Photon Unity Networking 2

v2.0

Generated by Doxygen 1.8.10

Tue Aug 7 2018 15:50:24

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 multiple client SDKs for major platforms.

Photon **Unity Network (PUN)** is our is our take on a Unity specific, high-level solution: Matchmaking, easy to use callbacks, components to synchronize GameObjects, Remote Procedure Calls (RPCs) and similar features provide a great start. Beyond that is a solid, extensive API for more advanced control.

Full source code is available, so you can scale this package to support any type of multiplayer game you come up with.

This package is compatible with the managed `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

- Lots of demos and an extensive `PUN Basics Tutorial`

- Server available as hosted service (free for development) or as "On Premise"

- Load-balanced! Scales across servers (with no extra effort)

- Outstanding performance of the Photon Server

- Dedicated servers. No NAT punch-through needed

- Offline mode: re-use your multiplayer code in singleplayer game modes

Documentation And Learning

There is an `Online Documentation`, which is considered a manual for PUN. This might become your primary source for information.

This is the Reference Documentation for PUN. It summarizes the most important classes in the Public API module and explains each class, method and field individually. This is generated from the source of PUN and should be used to look up details on usage and parameters.

Aside from that, there are also Demos in the PUN package itself and a `PUN Basics Tutorial` online, which you should check out.

First Steps

Import PUN into a new, empty project. Register via the pop up "wizard" (ALT+P) to get you a free Photon Cloud subscription (saving an initial AppId for you). Now you're ready to run and dissect the Demos.

Make sure to open and code the `PUN Basics Tutorial`.

Chapter 2

General Documentation

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

Photon Unity Networking - First steps

When you import PUN, the "Wizard" window will pop up. If not, find it in the Window menu as "Photon Unity Networking". In the Wizard, either enter your email address to register for the Photon Cloud, enter the AppId of an existing account or skip this step for the time being.

The Wizard creates a configuration in the project, named: PhotonServerSettings.

PUN consists of quite a few files, however most functionality is concentrated into: **Photon.Pun.PhotonNetwork**. This class contains all functions and variables typically 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 learn how this API works, visit the `online documentation for PUN`

2.1 Photon

Photon Unity Networking (PUN) always connects to a dedicated Photon server, which provides matchmaking, load balancing and in-room communication for players.

Behind the scenes PUN uses more than one server: A "Name Server" acts as point of entry and provides a list of regional "Master Servers". A Master Server keeps track of rooms and provides the Matchmaking, while several "Game Servers" run the actual rooms (matches).

Exit Games Cloud

The Exit Games Cloud 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 Public Cloud service runs a fixed logic, so the clients need to be authoritative.

Clients are separated by "application id" (identifies your game title) and a "game version". Changing the game version helps separate players with new and old client builds.

Subscriptions bought in Asset Store

If you bought a package with Photon Cloud Subscription in the Asset Store:

- Register a Photon Cloud Account `at this link`

- Create an App and get your AppID from the `Dashboard`

- Send a Mail to: `developer@photonengine.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 `at this link`

Read about how to start the server `here`.

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 Photon.Pun.UtilityScripts.PhotonLagSimulationGui 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 Photon.Pun.UtilityScripts.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.

Button "Stats On" (Enabling Traffic Stats)

Chapter 5

Public API Module

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

The classes which are most commonly used, are grouped into a Public API module, which is only a documentation structure. Classes like Photon.Pun.PhotonNetwork and Photon.Pun.MonoBehaviourPunCallbacks are good entry points to learn how to code with PUN.

Typically, classes for internal use are not public but there are a few exceptions to this where access may be of use, if you know what you're doing.

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

- class Photon.Pun.PhotonNetwork

  The main class to use the PhotonNetwork plugin. This class is static.
- class Photon.Pun.PhotonView

  PUN's NetworkView replacement class for networking. Use it like a NetworkView.
- struct Photon.Pun.PhotonMessageInfo

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

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

Enumerations

- enum Photon.Realtime.ClientState

  State values for a client, which handles switching Photon server types, some operations, etc.
- enum Photon.Pun.PunLogLevel

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

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

Functions

- void Photon.Pun.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.

