns

Disabled/Discrete/Continuous

8.17.3.6 List<SynchronizedLayer> PhotonAnimatorView.GetSynchronizedLayers ( )

Get a list of all synchronized layers

Returns

List of SynchronizedLayer objects

8.17.3.7 List<SynchronizedParameter> PhotonAnimatorView.GetSynchronizedParameters ( )

Get a list of all synchronized parameters

Returns

List of SynchronizedParameter objects

8.17.3.8 void PhotonAnimatorView.SetLayerSynchronized (int *layerIndex*, SynchronizeType *synchronizeType*)

Sets the how a layer should be synchronized

Parameters

layerIndex	Index of the layer.
synchronizeType	Disabled/Discrete/Continuous

8.17.3.9 void PhotonAnimatorView.SetParameterSynchronized (string *name*, ParameterType *type*, SynchronizeType *synchronizeType*)

Sets the how a parameter should be synchronized

Parameters

name	The name of the parameter.
type	The type of the parameter.
synchronizeType	Disabled/Discrete/Continuous

8.18 PhotonLagSimulationGui Class Reference

This MonoBehaviour is a basic GUI for the Photon client's network-simulation feature. It can modify lag (fixed delay), jitter (random lag) and packet loss.

Inherits MonoBehaviour.

Public Member Functions

- void Start ()
- void OnGUI ()

Public Attributes

- Rect WindowRect = new Rect(0, 100, 120, 100)

  Positioning rect for window.
- int WindowId = 101

  Unity GUI Window ID (must be unique or will cause issues).
- bool Visible = true

  Shows or hides GUI (does not affect settings).

Properties

- PhotonPeer Peer `[get, set]`

  The peer currently in use (to set the network simulation).

8.18.1 Detailed Description

This MonoBehaviour is a basic GUI for the Photon client's network-simulation feature. It can modify lag (fixed delay), jitter (random lag) and packet loss.

8.18.2 Member Function Documentation

8.18.2.1 void PhotonLagSimulationGui.OnGUI ( )

8.18.2.2 void PhotonLagSimulationGui.Start ( )

8.18.3 Member Data Documentation

8.18.3.1 bool PhotonLagSimulationGui.Visible = true

Shows or hides GUI (does not affect settings).

8.18.3.2 int PhotonLagSimulationGui.WindowId = 101

Unity GUI Window ID (must be unique or will cause issues).

8.18.3.3 Rect PhotonLagSimulationGui.WindowRect = new Rect(0, 100, 120, 100)

Positioning rect for window.

8.18.4 Property Documentation

8.18.4.1 PhotonPeer PhotonLagSimulationGui.Peer `[get],[set]`

The peer currently in use (to set the network simulation).

8.19 PhotonMessageInfo Class Reference

Container class for info about a particular message, RPC or update.

Public Member Functions

- PhotonMessageInfo ()

  Initializes a new instance of the PhotonMessageInfo class. To create an empty messageinfo only!
- PhotonMessageInfo (PhotonPlayer player, int timestamp, PhotonView view)
- override string ToString ()

Public Attributes

- PhotonPlayer sender
- PhotonView photonView

Properties

- double timestamp [get]

8.19.1 Detailed Description

Container class for info about a particular message, RPC or update.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 PhotonMessageInfo.PhotonMessageInfo ( )

Initializes a new instance of the PhotonMessageInfo class. To create an empty messageinfo only!

8.19.2.2 PhotonMessageInfo.PhotonMessageInfo (PhotonPlayer *player,* int *timestamp,* PhotonView *view*)

8.19.3 Member Function Documentation

8.19.3.1 override string PhotonMessageInfo.ToString ( )

8.19.4 Member Data Documentation

8.19.4.1 PhotonView PhotonMessageInfo.photonView

8.19.4.2 PhotonPlayer PhotonMessageInfo.sender

8.19.5 Property Documentation

8.19.5.1 double PhotonMessageInfo.timestamp [get]

8.20 PhotonNetwork Class Reference

The main class to use the PhotonNetwork plugin. This class is static.

Public Member Functions

- delegate void EventCallback (byte eventCode, object content, int senderId)

  Defines the delegate usable in OnEventCall.

Static Public Member Functions

- static void SwitchToProtocol (ConnectionProtocol cp)

  While offline, the network protocol can be switched (which affects the ports you can use to connect).
- static bool ConnectUsingSettings (string gameVersion)

  Connect to Photon as configured in the editor (saved in PhotonServerSettings file).
- static bool ConnectToMaster (string masterServerAddress, int port, string appID, string gameVersion)

  Connect to a Photon Master Server by address, port, appID and game(client) version.
- static bool Reconnect ()

  Can be used to reconnect to the master server after a disconnect.
- static bool ReconnectAndRejoin ()

  When the client lost connection during gameplay, this method attempts to reconnect and rejoin the room.
- static bool ConnectToBestCloudServer (string gameVersion)

  Connect to the Photon Cloud region with the lowest ping (on platforms that support Unity's Ping).
- static bool ConnectToRegion (CloudRegionCode region, string gameVersion)

  Connects to the Photon Cloud region of choice.
- static void OverrideBestCloudServer (CloudRegionCode region)

  Overwrites the region that is used for ConnectToBestCloudServer(string gameVersion).
- static void RefreshCloudServerRating ()

  Pings all cloud servers again to find the one with best ping (currently).
- static void NetworkStatisticsReset ()

  Resets the traffic stats and re-enables them.
- static string NetworkStatisticsToString ()

  Only available when NetworkStatisticsEnabled was used to gather some stats.
- static void InitializeSecurity ()

  Used for compatibility with Unity networking only. Encryption is automatically initialized while connecting.
- static void Disconnect ()

  Makes this client disconnect from the photon server, a process that leaves any room and calls OnDisconnected↩ FromPhoton on completion.
- static bool FindFriends (string[] friendsToFind)

  Requests the rooms and online status for a list of friends and saves the result in PhotonNetwork.Friends.
- static bool CreateRoom (string roomName)

  Creates a room with given name but fails if this room(name) is existing already. Creates random name for roomName null.
- static bool CreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby)

  Creates a room but fails if this room is existing already. Can only be called on Master Server.
- static bool CreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby, string[] expectedUsers)

  Creates a room but fails if this room is existing already. Can only be called on Master Server.
- static bool JoinRoom (string roomName)

  Join room by roomname and on success calls OnJoinedRoom(). This is not affected by lobbies.
- static bool JoinRoom (string roomName, string[] expectedUsers)

  Join room by roomname and on success calls OnJoinedRoom(). This is not affected by lobbies.
- static bool JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby)

  Lets you either join a named room or create it on the fly - you don't have to know if someone created the room already.
- static bool JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby, string[] expectedUsers)

  Lets you either join a named room or create it on the fly - you don't have to know if someone created the room already.
- static bool JoinRandomRoom ()

  Joins any available room of the currently used lobby and fails if none is available.
- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)

  Attempts to join an open room with fitting, custom properties but fails if none is currently available.

- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType, TypedLobby typedLobby, string sqlLobbyFilter, string[] expectedUsers=null)

  Attempts to join an open room with fitting, custom properties but fails if none is currently available.
- static bool ReJoinRoom (string roomName)

  Can be used to return to a room after a disconnect and reconnect.
- static bool JoinLobby ()

  On MasterServer this joins the default lobby which list rooms currently in use.
- static bool JoinLobby (TypedLobby typedLobby)

  On a Master Server you can join a lobby to get lists of available rooms.
- static bool LeaveLobby ()

  Leave a lobby to stop getting updates about available rooms.
- static bool LeaveRoom ()

  Leave the current room and return to the Master Server where you can join or create rooms (see remarks).
- static RoomInfo[] GetRoomList ()

  Gets currently known rooms as RoomInfo array. This is available and updated while in a lobby (check insideLobby).
- static void SetPlayerCustomProperties (Hashtable customProperties)

  Sets this (local) player's properties and synchronizes them to the other players (don't modify them directly).
- static void RemovePlayerCustomProperties (string[] customPropertiesToDelete)

  Locally removes Custom Properties of "this" player. Important: This does not synchronize the change! Useful when you switch rooms.
- static bool RaiseEvent (byte eventCode, object eventContent, bool sendReliable, RaiseEventOptions options)

  Sends fully customizable events in a room. Events consist of at least an EventCode (0..199) and can have content.
- static int AllocateViewID ()

  Allocates a viewID that's valid for the current/local player.
- static int AllocateSceneViewID ()

  Enables the Master Client to allocate a viewID that is valid for scene objects.
- static void UnAllocateViewID (int viewID)

  Unregister a viewID (of manually instantiated and destroyed networked objects).
- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group)

  Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.
- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)

  Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.
- static GameObject InstantiateSceneObject (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)

  Instantiate a scene-owned prefab over the network. The PhotonViews will be controllable by the MasterClient. This prefab needs to be located in the root of a "Resources" folder.
- static int GetPing ()

  The current roundtrip time to the photon server.
- static void FetchServerTimestamp ()

  Refreshes the server timestamp (async operation, takes a roundtrip).
- static void SendOutgoingCommands ()

  Can be used to immediately send the RPCs and Instantiates just called, so they are on their way to the other players.
- static bool CloseConnection (PhotonPlayer kickPlayer)

  Request a client to disconnect (KICK). Only the master client can do this
- static bool SetMasterClient (PhotonPlayer masterClientPlayer)

  Asks the server to assign another player as Master Client of your current room.
- static void Destroy (PhotonView targetView)

  Network-Destroy the GameObject associated with the PhotonView, unless the PhotonView is static or not under this client's control.
- static void Destroy (GameObject targetGo)

Network-Destroy the GameObject, unless it is static or not under this client's control.

- static void DestroyPlayerObjects (PhotonPlayer targetPlayer)

  Network-Destroy all GameObjects, PhotonViews and their RPCs of targetPlayer. Can only be called on local player (for "self") or Master Client (for anyone).
- static void DestroyPlayerObjects (int targetPlayerId)

  Network-Destroy all GameObjects, PhotonViews and their RPCs of this player (by ID). Can only be called on local player (for "self") or Master Client (for anyone).
- static void DestroyAll ()

  Network-Destroy all GameObjects, PhotonViews and their RPCs in the room. Removes anything buffered from the server. Can only be called by Master Client (for anyone).
- static void RemoveRPCs (PhotonPlayer targetPlayer)

  Remove all buffered RPCs from server that were sent by targetPlayer. Can only be called on local player (for "self") or Master Client (for anyone).
- static void RemoveRPCs (PhotonView targetPhotonView)

  Remove all buffered RPCs from server that were sent via targetPhotonView. The Master Client and the owner of the targetPhotonView may call this.
- static void RemoveRPCsInGroup (int targetGroup)

  Remove all buffered RPCs from server that were sent in the targetGroup, if this is the Master Client or if this controls the individual PhotonView.
- static void CacheSendMonoMessageTargets (Type type)

  Populates SendMonoMessageTargets with currently existing GameObjects that have a Component of type.
- static HashSet< GameObject > FindGameObjectsWithComponent (Type type)

  Finds the GameObjects with Components of a specific type (using FindObjectsOfType).
- static void SetReceivingEnabled (int group, bool enabled)

  Enable/disable receiving on given group (applied to PhotonViews)
- static void SetReceivingEnabled (int[] enableGroups, int[] disableGroups)

  Enable/disable receiving on given groups (applied to PhotonViews)
- static void SetSendingEnabled (int group, bool enabled)

  Enable/disable sending on given group (applied to PhotonViews)
- static void SetSendingEnabled (int[] enableGroups, int[] disableGroups)

  Enable/disable sending on given groups (applied to PhotonViews)
- static void SetLevelPrefix (short prefix)

  Sets level prefix for PhotonViews instantiated later on. Don't set it if you need only one!
- static void LoadLevel (int levelNumber)

  Wraps loading a level to pause the network mesage-queue. Optionally syncs the loaded level in a room.
- static void LoadLevel (string levelName)

  Wraps loading a level to pause the network mesage-queue. Optionally syncs the loaded level in a room.
- static bool WebRpc (string name, object parameters)

  This operation makes Photon call your custom web-service by name (path) with the given parameters.

Public Attributes

- const string versionPUN = "1.72"

  Version number of PUN. Also used in GameVersion to separate client version from each other.

Static Public Attributes

- static readonly int MAX_VIEW_IDS = 1000

  The maximum number of assigned PhotonViews per player (or scene). See the General Documentation topic "Limitations" on how to raise this limitation.
- static ServerSettings PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.serverSettingsAssetFile, typeof(ServerSettings))

Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.

- static bool InstantiateInRoomOnly = true

  If true, Instantiate methods will check if you are in a room and fail if you are not.

- static PhotonLogLevel logLevel = PhotonLogLevel.ErrorsOnly

  Network log level. Controls how verbose PUN is.

- static float precisionForVectorSynchronization = 0.000099f

  The minimum difference that a Vector2 or Vector3(e.g. a transforms rotation) needs to change before we send it via a PhotonView's OnSerialize/ObservingComponent.

- static float precisionForQuaternionSynchronization = 1.0f

  The minimum angle that a rotation needs to change before we send it via a PhotonView's OnSerialize/Observing↩ Component.

- static float precisionForFloatSynchronization = 0.01f

  The minimum difference between floats before we send it via a PhotonView's OnSerialize/ObservingComponent.

- static bool UseRpcMonoBehaviourCache

  While enabled, the MonoBehaviours on which we call RPCs are cached, avoiding costly GetComponents<Mono↩ Behaviour>() calls.

- static bool UsePrefabCache = true

  While enabled (true), Instantiate uses PhotonNetwork.PrefabCache to keep game objects in memory (improving instantiation of the same prefab).

- static Dictionary< string, GameObject > PrefabCache = new Dictionary<string, GameObject>()

  Keeps references to GameObjects for frequent instantiation (out of memory instead of loading the Resources).

- static HashSet< GameObject > SendMonoMessageTargets

  If not null, this is the (exclusive) list of GameObjects that get called by PUN SendMonoMessage().

- static Type SendMonoMessageTargetType = typeof(MonoBehaviour)

  Defines which classes can contain PUN Callback implementations.

- static bool StartRpcsAsCoroutine = true

  Can be used to skip starting RPCs as Coroutine, which can be a performance issue.

- static int maxConnections

  Only used in Unity Networking. In PUN, set the number of players in PhotonNetwork.CreateRoom.

- static float BackgroundTimeout = 60.0f

  Defines how long PUN keeps running a "fallback thread" to keep the connection after Unity's OnApplication↩ Pause(true) call.

- static EventCallback OnEventCall

  Register your RaiseEvent handling methods here by using "+=".

Properties

- static string gameVersion `[get, set]`

  Version string for your this build. Can be used to separate incompatible clients. Sent during connect.

- static string ServerAddress `[get]`

  Currently used server address (no matter if master or game server).

- static bool connected `[get]`

  False until you connected to Photon initially. True in offline mode, while connected to any server and even while switching servers.

- static bool connecting `[get]`

  True when you called ConnectUsingSettings (or similar) until the low level connection to Photon gets established.

- static bool connectedAndReady `[get]`

  A refined version of connected which is true only if your connection to the server is ready to accept operations like join, leave, etc.

- static ConnectionState connectionState `[get]`

  Simplified connection state

- static PeerState connectionStateDetailed [get]

  Detailed connection state (ignorant of PUN, so it can be "disconnected" while switching servers).
- static ServerConnection Server [get]

  The server (type) this client is currently connected or connecting to.
- static AuthenticationValues AuthValues [get, set]

  A user's authentication values used during connect for Custom Authentication with Photon (and a custom service/community). Set these before calling Connect if you want custom authentication.
- static Room room [get]

  Get the room we're currently in. Null if we aren't in any room.
- static PhotonPlayer player [get]

  The local PhotonPlayer. Always available and represents this player. CustomProperties can be set before entering a room and will be synced as well.
- static PhotonPlayer masterClient [get]

  The Master Client of the current room or null (outside of rooms).
- static string playerName [get, set]

  Set to synchronize the player's nickname with everyone in the room(s) you enter. This sets PhotonPlayer.name.
- static PhotonPlayer[] playerList [get]

  The list of players in the current room, including the local player.
- static PhotonPlayer[] otherPlayers [get]

  The list of players in the current room, excluding the local player.
- static List< FriendInfo > Friends [get, set]

  Read-only list of friends, their online status and the room they are in. Null until initialized by a FindFriends call.
- static int FriendsListAge [get]

  Age of friend list info (in milliseconds). It's 0 until a friend list is fetched.
- static IPunPrefabPool PrefabPool [get, set]

  An Object Pool can be used to keep and reuse instantiated object instances. It replaced Unity's default Instantiate and Destroy methods.
- static bool offlineMode [get, set]

  Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on PhotonNetwork will not create any connections and there is near to no overhead. Mostly usefull for reusing RPC's and PhotonNetwork.Instantiate
- static bool automaticallySyncScene [get, set]

  Defines if all clients in a room should load the same level as the Master Client (if that used PhotonNetwork.LoadLevel).
- static bool autoCleanUpPlayerObjects [get, set]

  This setting defines per room, if network-instantiated GameObjects (with PhotonView) get cleaned up when the creator of it leaves.
- static bool autoJoinLobby [get, set]

  Set in PhotonServerSettings asset. Defines if the PhotonNetwork should join the "lobby" when connected to the Master server.
- static bool EnableLobbyStatistics [get, set]

  Set in PhotonServerSettings asset. Enable to get a list of active lobbies from the Master Server.
- static List< TypedLobbyInfo > LobbyStatistics [get, set]

  If turned on, the Master Server will provide information about active lobbies for this application.
- static bool insideLobby [get]

  True while this client is in a lobby.
- static TypedLobby lobby [get, set]

  The lobby that will be used when PUN joins a lobby or creates a game.
- static int sendRate [get, set]

  Defines how many times per second PhotonNetwork should send a package. If you change this, do not forget to also change 'sendRateOnSerialize'.
- static int sendRateOnSerialize [get, set]

  Defines how many times per second OnPhotonSerialize should be called on PhotonViews.

- static bool isMessageQueueRunning [get, set]

  Can be used to pause dispatching of incoming evtents (RPCs, Instantiates and anything else incoming).
- static int unreliableCommandsLimit [get, set]

  Used once per dispatch to limit unreliable commands per channel (so after a pause, many channels can still cause a lot of unreliable commands)
- static double time [get]

  Photon network time, synched with the server.
- static int ServerTimestamp [get]

  The current server's millisecond timestamp.
- static bool isMasterClient [get]

  Are we the master client?
- static bool inRoom [get]

  Is true while being in a room (connectionStateDetailed == PeerState.Joined).
- static bool isNonMasterClientInRoom [get]

  True if we are in a room (client) and NOT the room's masterclient
- static int countOfPlayersOnMaster [get]

  The count of players currently looking for a room (available on MasterServer in 5sec intervals).
- static int countOfPlayersInRooms [get]

  Count of users currently playing your app in some room (sent every 5sec by Master Server). Use playerList.Count to get the count of players in the room you're in!
- static int countOfPlayers [get]

  The count of players currently using this application (available on MasterServer in 5sec intervals).
- static int countOfRooms [get]

  The count of rooms currently in use (available on MasterServer in 5sec intervals).
- static bool NetworkStatisticsEnabled [get, set]

  Enables or disables the collection of statistics about this client's traffic.
- static int ResentReliableCommands [get]

  Count of commands that got repeated (due to local repeat-timing before an ACK was received).
- static bool CrcCheckEnabled [get, set]

  Crc checks can be useful to detect and avoid issues with broken datagrams. Can be enabled while not connected.
- static int PacketLossByCrcCheck [get]

  If CrcCheckEnabled, this counts the incoming packages that don't have a valid CRC checksum and got rejected.
- static int MaxResendsBeforeDisconnect [get, set]

  Defines the number of times a reliable message can be resent before not getting an ACK for it will trigger a disconnect. Default: 5.
- static int QuickResends [get, set]

  In case of network loss, reliable messages can be repeated quickly up to 3 times.

8.20.1 Detailed Description

The main class to use the PhotonNetwork plugin. This class is static.

8.20.2 Member Function Documentation

8.20.2.1 static int PhotonNetwork.AllocateSceneViewID ( ) [static]

Enables the Master Client to allocate a viewID that is valid for scene objects.

Returns

A viewID that can be used for a new PhotonView or -1 in case of an error.

8.20.2.2 static int PhotonNetwork.AllocateViewID ( ) [static]

Allocates a viewID that's valid for the current/local player.

Returns

A viewID that can be used for a new PhotonView.

8.20.2.3 static void PhotonNetwork.CacheSendMonoMessageTargets (Type *type*) [static]

Populates SendMonoMessageTargets with currently existing GameObjects that have a Component of type.

Parameters

type	If null, this will use SendMonoMessageTargets as component-type (MonoBehaviour by default).

8.20.2.4 static bool PhotonNetwork.CloseConnection (PhotonPlayer *kickPlayer*) [static]

Request a client to disconnect (KICK). Only the master client can do this

Only the target player gets this event. That player will disconnect automatically, which is what the others will notice, too.

Parameters

kickPlayer	The PhotonPlayer to kick.

8.20.2.5 static bool PhotonNetwork.ConnectToBestCloudServer (string *gameVersion*) [static]

Connect to the Photon Cloud region with the lowest ping (on platforms that support Unity's Ping).

Will save the result of pinging all cloud servers in PlayerPrefs. Calling this the first time can take +-2 seconds. The ping result can be overridden via PhotonNetwork.OverrideBestCloudServer(..) This call can take up to 2 seconds if it is the first time you are using this, all cloud servers will be pinged to check for the best region.

The PUN Setup Wizard stores your appID in a settings file and applies a server address/port. To connect to the Photon Cloud, a valid AppId must be in the settings file (shown in the Photon Cloud Dashboard). https://www.photonengine.com/dashboard

Connecting to the Photon Cloud might fail due to:

- Invalid AppId (calls: OnFailedToConnectToPhoton(). check exact AppId value)
- Network issues (calls: OnFailedToConnectToPhoton())
- Invalid region (calls: OnConnectionFail() with DisconnectCause.InvalidRegion)
- Subscription CCU limit reached (calls: OnConnectionFail() with DisconnectCause.MaxCcuReached. also calls: OnPhotonMaxCccuReached())

More about the connection limitations: http://doc.exitgames.com/en/pun

Parameters

gameVersion	This client's version number. Users are separated from each other by gameversion (which allows you to make breaking changes).

Returns

If this client is going to connect to cloud server based on ping. Even if true, this does not guarantee a connection but the attempt is being made.

8.20.2.6 static bool PhotonNetwork.ConnectToMaster (string *masterServerAddress,* int *port,* string *appID,* string *gameVersion*) [static]

Connect to a Photon Master Server by address, port, appID and game(client) version.

To connect to the Photon Cloud, a valid AppId must be in the settings file (shown in the Photon Cloud Dashboard). https://www.photonengine.com/dashboard

Connecting to the Photon Cloud might fail due to:

- Invalid AppId (calls: OnFailedToConnectToPhoton(). check exact AppId value)
- Network issues (calls: OnFailedToConnectToPhoton())
- Invalid region (calls: OnConnectionFail() with DisconnectCause.InvalidRegion)
- Subscription CCU limit reached (calls: OnConnectionFail() with DisconnectCause.MaxCcuReached. also calls: OnPhotonMaxCccuReached())

More about the connection limitations: http://doc.exitgames.com/en/pun

Parameters

masterServer↩ Address	The server's address (either your own or Photon Cloud address).
port	The server's port to connect to.
appID	Your application ID (Photon Cloud provides you with a GUID for your game).
gameVersion	This client's version number. Users are separated by gameversion (which allows you to make breaking changes).

8.20.2.7 static bool PhotonNetwork.ConnectToRegion (CloudRegionCode *region,* string *gameVersion*) [static]

Connects to the Photon Cloud region of choice.

8.20.2.8 static bool PhotonNetwork.ConnectUsingSettings (string *gameVersion*) [static]

Connect to Photon as configured in the editor (saved in PhotonServerSettings file).

This method will disable offlineMode (which won't destroy any instantiated GOs) and it will set isMessageQueue↩ Running to true.

Your server configuration is created by the PUN Wizard and contains the AppId and region for Photon Cloud games and the server address if you host Photon yourself. These settings usually don't change often.