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 Photon.Realtime.ClientState `[strong]`

State values for a client, which handles switching Photon server types, some operations, etc.

Enumerator

PeerCreated   Peer is created but not used yet.

Authenticating   Transition state while connecting to a server. On the Photon Cloud this sends the AppId and
AuthenticationValues (UserID).

Authenticated   Transition state while connecting to a server. Leads to state ConnectedToMasterserver or
JoinedLobby.

JoiningLobby   The client sent an OpJoinLobby and if this was done on the Master Server, it will result in.
Depending on the lobby, it gets room listings.

JoinedLobby   The client is in a lobby, connected to the MasterServer. Depending on the lobby, it gets room
listings.

DisconnectingFromMasterserver   Transition from MasterServer to GameServer.

ConnectingToGameserver   Transition to GameServer (client authenticates and joins/creates a room).

ConnectedToGameserver   Connected to GameServer (going to auth and join game).

Joining   Transition state while joining or creating a room on GameServer.

Joined   The client entered a room. The CurrentRoom and Players are known and you can now raise events.

Leaving   Transition state when leaving a room.

DisconnectingFromGameserver   Transition from GameServer to MasterServer (after leaving a room/game).

ConnectingToMasterserver   Connecting to MasterServer (includes sending authentication values).

Disconnecting   The client disconnects (from any server). This leads to state Disconnected.

Disconnected   The client is no longer connected (to any server). Connect to MasterServer to go on.

ConnectedToMasterserver   Connected to MasterServer. You might use matchmaking or join a lobby now.

ConnectedToMaster   Connected to MasterServer. You might use matchmaking or join a lobby now.

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   Clients disconnects (specifically) from the NameServer (usually to connect
to the MasterServer).

6.1.2.2 enum Photon.Pun.PunLogLevel `[strong]`

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.

6.1.2.3   enum Photon.Pun.RpcTarget `[strong]`

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 Photon.Pun.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.SerializationRate 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 Photon↩
View.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 Photon.Pun.PhotonAnimatorView, Photon.Pun.UtilityScripts.CullingHandler, Photon.Pun.Photon↩
TransformView, Photon.Pun.PhotonRigidbodyView, Photon.Pun.PhotonRigidbody2DView, and Photon.Pun.↩
UtilityScripts.SmoothSyncMovement.

6.2 Optional Gui Elements

Useful GUI elements for PUN.

Classes

- class Photon.Pun.UtilityScripts.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 Photon.Pun.UtilityScripts.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.

6.3 Callbacks

Callback Interfaces.

Classes

- interface Photon.Realtime.IConnectionCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover: Connection and Regions.
- interface Photon.Realtime.ILobbyCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover the Lobby.
- interface Photon.Realtime.IMatchmakingCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover Matchmaking.
- interface Photon.Realtime.IInRoomCallbacks

    Collection of "in room" callbacks for the Realtime Api to cover: Players entering or leaving, property updates and Master Client switching.
- interface Photon.Realtime.IOnEventCallback

    Event callback for the Realtime Api. Covers events from the server and those sent by clients via OpRaiseEvent.
- interface Photon.Realtime.IWebRpcCallback

    Interface for "WebRpc" callbacks for the Realtime Api. Currently includes only responses for Web RPCs.
- interface Photon.Pun.IPunObservable

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

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

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

6.3.1 Detailed Description

Callback Interfaces.

Chapter 7

Namespace Documentation

7.1 Photon Namespace Reference

Namespaces

- namespace Chat
- namespace Pun
- namespace Realtime

7.2 Photon.Chat Namespace Reference

Classes

- class AuthenticationValues

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

    A channel of communication in Photon Chat, updated by ChatClient and provided as READ ONLY.
- class ChatClient

    Central class of the Photon Chat API to connect, handle channels and messages.
- class ChatEventCode

    Wraps up internally used constants in Photon Chat events. You don't have to use them directly usually.
- class ChatOperationCode

    Wraps up codes for operations used internally in Photon Chat. You don't have to use them directly usually.
- class ChatParameterCode

    Wraps up codes for parameters (in operations and events) used internally in Photon Chat. You don't have to use them directly usually.
- class ChatPeer

    Provides basic operations of the Photon Chat server. This internal class is used by public ChatClient.
- class ChatUserStatus

    Contains commonly used status values for SetOnlineStatus. You can define your own.
- class ErrorCode

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

    Callback interface for Chat client side. Contains callback methods to notify your app about updates. Must be provided to new ChatClient in constructor
- class ParameterCode

    Class for constants. Codes for parameters of Operations and Events.