To ignore the config file and connect anywhere call: PhotonNetwork.ConnectToMaster.

To connect to the Photon Cloud, a valid AppId must be in the settings file (shown in the Photon Cloud Dashboard). https://www.photonengine.com/dashboard

Connecting to the Photon Cloud might fail due to:

- Invalid AppId (calls: OnFailedToConnectToPhoton(). check exact AppId value)
- Network issues (calls: OnFailedToConnectToPhoton())
- Invalid region (calls: OnConnectionFail() with DisconnectCause.InvalidRegion)
- Subscription CCU limit reached (calls: OnConnectionFail() with DisconnectCause.MaxCcuReached. also calls: OnPhotonMaxCccuReached())

More about the connection limitations: http://doc.exitgames.com/en/pun

Parameters

gameVersion	This client's version number. Users are separated from each other by gameversion (which allows you to make breaking changes).

8.20.2.9 static bool PhotonNetwork.CreateRoom (string *roomName*) [static]

Creates a room with given name but fails if this room(name) is existing already. Creates random name for room↩ Name null.

If you don't want to create a unique room-name, pass null or "" as name and the server will assign a roomName (a GUID as string).

The created room is automatically placed in the currently used lobby (if any) or the default-lobby if you didn't explicitly join one.

Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName.

PhotonNetwork.autoCleanUpPlayerObjects will become this room's AutoCleanUp property and that's used by all clients that join this room.

Parameters

roomName	Unique name of the room to create.

Returns

If the operation got queued and will be sent.

8.20.2.10 static bool PhotonNetwork.CreateRoom (string *roomName,* RoomOptions *roomOptions,* TypedLobby *typedLobby*) [static]

Creates a room but fails if this room is existing already. Can only be called on Master Server.

When successful, this calls the callbacks OnCreatedRoom and OnJoinedRoom (the latter, cause you join as first player). If the room can't be created (because it exists already), OnPhotonCreateRoomFailed gets called.

If you don't want to create a unique room-name, pass null or "" as name and the server will assign a roomName (a GUID as string).

Rooms can be created in any number of lobbies. Those don't have to exist before you create a room in them (they get auto-created on demand). Lobbies can be useful to split room lists on the server-side already. That can help keep the room lists short and manageable. If you set a typedLobby parameter, the room will be created in that lobby (no matter if you are active in any). If you don't set a typedLobby, the room is automatically placed in the currently active lobby (if any) or the default-lobby.

Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName.

PhotonNetwork.autoCleanUpPlayerObjects will become this room's autoCleanUp property and that's used by all clients that join this room.

Parameters

roomName	Unique name of the room to create. Pass null or "" to make the server generate a name.
roomOptions	Common options for the room like maxPlayers, initial custom room properties and similar. See RoomOptions type..

typedLobby	If null, the room is automatically created in the currently used lobby (which is "default" when you didn't join one explicitly).

Returns

If the operation got queued and will be sent.

8.20.2.11 static bool PhotonNetwork.CreateRoom (string *roomName,* RoomOptions *roomOptions,* TypedLobby *typedLobby,* string[] *expectedUsers*) [static]

Creates a room but fails if this room is existing already. Can only be called on Master Server.

When successful, this calls the callbacks OnCreatedRoom and OnJoinedRoom (the latter, cause you join as first player). If the room can't be created (because it exists already), OnPhotonCreateRoomFailed gets called.

If you don't want to create a unique room-name, pass null or "" as name and the server will assign a roomName (a GUID as string).

Rooms can be created in any number of lobbies. Those don't have to exist before you create a room in them (they get auto-created on demand). Lobbies can be useful to split room lists on the server-side already. That can help keep the room lists short and manageable. If you set a typedLobby parameter, the room will be created in that lobby (no matter if you are active in any). If you don't set a typedLobby, the room is automatically placed in the currently active lobby (if any) or the default-lobby.

Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName.

PhotonNetwork.autoCleanUpPlayerObjects will become this room's autoCleanUp property and that's used by all clients that join this room.

You can define an array of expectedUsers, to block player slots in the room for these users. The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages.

Parameters

roomName	Unique name of the room to create. Pass null or "" to make the server generate a name.
roomOptions	Common options for the room like maxPlayers, initial custom room properties and similar. See RoomOptions type..
typedLobby	If null, the room is automatically created in the currently used lobby (which is "default" when you didn't join one explicitly).
expectedUsers	Optional list of users (by UserId) who are expected to join this game and who you want to block a slot for.

Returns

If the operation got queued and will be sent.

8.20.2.12 static void PhotonNetwork.Destroy (PhotonView *targetView*) [static]

Network-Destroy the GameObject associated with the PhotonView, unless the PhotonView is static or not under this client's control.

Destroying a networked GameObject while in a Room includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the PhotonNetwork.Instantiate call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Usually, when you leave a room, the GOs get destroyed automatically. If you have to destroy a GO while not in a room, the Destroy is only done locally.

Destroying networked objects works only if they got created with PhotonNetwork.Instantiate(). Objects loaded with a scene are ignored, no matter if they have PhotonView components.

The GameObject must be under this client's control:

- Instantiated and owned by this client.
- Instantiated objects of players who left the room are controlled by the Master Client.
- Scene-owned game objects are controlled by the Master Client.
- GameObject can be destroyed while client is not in a room.

Returns

Nothing. Check error debug log for any issues.

8.20.2.13 static void PhotonNetwork.Destroy (GameObject *targetGo*) [static]

Network-Destroy the GameObject, unless it is static or not under this client's control.

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the PhotonNetwork.Instantiate call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Usually, when you leave a room, the GOs get destroyed automatically. If you have to destroy a GO while not in a room, the Destroy is only done locally.

Destroying networked objects works only if they got created with PhotonNetwork.Instantiate(). Objects loaded with a scene are ignored, no matter if they have PhotonView components.

The GameObject must be under this client's control:

- Instantiated and owned by this client.
- Instantiated objects of players who left the room are controlled by the Master Client.
- Scene-owned game objects are controlled by the Master Client.
- GameObject can be destroyed while client is not in a room.

Returns

Nothing. Check error debug log for any issues.

8.20.2.14 static void PhotonNetwork.DestroyAll ( ) [static]

Network-Destroy all GameObjects, PhotonViews and their RPCs in the room. Removes anything buffered from the server. Can only be called by Master Client (for anyone).

Can only be called by Master Client (for anyone). Unlike the Destroy methods, this will remove anything from the server's room buffer. If your game buffers anything beyond Instantiate and RPC calls, that will be cleaned as well from server.

Destroying all includes:

- Remove anything from the server's room buffer (Instantiate, RPCs, anything buffered).
- Sending a message to other clients to destroy everything locally, too (affected by network lag).

Destroying networked objects works only if they got created with PhotonNetwork.Instantiate(). Objects loaded with a scene are ignored, no matter if they have PhotonView components.

Returns

Nothing. Check error debug log for any issues.

8.20.2.15 static void PhotonNetwork.DestroyPlayerObjects (PhotonPlayer *targetPlayer*) [static]

Network-Destroy all GameObjects, PhotonViews and their RPCs of targetPlayer. Can only be called on local player (for "self") or Master Client (for anyone).

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the PhotonNetwork.Instantiate call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Destroying networked objects works only if they got created with PhotonNetwork.Instantiate(). Objects loaded with a scene are ignored, no matter if they have PhotonView components.

Returns

Nothing. Check error debug log for any issues.

8.20.2.16 static void PhotonNetwork.DestroyPlayerObjects (int *targetPlayerId*) [static]

Network-Destroy all GameObjects, PhotonViews and their RPCs of this player (by ID). Can only be called on local player (for "self") or Master Client (for anyone).

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the PhotonNetwork.Instantiate call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Destroying networked objects works only if they got created with PhotonNetwork.Instantiate(). Objects loaded with a scene are ignored, no matter if they have PhotonView components.

Returns

Nothing. Check error debug log for any issues.

8.20.2.17 static void PhotonNetwork.Disconnect ( ) [static]

Makes this client disconnect from the photon server, a process that leaves any room and calls OnDisconnected↩ FromPhoton on completion.

When you disconnect, the client will send a "disconnecting" message to the server. This speeds up leave/disconnect messages for players in the same room as you (otherwise the server would timeout this client's connection). When used in offlineMode, the state-change and event-call OnDisconnectedFromPhoton are immediate. Offline mode is set to false as well. Once disconnected, the client can connect again. Use ConnectUsingSettings.

8.20.2.18 delegate void PhotonNetwork.EventCallback (byte *eventCode,* object *content,* int *senderId*)

Defines the delegate usable in OnEventCall.

Any eventCode < 200 will be forwarded to your delegate(s).

Parameters

eventCode	The code assigend to the incoming event.
content	The content the sender put into the event.
senderId	The ID of the player who sent the event. It might be 0, if the "room" sent the event.

8.20.2.19 static void PhotonNetwork.FetchServerTimestamp ( ) [static]

Refreshes the server timestamp (async operation, takes a roundtrip).

Can be useful if a bad connection made the timestamp unusable or imprecise.

8.20.2.20 static bool PhotonNetwork.FindFriends (string[] *friendsToFind*) [static]

Requests the rooms and online status for a list of friends and saves the result in PhotonNetwork.Friends.

Works only on Master Server to find the rooms played by a selected list of users.

The result will be stored in PhotonNetwork.Friends when available. That list is initialized on first use of OpFindFriends (before that, it is null). To refresh the list, call FindFriends again (in 5 seconds or 10 or 20).

Users identify themselves by setting a unique username via PhotonNetwork.playerName or by PhotonNetwork.AuthValues. The user id set in AuthValues overrides the playerName, so make sure you know the ID your friends use to authenticate. The AuthValues are sent in OpAuthenticate when you connect, so the AuthValues must be set before you connect!

Note: Changing a player's name doesn't make sense when using a friend list.

The list of friends must be fetched from some other source (not provided by Photon).

Internal: The server response includes 2 arrays of info (each index matching a friend from the request): ParameterCode.FindFriendsResponseOnlineList = bool[] of online states ParameterCode.FindFriendsResponseRoomIdList = string[] of room names (empty string if not in a room)

Parameters

friendsToFind	Array of friend (make sure to use unique playerName or AuthValues).

Returns

If the operation could be sent (requires connection, only one request is allowed at any time). Always false in offline mode.

8.20.2.21 static HashSet<GameObject> PhotonNetwork.FindGameObjectsWithComponent (Type *type*) [static]

Finds the GameObjects with Components of a specific type (using FindObjectsOfType).

Parameters

type	Type must be a Component

Returns

HashSet with GameObjects that have a specific type of Component.

8.20.2.22 static int PhotonNetwork.GetPing ( ) [static]

The current roundtrip time to the photon server.

Returns

Roundtrip time (to server and back).

8.20.2.23 static RoomInfo [] PhotonNetwork.GetRoomList ( ) [static]

Gets currently known rooms as RoomInfo array. This is available and updated while in a lobby (check insideLobby).

This list is a cached copy of the internal rooms list so it can be accessed each frame if needed. Per RoomInfo you can check if the room is full by comparing playerCount and maxPlayers before you allow a join.

The name of a room must be used to join it (via JoinRoom).

Closed rooms are also listed by lobbies but they can't be joined. While in a room, any player can set Room.visible and Room.open to hide rooms from matchmaking and close them.

Returns

RoomInfo[] of current rooms in lobby.

8.20.2.24 static void PhotonNetwork.InitializeSecurity ( ) [static]

Used for compatibility with Unity networking only. Encryption is automatically initialized while connecting.

8.20.2.25 static GameObject PhotonNetwork.Instantiate (string *prefabName*, Vector3 *position*, Quaternion *rotation*, int *group*) [static]

Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.

Instead of using prefabs in the Resources folder, you can manually Instantiate and assign PhotonViews. See doc.

Parameters

prefabName	Name of the prefab to instantiate.
position	Position Vector3 to apply on instantiation.
rotation	Rotation Quaternion to apply on instantiation.
group	The group for this PhotonView.

Returns

The new instance of a GameObject with initialized PhotonView.

8.20.2.26 static GameObject PhotonNetwork.Instantiate (string *prefabName*, Vector3 *position*, Quaternion *rotation*, int *group*, object[] *data*) [static]

Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.

Instead of using prefabs in the Resources folder, you can manually Instantiate and assign PhotonViews. See doc.

Parameters

prefabName	Name of the prefab to instantiate.
position	Position Vector3 to apply on instantiation.
rotation	Rotation Quaternion to apply on instantiation.
group	The group for this PhotonView.
data	Optional instantiation data. This will be saved to it's PhotonView.instantiationData.

Returns

The new instance of a GameObject with initialized PhotonView.

8.20.2.27 static GameObject PhotonNetwork.InstantiateSceneObject (string *prefabName*, Vector3 *position*, Quaternion *rotation*, int *group*, object[] *data*) [static]

Instantiate a scene-owned prefab over the network. The PhotonViews will be controllable by the MasterClient. This prefab needs to be located in the root of a "Resources" folder.

Only the master client can Instantiate scene objects. Instead of using prefabs in the Resources folder, you can manually Instantiate and assign PhotonViews. See doc.

Parameters

prefabName	Name of the prefab to instantiate.
position	Position Vector3 to apply on instantiation.
rotation	Rotation Quaternion to apply on instantiation.
group	The group for this PhotonView.
data	Optional instantiation data. This will be saved to it's PhotonView.instantiationData.

Returns

The new instance of a GameObject with initialized PhotonView.

8.20.2.28 static bool PhotonNetwork.JoinLobby ( ) [static]

On MasterServer this joins the default lobby which list rooms currently in use.

The room list is sent and refreshed by the server. You can access this cached list by PhotonNetwork.GetRoomList().

Per room you should check if it's full or not before joining. Photon also lists rooms that are full, unless you close and hide them (room.open = false and room.visible = false).

In best case, you make your clients join random games, as described here: http://doc.exitgames.com/en/realtime/current/reference/matchmaking-and-lobby

You can show your current players and room count without joining a lobby (but you must be on the master server). Use: countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms and countOfRooms.

You can use more than one lobby to keep the room lists shorter. See JoinLobby(TypedLobby lobby). When creating new rooms, they will be "attached" to the currently used lobby or the default lobby.

You can use JoinRandomRoom without being in a lobby! Set autoJoinLobby = false before you connect, to not join a lobby. In that case, the connect-workflow will call OnConnectedToMaster (if you implement it) when it's done.

8.20.2.29 static bool PhotonNetwork.JoinLobby (TypedLobby *typedLobby*) [static]

On a Master Server you can join a lobby to get lists of available rooms.

The room list is sent and refreshed by the server. You can access this cached list by PhotonNetwork.GetRoomList().

Any client can "make up" any lobby on the fly. Splitting rooms into multiple lobbies will keep each list shorter. However, having too many lists might ruin the matchmaking experience.

In best case, you create a limited number of lobbies. For example, create a lobby per game-mode: "koth" for king of the hill and "ffa" for free for all, etc.

There is no listing of lobbies at the moment.

Sql-typed lobbies offer a different filtering model for random matchmaking. This might be more suited for skillbased-games. However, you will also need to follow the conventions for naming filterable properties in sql-lobbies! Both is explained in the matchmaking doc linked below.

In best case, you make your clients join random games, as described here: http://confluence.↩ exitgames.com/display/PTN/Op+JoinRandomGame

Per room you should check if it's full or not before joining. Photon does list rooms that are full, unless you close and hide them (room.open = false and room.visible = false).

You can show your games current players and room count without joining a lobby (but you must be on the master server). Use: countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms and countOfRooms.

When creating new rooms, they will be "attached" to the currently used lobby or the default lobby.

You can use JoinRandomRoom without being in a lobby! Set autoJoinLobby = false before you connect, to not join a lobby. In that case, the connect-workflow will call OnConnectedToMaster (if you implement it) when it's done.

Parameters

typedLobby	A typed lobby to join (must have name and type).

8.20.2.30 static bool PhotonNetwork.JoinOrCreateRoom (string *roomName,* RoomOptions *roomOptions,* TypedLobby *typedLobby*) [static]

Lets you either join a named room or create it on the fly - you don't have to know if someone created the room already.

This makes it easier for groups of players to get into the same room. Once the group exchanged a roomName, any player can call JoinOrCreateRoom and it doesn't matter who actually joins or creates the room.

The parameters roomOptions and typedLobby are only used when the room actually gets created by this client. You know if this client created a room, if you get a callback OnCreatedRoom (before OnJoinedRoom gets called as well).

Parameters

roomName	Name of the room to join. Must be non null.
roomOptions	Options for the room, in case it does not exist yet. Else these values are ignored.
typedLobby	Lobby you want a new room to be listed in. Ignored if the room was existing and got joined.

Returns

If the operation got queued and will be sent.

8.20.2.31 static bool PhotonNetwork.JoinOrCreateRoom (string *roomName,* RoomOptions *roomOptions,* TypedLobby *typedLobby,* string[] *expectedUsers*) [static]

Lets you either join a named room or create it on the fly - you don't have to know if someone created the room already.

This makes it easier for groups of players to get into the same room. Once the group exchanged a roomName, any player can call JoinOrCreateRoom and it doesn't matter who actually joins or creates the room.

The parameters roomOptions and typedLobby are only used when the room actually gets created by this client. You know if this client created a room, if you get a callback OnCreatedRoom (before OnJoinedRoom gets called as

well).

You can define an array of expectedUsers, to block player slots in the room for these users. The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages.

Parameters

roomName	Name of the room to join. Must be non null.
roomOptions	Options for the room, in case it does not exist yet. Else these values are ignored.
typedLobby	Lobby you want a new room to be listed in. Ignored if the room was existing and got joined.
expectedUsers	Optional list of users (by UserId) who are expected to join this game and who you want to block a slot for.

Returns

If the operation got queued and will be sent.

8.20.2.32 static bool PhotonNetwork.JoinRandomRoom ( ) [static]

Joins any available room of the currently used lobby and fails if none is available.

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby. Use the JoinRandomRoom overload with TypedLobby parameter.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the currently active lobby or (if no lobby is joined) in the default lobby.

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

8.20.2.33 static bool PhotonNetwork.JoinRandomRoom (Hashtable *expectedCustomRoomProperties*, byte *expectedMaxPlayers*) [static]

Attempts to join an open room with fitting, custom properties but fails if none is currently available.

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby. Use the JoinRandomRoom overload with TypedLobby parameter.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the currently active lobby or (if no lobby is joined) in the default lobby.

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

Parameters

expectedCustomRoomProperties	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
expectedMaxPlayers	Filters for a particular maxplayer setting. Use 0 to accept any maxPlayer value.

Returns

If the operation got queued and will be sent.

8.20.2.34 static bool PhotonNetwork.JoinRandomRoom (Hashtable *expectedCustomRoomProperties,* byte *expectedMaxPlayers,* MatchmakingMode *matchingType,* TypedLobby *typedLobby,* string *sqlLobbyFilter,* string[] *expectedUsers* = `null`) `[static]`

Attempts to join an open room with fitting, custom properties but fails if none is currently available.

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby with this overload.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the specified lobby or the currently active lobby (if none specified) or in the default lobby (if none active).

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

In offlineMode, a room will be created but no properties will be set and all parameters of this JoinRandomRoom call are ignored. The event/callback OnJoinedRoom gets called (see enum PhotonNetworkingMessage).

You can define an array of expectedUsers, to block player slots in the room for these users. The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages.

Parameters

expectedCustomRoomProperties	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
expectedMaxPlayers	Filters for a particular maxplayer setting. Use 0 to accept any maxPlayer value.
matchingType	Selects one of the available matchmaking algorithms. See MatchmakingMode enum for options.
typedLobby	The lobby in which you want to lookup a room. Pass null, to use the default lobby. This does not join that lobby and neither sets the lobby property.
sqlLobbyFilter	A filter-string for SQL-typed lobbies.
expectedUsers	Optional list of users (by UserId) who are expected to join this game and who you want to block a slot for.

Returns

If the operation got queued and will be sent.

8.20.2.35 static bool PhotonNetwork.JoinRoom (string *roomName*) `[static]`

Join room by roomname and on success calls OnJoinedRoom(). This is not affected by lobbies.

On success, the method OnJoinedRoom() is called on any script. You can implement it to react to joining a room.

JoinRoom fails if the room is either full or no longer available (it might become empty while you attempt to join). Implement OnPhotonJoinRoomFailed() to get a callback in error case.

To join a room from the lobby's listing, use RoomInfo.name as roomName here. Despite using multiple lobbies, a roomName is always "global" for your application and so you don't have to specify which lobby it's in. The Master Server will find the room. In the Photon Cloud, an application is defined by AppId, Game- and PUN-version.

PhotonNetworkingMessage.OnPhotonJoinRoomFailed PhotonNetworkingMessage.OnJoinedRoom

Parameters

roomName	Unique name of the room to join.

Returns

If the operation got queued and will be sent.

8.20.2.36 static bool PhotonNetwork.JoinRoom (string *roomName,* string[] *expectedUsers*) [static]

Join room by roomname and on success calls OnJoinedRoom(). This is not affected by lobbies.

On success, the method OnJoinedRoom() is called on any script. You can implement it to react to joining a room.

JoinRoom fails if the room is either full or no longer available (it might become empty while you attempt to join). Implement OnPhotonJoinRoomFailed() to get a callback in error case.

To join a room from the lobby's listing, use RoomInfo.name as roomName here. Despite using multiple lobbies, a roomName is always "global" for your application and so you don't have to specify which lobby it's in. The Master Server will find the room. In the Photon Cloud, an application is defined by AppId, Game- and PUN-version.

You can define an array of expectedUsers, to block player slots in the room for these users. The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages.

PhotonNetworkingMessage.OnPhotonJoinRoomFailed PhotonNetworkingMessage.OnJoinedRoom

Parameters

roomName	Unique name of the room to join.
expectedUsers	Optional list of users (by UserId) who are expected to join this game and who you want to block a slot for.

Returns

If the operation got queued and will be sent.

8.20.2.37 static bool PhotonNetwork.LeaveLobby ( ) [static]

Leave a lobby to stop getting updates about available rooms.

This does not reset PhotonNetwork.lobby! This allows you to join this particular lobby later easily.

The values countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms and countOfRooms are received even without being in a lobby.

You can use JoinRandomRoom without being in a lobby. Use autoJoinLobby to not join a lobby when you connect.

8.20.2.38 static bool PhotonNetwork.LeaveRoom ( ) [static]

Leave the current room and return to the Master Server where you can join or create rooms (see remarks).

This will clean up all (network) GameObjects with a PhotonView, unless you changed autoCleanUp to false. Returns to the Master Server.

In OfflineMode, the local "fake" room gets cleaned up and OnLeftRoom gets called immediately.

8.20.2.39 static void PhotonNetwork.LoadLevel (int *levelNumber*) [static]

Wraps loading a level to pause the network mesage-queue. Optionally syncs the loaded level in a room.

To sync the loaded level in a room, set PhotonNetwork.automaticallySyncScene to true. The Master Client of a room will then sync the loaded level with every other player in the room.

While loading levels, it makes sense to not dispatch messages received by other players. This method takes care of that by setting PhotonNetwork.isMessageQueueRunning = false and enabling the queue when the level was loaded.

You should make sure you don't fire RPCs before you load another scene (which doesn't contain the same GameObjects and PhotonViews). You can call this in OnJoinedRoom.

This uses Application.LoadLevel.

Parameters

levelNumber	Number of the level to load. When using level numbers, make sure they are identical on all clients.

8.20.2.40 static void PhotonNetwork.LoadLevel (string *levelName*) [static]

Wraps loading a level to pause the network mesage-queue. Optionally syncs the loaded level in a room.

While loading levels, it makes sense to not dispatch messages received by other players. This method takes care of that by setting PhotonNetwork.isMessageQueueRunning = false and enabling the queue when the level was loaded.

To sync the loaded level in a room, set PhotonNetwork.automaticallySyncScene to true. The Master Client of a room will then sync the loaded level with every other player in the room.

You should make sure you don't fire RPCs before you load another scene (which doesn't contain the same GameObjects and PhotonViews). You can call this in OnJoinedRoom.

This uses Application.LoadLevel.

Parameters

levelName	Name of the level to load. Make sure it's available to all clients in the same room.

8.20.2.41 static void PhotonNetwork.NetworkStatisticsReset ( ) [static]

Resets the traffic stats and re-enables them.

8.20.2.42 static string PhotonNetwork.NetworkStatisticsToString ( ) [static]

Only available when NetworkStatisticsEnabled was used to gather some stats.

Returns

A string with vital networking statistics.

8.20.2.43 static void PhotonNetwork.OverrideBestCloudServer (CloudRegionCode *region*) [static]

Overwrites the region that is used for ConnectToBestCloudServer(string gameVersion).

This will overwrite the result of pinging all cloud servers.
Use this to allow your users to save a manually selected region in the player preferences.
Note: You can also use PhotonNetwork.ConnectToRegion to (temporarily) connect to a specific region.

8.20.2.44 static bool PhotonNetwork.RaiseEvent (byte *eventCode*, object *eventContent*, bool *sendReliable*, RaiseEventOptions *options*) [static]

Sends fully customizable events in a room. Events consist of at least an EventCode (0..199) and can have content.

To receive the events someone sends, register your handling method in PhotonNetwork.OnEventCall.

Example: private void OnEventHandler(byte eventCode, object content, int senderId) { Debug.Log("OnEventHandler"); }

PhotonNetwork.OnEventCall += this.OnEventHandler;

With the senderId, you can look up the PhotonPlayer who sent the event. It is best practice to assign a eventCode for each different type of content and action. You have to cast the content.

The eventContent is optional. To be able to send something, it must be a "serializable type", something that the client can turn into a byte[] basically. Most basic types and arrays of them are supported, including Unity's Vector2, Vector3, Quaternion. Transforms or classes some project defines are NOT supported! You can make your own class a "serializable type" by following the example in CustomTypes.cs.

The RaiseEventOptions have some (less intuitive) combination rules: If you set targetActors (an array of PhotonPlayer.ID values), the receivers parameter gets ignored. When using event caching, the targetActors, receivers and interestGroup can't be used. Buffered events go to all. When using cachingOption removeFromRoomCache, the eventCode and content are actually not sent but used as filter.

Parameters

eventCode	A byte identifying the type of event. You might want to use a code per action or to signal which content can be expected. Allowed: 0..199.
eventContent	Some serializable object like string, byte, integer, float (etc) and arrays of those. Hashtables with byte keys are good to send variable content.
sendReliable	Makes sure this event reaches all players. It gets acknowledged, which requires bandwidth and it can't be skipped (might add lag in case of loss).
options	Allows more complex usage of events. If null, RaiseEventOptions.Default will be used (which is fine).

Returns

False if event could not be sent

8.20.2.45 static bool PhotonNetwork.Reconnect ( ) [static]

Can be used to reconnect to the master server after a disconnect.

After losing connection, you can use this to connect a client to the region Master Server again. Cache the room name you're in and use ReJoin(roomname) to return to a game. Common use case: Press the Lock Button on a iOS device and you get disconnected immediately.

8.20.2.46 static bool PhotonNetwork.ReconnectAndRejoin ( ) [static]

When the client lost connection during gameplay, this method attempts to reconnect and rejoin the room.

This method re-connects directly to the game server which was hosting the room PUN was in before. If the room was shut down in the meantime, PUN will call OnPhotonJoinRoomFailed and return this client to the Master Server.

Check the return value, if this client will attempt a reconnect and rejoin (if the conditions are met). If ReconnectAndRejoin returns false, you can still attempt a Reconnect and ReJoin.

Similar to PhotonNetwork.ReJoin, this requires you to use unique IDs per player (the UserID).

Returns

False, if there is no known room or game server to return to. Then, this client does not attempt the ReconnectAndRejoin.

8.20.2.47 static void PhotonNetwork.RefreshCloudServerRating ( ) [static]

Pings all cloud servers again to find the one with best ping (currently).

8.20.2.48 static bool PhotonNetwork.ReJoinRoom (string *roomName*) [static]

Can be used to return to a room after a disconnect and reconnect.

After losing connection, you might be able to return to a room and continue playing, if the client is reconnecting fast enough. Use Reconnect() and this method. Cache the room name you're in and use ReJoin(roomname) to return to a game.

Note: To be able to ReJoin any room, you need to use UserIDs! You also need to set RoomOptions.PlayerTtl.

Important: Instantiate() and use of RPCs is not yet supported. The ownership rules of PhotonViews prevent a seamless return to a game. Use Custom Properties and RaiseEvent with event caching instead.

Common use case: Press the Lock Button on a iOS device and you get disconnected immediately.

8.20.2.49 static void PhotonNetwork.RemovePlayerCustomProperties (string[] *customPropertiesToDelete*) [static]

Locally removes Custom Properties of "this" player. Important: This does not synchronize the change! Useful when you switch rooms.

Use this method with care. It can create inconsistencies of state between players! This only changes the player.↩ customProperties locally. This can be useful to clear your Custom Properties between games (let's say they store which turn you made, kills, etc).

SetPlayerCustomProperties() syncs and can be used to set values to null while in a room. That can be considered "removed" while in a room.

If customPropertiesToDelete is null or has 0 entries, all Custom Properties are deleted (replaced with a new Hashtable). If you specify keys to remove, those will be removed from the Hashtable but other keys are unaffected.

Parameters

custom↩ PropertiesTo↩ Delete	List of Custom Property keys to remove. See remarks.

8.20.2.50 static void PhotonNetwork.RemoveRPCs (PhotonPlayer *targetPlayer*) [static]

Remove all buffered RPCs from server that were sent by targetPlayer. Can only be called on local player (for "self") or Master Client (for anyone).

This method requires either:

- This is the targetPlayer's client.
- This client is the Master Client (can remove any PhotonPlayer's RPCs).

If the targetPlayer calls RPCs at the same time that this is called, network lag will determine if those get buffered or cleared like the rest.

Parameters

targetPlayer	This player's buffered RPCs get removed from server buffer.

8.20.2.51 static void PhotonNetwork.RemoveRPCs (PhotonView *targetPhotonView*) [static]

Remove all buffered RPCs from server that were sent via targetPhotonView. The Master Client and the owner of the targetPhotonView may call this.

This method requires either:

- The targetPhotonView is owned by this client (Instantiated by it).
- This client is the Master Client (can remove any PhotonView's RPCs).

Parameters

targetPhoton↩ View	RPCs buffered for this PhotonView get removed from server buffer.

8.20.2.52 static void PhotonNetwork.RemoveRPCsInGroup (int *targetGroup*) [static]

Remove all buffered RPCs from server that were sent in the targetGroup, if this is the Master Client or if this controls the individual PhotonView.

This method requires either:

- This client is the Master Client (can remove any RPCs per group).
- Any other client: each PhotonView is checked if it is under this client's control. Only those RPCs are removed.

Parameters

targetGroup	Interest group that gets all RPCs removed.

8.20.2.53 static void PhotonNetwork.SendOutgoingCommands ( ) [static]

Can be used to immediately send the RPCs and Instantiates just called, so they are on their way to the other players.

This could be useful if you do a RPC to load a level and then load it yourself. While loading, no RPCs are sent to others, so this would delay the "load" RPC. You can send the RPC to "others", use this method, disable the message queue (by isMessageQueueRunning) and then load.

8.20.2.54 static void PhotonNetwork.SetLevelPrefix (short *prefix*) [static]

Sets level prefix for PhotonViews instantiated later on. Don't set it if you need only one!

Important: If you don't use multiple level prefixes, simply don't set this value. The default value is optimized out of the traffic.

This won't affect existing PhotonViews (they can't be changed yet for existing PhotonViews).

Messages sent with a different level prefix will be received but not executed. This affects RPCs, Instantiates and synchronization.

Be aware that PUN never resets this value, you'll have to do so yourself.

Parameters

prefix	Max value is short.MaxValue = 32767

8.20.2.55 static bool PhotonNetwork.SetMasterClient (PhotonPlayer *masterClientPlayer*) [static]

Asks the server to assign another player as Master Client of your current room.

RPCs and RaiseEvent have the option to send messages only to the Master Client of a room. SetMasterClient affects which client gets those messages.

This method calls an operation on the server to set a new Master Client, which takes a roundtrip. In case of success, this client and the others get the new Master Client from the server.

SetMasterClient tells the server which current Master Client should be replaced with the new one. It will fail, if anything switches the Master Client moments earlier. There is no callback for this error. All clients should get the new Master Client assigned by the server anyways.

See also: PhotonNetwork.masterClient

On v3 servers: The ReceiverGroup.MasterClient (usable in RPCs) is not affected by this (still points to lowest player.ID in room). Avoid using this enum value (and send to a specific player instead).

If the current Master Client leaves, PUN will detect a new one by "lowest player ID". Implement OnMasterClient↩ Switched to get a callback in this case. The PUN-selected Master Client might assign a new one.

Make sure you don't create an endless loop of Master-assigning! When selecting a custom Master Client, all clients should point to the same player, no matter who actually assigns this player.

Locally the Master Client is immediately switched, while remote clients get an event. This means the game is tempoarily without Master Client like when a current Master Client leaves.

When switching the Master Client manually, keep in mind that this user might leave and not do it's work, just like any Master Client.

Parameters

masterClient↩ Player	The player to become the next Master Client.

Returns

False when this operation couldn't be done. Must be in a room (not in offlineMode).

8.20.2.56 static void PhotonNetwork.SetPlayerCustomProperties (Hashtable *customProperties*) [static]

Sets this (local) player's properties and synchronizes them to the other players (don't modify them directly).

While in a room, your properties are synced with the other players. CreateRoom, JoinRoom and JoinRandomRoom will all apply your player's custom properties when you enter the room. The whole Hashtable will get sent. Minimize the traffic by setting only updated key/values.

If the Hashtable is null, the custom properties will be cleared. Custom properties are never cleared automatically, so they carry over to the next room, if you don't change them.

Don't set properties by modifying PhotonNetwork.player.customProperties!

Parameters

custom↩ Properties	Only string-typed keys will be used from this hashtable. If null, custom properties are all deleted.

8.20.2.57 static void PhotonNetwork.SetReceivingEnabled (int *group,* bool *enabled*) [static]

Enable/disable receiving on given group (applied to PhotonViews)

Parameters

group	The interest group to affect.
enabled	Sets if receiving from group to enabled (or not).

8.20.2.58 static void PhotonNetwork.SetReceivingEnabled (int[] *enableGroups,* int[] *disableGroups*) [static]

Enable/disable receiving on given groups (applied to PhotonViews)

Parameters

enableGroups	The interest groups to enable (or null).
disableGroups	The interest groups to disable (or null).

8.20.2.59 static void PhotonNetwork.SetSendingEnabled (int *group*, bool *enabled*) [static]

Enable/disable sending on given group (applied to PhotonViews)

Parameters

group	The interest group to affect.
enabled	Sets if sending to group is enabled (or not).

8.20.2.60 static void PhotonNetwork.SetSendingEnabled (int[] *enableGroups*, int[] *disableGroups*) [static]

Enable/disable sending on given groups (applied to PhotonViews)

Parameters

enableGroups	The interest groups to enable sending on (or null).
disableGroups	The interest groups to disable sending on (or null).

8.20.2.61 static void PhotonNetwork.SwitchToProtocol (ConnectionProtocol *cp*) [static]

While offline, the network protocol can be switched (which affects the ports you can use to connect).

When you switch the protocol, make sure to also switch the port for the master server. Default ports are: TCP: 4530 UDP: 5055

This could look like this:
Connect(serverAddress, <udpport|tcpport>, appID, gameVersion)

Or when you use ConnectUsingSettings(), the PORT in the settings can be switched like so:
PhotonNetwork.PhotonServerSettings.ServerPort = 4530;

The current protocol can be read this way:
PhotonNetwork.networkingPeer.UsedProtocol

This does not work with the native socket plugin of PUN+ on mobile!

Parameters

cp	Network protocol to use as low level connection. UDP is default. TCP is not available on all platforms (see remarks).

8.20.2.62 static void PhotonNetwork.UnAllocateViewID (int *viewID*) [static]

Unregister a viewID (of manually instantiated and destroyed networked objects).

Parameters

viewID	A viewID manually allocated by this player.

8.20.2.63 static bool PhotonNetwork.WebRpc (string *name*, object *parameters*) [static]

This operation makes Photon call your custom web-service by name (path) with the given parameters.

This is a server-side feature which must be setup in the Photon Cloud Dashboard prior to use.
See the Turnbased Feature Overview for a short intro.
http://doc.photonengine.com/en/turnbased/current/getting-started/feature-overview
br/> The Parameters will be converted into JSon format, so make sure your parameters are compatible.

See PhotonNetworkingMessage.OnWebRpcResponse on how to get a response.

It's important to understand that the OperationResponse only tells if the WebRPC could be called. The content of the response contains any values your web-service sent and the error/success code. In case the web-service failed, an error code and a debug message are usually inside the OperationResponse.

The class WebRpcResponse is a helper-class that extracts the most valuable content from the WebRPC response.

Example callback implementation:

```
public void OnWebRpcResponse(OperationResponse response)
{
    WebRpcResponse webResponse = new WebRpcResponse(operationResponse);
    if (webResponse.ReturnCode != 0) { //...
    }


    switch (webResponse.Name) { //...
    }
    // and so on
}
```

8.20.3 Member Data Documentation

8.20.3.1 float PhotonNetwork.BackgroundTimeout = 60.0f [static]

Defines how long PUN keeps running a "fallback thread" to keep the connection after Unity's OnApplication↩ Pause(true) call.

If you set BackgroundTimeout PUN will stop keeping the connection, BackgroundTimeout seconds after On↩ ApplicationPause(true) got called. That means: After the set time, a regular timeout can happen. Your application will notice that timeout when it becomes active again.

To handle the timeout, implement: OnConnectionFail() (this case will use the cause: DisconnectByServerTimeout).

It's best practice to let inactive apps/connections time out after a while but allow taking calls, etc. So a reasonable value should be found. We think it could be 60 seconds.

Set a value greater than 0.001f, if you want to limit how long an app can keep the connection in background.

Info: PUN is running a "fallback thread" to send ACKs to the server, even when Unity is not calling Update() regularly. This helps keeping the connection while loading scenes and assets and when the app is in the background.

Note: Some platforms (e.g. iOS) don't allow to keep a connection while the app is in background. In those cases, this value does not change anything, the app immediately loses connection in background.

Unity's OnApplicationPause() callback is broken in some exports (Android) of some Unity versions. Make sure OnApplicationPause() gets the callbacks you'd expect on the platform you target! Check PhotonHandler.On↩ ApplicationPause(bool pause), to see the implementation.

8.20.3.2 bool PhotonNetwork.InstantiateInRoomOnly = true [static]

If true, Instantiate methods will check if you are in a room and fail if you are not.

Instantiating anything outside of a specific room is very likely to break things. Turn this off only if you know what you do.

8.20.3.3 PhotonLogLevel PhotonNetwork.logLevel = PhotonLogLevel.ErrorsOnly [static]

Network log level. Controls how verbose PUN is.

8.20.3.4 readonly int PhotonNetwork.MAX_VIEW_IDS = 1000 [static]

The maximum number of assigned PhotonViews *per player* (or scene). See the General Documentation topic "Limitations" on how to raise this limitation.

8.20.3.5 int PhotonNetwork.maxConnections [static]

Only used in Unity Networking. In PUN, set the number of players in PhotonNetwork.CreateRoom.

8.20.3.6 EventCallback PhotonNetwork.OnEventCall [static]

Register your RaiseEvent handling methods here by using "+=".

Any eventCode $<$ 200 will be forwarded to your delegate(s).

RaiseEvent

8.20.3.7 ServerSettings PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.↩ serverSettingsAssetFile, typeof(ServerSettings)) [static]

Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.

8.20.3.8 float PhotonNetwork.precisionForFloatSynchronization = 0.01f [static]

The minimum difference between floats before we send it via a PhotonView's OnSerialize/ObservingComponent.

8.20.3.9 float PhotonNetwork.precisionForQuaternionSynchronization = 1.0f [static]

The minimum angle that a rotation needs to change before we send it via a PhotonView's OnSerialize/Observing↩ Component.

8.20.3.10 float PhotonNetwork.precisionForVectorSynchronization = 0.000099f [static]

The minimum difference that a Vector2 or Vector3(e.g. a transforms rotation) needs to change before we send it via a PhotonView's OnSerialize/ObservingComponent.

Note that this is the sqrMagnitude. E.g. to send only after a 0.01 change on the Y-axix, we use 0.01f∗0.01f=0.0001f. As a remedy against float inaccuracy we use 0.000099f instead of 0.0001f.

8.20.3.11 Dictionary$<$string, GameObject$>$ PhotonNetwork.PrefabCache = new Dictionary$<$string, GameObject$>$() [static]

Keeps references to GameObjects for frequent instantiation (out of memory instead of loading the Resources).

You should be able to modify the cache anytime you like, except while Instantiate is used. Best do it only in the main-Thread.

8.20.3.12 HashSet<GameObject> PhotonNetwork.SendMonoMessageTargets [static]

If not null, this is the (exclusive) list of GameObjects that get called by PUN SendMonoMessage().

For all callbacks defined in PhotonNetworkingMessage, PUN will use SendMonoMessage and call FindObjectsOfType() to find all scripts and GameObjects that might want a callback by PUN.

PUN callbacks are not very frequent (in-game, property updates are most frequent) but FindObjectsOfType is time consuming and with a large number of GameObjects, performance might suffer.

Optionally, SendMonoMessageTargets can be used to supply a list of target GameObjects. This skips the FindObjectsOfType() but any GameObject that needs callbacks will have to Add itself to this list.

If null, the default behaviour is to do a SendMessage on each GameObject with a MonoBehaviour.

8.20.3.13 Type PhotonNetwork.SendMonoMessageTargetType = typeof(MonoBehaviour) [static]

Defines which classes can contain PUN Callback implementations.

This provides the option to optimize your runtime for speed.
The more specific this Type is, the fewer classes will be checked with reflection for callback methods.

8.20.3.14 bool PhotonNetwork.StartRpcsAsCoroutine = true [static]

Can be used to skip starting RPCs as Coroutine, which can be a performance issue.

8.20.3.15 bool PhotonNetwork.UsePrefabCache = true [static]

While enabled (true), Instantiate uses PhotonNetwork.PrefabCache to keep game objects in memory (improving instantiation of the same prefab).

Setting UsePrefabCache to false during runtime will not clear PrefabCache but will ignore it right away. You could clean and modify the cache yourself. Read its comments.

8.20.3.16 bool PhotonNetwork.UseRpcMonoBehaviourCache [static]

While enabled, the MonoBehaviours on which we call RPCs are cached, avoiding costly GetComponents<MonoBehaviour>() calls.

RPCs are called on the MonoBehaviours of a target PhotonView. Those have to be found via GetComponents.

When set this to true, the list of MonoBehaviours gets cached in each PhotonView. You can use photonView.RefreshRpcMonoBehaviourCache() to manually refresh a PhotonView's list of MonoBehaviours on demand (when a new MonoBehaviour gets added to a networked GameObject, e.g.).

8.20.3.17 const string PhotonNetwork.versionPUN = "1.72"

Version number of PUN. Also used in GameVersion to separate client version from each other.

8.20.4 Property Documentation

8.20.4.1 AuthenticationValues PhotonNetwork.AuthValues [static], [get], [set]

A user's authentication values used during connect for Custom Authentication with Photon (and a custom service/community). Set these before calling Connect if you want custom authentication.

If authentication fails for any values, PUN will call your implementation of OnCustomAuthenticationFailed(string debugMsg). See: PhotonNetworkingMessage.OnCustomAuthenticationFailed

8.20.4.2 bool PhotonNetwork.autoCleanUpPlayerObjects [static],[get],[set]

This setting defines per room, if network-instantiated GameObjects (with PhotonView) get cleaned up when the creator of it leaves.

This setting is done per room. It can't be changed in the room and it will override the settings of individual clients.

If room.AutoCleanUp is enabled in a room, the PUN clients will destroy a player's GameObjects on leave. This includes GameObjects manually instantiated (via RPCs, e.g.). When enabled, the server will clean RPCs, instantiated GameObjects and PhotonViews of the leaving player, too. and Players who join after someone left, won't get the events of that player anymore.

Under the hood, this setting is stored as a Custom Room Property. Enabled by default.

8.20.4.3 bool PhotonNetwork.autoJoinLobby [static],[get],[set]

Set in PhotonServerSettings asset. Defines if the PhotonNetwork should join the "lobby" when connected to the Master server.

If this is false, OnConnectedToMaster() will be called when connection to the Master is available. OnJoinedLobby() will NOT be called if this is false.

Enabled by default.

The room listing will not become available. Rooms can be created and joined (randomly) without joining the lobby (and getting sent the room list).

8.20.4.4 bool PhotonNetwork.automaticallySyncScene [static],[get],[set]

Defines if all clients in a room should load the same level as the Master Client (if that used PhotonNetwork.Load↩Level).

To synchronize the loaded level, the Master Client should use PhotonNetwork.LoadLevel. All clients will load the new scene when they get the update or when they join.

Internally, a Custom Room Property is set for the loaded scene. When a client reads that and is not in the same scene yet, it will immediately pause the Message Queue (PhotonNetwork.isMessageQueueRunning = false) and load. When the scene finished loading, PUN will automatically re-enable the Message Queue.

8.20.4.5 bool PhotonNetwork.connected [static],[get]

False until you connected to Photon initially. True in offline mode, while connected to any server and even while switching servers.

8.20.4.6 bool PhotonNetwork.connectedAndReady [static],[get]

A refined version of connected which is true only if your connection to the server is ready to accept operations like join, leave, etc.

8.20.4.7 bool PhotonNetwork.connecting [static],[get]

True when you called ConnectUsingSettings (or similar) until the low level connection to Photon gets established.

8.20.4.8 ConnectionState PhotonNetwork.connectionState [static],[get]

Simplified connection state

8.20.4.9 PeerState PhotonNetwork.connectionStateDetailed [static], [get]

Detailed connection state (ignorant of PUN, so it can be "disconnected" while switching servers).

In OfflineMode, this is PeerState.Joined (after create/join) or it is ConnectedToMaster in all other cases.

8.20.4.10 int PhotonNetwork.countOfPlayers [static], [get]

The count of players currently using this application (available on MasterServer in 5sec intervals).

8.20.4.11 int PhotonNetwork.countOfPlayersInRooms [static], [get]

Count of users currently playing your app in some room (sent every 5sec by Master Server). Use playerList.Count to get the count of players in the room you're in!

8.20.4.12 int PhotonNetwork.countOfPlayersOnMaster [static], [get]

The count of players currently looking for a room (available on MasterServer in 5sec intervals).

8.20.4.13 int PhotonNetwork.countOfRooms [static], [get]

The count of rooms currently in use (available on MasterServer in 5sec intervals).

While inside the lobby you can also check the count of listed rooms as: PhotonNetwork.GetRoomList().Length. Since PUN v1.25 this is only based on the statistic event Photon sends (counting all rooms).

8.20.4.14 bool PhotonNetwork.CrcCheckEnabled [static], [get], [set]

Crc checks can be useful to detect and avoid issues with broken datagrams. Can be enabled while not connected.

8.20.4.15 bool PhotonNetwork.EnableLobbyStatistics [static], [get], [set]

Set in PhotonServerSettings asset. Enable to get a list of active lobbies from the Master Server.

Lobby Statistics can be useful if a game uses multiple lobbies and you want to show activity of each to players.

This value is stored in PhotonServerSettings.

PhotonNetwork.LobbyStatistics is updated when you connect to the Master Server. There is also a callback PunBehaviour.

8.20.4.16 List<FriendInfo> PhotonNetwork.Friends [static], [get], [set]

Read-only list of friends, their online status and the room they are in. Null until initialized by a FindFriends call.

Do not modify this list! It is internally handled by FindFriends and only available to read the values. The value of FriendsListAge tells you how old the data is in milliseconds.

Don't get this list more often than useful (> 10 seconds). In best case, keep the list you fetch really short. You could (e.g.) get the full list only once, then request a few updates only for friends who are online. After a while (e.g. 1 minute), you can get the full list again (to update online states).

8.20.4.17 int PhotonNetwork.FriendsListAge [static], [get]

Age of friend list info (in milliseconds). It's 0 until a friend list is fetched.