Enumerations

- enum ChatDisconnectCause

  Enumaration of causes for Disconnects (used in ChatClient.DisconnectedCause).

- enum CustomAuthenticationType : byte

  Options for optional "Custom Authentication" services used with Photon. Used by OpAuthenticate after connecting to Photon.

- enum ChatState

  Possible states for a Chat Client.

7.2.1 Enumeration Type Documentation

7.2.1.1 enum Photon.Chat.ChatDisconnectCause `[strong]`

Enumaration of causes for Disconnects (used in ChatClient.DisconnectedCause).

Read the individual descriptions to find out what to do about this type of disconnect.

Enumerator

None No error was tracked.

DisconnectByServerUserLimit OnStatusChanged: The CCUs count of your Photon Server License is exausted (temporarily).

ExceptionOnConnect OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

DisconnectByServer OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).

TimeoutDisconnect OnStatusChanged: This client detected that the server's responses are not received in due time. Maybe you send / receive too much?

Exception OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

InvalidAuthentication OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.

MaxCcuReached OnOperationResponse: Authenticate (temporarily) failed when using a Photon Cloud subscription without CCU Burst. Update your subscription.

InvalidRegion OnOperationResponse: Authenticate when the app's Photon Cloud subscription is locked to some (other) region(s). Update your subscription or change region.

OperationNotAllowedInCurrentState OnOperationResponse: Operation that's (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.

CustomAuthenticationFailed OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.

7.2.1.2 enum Photon.Chat.ChatState `[strong]`

Possible states for a Chat Client.

Enumerator

Uninitialized Peer is created but not used yet.

ConnectingToNameServer Connecting to name server.

ConnectedToNameServer Connected to name server.

Authenticating Authenticating on current server.

Authenticated Finished authentication on current server.

DisconnectingFromNameServer Disconnecting from name server. This is usually a transition from name server to frontend server.

ConnectingToFrontEnd Connecting to frontend server.

ConnectedToFrontEnd Connected to frontend server.

DisconnectingFromFrontEnd Disconnecting from frontend server.

QueuedComingFromFrontEnd Currently not used.

Disconnecting The client disconnects (from any server).

Disconnected The client is no longer connected (to any server).

7.2.1.3   enum Photon.Chat.CustomAuthenticationType : byte `[strong]`

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!

Oculus Authenticates users by their Oculus Account and token.

PlayStation Authenticates users by their PSN Account and token.

Xbox Authenticates users by their Xbox Account and XSTS token.

None Disables custom authentification. Same as not providing any AuthenticationValues for connect (more precisely for: OpAuthenticate).

7.3   Photon.Pun Namespace Reference

Namespaces

- namespace UtilityScripts

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.
- interface IPunInstantiateMagicCallback
- interface IPunObservable

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

  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 MonoBehaviourPun

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

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

- 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 **PhotonHandler**

  Internal Monobehaviour that allows Photon to run an Update loop.

- struct PhotonMessageInfo

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

- class PhotonNetwork

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

- class PhotonRigidbody2DView
- class PhotonRigidbodyView
- class PhotonStream

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

- 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.

- class PhotonTransformView
- class PhotonView

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

- class **PunEvent**

  Defines Photon event-codes as used by PUN.

- class PunExtensions

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

- class PunRPC

  Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

- class SceneManagerHelper
- class ServerSettings

  Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

Typedefs

- using **Debug** = UnityEngine.Debug
- using **Hashtable** = ExitGames.Client.Photon.Hashtable
- using **SupportClassPun** = ExitGames.Client.Photon.SupportClass

Enumerations

- enum ConnectMethod

  Which PhotonNetwork method was called to connect (which influences the regions we want pinged).