8.20.4.18 string PhotonNetwork.gameVersion [static], [get], [set]

Version string for your this build. Can be used to separate incompatible clients. Sent during connect.

This is only sent when you connect so that is also the place you set it usually (e.g. in ConnectUsingSettings).

8.20.4.19 bool PhotonNetwork.inRoom [static], [get]

Is true while being in a room (connectionStateDetailed == PeerState.Joined).

Many actions can only be executed in a room, like Instantiate or Leave, etc. You can join a room in offline mode, too.

8.20.4.20 bool PhotonNetwork.insideLobby [static], [get]

True while this client is in a lobby.

Implement IPunCallbacks.OnReceivedRoomListUpdate() for a notification when the list of rooms becomes available or updated.

You are automatically leaving any lobby when you join a room! Lobbies only exist on the Master Server (whereas rooms are handled by Game Servers).

8.20.4.21 bool PhotonNetwork.isMasterClient [static], [get]

Are we the master client?

8.20.4.22 bool PhotonNetwork.isMessageQueueRunning [static], [get], [set]

Can be used to pause dispatching of incoming evtents (RPCs, Instantiates and anything else incoming).

While IsMessageQueueRunning == false, the OnPhotonSerializeView calls are not done and nothing is sent by a client. Also, incoming messages will be queued until you re-activate the message queue.

This can be useful if you first want to load a level, then go on receiving data of PhotonViews and RPCs. The client will go on receiving and sending acknowledgements for incoming packages and your RPCs/Events. This adds "lag" and can cause issues when the pause is longer, as all incoming messages are just queued.

8.20.4.23 bool PhotonNetwork.isNonMasterClientInRoom [static], [get]

True if we are in a room (client) and NOT the room's masterclient

8.20.4.24 TypedLobby PhotonNetwork.lobby [static], [get], [set]

The lobby that will be used when PUN joins a lobby or creates a game.

The default lobby uses an empty string as name. PUN will enter a lobby on the Master Server if autoJoinLobby is set to true. So when you connect or leave a room, PUN automatically gets you into a lobby again.

Check PhotonNetwork.insideLobby if the client is in a lobby. (Master Server And Lobby)

8.20.4.25 List<TypedLobbyInfo> PhotonNetwork.LobbyStatistics [static], [get], [set]

If turned on, the Master Server will provide information about active lobbies for this application.

Lobby Statistics can be useful if a game uses multiple lobbies and you want to show activity of each to players. Per lobby, you get: name, type, room- and player-count.

PhotonNetwork.LobbyStatistics is updated when you connect to the Master Server. There is also a callback PunBehaviour.OnLobbyStatisticsUpdate, which you should implement to update your UI (e.g.).

Lobby Statistics are not turned on by default. Enable them in the PhotonServerSettings file of the project.

8.20.4.26 PhotonPlayer PhotonNetwork.masterClient [static], [get]

The Master Client of the current room or null (outside of rooms).

Can be used as "authoritative" client/player to make descisions, run AI or other.

If the current Master Client leaves the room (leave/disconnect), the server will quickly assign someone else. If the current Master Client times out (closed app, lost connection, etc), messages sent to this client are effectively lost for the others! A timeout can take 10 seconds in which no Master Client is active.

Implement the method IPunCallbacks.OnMasterClientSwitched to be called when the Master Client switched.

Use PhotonNetwork.SetMasterClient, to switch manually to some other player / client.

With offlineMode == true, this always returns the PhotonNetwork.player.

8.20.4.27 int PhotonNetwork.MaxResendsBeforeDisconnect [static], [get], [set]

Defines the number of times a reliable message can be resent before not getting an ACK for it will trigger a disconnect. Default: 5.

Less resends mean quicker disconnects, while more can lead to much more lag without helping. Min: 3. Max: 10.

8.20.4.28 bool PhotonNetwork.NetworkStatisticsEnabled [static], [get], [set]

Enables or disables the collection of statistics about this client's traffic.

If you encounter issues with clients, the traffic stats are a good starting point to find solutions. Only with enabled stats, you can use GetVitalStats

8.20.4.29 bool PhotonNetwork.offlineMode [static], [get], [set]

Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on PhotonNetwork will not create any connections and there is near to no overhead. Mostly usefull for reusing RPC's and PhotonNetwork.Instantiate

8.20.4.30 PhotonPlayer [] PhotonNetwork.otherPlayers [static], [get]

The list of players in the current room, excluding the local player.

This list is only valid, while the client is in a room. It automatically gets updated when someone joins or leaves.

This can be used to list all other players in a room. Each player's PhotonPlayer.customProperties are accessible (set and synchronized via PhotonPlayer.SetCustomProperties).

You can use a PhotonPlayer.TagObject to store an arbitrary object for reference. That is not synchronized via the network.

8.20.4.31 int PhotonNetwork.PacketLossByCrcCheck [static], [get]

If CrcCheckEnabled, this counts the incoming packages that don't have a valid CRC checksum and got rejected.

8.20.4.32 PhotonPlayer PhotonNetwork.player [static], [get]

The local PhotonPlayer. Always available and represents this player. CustomProperties can be set before entering a room and will be synced as well.

8.20.4.33 PhotonPlayer [] PhotonNetwork.playerList [static], [get]

The list of players in the current room, including the local player.

This list is only valid, while the client is in a room. It automatically gets updated when someone joins or leaves.

This can be used to list all players in a room. Each player's PhotonPlayer.customProperties are accessible (set and synchronized via PhotonPlayer.SetCustomProperties).

You can use a PhotonPlayer.TagObject to store an arbitrary object for reference. That is not synchronized via the network.

8.20.4.34 string PhotonNetwork.playerName [static], [get], [set]

Set to synchronize the player's nickname with everyone in the room(s) you enter. This sets PhotonPlayer.name.

The playerName is just a nickname and does not have to be unique or backed up with some account.
Set the value any time (e.g. before you connect) and it will be available to everyone you play with.
Access the names of players by: PhotonPlayer.name.
PhotonNetwork.otherPlayers is a list of other players - each contains the playerName the remote player set.

8.20.4.35 IPunPrefabPool PhotonNetwork.PrefabPool [static], [get], [set]

An Object Pool can be used to keep and reuse instantiated object instances. It replaced Unity's default Instantiate and Destroy methods.

To use a GameObject pool, implement IPunPrefabPool and assign it here. Prefabs are identified by name.

8.20.4.36 int PhotonNetwork.QuickResends [static], [get], [set]

In case of network loss, reliable messages can be repeated quickly up to 3 times.

When reliable messages get lost more than once, subsequent repeats are delayed a bit to allow the network to recover.
With this option, the repeats 2 and 3 can be sped up. This can help avoid timeouts but also it increases the speed in which gaps are closed.
When you set this, increase PhotonNetwork.MaxResendsBeforeDisconnect to 6 or 7.

8.20.4.37 int PhotonNetwork.ResentReliableCommands [static], [get]

Count of commands that got repeated (due to local repeat-timing before an ACK was received).

If this value increases a lot, there is a good chance that a timeout disconnect will happen due to bad conditions.

8.20.4.38 Room PhotonNetwork.room [static], [get]

Get the room we're currently in. Null if we aren't in any room.

8.20.4.39 int PhotonNetwork.sendRate [static], [get], [set]

Defines how many times per second PhotonNetwork should send a package. If you change this, do not forget to also change 'sendRateOnSerialize'.

Less packages are less overhead but more delay. Setting the sendRate to 50 will create up to 50 packages per second (which is a lot!). Keep your target platform in mind: mobile networks are slower and less reliable.

8.20.4.40 int PhotonNetwork.sendRateOnSerialize [static], [get], [set]

Defines how many times per second OnPhotonSerialize should be called on PhotonViews.

Choose this value in relation to PhotonNetwork.sendRate. OnPhotonSerialize will create updates and messages to be sent.
A lower rate takes up less performance but will cause more lag.

8.20.4.41 ServerConnection PhotonNetwork.Server [static], [get]

The server (type) this client is currently connected or connecting to.

Photon uses 3 different roles of servers: Name Server, Master Server and Game Server.

8.20.4.42 string PhotonNetwork.ServerAddress [static], [get]

Currently used server address (no matter if master or game server).

8.20.4.43 int PhotonNetwork.ServerTimestamp [static], [get]

The current server's millisecond timestamp.

This can be useful to sync actions and events on all clients in one room. The timestamp is based on the server's Environment.TickCount.

It will overflow from a positive to a negative value every so often, so be careful to use only time-differences to check the time delta when things happen.

This is the basis for PhotonNetwork.time.

8.20.4.44 double PhotonNetwork.time [static], [get]

Photon network time, synched with the server.

v1.55
This time value depends on the server's Environment.TickCount. It is different per server but inside a Room, all clients should have the same value (Rooms are on one server only).
This is not a DateTime!

Use this value with care:
It can start with any positive value.
It will "wrap around" from 4294967.295 to 0!

8.20.4.45 int PhotonNetwork.unreliableCommandsLimit [static], [get], [set]

Used once per dispatch to limit unreliable commands per channel (so after a pause, many channels can still cause a lot of unreliable commands)

8.21 PhotonPingManager Class Reference

Public Member Functions

- IEnumerator PingSocket (Region region)

Static Public Member Functions

- static string ResolveHost (string hostName)

  Attempts to resolve a hostname into an IP string or returns empty string if that fails.

Public Attributes

- bool UseNative

Static Public Attributes

- static int Attempts = 5
- static bool IgnoreInitialAttempt = true
- static int MaxMilliseconsPerPing = 800

Properties

- Region BestRegion `[get]`
- bool Done `[get]`

8.21.1 Member Function Documentation

8.21.1.1 IEnumerator PhotonPingManager.PingSocket (Region *region*)

Affected by frame-rate of app, as this Coroutine checks the socket for a result once per frame.

8.21.1.2 static string PhotonPingManager.ResolveHost (string *hostName*) `[static]`

Attempts to resolve a hostname into an IP string or returns empty string if that fails.

To be compatible with most platforms, the address family is checked like this: if (ipAddress.AddressFamily.To↩ String().Contains("6")) // ipv6... </reamrks>

Parameters

hostName	Hostname to resolve.

Returns

IP string or empty string if resolution fails

8.21.2 Member Data Documentation

8.21.2.1 int PhotonPingManager.Attempts = 5 `[static]`

8.21.2.2 bool PhotonPingManager.IgnoreInitialAttempt = true `[static]`

8.21.2.3 int PhotonPingManager.MaxMilliseconsPerPing = 800 `[static]`

8.21.2.4 bool PhotonPingManager.UseNative

8.21.3 Property Documentation

8.21.3.1 Region PhotonPingManager.BestRegion [get]

8.21.3.2 bool PhotonPingManager.Done [get]

8.22 PhotonPlayer Class Reference

Summarizes a "player" within a room, identified (in that room) by actorID.

Public Member Functions

- PhotonPlayer (bool isLocal, int actorID, string name)

  Creates a PhotonPlayer instance.
- override bool Equals (object p)

  Makes PhotonPlayer comparable
- override int GetHashCode ()
- void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues=null, bool webForward=false)

  Updates the this player's Custom Properties with new/updated key-values.
- PhotonPlayer Get (int id)
- PhotonPlayer GetNext ()
- PhotonPlayer GetNextFor (PhotonPlayer currentPlayer)
- PhotonPlayer GetNextFor (int currentPlayerId)
- override string ToString ()

  Brief summary string of the PhotonPlayer. Includes name or player.ID and if it's the Master Client.
- string ToStringFull ()

  String summary of the PhotonPlayer: player.ID, name and all custom properties of this user.

Static Public Member Functions

- static PhotonPlayer Find (int ID)

  Try to get a specific player by id.

Public Attributes

- readonly bool isLocal = false

  Only one player is controlled by each client. Others are not local.
- object TagObject

  Can be used to store a reference that's useful to know "by player".

Protected Member Functions

- PhotonPlayer (bool isLocal, int actorID, Hashtable properties)

  Internally used to create players from event Join

Properties

- int ID [get]

  This player's actorID
- string name [get, set]

  Nickname of this player.
- string userId [get, set]

  UserId of the player, available when the room got created with RoomOptions.publishUserId = true.
- bool isMasterClient [get]

  True if this player is the Master Client of the current room.
- bool isInactive [get, set]

  Players might be inactive in a room when PlayerTTL for a room is > 0. If true, the player is not getting events from this room (now) but can return later.
- Hashtable customProperties [get, set]

  Read-only cache for custom properties of player. Set via PhotonPlayer.SetCustomProperties.
- Hashtable allProperties [get]

  Creates a Hashtable with all properties (custom and "well known" ones).

8.22.1 Detailed Description

Summarizes a "player" within a room, identified (in that room) by actorID.

Each player has an actorId (or ID), valid for that room. It's -1 until it's assigned by server. Each client can set it's player's custom properties with SetCustomProperties, even before being in a room. They are synced when joining a room.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 PhotonPlayer.PhotonPlayer (bool *isLocal*, int *actorID*, string *name*)

Creates a PhotonPlayer instance.

Parameters

isLocal	If this is the local peer's player (or a remote one).
actorID	ID or ActorNumber of this player in the current room (a shortcut to identify each player in room)
name	Name of the player (a "well known property").

8.22.2.2 PhotonPlayer.PhotonPlayer (bool *isLocal*, int *actorID*, Hashtable *properties*) [protected]

Internally used to create players from event Join

8.22.3 Member Function Documentation

8.22.3.1 override bool PhotonPlayer.Equals (object *p*)

Makes PhotonPlayer comparable

8.22.3.2 static PhotonPlayer PhotonPlayer.Find (int *ID*) [static]

Try to get a specific player by id.

Parameters

ID	ActorID

Returns

The player with matching actorID or null, if the actorID is not in use.

8.22.3.3 PhotonPlayer PhotonPlayer.Get (int *id*)

8.22.3.4 override int PhotonPlayer.GetHashCode ( )

8.22.3.5 PhotonPlayer PhotonPlayer.GetNext ( )

8.22.3.6 PhotonPlayer PhotonPlayer.GetNextFor (PhotonPlayer *currentPlayer*)

8.22.3.7 PhotonPlayer PhotonPlayer.GetNextFor (int *currentPlayerId*)

8.22.3.8 void PhotonPlayer.SetCustomProperties (Hashtable *propertiesToSet*, Hashtable *expectedValues* = `null`, bool *webForward* = `false`)

Updates the this player's Custom Properties with new/updated key-values.

Custom Properties are a key-value set (Hashtable) which is available to all players in a room. They can relate to the room or individual players and are useful when only the current value of something is of interest. For example: The map of a room. All keys must be strings.

The Room and the PhotonPlayer class both have SetCustomProperties methods. Also, both classes offer access to current key-values by: customProperties.

Always use SetCustomProperties to change values. To reduce network traffic, set only values that actually changed. New properties are added, existing values are updated. Other values will not be changed, so only provide values that changed or are new.

To delete a named (custom) property of this room, use null as value.

Locally, SetCustomProperties will update it's cache without delay. Other clients are updated through Photon (the server) with a fitting operation.

Check and Swap

SetCustomProperties have the option to do a server-side Check-And-Swap (CAS): Values only get updated if the expected values are correct. The expectedValues can be different key/values than the propertiesToSet. So you can check some key and set another key's value (if the check succeeds).

If the client's knowledge of properties is wrong or outdated, it can't set values with CAS. This can be useful to keep players from concurrently setting values. For example: If all players try to pickup some card or item, only one should get it. With CAS, only the first SetProperties gets executed server-side and any other (sent at the same time) fails.

The server will broadcast successfully changed values and the local "cache" of customProperties only gets updated after a roundtrip (if anything changed).

You can do a "webForward": Photon will send the changed properties to a WebHook defined for your application.

OfflineMode

While PhotonNetwork.offlineMode is true, the expectedValues and webForward parameters are ignored. In Offline↩ Mode, the local customProperties values are immediately updated (without the roundtrip).

Parameters

propertiesToSet	The new properties to be set.
expectedValues	At least one property key/value set to check server-side. Key and value must be correct. Ignored in OfflineMode.
webForward	Set to true, to forward the set properties to a WebHook, defined for this app (in Dashboard). Ignored in OfflineMode.

8.22.3.9 override string PhotonPlayer.ToString ( )

Brief summary string of the PhotonPlayer. Includes name or player.ID and if it's the Master Client.

8.22.3.10 string PhotonPlayer.ToStringFull ( )

String summary of the PhotonPlayer: player.ID, name and all custom properties of this user.

Use with care and not every frame! Converts the customProperties to a String on every single call.

8.22.4 Member Data Documentation

8.22.4.1 readonly bool PhotonPlayer.isLocal = false

Only one player is controlled by each client. Others are not local.

8.22.4.2 object PhotonPlayer.TagObject

Can be used to store a reference that's useful to know "by player".

Example: Set a player's character as Tag by assigning the GameObject on Instantiate.

8.22.5 Property Documentation

8.22.5.1 Hashtable PhotonPlayer.allProperties [get]

Creates a Hashtable with all properties (custom and "well known" ones).

If used more often, this should be cached.

8.22.5.2 Hashtable PhotonPlayer.customProperties [get], [set]

Read-only cache for custom properties of player. Set via PhotonPlayer.SetCustomProperties.

Don't modify the content of this Hashtable. Use SetCustomProperties and the properties of this class to modify values. When you use those, the client will sync values with the server.

SetCustomProperties

8.22.5.3 int PhotonPlayer.ID [get]

This player's actorID

8.22.5.4 bool PhotonPlayer.isInactive [get], [set]

Players might be inactive in a room when PlayerTTL for a room is > 0. If true, the player is not getting events from this room (now) but can return later.

8.22.5.5 bool PhotonPlayer.isMasterClient [get]

True if this player is the Master Client of the current room.

See also: PhotonNetwork.masterClient.

8.22.5.6 string PhotonPlayer.name [get], [set]

Nickname of this player.

Set the PhotonNetwork.playerName to make the name synchronized in a room.

8.22.5.7 string PhotonPlayer.userId [get], [set]

UserId of the player, available when the room got created with RoomOptions.publishUserId = true.

Useful for PhotonNetwork.FindFriends and blocking slots in a room for expected players (e.g. in PhotonNetwork.↩ CreateRoom).

8.23 PhotonRigidbody2DView Class Reference

This class helps you to synchronize the velocities of a 2d physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbody2DView is added to the list of observed components

Inherits MonoBehaviour.

8.23.1 Detailed Description

This class helps you to synchronize the velocities of a 2d physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbody2DView is added to the list of observed components

8.24 PhotonRigidbodyView Class Reference

This class helps you to synchronize the velocities of a physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbodyView is added to the list of observed components

Inherits MonoBehaviour.

8.24.1 Detailed Description

This class helps you to synchronize the velocities of a physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbodyView is added to the list of observed components

8.25 PhotonStatsGui Class Reference

Basic GUI to show traffic and health statistics of the connection to Photon, toggled by shift+tab.

Inherits MonoBehaviour.

Public Member Functions

- void Start ()
- void Update ()

  Checks for shift+tab input combination (to toggle statsOn).
- void OnGUI ()
- void TrafficStatsWindow (int windowID)

Public Attributes

- bool statsWindowOn = true

  Shows or hides GUI (does not affect if stats are collected).
- bool statsOn = true

  Option to turn collecting stats on or off (used in Update()).
- bool healthStatsVisible

  Shows additional "health" values of connection.
- bool trafficStatsOn

  Shows additional "lower level" traffic stats.
- bool buttonsOn

  Show buttons to control stats and reset them.
- Rect statsRect = new Rect(0, 100, 200, 50)

  Positioning rect for window.
- int WindowId = 100

  Unity GUI Window ID (must be unique or will cause issues).

8.25.1 Detailed Description

Basic GUI to show traffic and health statistics of the connection to Photon, toggled by shift+tab.

The shown health values can help identify problems with connection losses or performance. Example: If the time delta between two consecutive SendOutgoingCommands calls is a second or more, chances rise for a disconnect being caused by this (because acknowledgements to the server need to be sent in due time).

8.25.2 Member Function Documentation

8.25.2.1 void PhotonStatsGui.OnGUI ( )

8.25.2.2 void PhotonStatsGui.Start ( )

8.25.2.3 void PhotonStatsGui.TrafficStatsWindow (int *windowID*)

8.25.2.4 void PhotonStatsGui.Update ( )

Checks for shift+tab input combination (to toggle statsOn).

8.25.3 Member Data Documentation

8.25.3.1 bool PhotonStatsGui.buttonsOn

Show buttons to control stats and reset them.

8.25.3.2 bool PhotonStatsGui.healthStatsVisible

Shows additional "health" values of connection.

8.25.3.3 bool PhotonStatsGui.statsOn = true

Option to turn collecting stats on or off (used in Update()).

8.25.3.4 Rect PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)

Positioning rect for window.

8.25.3.5 bool PhotonStatsGui.statsWindowOn = true

Shows or hides GUI (does not affect if stats are collected).

8.25.3.6 bool PhotonStatsGui.trafficStatsOn

Shows additional "lower level" traffic stats.

8.25.3.7 int PhotonStatsGui.WindowId = 100

Unity GUI Window ID (must be unique or will cause issues).

8.26 PhotonStream Class Reference

This container is used in OnPhotonSerializeView() to either provide incoming data of a PhotonView or for you to provide it.

Public Member Functions

- PhotonStream (bool write, object[] incomingData)

  Creates a stream and initializes it. Used by PUN internally.
- object ReceiveNext ()

  Read next piece of data from the stream when isReading is true.
- object PeekNext ()

  Read next piece of data from the stream without advancing the "current" item.
- void SendNext (object obj)

  Add another piece of data to send it when isWriting is true.
- object[] ToArray ()

  Turns the stream into a new object[].
- void Serialize (ref bool myBool)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref int myInt)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref string value)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref char value)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref short value)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref float obj)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref PhotonPlayer obj)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref Vector3 obj)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref Vector2 obj)

  Will read or write the value, depending on the stream's isWriting value.
- void Serialize (ref Quaternion obj)

  Will read or write the value, depending on the stream's isWriting value.

Properties

- bool isWriting `[get]`

  If true, this client should add data to the stream to send it.
- bool isReading `[get]`

  If true, this client should read data send by another client.
- int Count `[get]`

  Count of items in the stream.

8.26.1 Detailed Description

This container is used in OnPhotonSerializeView() to either provide incoming data of a PhotonView or for you to provide it.

The isWriting property will be true if this client is the "owner" of the PhotonView (and thus the GameObject). Add data to the stream and it's sent via the server to the other players in a room. On the receiving side, isWriting is false and the data should be read.

Send as few data as possible to keep connection quality up. An empty PhotonStream will not be sent.

Use either Serialize() for reading and writing or SendNext() and ReceiveNext(). The latter two are just explicit read and write methods but do about the same work as Serialize(). It's a matter of preference which methods you use.

See also

PhotonNetworkingMessage

8.26.2 Constructor & Destructor Documentation

8.26.2.1 PhotonStream.PhotonStream (bool *write,* object[ ] *incomingData*)

Creates a stream and initializes it. Used by PUN internally.

8.26.3 Member Function Documentation

8.26.3.1 object PhotonStream.PeekNext ( )

Read next piece of data from the stream without advancing the "current" item.

8.26.3.2 object PhotonStream.ReceiveNext ( )

Read next piece of data from the stream when isReading is true.

8.26.3.3 void PhotonStream.SendNext (object *obj*)

Add another piece of data to send it when isWriting is true.

8.26.3.4 void PhotonStream.Serialize (ref bool *myBool*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.5 void PhotonStream.Serialize (ref int *myInt*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.6 void PhotonStream.Serialize (ref string *value*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.7 void PhotonStream.Serialize (ref char *value*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.8 void PhotonStream.Serialize (ref short *value*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.9 void PhotonStream.Serialize (ref float *obj*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.10 void PhotonStream.Serialize (ref PhotonPlayer *obj*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.11 void PhotonStream.Serialize (ref Vector3 *obj*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.12 void PhotonStream.Serialize (ref Vector2 *obj*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.13 void PhotonStream.Serialize (ref Quaternion *obj*)

Will read or write the value, depending on the stream's isWriting value.

8.26.3.14 object [ ] PhotonStream.ToArray ( )

Turns the stream into a new object[].

8.26.4 Property Documentation

8.26.4.1 int PhotonStream.Count [get]

Count of items in the stream.

8.26.4.2 bool PhotonStream.isReading [get]

If true, this client should read data send by another client.

8.26.4.3 bool PhotonStream.isWriting [get]

If true, this client should add data to the stream to send it.

8.27 PhotonStreamQueue Class Reference

The PhotonStreamQueue helps you poll object states at higher frequencies then what PhotonNetwork.sendRate dictates and then sends all those states at once when Serialize() is called. On the receiving end you can call Deserialize() and then the stream will roll out the received object states in the same order and timeStep they were recorded in.

Public Member Functions

- PhotonStreamQueue (int sampleRate)

  Initializes a new instance of the PhotonStreamQueue class.
- void Reset ()

  Resets the PhotonStreamQueue. You need to do this whenever the amount of objects you are observing changes
- void SendNext (object obj)

  Adds the next object to the queue. This works just like PhotonStream.SendNext

- bool HasQueuedObjects ()

  Determines whether the queue has stored any objects
- object ReceiveNext ()

  Receives the next object from the queue. This works just like PhotonStream.ReceiveNext
- void Serialize (PhotonStream stream)

  Serializes the specified stream. Call this in your OnPhotonSerializeView method to send the whole recorded stream.
- void Deserialize (PhotonStream stream)

  Deserializes the specified stream. Call this in your OnPhotonSerializeView method to receive the whole recorded stream.

8.27.1 Detailed Description

The PhotonStreamQueue helps you poll object states at higher frequencies then what PhotonNetwork.sendRate dictates and then sends all those states at once when Serialize() is called. On the receiving end you can call Deserialize() and then the stream will roll out the received object states in the same order and timeStep they were recorded in.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 PhotonStreamQueue.PhotonStreamQueue (int *sampleRate*)

Initializes a new instance of the PhotonStreamQueue class.

Parameters

sampleRate	How many times per second should the object states be sampled

8.27.3 Member Function Documentation

8.27.3.1 void PhotonStreamQueue.Deserialize (PhotonStream *stream*)

Deserializes the specified stream. Call this in your OnPhotonSerializeView method to receive the whole recorded stream.

Parameters

stream	The PhotonStream you receive as a parameter in OnPhotonSerializeView

8.27.3.2 bool PhotonStreamQueue.HasQueuedObjects ( )

Determines whether the queue has stored any objects

8.27.3.3 object PhotonStreamQueue.ReceiveNext ( )

Receives the next object from the queue. This works just like PhotonStream.ReceiveNext

Returns

8.27.3.4 void PhotonStreamQueue.Reset ( )

Resets the PhotonStreamQueue. You need to do this whenever the amount of objects you are observing changes

8.27.3.5 void PhotonStreamQueue.SendNext (object *obj*)

Adds the next object to the queue. This works just like PhotonStream.SendNext

Parameters

obj	The object you want to add to the queue

8.27.3.6 void PhotonStreamQueue.Serialize (PhotonStream *stream*)

Serializes the specified stream. Call this in your OnPhotonSerializeView method to send the whole recorded stream.

Parameters

stream	The PhotonStream you receive as a parameter in OnPhotonSerializeView

8.28 PhotonTransformView Class Reference

This class helps you to synchronize position, rotation and scale of a GameObject. It also gives you many different options to make the synchronized values appear smooth, even when the data is only send a couple of times per second. Simply add the component to your GameObject and make sure that the PhotonTransformView is added to the list of observed components

Inherits MonoBehaviour, and IPunObservable.

Public Member Functions

- void SetSynchronizedValues (Vector3 speed, float turnSpeed)

  These values are synchronized to the remote objects if the interpolation mode or the extrapolation mode SynchronizeValues is used. Your movement script should pass on the current speed (in units/second) and turning speed (in angles/second) so the remote object can use them to predict the objects movement.
- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

  Called by PUN several times per second, so that your script can write and read synchronization data for the PhotonView.

8.28.1 Detailed Description

This class helps you to synchronize position, rotation and scale of a GameObject. It also gives you many different options to make the synchronized values appear smooth, even when the data is only send a couple of times per second. Simply add the component to your GameObject and make sure that the PhotonTransformView is added to the list of observed components

8.28.2 Member Function Documentation

8.28.2.1 void PhotonTransformView.OnPhotonSerializeView (PhotonStream *stream,* PhotonMessageInfo *info*)

Called by PUN several times per second, so that your script can write and read synchronization data for the PhotonView.

This method will be called in scripts that are assigned as Observed component of a PhotonView.
PhotonNetwork.sendRateOnSerialize affects how often this method is called.
PhotonNetwork.sendRate affects how often packages are sent by this client.

Implementing this method, you can customize which data a PhotonView regularly synchronizes. Your code defines what is being sent (content) and how your data is used by receiving clients.

Unlike other callbacks, *OnPhotonSerializeView only gets called when it is assigned to a PhotonView* as PhotonView.observed script.

To make use of this method, the PhotonStream is essential. It will be in "writing" mode" on the client that controls a PhotonView (PhotonStream.isWriting == true) and in "reading mode" on the remote clients that just receive that the controlling client sends.

If you skip writing any value into the stream, PUN will skip the update. Used carefully, this can conserve bandwidth and messages (which have a limit per room/second).

Note that OnPhotonSerializeView is not called on remote clients when the sender does not send any update. This can't be used as "x-times per second Update()".

Implements IPunObservable.

8.28.2.2 void PhotonTransformView.SetSynchronizedValues (Vector3 *speed,* float *turnSpeed*)

These values are synchronized to the remote objects if the interpolation mode or the extrapolation mode SynchronizeValues is used. Your movement script should pass on the current speed (in units/second) and turning speed (in angles/second) so the remote object can use them to predict the objects movement.

Parameters

speed	The current movement vector of the object in units/second.
turnSpeed	The current turn speed of the object in angles/second.

8.29 PhotonTransformViewPositionControl Class Reference

Public Member Functions

- PhotonTransformViewPositionControl (PhotonTransformViewPositionModel model)
- void SetSynchronizedValues (Vector3 speed, float turnSpeed)

  These values are synchronized to the remote objects if the interpolation mode or the extrapolation mode SynchronizeValues is used. Your movement script should pass on the current speed (in units/second) and turning speed (in angles/second) so the remote object can use them to predict the objects movement.
- Vector3 UpdatePosition (Vector3 currentPosition)

  Calculates the new position based on the values setup in the inspector
- Vector3 GetNetworkPosition ()

  Gets the last position that was received through the network
- Vector3 GetExtrapolatedPositionOffset ()

  Calculates an estimated position based on the last synchronized position, the time when the last position was received and the movement speed of the object
- void OnPhotonSerializeView (Vector3 currentPosition, PhotonStream stream, PhotonMessageInfo info)

8.29.1 Constructor & Destructor Documentation

8.29.1.1 PhotonTransformViewPositionControl.PhotonTransformViewPositionControl (PhotonTransformViewPositionModel *model*)

8.29.2 Member Function Documentation

8.29.2.1 Vector3 PhotonTransformViewPositionControl.GetExtrapolatedPositionOffset ( )

Calculates an estimated position based on the last synchronized position, the time when the last position was received and the movement speed of the object

Returns

Estimated position of the remote object

8.29.2.2 Vector3 PhotonTransformViewPositionControl.GetNetworkPosition ( )

Gets the last position that was received through the network

Returns

8.29.2.3 void PhotonTransformViewPositionControl.OnPhotonSerializeView (Vector3 *currentPosition,* PhotonStream *stream,* PhotonMessageInfo *info*)

8.29.2.4 void PhotonTransformViewPositionControl.SetSynchronizedValues (Vector3 *speed,* float *turnSpeed*)

These values are synchronized to the remote objects if the interpolation mode or the extrapolation mode SynchronizeValues is used. Your movement script should pass on the current speed (in units/second) and turning speed (in angles/second) so the remote object can use them to predict the objects movement.

Parameters

speed	The current movement vector of the object in units/second.
turnSpeed	The current turn speed of the object in angles/second.

8.29.2.5 Vector3 PhotonTransformViewPositionControl.UpdatePosition (Vector3 *currentPosition*)

Calculates the new position based on the values setup in the inspector

Parameters

currentPosition	The current position.

Returns

The new position.

8.30 PhotonTransformViewPositionModel Class Reference

Public Types

- enum InterpolateOptions { InterpolateOptions.Disabled, InterpolateOptions.FixedSpeed, InterpolateOptions.EstimatedSpeed, InterpolateOptions.SynchronizeValues, InterpolateOptions.Lerp }
- enum ExtrapolateOptions { ExtrapolateOptions.Disabled, ExtrapolateOptions.SynchronizeValues, ExtrapolateOptions.EstimateSpeedAndTurn, ExtrapolateOptions.FixedSpeed }

Public Attributes

- bool SynchronizeEnabled
- bool TeleportEnabled = true
- float TeleportIfDistanceGreaterThan = 3f
- InterpolateOptions InterpolateOption = InterpolateOptions.EstimatedSpeed
- float InterpolateMoveTowardsSpeed = 1f
- float InterpolateLerpSpeed = 1f
- float InterpolateMoveTowardsAcceleration = 2

- float InterpolateMoveTowardsDeceleration = 2
- AnimationCurve InterpolateSpeedCurve
- ExtrapolateOptions ExtrapolateOption = ExtrapolateOptions.Disabled
- float ExtrapolateSpeed = 1f
- bool ExtrapolateIncludingRoundTripTime = true
- int ExtrapolateNumberOfStoredPositions = 1
- bool DrawErrorGizmo = true

8.30.1 Member Enumeration Documentation

8.30.1.1 enum PhotonTransformViewPositionModel.ExtrapolateOptions

Enumerator

Disabled

SynchronizeValues

EstimateSpeedAndTurn

FixedSpeed

8.30.1.2 enum PhotonTransformViewPositionModel.InterpolateOptions

Enumerator

Disabled

FixedSpeed

EstimatedSpeed

SynchronizeValues

Lerp

8.30.2 Member Data Documentation

8.30.2.1 bool PhotonTransformViewPositionModel.DrawErrorGizmo = true

8.30.2.2 bool PhotonTransformViewPositionModel.ExtrapolateIncludingRoundTripTime = true

8.30.2.3 int PhotonTransformViewPositionModel.ExtrapolateNumberOfStoredPositions = 1

8.30.2.4 ExtrapolateOptions PhotonTransformViewPositionModel.ExtrapolateOption = ExtrapolateOptions.Disabled

8.30.2.5 float PhotonTransformViewPositionModel.ExtrapolateSpeed = 1f

8.30.2.6 float PhotonTransformViewPositionModel.InterpolateLerpSpeed = 1f

8.30.2.7 float PhotonTransformViewPositionModel.InterpolateMoveTowardsAcceleration = 2

8.30.2.8 float PhotonTransformViewPositionModel.InterpolateMoveTowardsDeceleration = 2

8.30.2.9 float PhotonTransformViewPositionModel.InterpolateMoveTowardsSpeed = 1f

8.30.2.10 InterpolateOptions PhotonTransformViewPositionModel.InterpolateOption = InterpolateOptions.EstimatedSpeed

8.30.2.11 AnimationCurve PhotonTransformViewPositionModel.InterpolateSpeedCurve

Initial value:

```
= new AnimationCurve( new Keyframe[] {
                                                                    new Keyframe( -1, 0, 0, Mathf
      .Infinity ),
                                                                    new Keyframe( 0, 1, 0, 0 ),
                                                                    new Keyframe( 1, 1, 0, 1 ),
                                                                    new Keyframe( 4, 4, 1, 0 ) }
      )
```

8.30.2.12 bool PhotonTransformViewPositionModel.SynchronizeEnabled

8.30.2.13 bool PhotonTransformViewPositionModel.TeleportEnabled = true

8.30.2.14 float PhotonTransformViewPositionModel.TeleportIfDistanceGreaterThan = 3f

8.31 PhotonTransformViewRotationControl Class Reference

Public Member Functions

- PhotonTransformViewRotationControl (PhotonTransformViewRotationModel model)
- Quaternion GetRotation (Quaternion currentRotation)
- void OnPhotonSerializeView (Quaternion currentRotation, PhotonStream stream, PhotonMessageInfo info)

8.31.1 Constructor & Destructor Documentation

8.31.1.1 PhotonTransformViewRotationControl.PhotonTransformViewRotationControl (PhotonTransformViewRotationModel *model*)

8.31.2 Member Function Documentation

8.31.2.1 Quaternion PhotonTransformViewRotationControl.GetRotation (Quaternion *currentRotation*)

8.31.2.2 void PhotonTransformViewRotationControl.OnPhotonSerializeView (Quaternion *currentRotation,* PhotonStream *stream,* PhotonMessageInfo *info*)

8.32 PhotonTransformViewRotationModel Class Reference

Public Types

- enum InterpolateOptions { InterpolateOptions.Disabled, InterpolateOptions.RotateTowards, InterpolateOptions.Lerp }

Public Attributes

- bool SynchronizeEnabled
- InterpolateOptions InterpolateOption = InterpolateOptions.RotateTowards
- float InterpolateRotateTowardsSpeed = 180
- float InterpolateLerpSpeed = 5

8.32.1 Member Enumeration Documentation

8.32.1.1 enum PhotonTransformViewRotationModel.InterpolateOptions

Enumerator

Disabled

RotateTowards

Lerp

8.32.2 Member Data Documentation

8.32.2.1 float PhotonTransformViewRotationModel.InterpolateLerpSpeed = 5

8.32.2.2 InterpolateOptions PhotonTransformViewRotationModel.InterpolateOption = InterpolateOptions.RotateTowards

8.32.2.3 float PhotonTransformViewRotationModel.InterpolateRotateTowardsSpeed = 180

8.32.2.4 bool PhotonTransformViewRotationModel.SynchronizeEnabled

8.33 PhotonTransformViewScaleControl Class Reference

Public Member Functions

- PhotonTransformViewScaleControl (PhotonTransformViewScaleModel model)
- Vector3 GetScale (Vector3 currentScale)
- void OnPhotonSerializeView (Vector3 currentScale, PhotonStream stream, PhotonMessageInfo info)

8.33.1 Constructor & Destructor Documentation

8.33.1.1 PhotonTransformViewScaleControl.PhotonTransformViewScaleControl (PhotonTransformViewScaleModel *model*)

8.33.2 Member Function Documentation

8.33.2.1 Vector3 PhotonTransformViewScaleControl.GetScale (Vector3 *currentScale*)

8.33.2.2 void PhotonTransformViewScaleControl.OnPhotonSerializeView (Vector3 *currentScale,* PhotonStream *stream,* PhotonMessageInfo *info*)

8.34 PhotonTransformViewScaleModel Class Reference

Public Types

- enum InterpolateOptions { InterpolateOptions.Disabled, InterpolateOptions.MoveTowards, InterpolateOptions.Lerp }

Public Attributes

- bool SynchronizeEnabled
- InterpolateOptions InterpolateOption = InterpolateOptions.Disabled
- float InterpolateMoveTowardsSpeed = 1f
- float InterpolateLerpSpeed

8.34.1 Member Enumeration Documentation

8.34.1.1 enum PhotonTransformViewScaleModel.InterpolateOptions

Enumerator

Disabled

MoveTowards

Lerp

8.34.2 Member Data Documentation

8.34.2.1 float PhotonTransformViewScaleModel.InterpolateLerpSpeed

8.34.2.2 float PhotonTransformViewScaleModel.InterpolateMoveTowardsSpeed = 1f

8.34.2.3 InterpolateOptions PhotonTransformViewScaleModel.InterpolateOption = InterpolateOptions.Disabled

8.34.2.4 bool PhotonTransformViewScaleModel.SynchronizeEnabled

8.35 PhotonView Class Reference

PUN's NetworkView replacement class for networking. Use it like a NetworkView.

Inherits Photon.MonoBehaviour.

Public Member Functions

- void RequestOwnership ()

  Depending on the PhotonView's ownershipTransfer setting, any client can request to become owner of the PhotonView.
- void TransferOwnership (PhotonPlayer newOwner)

  Transfers the ownership of this PhotonView (and GameObject) to another player.
- void TransferOwnership (int newOwnerId)

  Transfers the ownership of this PhotonView (and GameObject) to another player.
- void SerializeView (PhotonStream stream, PhotonMessageInfo info)
- void DeserializeView (PhotonStream stream, PhotonMessageInfo info)
- void RefreshRpcMonoBehaviourCache ()

  Can be used to refesh the list of MonoBehaviours on this GameObject while PhotonNetwork.UseRpcMonoBehaviourCache is true.
- void RPC (string methodName, PhotonTargets target, params object[] parameters)

  Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).
- void RpcSecure (string methodName, PhotonTargets target, bool encrypt, params object[] parameters)

  Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).
- void RPC (string methodName, PhotonPlayer targetPlayer, params object[] parameters)

  Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).
- void RpcSecure (string methodName, PhotonPlayer targetPlayer, bool encrypt, params object[] parameters)

  Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).
- override string ToString ()

Static Public Member Functions

- static PhotonView Get (Component component)
- static PhotonView Get (GameObject gameObj)
- static PhotonView Find (int viewID)

Public Attributes

- int ownerId
- int group = 0
- int prefixBackup = -1
- Component observed
- ViewSynchronization synchronization
- OnSerializeTransform onSerializeTransformOption = OnSerializeTransform.PositionAndRotation
- OnSerializeRigidBody onSerializeRigidBodyOption = OnSerializeRigidBody.All
- OwnershipOption ownershipTransfer = OwnershipOption.Fixed

  Defines if ownership of this PhotonView is fixed, can be requested or simply taken.
- List< Component > ObservedComponents
- int instantiationId

Properties

- int prefix `[get, set]`
- object[] instantiationData `[get, set]`

  This is the instantiationData that was passed when calling PhotonNetwork.Instantiate (if that was used to spawn this prefab)*
- int viewID `[get, set]`

  The ID of the PhotonView. Identifies it in a networked game (per room).
- bool isSceneView `[get]`

  True if the PhotonView was loaded with the scene (game object) or instantiated with InstantiateSceneObject.
- PhotonPlayer owner `[get]`

  The owner of a PhotonView is the player who created the GameObject with that view. Objects in the scene don't have an owner.
- int OwnerActorNr `[get]`
- bool isOwnerActive `[get]`
- int CreatorActorNr `[get]`
- bool isMine `[get]`

  True if the PhotonView is "mine" and can be controlled by this client.

8.35.1 Detailed Description

PUN's NetworkView replacement class for networking. Use it like a NetworkView.

8.35.2 Member Function Documentation

8.35.2.1 void PhotonView.DeserializeView (PhotonStream *stream*, PhotonMessageInfo *info*)

8.35.2.2 static PhotonView PhotonView.Find (int *viewID*) `[static]`

8.35.2.3 static PhotonView PhotonView.Get (Component *component*) `[static]`

8.35.2.4 static PhotonView PhotonView.Get (GameObject *gameObj*) `[static]`

8.35.2.5 void PhotonView.RefreshRpcMonoBehaviourCache ( )

Can be used to refesh the list of MonoBehaviours on this GameObject while PhotonNetwork.UseRpcMonoBehaviourCache is true.

Set PhotonNetwork.UseRpcMonoBehaviourCache to true to enable the caching. Uses this.GetComponents<MonoBehaviour>() to get a list of MonoBehaviours to call RPCs on (potentially).

While PhotonNetwork.UseRpcMonoBehaviourCache is false, this method has no effect, because the list is refreshed when a RPC gets called.

8.35.2.6 void PhotonView.RequestOwnership ( )

Depending on the PhotonView's ownershipTransfer setting, any client can request to become owner of the PhotonView.

Requesting ownership can give you control over a PhotonView, if the ownershipTransfer setting allows that. The current owner might have to implement IPunCallbacks.OnOwnershipRequest to react to the ownership request.

The owner/controller of a PhotonView is also the client which sends position updates of the GameObject.

8.35.2.7 void PhotonView.RPC (string *methodName*, PhotonTargets *target*, params object[] *parameters*)

Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).

Remote Procedure Calls are an essential tool in making multiplayer games with PUN. It enables you to make every client in a room call a specific method.

RPC calls can target "All" or the "Others". Usually, the target "All" gets executed locally immediately after sending the RPC. The "∗ViaServer" options send the RPC to the server and execute it on this client when it's sent back. Of course, calls are affected by this client's lag and that of remote clients.

Each call automatically is routed to the same PhotonView (and GameObject) that was used on the originating client.

See: Remote Procedure Calls.

Parameters

methodName	The name of a fitting method that was has the RPC attribute.
target	The group of targets and the way the RPC gets sent.
parameters	The parameters that the RPC method has (must fit this call!).

8.35.2.8 void PhotonView.RPC (string *methodName*, PhotonPlayer *targetPlayer*, params object[] *parameters*)

Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).

Remote Procedure Calls are an essential tool in making multiplayer games with PUN. It enables you to make every client in a room call a specific method.

This method allows you to make an RPC calls on a specific player's client. Of course, calls are affected by this client's lag and that of remote clients.

Each call automatically is routed to the same PhotonView (and GameObject) that was used on the originating client.

See: Remote Procedure Calls.

Parameters

methodName	The name of a fitting method that was has the RPC attribute.
targetPlayer	The group of targets and the way the RPC gets sent.
parameters	The parameters that the RPC method has (must fit this call!).

8.35.2.9 void PhotonView.RpcSecure (string *methodName*, PhotonTargets *target*, bool *encrypt*, params object[] *parameters*)

Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).

Remote Procedure Calls are an essential tool in making multiplayer games with PUN. It enables you to make every client in a room call a specific method.

RPC calls can target "All" or the "Others". Usually, the target "All" gets executed locally immediately after sending the RPC. The "∗ViaServer" options send the RPC to the server and execute it on this client when it's sent back. Of course, calls are affected by this client's lag and that of remote clients.

Each call automatically is routed to the same PhotonView (and GameObject) that was used on the originating client.

See: Remote Procedure Calls.

param name="methodName"＞The name of a fitting method that was has the RPC attribute.

param name="target"＞The group of targets and the way the RPC gets sent.

param name="encrypt"＞

param name="parameters"＞The parameters that the RPC method has (must fit this call!).

8.35.2.10 void PhotonView.RpcSecure (string *methodName,* PhotonPlayer *targetPlayer,* bool *encrypt,* params object[] *parameters*)

Call a RPC method of this GameObject on remote clients of this room (or on all, inclunding this client).

Remote Procedure Calls are an essential tool in making multiplayer games with PUN. It enables you to make every client in a room call a specific method.

This method allows you to make an RPC calls on a specific player's client. Of course, calls are affected by this client's lag and that of remote clients.

Each call automatically is routed to the same PhotonView (and GameObject) that was used on the originating client.

See: Remote Procedure Calls.

param name="methodName"＞The name of a fitting method that was has the RPC attribute.

param name="targetPlayer"＞The group of targets and the way the RPC gets sent.

param name="encrypt"＞

param name="parameters"＞The parameters that the RPC method has (must fit this call!).

8.35.2.11 void PhotonView.SerializeView (PhotonStream *stream,* PhotonMessageInfo *info*)

8.35.2.12 override string PhotonView.ToString ( )

8.35.2.13 void PhotonView.TransferOwnership (PhotonPlayer *newOwner*)

Transfers the ownership of this PhotonView (and GameObject) to another player.

The owner/controller of a PhotonView is also the client which sends position updates of the GameObject.

8.35.2.14 void PhotonView.TransferOwnership (int *newOwnerId*)

Transfers the ownership of this PhotonView (and GameObject) to another player.

The owner/controller of a PhotonView is also the client which sends position updates of the GameObject.

8.35.3 Member Data Documentation

8.35.3.1 int PhotonView.group = 0

8.35.3.2 int PhotonView.instantiationId

8.35.3.3 Component PhotonView.observed

8.35.3.4 List<Component> PhotonView.ObservedComponents

8.35.3.5 OnSerializeRigidBody PhotonView.onSerializeRigidBodyOption = OnSerializeRigidBody.All

8.35.3.6 OnSerializeTransform PhotonView.onSerializeTransformOption = OnSerializeTransform.PositionAndRotation

8.35.3.7 int PhotonView.ownerId

8.35.3.8 OwnershipOption PhotonView.ownershipTransfer = OwnershipOption.Fixed

Defines if ownership of this PhotonView is fixed, can be requested or simply taken.