- enum PunLogLevel

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

- enum RpcTarget

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

- enum **ViewSynchronization**
- enum OwnershipOption

  Options to define how Ownership Transfer is handled per PhotonView.

7.3.1 Enumeration Type Documentation

7.3.1.1 enum Photon.Pun.ConnectMethod `[strong]`

Which PhotonNetwork method was called to connect (which influences the regions we want pinged).

PhotonNetwork.ConnectUsingSettings will call either ConnectToMaster, ConnectToRegion or ConnectToBest, depending on the settings.

7.3.1.2 enum Photon.Pun.OwnershipOption `[strong]`

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.

7.4 Photon.Pun.UtilityScripts Namespace Reference

Classes

- class ButtonInsideScrollList

  Button inside scroll list will stop scrolling ability of scrollRect container, so that when pressing down on a button and draggin up and down will not affect scrolling. this doesn't do anything if no scrollRect component found in Parent Hierarchy.

- class CellTree

  Represents the tree accessible from its root node.

- class CellTreeNode

  Represents a single node of the tree.

- class ConnectAndJoinRandom

  Simple component to call ConnectUsingSettings and to get into a PUN room easily.

- class CountdownTimer

  This is a basic CountdownTimer. In order to start the timer, the MasterClient can add a certain entry to the Custom Room Properties, which contains the property's name 'StartTime' and the actual start time describing the moment, the timer has been started. To have a synchronized timer, the best practice is to use PhotonNetwork.Time. In order to subscribe to the CountdownTimerHasExpired event you can call CountdownTimer.OnCountdownTimer↩ HasExpired += OnCountdownTimerIsExpired; from Unity's OnEnable function for example. For unsubscribing simply call CountdownTimer.OnCountdownTimerHasExpired -= OnCountdownTimerIsExpired;. You can do this from Unity's OnDisable function for example.

- class CullArea

  Represents the cull area used for network culling.

- class CullingHandler

  Handles the network culling.

- class EventSystemSpawner

  Event system spawner. Will add an EventSystem GameObject with an EventSystem component and a Standalone↩ InputModule component Use this in additive scene loading context where you would otherwise get a "Multiple eventsystem in scene... this is not supported" error from Unity

- class GraphicToggleIsOnTransition

Use this on toggles texts to have some color transition on the text depending on the isOn State.

- interface IPunTurnManagerCallbacks
- class MoveByKeys

    Very basic component to move a GameObject by WASD and Space.

- class OnClickDestroy

    Implements OnClick to destroy the GameObject it's attached to. Optionally a RPC is sent to do this.

- class OnClickInstantiate

    This component will instantiate a network GameObject when in a room and the user click on that component's Gameobject Uses PhysicsRaycaster for positioning

- class OnEscapeQuit

    This component will quit the application when escape key is pressed

- class OnJoinedInstantiate

    This component will instantiate a network GameObject when a room is joined

- class OnPointerOverTooltip

    Set focus to a given photonView when pointed is over

- class OnStartDelete

    This component will destroy the GameObject it is attached to (in Start()).

- 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.

- class PlayerNumbering

    Implements consistent numbering in a room/game with help of room properties. Access them by Player.GetPlayer↩ Number() extension.

- class PlayerNumberingExtensions

    Extension used for PlayerRoomIndexing and Player class.

- class PointedAtGameObjectInfo

    Display ViewId, OwnerActorNr, IsCeneView and IsMine when clicked.

- class PunPlayerScores

    Scoring system for PhotonPlayer

- class PunTeams

    Implements teams in a room/game with help of player properties. Access them by Player.GetTeam extension.

- class PunTurnManager

    Pun turnBased Game manager. Provides an Interface (IPunTurnManagerCallbacks) for the typical turn flow and logic, between players Provides Extensions for Player, Room and RoomInfo to feature dedicated api for TurnBased Needs

- class ScoreExtensions
- class SmoothSyncMovement

    Smoothed out movement for network gameobjects

- class StatesGui

    Output detailed information about Pun Current states, using the old Unity UI framework.

- class TabViewManager

    Tab view manager. Handles Tab views activation and deactivation, and provides a Unity Event Callback when a tab was selected.