Note that you can't edit this value at runtime. The options are described in enum OwnershipOption. The current owner has to implement IPunCallbacks.OnOwnershipRequest to react to the ownership request.

8.35.3.9 int PhotonView.prefixBackup = -1

8.35.3.10 ViewSynchronization PhotonView.synchronization

8.35.4 Property Documentation

8.35.4.1 int PhotonView.CreatorActorNr [get]

8.35.4.2 object [] PhotonView.instantiationData [get], [set]

This is the instantiationData that was passed when calling PhotonNetwork.Instantiate* (if that was used to spawn this prefab)

8.35.4.3 bool PhotonView.isMine [get]

True if the PhotonView is "mine" and can be controlled by this client.

PUN has an ownership concept that defines who can control and destroy each PhotonView. True in case the owner matches the local PhotonPlayer. True if this is a scene photonview on the Master client.

8.35.4.4 bool PhotonView.isOwnerActive [get]

8.35.4.5 bool PhotonView.isSceneView [get]

True if the PhotonView was loaded with the scene (game object) or instantiated with InstantiateSceneObject.

Scene objects are not owned by a particular player but belong to the scene. Thus they don't get destroyed when their creator leaves the game and the current Master Client can control them (whoever that is). The ownerId is 0 (player IDs are 1 and up).

8.35.4.6 PhotonPlayer PhotonView.owner [get]

The owner of a PhotonView is the player who created the GameObject with that view. Objects in the scene don't have an owner.

The owner/controller of a PhotonView is also the client which sends position updates of the GameObject.

Ownership can be transferred to another player with PhotonView.TransferOwnership or any player can request ownership by calling the PhotonView's RequestOwnership method. The current owner has to implement IPunCallbacks.OnOwnershipRequest to react to the ownership request.

8.35.4.7 int PhotonView.OwnerActorNr [get]

8.35.4.8 int PhotonView.prefix [get],[set]

8.35.4.9 int PhotonView.viewID [get],[set]

The ID of the PhotonView. Identifies it in a networked game (per room).

See: Network Instantiation

8.36 PingMonoEditor Class Reference

Uses C# Socket class from System.Net.Sockets (as Unity usually does).

Inherits PhotonPing.

Public Member Functions

- override bool StartPing (string ip)

  Sends a "Photon Ping" to a server.
- override bool Done ()
- override void Dispose ()

8.36.1 Detailed Description

Uses C# Socket class from System.Net.Sockets (as Unity usually does).

Incompatible with Windows 8 Store/Phone API.

8.36.2 Member Function Documentation

8.36.2.1 override void PingMonoEditor.Dispose ( )

8.36.2.2 override bool PingMonoEditor.Done ( )

8.36.2.3 override bool PingMonoEditor.StartPing (string *ip*)

Sends a "Photon Ping" to a server.

Parameters

ip	Address in IPv4 or IPv6 format. An address containing a '.' will be interpretet as IPv4.

Returns

True if the Photon Ping could be sent.

8.37 Photon.PunBehaviour Class Reference

This class provides a .photonView and all callbacks/events that PUN can call. Override the events/methods you want to use.

Inherits Photon.MonoBehaviour, and IPunCallbacks.

Public Member Functions

- virtual void OnConnectedToPhoton ()

  Called when the initial connection got established but before you can use the server. OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.
- virtual void OnLeftRoom ()

  Called when the local user/client left a room.
- virtual void OnMasterClientSwitched (PhotonPlayer newMasterClient)

  Called after switching to a new MasterClient when the current one leaves.
- virtual void OnPhotonCreateRoomFailed (object[] codeAndMsg)

  Called when a CreateRoom() call failed. The parameter provides ErrorCode and message (as array).
- virtual void OnPhotonJoinRoomFailed (object[] codeAndMsg)

  Called when a JoinRoom() call failed. The parameter provides ErrorCode and message (as array).
- virtual void OnCreatedRoom ()

  Called when this client created a room and entered it. OnJoinedRoom() will be called as well.
- virtual void OnJoinedLobby ()

  Called on entering a lobby on the Master Server. The actual room-list updates will call OnReceivedRoomListUpdate().
- virtual void OnLeftLobby ()

  Called after leaving a lobby.
- virtual void OnFailedToConnectToPhoton (DisconnectCause cause)

  Called if a connect call to the Photon server failed before the connection was established, followed by a call to OnDisconnectedFromPhoton().
- virtual void OnDisconnectedFromPhoton ()

  Called after disconnecting from the Photon server.
- virtual void OnConnectionFail (DisconnectCause cause)

  Called when something causes the connection to fail (after it was established), followed by a call to OnDisconnectedFromPhoton().
- virtual void OnPhotonInstantiate (PhotonMessageInfo info)

  Called on all scripts on a GameObject (and children) that have been Instantiated using PhotonNetwork.Instantiate.
- virtual void OnReceivedRoomListUpdate ()

  Called for any update of the room-listing while in a lobby (PhotonNetwork.insideLobby) on the Master Server.
- virtual void OnJoinedRoom ()

  Called when entering a room (by creating or joining it). Called on all clients (including the Master Client).
- virtual void OnPhotonPlayerConnected (PhotonPlayer newPlayer)

  Called when a remote player entered the room. This PhotonPlayer is already added to the playerlist at this time.
- virtual void OnPhotonPlayerDisconnected (PhotonPlayer otherPlayer)

  Called when a remote player left the room. This PhotonPlayer is already removed from the playerlist at this time.
- virtual void OnPhotonRandomJoinFailed (object[] codeAndMsg)

  Called when a JoinRandom() call failed. The parameter provides ErrorCode and message.
- virtual void OnConnectedToMaster ()

  Called after the connection to the master is established and authenticated but only when PhotonNetwork.autoJoinLobby is false.
- virtual void OnPhotonMaxCccuReached ()

  Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting.
- virtual void OnPhotonCustomRoomPropertiesChanged (Hashtable propertiesThatChanged)

  Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.SetCustomProperties.
- virtual void OnPhotonPlayerPropertiesChanged (object[] playerAndUpdatedProps)

  Called when custom player-properties are changed. Player and the changed properties are passed as object[].
- virtual void OnUpdatedFriendList ()

  Called when the server sent the response to a FindFriends request and updated PhotonNetwork.Friends.
- virtual void OnCustomAuthenticationFailed (string debugMessage)

Called when the custom authentication failed. Followed by disconnect!

- virtual void OnCustomAuthenticationResponse (Dictionary< string, object > data)

  Called when your Custom Authentication service responds with additional data.

- virtual void OnWebRpcResponse (OperationResponse response)

  Called by PUN when the response to a WebRPC is available. See PhotonNetwork.WebRPC.

- virtual void OnOwnershipRequest (object[] viewAndPlayer)

  Called when another player requests ownership of a PhotonView from you (the current owner).

- virtual void OnLobbyStatisticsUpdate ()

  Called when the Master Server sent an update for the Lobby Statistics, updating PhotonNetwork.LobbyStatistics.

Additional Inherited Members

8.37.1 Detailed Description

This class provides a .photonView and all callbacks/events that PUN can call. Override the events/methods you want to use.

By extending this class, you can implement individual methods as override.

Visual Studio and MonoDevelop should provide the list of methods when you begin typing "override". **Your implementation does not have to call "base.method()".**

This class implements IPunCallbacks, which is used as definition of all PUN callbacks. Don't implement IPunCallbacks in your classes. Instead, implent PunBehaviour or individual methods.

8.37.2 Member Function Documentation

8.37.2.1 virtual void Photon.PunBehaviour.OnConnectedToMaster ( ) [virtual]

Called after the connection to the master is established and authenticated but only when PhotonNetwork.autoJoinLobby is false.

If you set PhotonNetwork.autoJoinLobby to true, OnJoinedLobby() will be called instead of this.

You can join rooms and create them even without being in a lobby. The default lobby is used in that case. The list of available rooms won't become available unless you join a lobby via PhotonNetwork.joinLobby.

Implements IPunCallbacks.

8.37.2.2 virtual void Photon.PunBehaviour.OnConnectedToPhoton ( ) [virtual]

Called when the initial connection got established but before you can use the server. OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.

This callback is only useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnFailedToConnectToPhoton() and OnDisconnectedFromPhoton().

OnJoinedLobby() or OnConnectedToMaster() are called when PUN is ready.

When this is called, the low level connection is established and PUN will send your AppId, the user, etc in the background. This is not called for transitions from the masterserver to game servers.

Implements IPunCallbacks.

8.37.2.3 virtual void Photon.PunBehaviour.OnConnectionFail (DisconnectCause *cause*) [virtual]

Called when something causes the connection to fail (after it was established), followed by a call to OnDisconnectedFromPhoton().

If the server could not be reached in the first place, OnFailedToConnectToPhoton is called instead. The reason for the error is provided as DisconnectCause.

Implements IPunCallbacks.

8.37.2.4 virtual void Photon.PunBehaviour.OnCreatedRoom ( ) [virtual]

Called when this client created a room and entered it. OnJoinedRoom() will be called as well.

This callback is only called on the client which created a room (see PhotonNetwork.CreateRoom).

As any client might close (or drop connection) anytime, there is a chance that the creator of a room does not execute OnCreatedRoom.

If you need specific room properties or a "start signal", it is safer to implement OnMasterClientSwitched() and to make the new MasterClient check the room's state.

Implements IPunCallbacks.

8.37.2.5 virtual void Photon.PunBehaviour.OnCustomAuthenticationFailed (string *debugMessage*) [virtual]

Called when the custom authentication failed. Followed by disconnect!

Custom Authentication can fail due to user-input, bad tokens/secrets. If authentication is successful, this method is not called. Implement OnJoinedLobby() or OnConnectedToMaster() (as usual).

During development of a game, it might also fail due to wrong configuration on the server side. In those cases, logging the debugMessage is very important.

Unless you setup a custom authentication service for your app (in the Dashboard), this won't be called!

Parameters

debugMessage	Contains a debug message why authentication failed. This has to be fixed during development time.

Implements IPunCallbacks.

8.37.2.6 virtual void Photon.PunBehaviour.OnCustomAuthenticationResponse (Dictionary< string, object > *data*) [virtual]

Called when your Custom Authentication service responds with additional data.

Custom Authentication services can include some custom data in their response. When present, that data is made available in this callback as Dictionary. While the keys of your data have to be strings, the values can be either string or a number (in Json). You need to make extra sure, that the value type is the one you expect. Numbers become (currently) int64.

Example: void OnCustomAuthenticationResponse(Dictionary<string, object> data) { ... }

https://doc.photonengine.com/en/realtime/current/reference/custom-authentication

Implements IPunCallbacks.

8.37.2.7 virtual void Photon.PunBehaviour.OnDisconnectedFromPhoton ( ) [virtual]

Called after disconnecting from the Photon server.

In some cases, other callbacks are called before OnDisconnectedFromPhoton is called. Examples: OnConnectionFail() and OnFailedToConnectToPhoton().

Implements IPunCallbacks.

8.37.2.8 virtual void Photon.PunBehaviour.OnFailedToConnectToPhoton (DisconnectCause *cause*) [virtual]

Called if a connect call to the Photon server failed before the connection was established, followed by a call to OnDisconnectedFromPhoton().

This is called when no connection could be established at all. It differs from OnConnectionFail, which is called when an existing connection fails.

Implements IPunCallbacks.

8.37.2.9 virtual void Photon.PunBehaviour.OnJoinedLobby ( ) [virtual]

Called on entering a lobby on the Master Server. The actual room-list updates will call OnReceivedRoomList↩Update().

Note: When PhotonNetwork.autoJoinLobby is false, OnConnectedToMaster() will be called and the room list won't become available.

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify). The room list gets available when OnReceivedRoomListUpdate() gets called after OnJoinedLobby().

Implements IPunCallbacks.

8.37.2.10 virtual void Photon.PunBehaviour.OnJoinedRoom ( ) [virtual]

Called when entering a room (by creating or joining it). Called on all clients (including the Master Client).

This method is commonly used to instantiate player characters. If a match has to be started "actively", you can call an PunRPC triggered by a user's button-press or a timer.

When this is called, you can usually already access the existing players in the room via PhotonNetwork.playerList. Also, all custom properties should be already available as Room.customProperties. Check Room.playerCount to find out if enough players are in the room to start playing.

Implements IPunCallbacks.

8.37.2.11 virtual void Photon.PunBehaviour.OnLeftLobby ( ) [virtual]

Called after leaving a lobby.

When you leave a lobby, CreateRoom and JoinRandomRoom automatically refer to the default lobby.

Implements IPunCallbacks.

8.37.2.12 virtual void Photon.PunBehaviour.OnLeftRoom ( ) [virtual]

Called when the local user/client left a room.

When leaving a room, PUN brings you back to the Master Server. Before you can use lobbies and join or create rooms, OnJoinedLobby() or OnConnectedToMaster() will get called again.

Implements IPunCallbacks.

8.37.2.13 virtual void Photon.PunBehaviour.OnLobbyStatisticsUpdate ( ) [virtual]

Called when the Master Server sent an update for the Lobby Statistics, updating PhotonNetwork.LobbyStatistics.

This callback has two preconditions: EnableLobbyStatistics must be set to true, before this client connects. And the client has to be connected to the Master Server, which is providing the info about lobbies.

Implements IPunCallbacks.

8.37.2.14 virtual void Photon.PunBehaviour.OnMasterClientSwitched (PhotonPlayer *newMasterClient*) [virtual]

Called after switching to a new MasterClient when the current one leaves.

This is not called when this client enters a room. The former MasterClient is still in the player list when this method get called.

Implements IPunCallbacks.

8.37.2.15 virtual void Photon.PunBehaviour.OnOwnershipRequest (object[] *viewAndPlayer*) [virtual]

Called when another player requests ownership of a PhotonView from you (the current owner).

The parameter viewAndPlayer contains:

PhotonView view = viewAndPlayer[0] as PhotonView;

PhotonPlayer requestingPlayer = viewAndPlayer[1] as PhotonPlayer;

Parameters

viewAndPlayer	The PhotonView is viewAndPlayer[0] and the requesting player is viewAndPlayer[1].

Implements IPunCallbacks.

8.37.2.16 virtual void Photon.PunBehaviour.OnPhotonCreateRoomFailed (object[] *codeAndMsg*) [virtual]

Called when a CreateRoom() call failed. The parameter provides ErrorCode and message (as array).

Most likely because the room name is already in use (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is a short ErrorCode and codeAndMsg[1] is a string debug msg.

Implements IPunCallbacks.

8.37.2.17 virtual void Photon.PunBehaviour.OnPhotonCustomRoomPropertiesChanged (Hashtable *propertiesThatChanged*) [virtual]

Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.SetCustomProperties.

Since v1.25 this method has one parameter: Hashtable propertiesThatChanged.
Changing properties must be done by Room.SetCustomProperties, which causes this callback locally, too.

Parameters

propertiesThat↩ Changed	

Implements IPunCallbacks.

8.37.2.18 virtual void Photon.PunBehaviour.OnPhotonInstantiate (PhotonMessageInfo *info*) [virtual]

Called on all scripts on a GameObject (and children) that have been Instantiated using PhotonNetwork.Instantiate.

PhotonMessageInfo parameter provides info about who created the object and when (based off PhotonNetworking.time).

Implements IPunCallbacks.

8.37.2.19 virtual void Photon.PunBehaviour.OnPhotonJoinRoomFailed (object[] *codeAndMsg*) [virtual]

Called when a JoinRoom() call failed. The parameter provides ErrorCode and message (as array).

Most likely error is that the room does not exist or the room is full (some other client was faster than you). PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg.

Implements IPunCallbacks.

8.37.2.20 virtual void Photon.PunBehaviour.OnPhotonMaxCccuReached ( ) [virtual]

Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting.

When this happens, the user might try again later. You can't create or join rooms in OnPhotonMaxCcuReached(), cause the client will be disconnecting. You can raise the CCU limits with a new license (when you host yourself) or extended subscription (when using the Photon Cloud). The Photon Cloud will mail you when the CCU limit was reached. This is also visible in the Dashboard (webpage).

Implements IPunCallbacks.

8.37.2.21 virtual void Photon.PunBehaviour.OnPhotonPlayerConnected (PhotonPlayer *newPlayer*) [virtual]

Called when a remote player entered the room. This PhotonPlayer is already added to the playerlist at this time.

If your game starts with a certain number of players, this callback can be useful to check the Room.playerCount and find out if you can start.

Implements IPunCallbacks.

8.37.2.22 virtual void Photon.PunBehaviour.OnPhotonPlayerDisconnected (PhotonPlayer *otherPlayer*) [virtual]

Called when a remote player left the room. This PhotonPlayer is already removed from the playerlist at this time.

When your client calls PhotonNetwork.leaveRoom, PUN will call this method on the remaining clients. When a remote client drops connection or gets closed, this callback gets executed. after a timeout of several seconds.

Implements IPunCallbacks.

8.37.2.23 virtual void Photon.PunBehaviour.OnPhotonPlayerPropertiesChanged (object[] *playerAndUpdatedProps*) [virtual]

Called when custom player-properties are changed. Player and the changed properties are passed as object[].

Since v1.25 this method has one parameter: object[] playerAndUpdatedProps, which contains two entries.
[0] is the affected PhotonPlayer.
[1] is the Hashtable of properties that changed.

We are using a object[] due to limitations of Unity's GameObject.SendMessage (which has only one optional parameter).

Changing properties must be done by PhotonPlayer.SetCustomProperties, which causes this callback locally, too.

Example:

```
void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) {
    PhotonPlayer player = playerAndUpdatedProps[0] as PhotonPlayer;
    Hashtable props = playerAndUpdatedProps[1] as Hashtable;
```

```
    //...
}
```

Parameters

playerAnd↩ UpdatedProps	Contains PhotonPlayer and the properties that changed See remarks.

Implements IPunCallbacks.

8.37.2.24 virtual void Photon.PunBehaviour.OnPhotonRandomJoinFailed (object[] *codeAndMsg*) [virtual]

Called when a JoinRandom() call failed. The parameter provides ErrorCode and message.

Most likely all rooms are full or no rooms are available.
When using multiple lobbies (via JoinLobby or TypedLobby), another lobby might have more/fitting rooms.
PUN logs some info if the PhotonNetwork.logLevel is >= PhotonLogLevel.Informational.

Parameters

codeAndMsg	codeAndMsg[0] is short ErrorCode. codeAndMsg[1] is string debug msg.

Implements IPunCallbacks.

8.37.2.25 virtual void Photon.PunBehaviour.OnReceivedRoomListUpdate ( ) [virtual]

Called for any update of the room-listing while in a lobby (PhotonNetwork.insideLobby) on the Master Server.

PUN provides the list of rooms by PhotonNetwork.GetRoomList().
Each item is a RoomInfo which might include custom properties (provided you defined those as lobby-listed when creating a room).

Not all types of lobbies provide a listing of rooms to the client. Some are silent and specialized for server-side matchmaking.

Implements IPunCallbacks.

8.37.2.26 virtual void Photon.PunBehaviour.OnUpdatedFriendList ( ) [virtual]

Called when the server sent the response to a FindFriends request and updated PhotonNetwork.Friends.

The friends list is available as PhotonNetwork.Friends, listing name, online state and the room a user is in (if any).

Implements IPunCallbacks.

8.37.2.27 virtual void Photon.PunBehaviour.OnWebRpcResponse (OperationResponse *response*) [virtual]

Called by PUN when the response to a WebRPC is available. See PhotonNetwork.WebRPC.

Important: The response.ReturnCode is 0 if Photon was able to reach your web-service. The content of the response is what your web-service sent. You can create a WebResponse instance from it. Example: WebRpc↩ Response webResponse = new WebRpcResponse(operationResponse);

Please note: Class OperationResponse is in a namespace which needs to be "used": using ExitGames.Client.↩ Photon; // includes OperationResponse (and other classes)

The OperationResponse.ReturnCode by Photon is:

```
 0 for "OK"
-3 for "Web-Service not configured" (see Dashboard / WebHooks)
-5 for "Web-Service does now have RPC path/name" (at least for Azure)
```

Implements IPunCallbacks.

8.38 PunRPC Class Reference

Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

Inherits Attribute.

8.38.1 Detailed Description

Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

8.39 RaiseEventOptions Class Reference

Aggregates several less-often used options for operation RaiseEvent. See field descriptions for usage details.

Public Attributes

- EventCaching CachingOption

  Defines if the server should simply send the event, put it in the cache or remove events that are like this one.
- byte InterestGroup

  The number of the Interest Group to send this to. 0 goes to all users but to get 1 and up, clients must subscribe to the group first.
- int[] TargetActors

  A list of PhotonPlayer.IDs to send this event to. You can implement events that just go to specific users this way.
- ReceiverGroup Receivers

  Sends the event to All, MasterClient or Others (default). Be careful with MasterClient, as the client might disconnect before it got the event and it gets lost.
- byte SequenceChannel

  Events are ordered per "channel". If you have events that are independent of others, they can go into another sequence or channel.
- bool ForwardToWebhook

  Events can be forwarded to Webhooks, which can evaluate and use the events to follow the game's state.
- bool Encrypt

Static Public Attributes

- static readonly RaiseEventOptions Default = new RaiseEventOptions()

  Default options: CachingOption: DoNotCache, InterestGroup: 0, targetActors: null, receivers: Others, sequence↩ Channel: 0.

8.39.1 Detailed Description

Aggregates several less-often used options for operation RaiseEvent. See field descriptions for usage details.

8.39.2 Member Data Documentation

8.39.2.1 EventCaching RaiseEventOptions.CachingOption

Defines if the server should simply send the event, put it in the cache or remove events that are like this one.

When using option: SliceSetIndex, SlicePurgeIndex or SlicePurgeUpToIndex, set a CacheSliceIndex. All other options except SequenceChannel get ignored.

8.39.2.2 readonly RaiseEventOptions RaiseEventOptions.Default = new RaiseEventOptions() [static]

Default options: CachingOption: DoNotCache, InterestGroup: 0, targetActors: null, receivers: Others, sequence↩ Channel: 0.

8.39.2.3 bool RaiseEventOptions.Encrypt

8.39.2.4 bool RaiseEventOptions.ForwardToWebhook

Events can be forwarded to Webhooks, which can evaluate and use the events to follow the game's state.

8.39.2.5 byte RaiseEventOptions.InterestGroup

The number of the Interest Group to send this to. 0 goes to all users but to get 1 and up, clients must subscribe to the group first.

8.39.2.6 ReceiverGroup RaiseEventOptions.Receivers

Sends the event to All, MasterClient or Others (default). Be careful with MasterClient, as the client might disconnect before it got the event and it gets lost.

8.39.2.7 byte RaiseEventOptions.SequenceChannel

Events are ordered per "channel". If you have events that are independent of others, they can go into another sequence or channel.

8.39.2.8 int [ ] RaiseEventOptions.TargetActors

A list of PhotonPlayer.IDs to send this event to. You can implement events that just go to specific users this way.

8.40 Region Class Reference

Public Member Functions

- override string ToString ()

Static Public Member Functions

- static CloudRegionCode Parse (string codeAsString)

Public Attributes

- CloudRegionCode Code
- string HostAndPort
- int Ping

8.40.1 Member Function Documentation

8.40.1.1 static CloudRegionCode Region.Parse (string *codeAsString*) [static]

8.40.1.2 override string Region.ToString ( )

8.40.2 Member Data Documentation

8.40.2.1 CloudRegionCode Region.Code

8.40.2.2 string Region.HostAndPort

8.40.2.3 int Region.Ping

8.41 Room Class Reference

This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a RoomInfo and you can close or hide "your" room.

Inherits RoomInfo.

Public Member Functions

- void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues=null, bool web↩ Forward=false)

  Updates the current room's Custom Properties with new/updated key-values.
- void SetPropertiesListedInLobby (string[] propsListedInLobby)

  Enables you to define the properties available in the lobby if not all properties are needed to pick a room.
- void ClearExpectedUsers ()

  Attempts to remove all current expected users from the server's Slot Reservation list.
- override string ToString ()

  Returns a summary of this Room instance as string.
- new string ToStringFull ()

  Returns a summary of this Room instance as longer string, including Custom Properties.

Properties

- new string name [get, set]

  The name of a room. Unique identifier (per Loadbalancing group) for a room/match.
- new bool open [get, set]

  Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "visible" to not list the room.
- new bool visible [get, set]

  Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.
- string[] propertiesListedInLobby [get, set]

  A list of custom properties that should be forwarded to the lobby and listed there.
- bool autoCleanUp [get]

  Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.
- new int maxPlayers [get, set]

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

- new int playerCount [get]

  Count of players in this room.

- string[] expectedUsers [get]

  List of users who are expected to join this room. In matchmaking, Photon blocks a slot for each of these UserIDs out of the MaxPlayers.

Additional Inherited Members

8.41.1 Detailed Description

This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a RoomInfo and you can close or hide "your" room.

8.41.2 Member Function Documentation

8.41.2.1 void Room.ClearExpectedUsers ( )

Attempts to remove all current expected users from the server's Slot Reservation list.

Note that this operation can conflict with new/other users joining. They might be adding users to the list of expected users before or after this client called ClearExpectedUsers.

This room's expectedUsers value will update, when the server sends a successful update.

Internals: This methods wraps up setting the ExpectedUsers property of a room.

8.41.2.2 void Room.SetCustomProperties (Hashtable *propertiesToSet*, Hashtable *expectedValues* = `null`, bool *webForward* = `false`)

Updates the current room's Custom Properties with new/updated key-values.

Custom Properties are a key-value set (Hashtable) which is available to all players in a room. They can relate to the room or individual players and are useful when only the current value of something is of interest. For example: The map of a room. All keys must be strings.

The Room and the PhotonPlayer class both have SetCustomProperties methods. Also, both classes offer access to current key-values by: customProperties.

Always use SetCustomProperties to change values. To reduce network traffic, set only values that actually changed. New properties are added, existing values are updated. Other values will not be changed, so only provide values that changed or are new.

To delete a named (custom) property of this room, use null as value.

Locally, SetCustomProperties will update it's cache without delay. Other clients are updated through Photon (the server) with a fitting operation.

Check and Swap

SetCustomProperties have the option to do a server-side Check-And-Swap (CAS): Values only get updated if the expected values are correct. The expectedValues can be different key/values than the propertiesToSet. So you can check some key and set another key's value (if the check succeeds).

If the client's knowledge of properties is wrong or outdated, it can't set values with CAS. This can be useful to keep players from concurrently setting values. For example: If all players try to pickup some card or item, only one should get it. With CAS, only the first SetProperties gets executed server-side and any other (sent at the same time) fails.

The server will broadcast successfully changed values and the local "cache" of customProperties only gets updated after a roundtrip (if anything changed).

You can do a "webForward": Photon will send the changed properties to a WebHook defined for your application.

OfflineMode

While PhotonNetwork.offlineMode is true, the expectedValues and webForward parameters are ignored. In Offline↩ Mode, the local customProperties values are immediately updated (without the roundtrip).

Parameters

propertiesToSet	The new properties to be set.
expectedValues	At least one property key/value set to check server-side. Key and value must be correct. Ignored in OfflineMode.
webForward	Set to true, to forward the set properties to a WebHook, defined for this app (in Dashboard). Ignored in OfflineMode.

8.41.2.3 void Room.SetPropertiesListedInLobby (string[] *propsListedInLobby*)

Enables you to define the properties available in the lobby if not all properties are needed to pick a room.

It makes sense to limit the amount of properties sent to users in the lobby as this improves speed and stability.

Parameters

propsListedIn↩ Lobby	An array of custom room property names to forward to the lobby.

8.41.2.4 override string Room.ToString ( )

Returns a summary of this Room instance as string.

Returns

Summary of this Room instance.

8.41.2.5 new string Room.ToStringFull ( )

Returns a summary of this Room instance as longer string, including Custom Properties.

Returns

Summary of this Room instance.

8.41.3 Property Documentation

8.41.3.1 bool Room.autoCleanUp [get]

Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.

8.41.3.2 string [] Room.expectedUsers [get]

List of users who are expected to join this room. In matchmaking, Photon blocks a slot for each of these UserIDs out of the MaxPlayers.

The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages. Define expected players in the PhotonNetwork methods: CreateRoom, JoinRoom and JoinOrCreateRoom.

8.41.3.3 new int Room.maxPlayers [get], [set]

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

8.41.3.4 new string Room.name [get], [set]

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

8.41.3.5 new bool Room.open [get], [set]

Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "visible" to not list the room.

8.41.3.6 new int Room.playerCount [get]

Count of players in this room.

8.41.3.7 string [] Room.propertiesListedInLobby [get], [set]

A list of custom properties that should be forwarded to the lobby and listed there.

8.41.3.8 new bool Room.visible [get], [set]

Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

8.42 RoomInfo Class Reference

A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc).

Inherited by Room.

Public Member Functions

- override bool Equals (object other)

  Makes RoomInfo comparable (by name).
- override int GetHashCode ()

  Accompanies Equals, using the name's HashCode as return.
- override string ToString ()

  Simple printingin method.
- string ToStringFull ()

  Simple printingin method.

Protected Attributes

- byte maxPlayersField = 0

  Backing field for property.
- string[] expectedUsersField

  Backing field for property.
- bool openField = true

  Backing field for property.
- bool visibleField = true

  Backing field for property.
- bool autoCleanUpField = PhotonNetwork.autoCleanUpPlayerObjects

  Backing field for property. False unless the GameProperty is set to true (else it's not sent).
- string nameField

  Backing field for property.

Properties

- bool removedFromList `[get, set]`

  Used internally in lobby, to mark rooms that are no longer listed.
- Hashtable customProperties `[get]`

  Read-only "cache" of custom properties of a room. Set via Room.SetCustomProperties (not available for RoomInfo class!).
- string name `[get]`

  The name of a room. Unique identifier (per Loadbalancing group) for a room/match.
- int playerCount `[get, set]`

  Only used internally in lobby, to display number of players in room (while you're not in).
- bool isLocalClientInside `[get, set]`

  State if the local client is already in the game or still going to join it on gameserver (in lobby always false).
- byte maxPlayers `[get]`

  Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.
- bool open `[get]`

  Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.
- bool visible `[get]`

  Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

8.42.1 Detailed Description

A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc).

This class resembles info about available rooms, as sent by the Master server's lobby. Consider all values as readonly. None are synced (only updated by events by server).

8.42.2 Member Function Documentation

8.42.2.1 override bool RoomInfo.Equals (object *other*)

Makes RoomInfo comparable (by name).

8.42.2.2 override int RoomInfo.GetHashCode ( )

Accompanies Equals, using the name's HashCode as return.

Returns

8.42.2.3 override string RoomInfo.ToString ( )

Simple printingin method.

Returns

Summary of this RoomInfo instance.

8.42.2.4 string RoomInfo.ToStringFull ( )

Simple printingin method.

Returns

Summary of this RoomInfo instance.

8.42.3 Member Data Documentation

8.42.3.1 bool RoomInfo.autoCleanUpField = PhotonNetwork.autoCleanUpPlayerObjects [protected]

Backing field for property. False unless the GameProperty is set to true (else it's not sent).

8.42.3.2 string [] RoomInfo.expectedUsersField [protected]

Backing field for property.

8.42.3.3 byte RoomInfo.maxPlayersField = 0 [protected]

Backing field for property.

8.42.3.4 string RoomInfo.nameField [protected]

Backing field for property.

8.42.3.5 bool RoomInfo.openField = true [protected]

Backing field for property.

8.42.3.6 bool RoomInfo.visibleField = true [protected]

Backing field for property.

8.42.4 Property Documentation

8.42.4.1 Hashtable RoomInfo.customProperties [get]

Read-only "cache" of custom properties of a room. Set via Room.SetCustomProperties (not available for RoomInfo class!).

All keys are string-typed and the values depend on the game/application.

Room.SetCustomProperties

8.42.4.2 bool RoomInfo.isLocalClientInside [get], [set]

State if the local client is already in the game or still going to join it on gameserver (in lobby always false).

8.42.4.3 byte RoomInfo.maxPlayers [get]

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

As part of RoomInfo this can't be set. As part of a Room (which the player joined), the setter will update the server and all clients.

8.42.4.4 string RoomInfo.name [get]

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

8.42.4.5 bool RoomInfo.open [get]

Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.

As part of RoomInfo this can't be set. As part of a Room (which the player joined), the setter will update the server and all clients.

8.42.4.6 int RoomInfo.playerCount [get], [set]

Only used internally in lobby, to display number of players in room (while you're not in).

8.42.4.7 bool RoomInfo.removedFromList [get], [set]

Used internally in lobby, to mark rooms that are no longer listed.

8.42.4.8 bool RoomInfo.visible [get]

Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

As part of RoomInfo this can't be set. As part of a Room (which the player joined), the setter will update the server and all clients.

8.43 RoomOptions Class Reference

Wraps up common room properties needed when you create rooms. Read the individual entries for more details.

Public Attributes

- byte maxPlayers

  Max number of players that can be in the room at any time. 0 means "no limit".
- int PlayerTtl

  Time To Live (TTL) for an 'actor' in a room. If a client disconnects, this actor is inactive first and removed after this timeout. In milliseconds.
- Hashtable customRoomProperties

  The room's custom properties to set. Use string keys!
- string[] customRoomPropertiesForLobby = new string[0]

  Defines the custom room properties that get listed in the lobby.
- string[] plugins

  Informs the server of the expected plugin setup.

Properties

- bool isVisible `[get, set]`

  Defines if this room is listed in the lobby. If not, it also is not joined randomly.
- bool isOpen `[get, set]`

  Defines if this room can be joined at all.
- bool cleanupCacheOnLeave `[get, set]`

  Time To Live (TTL) for a room when the last player leaves. Keeps room in memory for case a player re-joins soon. In milliseconds.
- bool suppressRoomEvents `[get]`

  Tells the server to skip room events for joining and leaving players.
- bool publishUserId `[get, set]`

  Defines if the UserIds of players get "published" in the room. Useful for FindFriends, if players want to play another game together.

8.43.1 Detailed Description

Wraps up common room properties needed when you create rooms. Read the individual entries for more details.

This directly maps to the fields in the Room class.

8.43.2 Member Data Documentation

8.43.2.1 Hashtable RoomOptions.customRoomProperties

The room's custom properties to set. Use string keys!

Custom room properties are any key-values you need to define the game's setup. The shorter your keys are, the better. Example: Map, Mode (could be "m" when used with "Map"), TileSet (could be "t").

8.43.2.2 string [ ] RoomOptions.customRoomPropertiesForLobby = new string[0]

Defines the custom room properties that get listed in the lobby.

Name the custom room properties that should be available to clients that are in a lobby. Use with care. Unless a custom property is essential for matchmaking or user info, it should not be sent to the lobby, which causes traffic and delays for clients in the lobby.

Default: No custom properties are sent to the lobby.

8.43.2.3 byte RoomOptions.maxPlayers

Max number of players that can be in the room at any time. 0 means "no limit".

8.43.2.4 int RoomOptions.PlayerTtl

Time To Live (TTL) for an 'actor' in a room. If a client disconnects, this actor is inactive first and removed after this timeout. In milliseconds.

8.43.2.5 string [ ] RoomOptions.plugins

Informs the server of the expected plugin setup.

The operation will fail in case of a plugin missmatch returning error code PluginMismatch 32757(0x7FFF - 10). Setting string[]{} means the client expects no plugin to be setup. Note: for backwards compatibility null omits any check.

8.43.3 Property Documentation

8.43.3.1 bool RoomOptions.cleanupCacheOnLeave [get],[set]

Time To Live (TTL) for a room when the last player leaves. Keeps room in memory for case a player re-joins soon. In milliseconds.

Removes a user's events and properties from the room when a user leaves.

This makes sense when in rooms where players can't place items in the room and just vanish entirely. When you disable this, the event history can become too long to load if the room stays in use indefinitely. Default: true. Cleans up the cache and props of leaving users.

8.43.3.2 bool RoomOptions.isOpen [get],[set]

Defines if this room can be joined at all.

If a room is closed, no player can join this. As example this makes sense when 3 of 4 possible players start their gameplay early and don't want anyone to join during the game. The room can still be listed in the lobby (set isVisible to control lobby-visibility).

8.43.3.3 bool RoomOptions.isVisible [get],[set]

Defines if this room is listed in the lobby. If not, it also is not joined randomly.

A room that is not visible will be excluded from the room lists that are sent to the clients in lobbies. An invisible room can be joined by name but is excluded from random matchmaking.

Use this to "hide" a room and simulate "private rooms". Players can exchange a roomname and create it invisble to avoid anyone else joining it.

8.43.3.4 bool RoomOptions.publishUserId [get], [set]

Defines if the UserIds of players get "published" in the room. Useful for FindFriends, if players want to play another game together.

When you set this to true, Photon will publish the UserIds of the players in that room. In that case, you can use PhotonPlayer.userId, to access any player's userID. This is useful for FindFriends and to set "expected users" to reserve slots in a room (see PhotonNetwork.JoinRoom e.g.).

8.43.3.5 bool RoomOptions.suppressRoomEvents [get]

Tells the server to skip room events for joining and leaving players.

Using this makes the client unaware of the other players in a room. That can save some traffic if you have some server logic that updates players but it can also limit the client's usability.

PUN will break if you use this, so it's not settable.

8.44 UnityEngine.SceneManagement.SceneManager Class Reference

Minimal implementation of the SceneManager for older Unity, up to v5.2.

Static Public Member Functions

- static void LoadScene (string name)
- static void LoadScene (int buildIndex)

8.44.1 Detailed Description

Minimal implementation of the SceneManager for older Unity, up to v5.2.

8.44.2 Member Function Documentation

8.44.2.1 static void UnityEngine.SceneManagement.SceneManager.LoadScene (string *name*) [static]

8.44.2.2 static void UnityEngine.SceneManagement.SceneManager.LoadScene (int *buildIndex*) [static]

8.45 SceneManagerHelper Class Reference

Properties

- static string ActiveSceneName [get]
- static int ActiveSceneBuildIndex [get]

8.45.1 Property Documentation

8.45.1.1 int SceneManagerHelper.ActiveSceneBuildIndex [static], [get]

8.45.1.2 string SceneManagerHelper.ActiveSceneName [static], [get]

8.46 ServerSettings Class Reference

Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

Inherits ScriptableObject.

Public Types

- enum HostingOption {
  HostingOption.NotSet = 0, HostingOption.PhotonCloud = 1, HostingOption.SelfHosted = 2, HostingOption.↩
  OfflineMode = 3,
  HostingOption.BestRegion = 4 }

Public Member Functions

- void UseCloudBestRegion (string cloudAppid)
- void UseCloud (string cloudAppid)
- void UseCloud (string cloudAppid, CloudRegionCode code)
- void UseMyServer (string serverAddress, int serverPort, string application)
- override string ToString ()

Public Attributes

- HostingOption HostType = HostingOption.NotSet
- ConnectionProtocol Protocol = ConnectionProtocol.Udp
- string ServerAddress = ""
- int ServerPort = 5055
- string AppID = ""
- CloudRegionCode PreferredRegion
- CloudRegionFlag EnabledRegions = (CloudRegionFlag)(-1)
- bool JoinLobby
- bool EnableLobbyStatistics
- List< string > RpcList = new List<string>()
- bool DisableAutoOpenWizard

8.46.1 Detailed Description

Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

8.46.2 Member Enumeration Documentation

8.46.2.1 enum ServerSettings.HostingOption

Enumerator

NotSet

PhotonCloud

SelfHosted

OfflineMode

BestRegion

8.46.3 Member Function Documentation

8.46.3.1 override string ServerSettings.ToString ( )

8.46.3.2 void ServerSettings.UseCloud (string *cloudAppid*)

8.46.3.3 void ServerSettings.UseCloud (string *cloudAppid,* CloudRegionCode *code*)

8.46.3.4 void ServerSettings.UseCloudBestRegion (string *cloudAppid*)

8.46.3.5 void ServerSettings.UseMyServer (string *serverAddress,* int *serverPort,* string *application*)

8.46.4 Member Data Documentation

8.46.4.1 string ServerSettings.AppID = ""

8.46.4.2 bool ServerSettings.DisableAutoOpenWizard

8.46.4.3 CloudRegionFlag ServerSettings.EnabledRegions = (CloudRegionFlag)(-1)

8.46.4.4 bool ServerSettings.EnableLobbyStatistics

8.46.4.5 HostingOption ServerSettings.HostType = HostingOption.NotSet

8.46.4.6 bool ServerSettings.JoinLobby

8.46.4.7 CloudRegionCode ServerSettings.PreferredRegion

8.46.4.8 ConnectionProtocol ServerSettings.Protocol = ConnectionProtocol.Udp

8.46.4.9 List<string> ServerSettings.RpcList = new List<string>()

8.46.4.10 string ServerSettings.ServerAddress = ""

8.46.4.11 int ServerSettings.ServerPort = 5055

8.47 PhotonAnimatorView.SynchronizedLayer Class Reference

Public Attributes

- SynchronizeType SynchronizeType
- int LayerIndex

8.47.1 Member Data Documentation

8.47.1.1 int PhotonAnimatorView.SynchronizedLayer.LayerIndex

8.47.1.2 SynchronizeType PhotonAnimatorView.SynchronizedLayer.SynchronizeType

8.48 PhotonAnimatorView.SynchronizedParameter Class Reference

Public Attributes

- ParameterType Type

- SynchronizeType SynchronizeType
- string Name

8.48.1 Member Data Documentation

8.48.1.1 string PhotonAnimatorView.SynchronizedParameter.Name

8.48.1.2 SynchronizeType PhotonAnimatorView.SynchronizedParameter.SynchronizeType

8.48.1.3 ParameterType PhotonAnimatorView.SynchronizedParameter.Type

8.49 TypedLobby Class Reference

Refers to a specific lobby (and type) on the server.

Inherited by TypedLobbyInfo.

Public Member Functions

- TypedLobby ()
- TypedLobby (string name, LobbyType type)
- override string ToString ()

Public Attributes

- string Name

  Name of the lobby this game gets added to. Default: null, attached to default lobby. Lobbies are unique per lobbyName plus lobbyType, so the same name can be used when several types are existing.

- LobbyType Type

  Type of the (named)lobby this game gets added to

Static Public Attributes

- static readonly TypedLobby Default = new TypedLobby()

Properties

- bool IsDefault `[get]`

8.49.1 Detailed Description

Refers to a specific lobby (and type) on the server.

The name and type are the unique identifier for a lobby.
Join a lobby via PhotonNetwork.JoinLobby(TypedLobby lobby).
The current lobby is stored in PhotonNetwork.lobby.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 TypedLobby.TypedLobby ( )

8.49.2.2 TypedLobby.TypedLobby (string *name,* LobbyType *type*)

8.49.3 Member Function Documentation

8.49.3.1 override string TypedLobby.ToString ( )

8.49.4 Member Data Documentation

8.49.4.1 readonly TypedLobby TypedLobby.Default = new TypedLobby() [static]

8.49.4.2 string TypedLobby.Name

Name of the lobby this game gets added to. Default: null, attached to default lobby. Lobbies are unique per lobbyName plus lobbyType, so the same name can be used when several types are existing.

8.49.4.3 LobbyType TypedLobby.Type

Type of the (named)lobby this game gets added to

8.49.5 Property Documentation

8.49.5.1 bool TypedLobby.IsDefault [get]

8.50 TypedLobbyInfo Class Reference

Inherits TypedLobby.

Public Member Functions

- override string ToString ()

Public Attributes

- int PlayerCount
- int RoomCount

Additional Inherited Members

8.50.1 Member Function Documentation

8.50.1.1 override string TypedLobbyInfo.ToString ( )

8.50.2 Member Data Documentation

8.50.2.1 int TypedLobbyInfo.PlayerCount

8.50.2.2 int TypedLobbyInfo.RoomCount

8.51 WebRpcResponse Class Reference

Reads an operation response of a WebRpc and provides convenient access to most common values.

Public Member Functions

- WebRpcResponse (OperationResponse response)

  An OperationResponse for a WebRpc is needed to read it's values.
- string ToStringFull ()

  Turns the response into an easier to read string.

Properties

- string Name `[get, set]`

  Name of the WebRpc that was called.
- int ReturnCode `[get, set]`

  ReturnCode of the WebService that answered the WebRpc.
- string DebugMessage `[get, set]`

  Might be empty or null.
- Dictionary< string, object > Parameters `[get, set]`

  Other key/values returned by the webservice that answered the WebRpc.

8.51.1 Detailed Description

Reads an operation response of a WebRpc and provides convenient access to most common values.

See method PhotonNetwork.WebRpc.
Create a WebRpcResponse to access common result values.
The operationResponse.OperationCode should be: OperationCode.WebRpc.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 WebRpcResponse.WebRpcResponse (OperationResponse *response*)

An OperationResponse for a WebRpc is needed to read it's values.

8.51.3 Member Function Documentation

8.51.3.1 string WebRpcResponse.ToStringFull ( )

Turns the response into an easier to read string.

Returns

String resembling the result.

8.51.4 Property Documentation

8.51.4.1 string WebRpcResponse.DebugMessage `[get],[set]`

Might be empty or null.

8.51.4.2 string WebRpcResponse.Name [get], [set]

Name of the WebRpc that was called.

8.51.4.3 Dictionary<string, object> WebRpcResponse.Parameters [get], [set]

Other key/values returned by the webservice that answered the WebRpc.

8.51.4.4 int WebRpcResponse.ReturnCode [get], [set]

ReturnCode of the WebService that answered the WebRpc.

0 is commonly used to signal success.
-1 tells you: Got no ReturnCode from WebRpc service.
Other ReturnCodes are defined by the individual WebRpc and service.

Chapter 9

File Documentation

9.1 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/general.md File Reference

9.2 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/main.md File Reference

9.3 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/optionalGui.md File Reference

9.4 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/photonStatsGui.md File Reference

9.5 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/_Doc/publicApi.md File Reference

9.6 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs File Reference

Sets up support for Unity-specific types. Can be a blueprint how to register your own Custom Types for sending.

Classes

- class **CustomTypes**

  Internally used class, containing de/serialization methods for various Unity-specific classes. Adding those to the Photon serialization protocol allows you to send them in events, etc.

9.6.1 Detailed Description

Sets up support for Unity-specific types. Can be a blueprint how to register your own Custom Types for sending.

9.7 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs File Reference

Wraps up several of the commonly used enumerations.