- class TeamExtensions

    Extension used for PunTeams and Player class. Wraps access to the player's custom property.

- class TextButtonTransition

    Use this on Button texts to have some color transition on the text as well without corrupting button's behaviour.

- class TextToggleIsOnTransition

    Use this on toggles texts to have some color transition on the text depending on the isOn State.

- class TurnExtensions

7.5   Photon.Realtime Namespace Reference

Classes

- class ActorProperties

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

- class AppSettings

    Settings for Photon application(s) and the server to connect to.

- class AuthenticationValues

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

- class **ConnectionCallbacksContainer**

    Container type for callbacks defined by IConnectionCallbacks. See LoadBalancingCallbackTargets.

- class ConnectionHandler

- class EncryptionDataParameters

- class EnterRoomParams

- class ErrorCode

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

- class EventCode

    Class for constants. These values are for events defined by Photon Loadbalancing.

- class Extensions

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

- class FriendInfo

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

- class GamePropertyKey

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

- interface IConnectionCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover: Connection and Regions.

- interface IInRoomCallbacks

    Collection of "in room" callbacks for the Realtime Api to cover: Players entering or leaving, property updates and Master Client switching.

- interface ILobbyCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover the Lobby.

- interface IMatchmakingCallbacks

    Collection of "organizational" callbacks for the Realtime Api to cover Matchmaking.

- class **InRoomCallbacksContainer**

    Container type for callbacks defined by IInRoomCallbacks. See InRoomCallbackTargets.

- interface IOnEventCallback

    Event callback for the Realtime Api. Covers events from the server and those sent by clients via OpRaiseEvent.

- interface IWebRpcCallback

    Interface for "WebRpc" callbacks for the Realtime Api. Currently includes only responses for Web RPCs.

- class LoadBalancingClient

    This class implements the Photon LoadBalancing workflow by using a LoadBalancingPeer. It keeps a state and will automatically execute transitions between the Master and Game Servers.

- class LoadBalancingPeer

    A LoadbalancingPeer provides the operations and enum definitions needed to use the loadbalancing server application which is also used in Photon Cloud.

- class **LobbyCallbacksContainer**

    Container type for callbacks defined by ILobbyCallbacks. See LobbyCallbackTargets.

- class **MatchMakingCallbacksContainer**

    Container type for callbacks defined by IMatchmakingCallbacks. See MatchMakingCallbackTargets.

- class OperationCode

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

- class OpJoinRandomRoomParams
- class ParameterCode

    Class for constants. Codes for parameters of Operations and Events.

- class PhotonPing
- class PingMono

    Uses C# Socket class from System.Net.Sockets (as Unity usually does).

- class Player

    Summarizes a "player" within a room, identified (in that room) by ID (or "actorNumber").

- class RaiseEventOptions

    Aggregates several less-often used options for operation RaiseEvent. See field descriptions for usage details.

- class Region
- class RegionHandler

    Provides methods to work with Photon's regions (Photon Cloud) and can be use to find the one with best ping.

- class RegionPinger
- class Room

    This class represents a room a client joins/joined.

- 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 (IsOpen, MaxPlayers, etc).

- class RoomOptions

    Wraps up common room properties needed when you create rooms. Read the individual entries for more details.

- class SupportLogger

    Helper class to debug log basic information about Photon client and vital traffic statistics.

- class TypedLobby

    Refers to a specific lobby (and type) on the server.

- class TypedLobbyInfo
- class WebFlags

    Optional flags to be used in Photon client SDKs with Op RaiseEvent and Op SetProperties. Introduced mainly for webhooks 1.2 to control behavior of forwarded HTTP requests.

- class **WebRpcCallbacksContainer**

    Container type for callbacks defined by IWebRpcCallback. See WebRpcCallbackTargets.

- class WebRpcResponse

    Reads an operation response of a WebRpc and provides convenient access to most common values.

Typedefs

- using **SupportClass** = ExitGames.Client.Photon.SupportClass

Enumerations

- enum ClientState

    State values for a client, which handles switching Photon server types, some operations, etc.

- enum DisconnectCause

    Enumaration of causes for Disconnects (used in LoadBalancingClient.DisconnectedCause).

- enum ServerConnection

    Available server (types) for internally used field: server.

- enum EncryptionMode

    Defines how the communication gets encrypted.

- 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.

- enum MatchmakingMode : byte

    Options for matchmaking rules for OpJoinRandom.

- 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.

- 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.

- enum PropertyTypeFlag : byte

    Flags for "types of properties", being used as filter in OpGetProperties.

- 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.

- enum AuthModeOption

    Options for authentication modes. From "classic" auth on each server to AuthOnce (on NameServer).

- enum CustomAuthenticationType : byte

    Options for optional "Custom Authentication" services used with Photon. Used by OpAuthenticate after connecting to Photon.

7.5.1 Enumeration Type Documentation

7.5.1.1 enum Photon.Realtime.AuthModeOption `[strong]`

Options for authentication modes. From "classic" auth on each server to AuthOnce (on NameServer).

7.5.1.2 enum Photon.Realtime.CustomAuthenticationType : byte `[strong]`

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!

>   ***Oculus*** Authenticates users by their Oculus Account and token.

>   ***PlayStation*** Authenticates users by their PSN Account and token.

>   ***Xbox*** Authenticates users by their Xbox Account and XSTS token.

>   ***None*** Disables custom authentification. Same as not providing any AuthenticationValues for connect (more precisely for: OpAuthenticate).

7.5.1.3 enum Photon.Realtime.DisconnectCause `[strong]`

Enumaration of causes for Disconnects (used in LoadBalancingClient.DisconnectedCause).

Read the individual descriptions to find out what to do about this type of disconnect.

Enumerator

>   ***None*** No error was tracked.

DisconnectByServerUserLimit OnStatusChanged: The CCUs count of your Photon Server License is exausted (temporarily).

ExceptionOnConnect OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

DisconnectByServer OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).

TimeoutDisconnect OnStatusChanged: This client detected that the server's responses are not received in due time. Maybe you send / receive too much?

Exception OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

InvalidAuthentication OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.

MaxCcuReached OnOperationResponse: Authenticate (temporarily) failed when using a Photon Cloud subscription without CCU Burst. Update your subscription.

InvalidRegion OnOperationResponse: Authenticate when the app's Photon Cloud subscription is locked to some (other) region(s). Update your subscription or master server address.

OperationNotAllowedInCurrentState OnOperationResponse: Operation that's (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.

CustomAuthenticationFailed OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.

DisconnectByServerLogic OnStatusChanged: The server disconnected this client from within the room's logic (the C# code).

AuthenticationTicketExpired The authentication ticket should provide access to any Photon Cloud server without doing another authentication-service call. However, the ticket expired.

DisconnectByClientLogic OnStatusChanged: The client disconnected from within the logic (the C# code).

7.5.1.4 enum Photon.Realtime.EncryptionMode `[strong]`

Defines how the communication gets encrypted.

Enumerator

PayloadEncryption This is the default encryption mode: Messages get encrypted only on demand (when you send operations with the "encrypt" parameter set to true).

DatagramEncryption With this encryption mode for UDP, the connection gets setup and all further datagrams get encrypted almost entirely. On-demand message encryption (like in PayloadEncryption) is skipped.

7.5.1.5 enum Photon.Realtime.EventCaching : byte `[strong]`

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.

7.5.1.6   enum Photon.Realtime.JoinMode : byte   `[strong]`

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.

7.5.1.7   enum Photon.Realtime.LobbyType : byte   `[strong]`

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.

7.5.1.8   enum Photon.Realtime.MatchmakingMode : byte   `[strong]`

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.

7.5.1.9   enum Photon.Realtime.PropertyTypeFlag : byte `[strong]`

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'

7.5.1.10   enum Photon.Realtime.ReceiverGroup : byte `[strong]`

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.

7.5.1.11   enum Photon.Realtime.ServerConnection `[strong]`

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).

Chapter 8

Class Documentation

8.1 Photon.Realtime.ActorProperties Class Reference

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

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 Photon.Realtime.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 Photon.Realtime.ActorProperties.PlayerName = 255

(255) Name of a player/actor.