Enumerations

- enum PhotonNetworkingMessage { PhotonNetworkingMessage.OnConnectedToPhoton, PhotonNetworkingMessage.OnLeftRoom, PhotonNetworkingMessage.OnMasterClientSwitched, PhotonNetworkingMessage.OnPhotonCreateRoomFailed, PhotonNetworkingMessage.OnPhotonJoinRoomFailed, PhotonNetworkingMessage.OnCreatedRoom, PhotonNetworkingMessage.OnJoinedLobby, PhotonNetworkingMessage.OnLeftLobby, PhotonNetworkingMessage.OnDisconnectedFromPhoton, PhotonNetworkingMessage.OnConnectionFail, PhotonNetworkingMessage.OnFailedToConnectToPhoton, PhotonNetworkingMessage.OnReceivedRoomListUpdate, PhotonNetworkingMessage.OnJoinedRoom, PhotonNetworkingMessage.OnPhotonPlayerConnected, PhotonNetworkingMessage.OnPhotonPlayerDisconnected, PhotonNetworkingMessage.OnPhotonRandomJoinFailed, PhotonNetworkingMessage.OnConnectedToMaster, PhotonNetworkingMessage.OnPhotonSerializeView, PhotonNetworkingMessage.OnPhotonInstantiate, PhotonNetworkingMessage.OnPhotonMaxCccuReached, PhotonNetworkingMessage.OnPhotonCustomRoomPropertiesChanged, PhotonNetworkingMessage.OnPhotonPlayerPropertiesChanged, PhotonNetworkingMessage.OnUpdatedFriendList, PhotonNetworkingMessage.OnCustomAuthenticationFailed, PhotonNetworkingMessage.OnCustomAuthenticationResponse, PhotonNetworkingMessage.OnWebRpcResponse, PhotonNetworkingMessage.OnOwnershipRequest, PhotonNetworkingMessage.OnLobbyStatisticsUpdate }

  This enum defines the set of MonoMessages Photon Unity Networking is using as callbacks. Implemented by PunBehaviour.
- enum PhotonLogLevel { PhotonLogLevel.ErrorsOnly, PhotonLogLevel.Informational, PhotonLogLevel.Full }

  Used to define the level of logging output created by the PUN classes. Either log errors, info (some more) or full.
- enum PhotonTargets { PhotonTargets.All, PhotonTargets.Others, PhotonTargets.MasterClient, PhotonTargets.AllBuffered, PhotonTargets.OthersBuffered, PhotonTargets.AllViaServer, PhotonTargets.AllBufferedViaServer }

  Enum of "target" options for RPCs. These define which remote clients get your RPC call.
- enum CloudRegionCode { CloudRegionCode.eu = 0, CloudRegionCode.us = 1, CloudRegionCode.asia = 2, CloudRegionCode.jp = 3, CloudRegionCode.au = 5, CloudRegionCode.none = 4 }

  Currently available `Photon Cloud regions` as enum.
- enum CloudRegionFlag { CloudRegionFlag.eu = 1 << 0, CloudRegionFlag.us = 1 << 1, CloudRegionFlag.asia = 1 << 2, CloudRegionFlag.jp = 1 << 3, CloudRegionFlag.au = 1 << 4 }

  Available regions as enum of flags. To be used as "enabled" flags for Best Region pinging.
- enum ServerConnection { ServerConnection.MasterServer, ServerConnection.GameServer, ServerConnection.NameServer }

  Available server (types) for internally used field: server.
- enum ConnectionState { ConnectionState.Disconnected, ConnectionState.Connecting, ConnectionState.Connected, ConnectionState.Disconnecting, ConnectionState.InitializingApplication }

  High level connection state of the client. Better use the more detailed PeerState.
- enum PeerState { PeerState.Uninitialized, PeerState.PeerCreated, PeerState.Queued, PeerState.Authenticated, PeerState.JoinedLobby, PeerState.DisconnectingFromMasterserver, PeerState.ConnectingToGameserver,

PeerState.ConnectedToGameserver,
PeerState.Joining, PeerState.Joined, PeerState.Leaving, PeerState.DisconnectingFromGameserver,
PeerState.ConnectingToMasterserver, PeerState.QueuedComingFromGameserver, PeerState.Disconnecting,
PeerState.Disconnected,
PeerState.ConnectedToMaster, PeerState.ConnectingToNameServer, PeerState.ConnectedToNameServer,
PeerState.DisconnectingFromNameServer,
PeerState.Authenticating }

Detailed connection / networking peer state. PUN implements a loadbalancing and authentication workflow "behind the scenes", so some states will automatically advance to some follow up state. Those states are commented with "(will-change)".

- enum DisconnectCause {
DisconnectCause.ExceptionOnConnect = StatusCode.ExceptionOnConnect, DisconnectCause.SecurityExceptionOnConnect = StatusCode.SecurityExceptionOnConnect, DisconnectCause.TimeoutDisconnect = StatusCode.TimeoutDisconnect, DisconnectCause.DisconnectByClientTimeout = StatusCode.TimeoutDisconnect,
DisconnectCause.InternalReceiveException = StatusCode.ExceptionOnReceive, DisconnectCause.DisconnectByServer = StatusCode.DisconnectByServer, DisconnectCause.DisconnectByServerTimeout = StatusCode.DisconnectByServer, DisconnectCause.DisconnectByServerLogic = StatusCode.DisconnectByServerLogic,
DisconnectCause.DisconnectByServerUserLimit = StatusCode.DisconnectByServerUserLimit, DisconnectCause.Exception = StatusCode.Exception, DisconnectCause.InvalidRegion = ErrorCode.InvalidRegion, DisconnectCause.MaxCcuReached = ErrorCode.MaxCcuReached,
DisconnectCause.InvalidAuthentication = ErrorCode.InvalidAuthentication, DisconnectCause.AuthenticationTicketExpired = 32753 }

Summarizes the cause for a disconnect. Used in: OnConnectionFail and OnFailedToConnectToPhoton.

9.7.1 Detailed Description

Wraps up several of the commonly used enumerations.

9.7.2 Enumeration Type Documentation

9.7.2.1 enum CloudRegionCode

Currently available `Photon Cloud regions` as enum.

This is used in PhotonNetwork.ConnectToRegion.

Enumerator

eu European servers in Amsterdam.

us US servers (East Coast).

asia Asian servers in Singapore.

jp Japanese servers in Tokyo.

au Australian servers in Melbourne.

none No region selectedcs.

9.7.2.2 enum CloudRegionFlag

Available regions as enum of flags. To be used as "enabled" flags for Best Region pinging.

Note that these enum values skip CloudRegionCode.none and their values are in strict order (power of 2).

Enumerator

eu

us

asia

jp

au

9.7.2.3 enum ConnectionState

High level connection state of the client. Better use the more detailed PeerState.

Enumerator

Disconnected

Connecting

Connected

Disconnecting

InitializingApplication

9.7.2.4 enum ServerConnection

Available server (types) for internally used field: server.

Photon uses 3 different roles of servers: Name Server, Master Server and Game Server.

Enumerator

MasterServer This server is where matchmaking gets done and where clients can get lists of rooms in lobbies.

GameServer This server handles a number of rooms to execute and relay the messages between players (in a room).

NameServer This server is used initially to get the address (IP) of a Master Server for a specific region. Not used for Photon OnPremise (self hosted).

9.8 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs File Reference

Classes

- class Extensions

  This static class defines some useful extension methods for several existing classes (e.g. Vector3, float and others).
- class GameObjectExtensions

  Small number of extension methods that make it easier for PUN to work cross-Unity-versions.

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.8.1 Typedef Documentation

9.8.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.8.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.9 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs File Reference

Classes

- class FriendInfo

  Used to store info about a friend's online state and in which room he/she is.

9.10 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/GizmoType.cs File Reference

Classes

- class ExitGames.Client.GUI.GizmoTypeDrawer

Namespaces

- package ExitGames.Client.GUI

Enumerations

- enum ExitGames.Client.GUI.GizmoType { ExitGames.Client.GUI.GizmoType.WireSphere, ExitGames.Client.GUI.GizmoType.Sphere, ExitGames.Client.GUI.GizmoType.WireCube, ExitGames.Client.GUI.GizmoType.Cube }

9.11 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs File Reference

Classes

- class **LoadBalancingPeer**

  Internally used by PUN. A LoadbalancingPeer provides the operations and enum definitions needed to use the load-balancing server application which is also used in Photon Cloud.
- class **OpJoinRandomRoomParams**
- class **EnterRoomParams**
- class ErrorCode

  ErrorCode defines the default codes associated with Photon client/server communication.
- class ActorProperties

  Class for constants. These (byte) values define "well known" properties for an Actor / Player. Pun uses these constants internally.
- class GamePropertyKey

  Class for constants. These (byte) values are for "well known" room/game properties used in Photon Loadbalancing. Pun uses these constants internally.

- class EventCode

  Class for constants. These values are for events defined by Photon Loadbalancing. Pun uses these constants internally.
- class ParameterCode

  Class for constants. Codes for parameters of Operations and Events. Pun uses these constants internally.
- class OperationCode

  Class for constants. Contains operation codes. Pun uses these constants internally.
- class RoomOptions

  Wraps up common room properties needed when you create rooms. Read the individual entries for more details.
- class RaiseEventOptions

  Aggregates several less-often used options for operation RaiseEvent. See field descriptions for usage details.
- class TypedLobby

  Refers to a specific lobby (and type) on the server.
- class TypedLobbyInfo
- class AuthenticationValues

  Container for user authentication in Photon. Set AuthValues before you connect - all else is handled.

Enumerations

- enum JoinMode : byte { JoinMode.Default = 0, JoinMode.CreateIfNotExists = 1, JoinMode.JoinOrRejoin = 2, JoinMode.RejoinOnly = 3 }

  Defines possible values for OpJoinRoom and OpJoinOrCreate. It tells the server if the room can be only be joined normally, created implicitly or found on a web-service for Turnbased games.
- enum MatchmakingMode : byte { MatchmakingMode.FillRoom = 0, MatchmakingMode.SerialMatching = 1, MatchmakingMode.RandomMatching = 2 }

  Options for matchmaking rules for OpJoinRandom.
- enum ReceiverGroup : byte { ReceiverGroup.Others = 0, ReceiverGroup.All = 1, ReceiverGroup.MasterClient = 2 }

  Lite - OpRaiseEvent lets you chose which actors in the room should receive events. By default, events are sent to "Others" but you can overrule this.
- enum EventCaching : byte { EventCaching.DoNotCache = 0, EventCaching.MergeCache = 1, EventCaching.ReplaceCache = 2, EventCaching.RemoveCache = 3, EventCaching.AddToRoomCache = 4, EventCaching.AddToRoomCacheGlobal = 5, EventCaching.RemoveFromRoomCache = 6, EventCaching.RemoveFromRoomCacheForActorsLeft = 7, EventCaching.SliceIncreaseIndex = 10, EventCaching.SliceSetIndex = 11, EventCaching.SlicePurgeIndex = 12, EventCaching.SlicePurgeUpToIndex = 13 }

  Lite - OpRaiseEvent allows you to cache events and automatically send them to joining players in a room. Events are cached per event code and player: Event 100 (example!) can be stored once per player. Cached events can be modified, replaced and removed.
- enum PropertyTypeFlag : byte { PropertyTypeFlag.None = 0x00, PropertyTypeFlag.Game = 0x01, PropertyTypeFlag.Actor = 0x02, PropertyTypeFlag.GameAndActor = Game | Actor }

  Flags for "types of properties", being used as filter in OpGetProperties.
- enum LobbyType : byte { LobbyType.Default = 0, LobbyType.SqlLobby = 2, LobbyType.AsyncRandomLobby = 3 }

  Options of lobby types available. Lobby types might be implemented in certain Photon versions and won't be available on older servers.
- enum CustomAuthenticationType : byte { CustomAuthenticationType.Custom = 0, CustomAuthenticationType.Steam = 1, CustomAuthenticationType.Facebook = 2, CustomAuthenticationType.None = byte.MaxValue }

  Options for optional "Custom Authentication" services used with Photon. Used by OpAuthenticate after connecting to Photon.

9.11.1 Enumeration Type Documentation

9.11.1.1 enum CustomAuthenticationType : byte

Options for optional "Custom Authentication" services used with Photon. Used by OpAuthenticate after connecting to Photon.

Enumerator

Custom Use a custom authentification service. Currently the only implemented option.

Steam Authenticates users by their Steam Account. Set auth values accordingly!

Facebook Authenticates users by their Facebook Account. Set auth values accordingly!

None Disables custom authentification. Same as not providing any AuthenticationValues for connect (more precisely for: OpAuthenticate).

9.11.1.2 enum EventCaching : byte

Lite - OpRaiseEvent allows you to cache events and automatically send them to joining players in a room. Events are cached per event code and player: Event 100 (example!) can be stored once per player. Cached events can be modified, replaced and removed.

Caching works only combination with ReceiverGroup options Others and All.

Enumerator

DoNotCache Default value (not sent).

MergeCache Will merge this event's keys with those already cached.

ReplaceCache Replaces the event cache for this eventCode with this event's content.

RemoveCache Removes this event (by eventCode) from the cache.

AddToRoomCache Adds an event to the room's cache

AddToRoomCacheGlobal Adds this event to the cache for actor 0 (becoming a "globally owned" event in the cache).

RemoveFromRoomCache Remove fitting event from the room's cache.

RemoveFromRoomCacheForActorsLeft Removes events of players who already left the room (cleaning up).

SliceIncreaseIndex Increase the index of the sliced cache.

SliceSetIndex Set the index of the sliced cache. You must set RaiseEventOptions.CacheSliceIndex for this.

SlicePurgeIndex Purge cache slice with index. Exactly one slice is removed from cache. You must set RaiseEventOptions.CacheSliceIndex for this.

SlicePurgeUpToIndex Purge cache slices with specified index and anything lower than that. You must set RaiseEventOptions.CacheSliceIndex for this.

9.11.1.3 enum JoinMode : byte

Defines possible values for OpJoinRoom and OpJoinOrCreate. It tells the server if the room can be only be joined normally, created implicitly or found on a web-service for Turnbased games.

These values are not directly used by a game but implicitly set.

Enumerator

Default Regular join. The room must exist.

CreateIfNotExists Join or create the room if it's not existing. Used for OpJoinOrCreate for example.

JoinOrRejoin The room might be out of memory and should be loaded (if possible) from a Turnbased web-service.

RejoinOnly Only re-join will be allowed. If the user is not yet in the room, this will fail.

9.11.1.4 enum LobbyType : byte

Options of lobby types available. Lobby types might be implemented in certain Photon versions and won't be available on older servers.

Enumerator

Default This lobby is used unless another is defined by game or JoinRandom. Room-lists will be sent and JoinRandomRoom can filter by matching properties.

SqlLobby This lobby type lists rooms like Default but JoinRandom has a parameter for SQL-like "where" clauses for filtering. This allows bigger, less, or and and combinations.

AsyncRandomLobby This lobby does not send lists of games. It is only used for OpJoinRandomRoom. It keeps rooms available for a while when there are only inactive users left.

9.11.1.5 enum MatchmakingMode : byte

Options for matchmaking rules for OpJoinRandom.

Enumerator

FillRoom Fills up rooms (oldest first) to get players together as fast as possible. Default.Makes most sense with MaxPlayers $>$ 0 and games that can only start with more players.

SerialMatching Distributes players across available rooms sequentially but takes filter into account. Without filter, rooms get players evenly distributed.

RandomMatching Joins a (fully) random room. Expected properties must match but aside from this, any available room might be selected.

9.11.1.6 enum PropertyTypeFlag : byte

Flags for "types of properties", being used as filter in OpGetProperties.

Enumerator

None (0x00) Flag type for no property type.

Game (0x01) Flag type for game-attached properties.

Actor (0x02) Flag type for actor related propeties.

GameAndActor (0x01) Flag type for game AND actor properties. Equal to 'Game'

9.11.1.7 enum ReceiverGroup : byte

Lite - OpRaiseEvent lets you chose which actors in the room should receive events. By default, events are sent to "Others" but you can overrule this.

Enumerator

Others Default value (not sent). Anyone else gets my event.

All Everyone in the current room (including this peer) will get this event.

MasterClient The server sends this event only to the actor with the lowest actorNumber.The "master client" does not have special rights but is the one who is in this room the longest time.

9.12 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/↩ Plugins/PhotonNetwork/NetworkingPeer.cs File Reference

Classes

- class **NetworkingPeer**

  Implements Photon LoadBalancing used in PUN. This class is used internally by PhotonNetwork and not intended as public API.

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.12.1 Typedef Documentation

9.12.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.12.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.13 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/↩ Plugins/PhotonNetwork/PhotonClasses.cs File Reference

Wraps up smaller classes that don't need their own file.

Classes

- interface IPunObservable

  Defines the OnPhotonSerializeView method to make it easy to implement correctly for observable scripts.
- interface IPunCallbacks

  This interface is used as definition of all callback methods of PUN, except OnPhotonSerializeView. Preferably, implement them individually.
- interface IPunPrefabPool

  Defines all the methods that a Object Pool must implement, so that PUN can use it.
- class Photon.MonoBehaviour

  This class adds the property photonView, while logging a warning when your game still uses the networkView.
- class Photon.PunBehaviour

  This class provides a .photonView and all callbacks/events that PUN can call. Override the events/methods you want to use.
- class PhotonMessageInfo

  Container class for info about a particular message, RPC or update.
- class **PunEvent**

  Defines Photon event-codes as used by PUN.
- class PhotonStream

  This container is used in OnPhotonSerializeView() to either provide incoming data of a PhotonView or for you to provide it.
- class HelpURL

  Empty implementation of the upcoming HelpURL of Unity 5.1. This one is only for compatibility of attributes.
- class UnityEngine.SceneManagement.SceneManager

  Minimal implementation of the SceneManager for older Unity, up to v5.2.

- class SceneManagerHelper
- class WebRpcResponse

  Reads an operation response of a WebRpc and provides convenient access to most common values.

Namespaces

- package Photon
- package UnityEngine.SceneManagement

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass
- using Photon.Hashtable = ExitGames.Client.Photon.Hashtable

9.13.1 Detailed Description

Wraps up smaller classes that don't need their own file.

9.13.2 Typedef Documentation

9.13.2.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.13.2.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.14 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs File Reference

Classes

- class **PhotonHandler**

  Internal Monobehaviour that allows Photon to run an Update loop.

Typedefs

- using Debug = UnityEngine.Debug
- using Hashtable = ExitGames.Client.Photon.Hashtable
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.14.1 Typedef Documentation

9.14.1.1 using Debug = UnityEngine.Debug

9.14.1.2 using Hashtable = ExitGames.Client.Photon.Hashtable

9.14.1.3 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.15 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/↩ Plugins/PhotonNetwork/PhotonLagSimulationGui.cs File Reference

Part of the Optional GUI.

Classes

- class PhotonLagSimulationGui

  This MonoBehaviour is a basic GUI for the Photon client's network-simulation feature. It can modify lag (fixed delay), jitter (random lag) and packet loss.

9.15.1 Detailed Description

Part of the Optional GUI.

9.16 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/↩ Plugins/PhotonNetwork/PhotonNetwork.cs File Reference

Classes

- class PhotonNetwork

  The main class to use the PhotonNetwork plugin. This class is static.

Typedefs

- using Debug = UnityEngine.Debug
- using Hashtable = ExitGames.Client.Photon.Hashtable

9.16.1 Typedef Documentation

9.16.1.1 using Debug = UnityEngine.Debug

9.16.1.2 using Hashtable = ExitGames.Client.Photon.Hashtable

9.17 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/↩ Plugins/PhotonNetwork/PhotonPlayer.cs File Reference

Classes

- class PhotonPlayer

  Summarizes a "player" within a room, identified (in that room) by actorID.

Typedefs

- using Hashtable = ExitGames.Client.Photon.Hashtable

9.17.1 Typedef Documentation

9.17.1.1 using Hashtable = ExitGames.Client.Photon.Hashtable

9.18 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs File Reference

Part of the Optional GUI.

Classes

- class PhotonStatsGui

  Basic GUI to show traffic and health statistics of the connection to Photon, toggled by shift+tab.

9.18.1 Detailed Description

Part of the Optional GUI.

9.19 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonStreamQueue.cs File Reference

Classes

- class PhotonStreamQueue

  The PhotonStreamQueue helps you poll object states at higher frequencies then what PhotonNetwork.sendRate dictates and then sends all those states at once when Serialize() is called. On the receiving end you can call Deserialize() and then the stream will roll out the received object states in the same order and timeStep they were recorded in.

9.20 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs File Reference

Classes

- class PhotonView

  PUN's NetworkView replacement class for networking. Use it like a NetworkView.

Enumerations

- enum ViewSynchronization { ViewSynchronization.Off, ViewSynchronization.ReliableDeltaCompressed, ViewSynchronization.Unreliable, ViewSynchronization.UnreliableOnChange }
- enum OnSerializeTransform { OnSerializeTransform.OnlyPosition, OnSerializeTransform.OnlyRotation, OnSerializeTransform.OnlyScale, OnSerializeTransform.PositionAndRotation, OnSerializeTransform.All }
- enum OnSerializeRigidBody { OnSerializeRigidBody.OnlyVelocity, OnSerializeRigidBody.OnlyAngularVelocity, OnSerializeRigidBody.All }
- enum OwnershipOption { OwnershipOption.Fixed, OwnershipOption.Takeover, OwnershipOption.Request }

  Options to define how Ownership Transfer is handled per PhotonView.

9.20.1 Enumeration Type Documentation

9.20.1.1 enum OnSerializeRigidBody

Enumerator

OnlyVelocity

OnlyAngularVelocity

All

9.20.1.2 enum OnSerializeTransform

Enumerator

OnlyPosition

OnlyRotation

OnlyScale

PositionAndRotation

All

9.20.1.3 enum OwnershipOption

Options to define how Ownership Transfer is handled per PhotonView.

This setting affects how RequestOwnership and TransferOwnership work at runtime.

Enumerator

Fixed Ownership is fixed. Instantiated objects stick with their creator, scene objects always belong to the Master Client.

Takeover Ownership can be taken away from the current owner who can't object.

Request Ownership can be requested with PhotonView.RequestOwnership but the current owner has to agree to give up ownership. The current owner has to implement IPunCallbacks.OnOwnershipRequest to react to the ownership request.

9.20.1.4 enum ViewSynchronization

Enumerator

Off

ReliableDeltaCompressed

Unreliable

UnreliableOnChange

9.21 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs File Reference

Classes

- class PingMonoEditor

  Uses C# Socket class from System.Net.Sockets (as Unity usually does).
- class PhotonPingManager

Typedefs

- using Debug = UnityEngine.Debug
- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.21.1 Typedef Documentation

9.21.1.1 using Debug = UnityEngine.Debug

9.21.1.2 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.22 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Room.cs File Reference

Classes

- class Room

  This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a RoomInfo and you can close or hide "your" room.

9.23 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs File Reference

Classes

- class RoomInfo

  A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc).

9.24 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/RPC.cs File Reference

Reimplements a RPC Attribute, as it's no longer in all versions of the UnityEngine assembly.

Classes

- class PunRPC

  Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

9.24.1 Detailed Description

Reimplements a RPC Attribute, as it's no longer in all versions of the UnityEngine assembly.

9.25 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs File Reference

Outdated. Here to overwrite older files on import.

9.25.1 Detailed Description

Outdated. Here to overwrite older files on import.

9.26 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs File Reference

ScriptableObject defining a server setup. An instance is created as **PhotonServerSettings**.

Classes

- class Region
- class ServerSettings

  Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

9.26.1 Detailed Description

ScriptableObject defining a server setup. An instance is created as **PhotonServerSettings**.

9.27 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/SocketWebTcp.cs File Reference

Typedefs

- using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.27.1 Typedef Documentation

9.27.1.1 using SupportClassPun = ExitGames.Client.Photon.SupportClass

9.28 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonAnimatorView.cs File Reference

Classes

- class PhotonAnimatorView

  This class helps you to synchronize Mecanim animations Simply add the component to your GameObject and make sure that the PhotonAnimatorView is added to the list of observed components
- class PhotonAnimatorView.SynchronizedParameter
- class PhotonAnimatorView.SynchronizedLayer

9.29 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonRigidbody2DView.cs File Reference

Classes

- class PhotonRigidbody2DView

This class helps you to synchronize the velocities of a 2d physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbody2DView is added to the list of observed components

9.30 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonRigidbodyView.cs File Reference

Classes

- class PhotonRigidbodyView

  This class helps you to synchronize the velocities of a physics RigidBody. Note that only the velocities are synchronized and because Unitys physics engine is not deterministic (ie. the results aren't always the same on all computers) - the actual positions of the objects may go out of sync. If you want to have the position of this object the same on all clients, you should also add a PhotonTransformView to synchronize the position. Simply add the component to your GameObject and make sure that the PhotonRigidbodyView is added to the list of observed components

9.31 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformView.cs File Reference

Classes

- class PhotonTransformView

  This class helps you to synchronize position, rotation and scale of a GameObject. It also gives you many different options to make the synchronized values appear smooth, even when the data is only send a couple of times per second. Simply add the component to your GameObject and make sure that the PhotonTransformView is added to the list of observed components

9.32 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewPositionControl.cs File Reference

Classes

- class PhotonTransformViewPositionControl

9.33 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewPositionModel.cs File Reference

Classes

- class PhotonTransformViewPositionModel

9.34 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewRotationControl.cs File Reference

Classes

- class PhotonTransformViewRotationControl

9.35 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewRotationModel.cs File Reference

Classes

- class PhotonTransformViewRotationModel

9.36 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewScaleControl.cs File Reference

Classes

- class PhotonTransformViewScaleControl

9.37 C:/Dev/photon-sdk-dotnet/Unity/PhotonNetworking/Assets/Photon Unity Networking/Plugins/PhotonNetwork/Views/PhotonTransformViewScaleModel.cs File Reference

Classes

- class PhotonTransformViewScaleModel