8.1.2.3 const byte Photon.Realtime.ActorProperties.UserId = 253

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

8.2 Photon.Realtime.AppSettings Class Reference

Settings for Photon application(s) and the server to connect to.

Public Member Functions

- string ToStringFull ()

  ToString but with more details.

Public Attributes

- string AppIdRealtime

  AppId for Realtime or PUN.
- string AppIdChat

  AppId for the Chat Api.
- string AppIdVoice

  AppId for use in the Voice Api.
- string AppVersion

  The AppVersion can be used to identify builds and will split the AppId distinct "Virtual AppIds" (important for match-making).
- bool UseNameServer = true

  If false, the app will attempt to connect to a Master Server (which is obsolete but sometimes still necessary).
- string FixedRegion

  Can be set to any of the Photon Cloud's region names to directly connect to that region.
- string Server

  The address (hostname or IP) of the server to connect to.
- int Port

  If not null, this sets the port of the first Photon server to connect to (that will "forward" the client as needed).
- ConnectionProtocol Protocol = ConnectionProtocol.Udp

  The network level protocol to use.
- bool EnableLobbyStatistics

  If true, the client will request the list of currently available lobbies.
- DebugLevel NetworkLogging = DebugLevel.ERROR

  Log level for the network lib.

Properties

- bool IsMasterServerAddress `[get]`

  If true, the Server field contains a Master Server address (if any address at all).
- bool IsBestRegion `[get]`

  If true, the client should fetch the region list from the Name Server and find the one with best ping.
- bool IsDefaultNameServer `[get]`

  If true, the default nameserver address for the Photon Cloud should be used.
- bool IsDefaultPort `[get]`

  If true, the default ports for a protocol will be used.

8.2.1 Detailed Description

Settings for Photon application(s) and the server to connect to.

This is Serializable for Unity, so it can be included in ScriptableObject instances.

8.2.2 Member Function Documentation

8.2.2.1 string Photon.Realtime.AppSettings.ToStringFull ( )

ToString but with more details.

8.2.3 Member Data Documentation

8.2.3.1 string Photon.Realtime.AppSettings.AppIdChat

AppId for the Chat Api.

8.2.3.2 string Photon.Realtime.AppSettings.AppIdRealtime

AppId for Realtime or PUN.

8.2.3.3 string Photon.Realtime.AppSettings.AppIdVoice

AppId for use in the Voice Api.

8.2.3.4 string Photon.Realtime.AppSettings.AppVersion

The AppVersion can be used to identify builds and will split the AppId distinct "Virtual AppIds" (important for match-making).

8.2.3.5 bool Photon.Realtime.AppSettings.EnableLobbyStatistics

If true, the client will request the list of currently available lobbies.

8.2.3.6 string Photon.Realtime.AppSettings.FixedRegion

Can be set to any of the Photon Cloud's region names to directly connect to that region.

if this IsNullOrEmpty() AND UseNameServer == true, use BestRegion. else, use a server

8.2.3.7 DebugLevel Photon.Realtime.AppSettings.NetworkLogging = DebugLevel.ERROR

Log level for the network lib.

8.2.3.8 int Photon.Realtime.AppSettings.Port

If not null, this sets the port of the first Photon server to connect to (that will "forward" the client as needed).

8.2.3.9 ConnectionProtocol Photon.Realtime.AppSettings.Protocol = ConnectionProtocol.Udp

The network level protocol to use.

8.2.3.10 string Photon.Realtime.AppSettings.Server

The address (hostname or IP) of the server to connect to.

8.2.3.11 bool Photon.Realtime.AppSettings.UseNameServer = true

If false, the app will attempt to connect to a Master Server (which is obsolete but sometimes still necessary).

if true, Server points to a NameServer (or is null, using the default), else it points to a MasterServer.

8.2.4 Property Documentation

8.2.4.1 bool Photon.Realtime.AppSettings.IsBestRegion `[get]`

If true, the client should fetch the region list from the Name Server and find the one with best ping.

See "Best Region" in the online docs.

8.2.4.2 bool Photon.Realtime.AppSettings.IsDefaultNameServer `[get]`

If true, the default nameserver address for the Photon Cloud should be used.

8.2.4.3 bool Photon.Realtime.AppSettings.IsDefaultPort `[get]`

If true, the default ports for a protocol will be used.

8.2.4.4 bool Photon.Realtime.AppSettings.IsMasterServerAddress `[get]`

If true, the Server field contains a Master Server address (if any address at all).

8.3 Photon.Realtime.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 SetAuthPostData (Dictionary< string, object > dictData)

    Sets data to be passed-on to the auth service as Json (Content-Type: "application/json") 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]`

    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..

8.3.1 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 AuthValues are sent in OpAuthenticate when you connect, so they must be set before you connect. Should you not set any AuthValues, PUN will create them and set the playerName as userId in them. If the AuthValues.userId is null or empty when it's sent to the server, then the Photon Server assigns a userId!

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

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Photon.Realtime.AuthenticationValues.AuthenticationValues ( )

Creates empty auth values without any info.

8.3.2.2 Photon.Realtime.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.

8.3.3 Member Function Documentation

8.3.3.1 virtual void Photon.Realtime.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.

8.3.3.2 virtual void Photon.Realtime.AuthenticationValues.SetAuthPostData (string *stringData*) `[virtual]`

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

AuthPostData is just one value. Each SetAuthPostData replaces any previous value. It can be either a string, a byte[] or a dictionary. Each SetAuthPostData replaces any previous value.

Parameters

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

8.3.3.3 virtual void Photon.Realtime.AuthenticationValues.SetAuthPostData (byte[ ] *byteData*) `[virtual]`

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

AuthPostData is just one value. Each SetAuthPostData replaces any previous value. It can be either a string, a byte[] or a dictionary. Each SetAuthPostData replaces any previous value.

Parameters

byteData	Binary token / auth-data to pass on.

8.3.3.4 virtual void Photon.Realtime.AuthenticationValues.SetAuthPostData (Dictionary< string, object > *dictData*) `[virtual]`

Sets data to be passed-on to the auth service as Json (Content-Type: "application/json") via Post.

AuthPostData is just one value. Each SetAuthPostData replaces any previous value. It can be either a string, a byte[] or a dictionary. Each SetAuthPostData replaces any previous value.

Parameters

dictData	A authentication-data dictionary will be converted to Json and passed to the Auth webservice via HTTP Post.

8.3.4   Property Documentation

8.3.4.1   string Photon.Realtime.AuthenticationValues.AuthGetParameters `[get],[set]`

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

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

8.3.4.2   object Photon.Realtime.AuthenticationValues.AuthPostData `[get]`

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

8.3.4.3 CustomAuthenticationType Photon.Realtime.AuthenticationValues.AuthType `[get],[set]`

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

8.3.4.4 string Photon.Realtime.AuthenticationValues.Token `[get],[set]`

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

8.3.4.5 string Photon.Realtime.AuthenticationValues.UserId `[get],[set]`

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

See remarks of AuthValues for info about how this is set and used.

8.4 Photon.Chat.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 ()

  Transform this object into string.

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]`

  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..

8.4.1 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←` `://dashboard.photonengine.com`

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Photon.Chat.AuthenticationValues.AuthenticationValues ( )

Creates empty auth values without any info.

8.4.2.2 Photon.Chat.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.

8.4.3 Member Function Documentation

8.4.3.1 virtual void Photon.Chat.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.

8.4.3.2 virtual void Photon.Chat.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 Auth← PostData to null.

8.4.3.3 virtual void Photon.Chat.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.

8.4.3.4 override string Photon.Chat.AuthenticationValues.ToString ( )

Transform this object into string.

Returns

> string representation of this object.

8.4.4 Property Documentation

8.4.4.1 string Photon.Chat.AuthenticationValues.AuthGetParameters `[get],[set]`

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

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

8.4.4.2 object Photon.Chat.AuthenticationValues.AuthPostData `[get]`

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

8.4.4.3 CustomAuthenticationType Photon.Chat.AuthenticationValues.AuthType `[get],[set]`

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

8.4.4.4 string Photon.Chat.AuthenticationValues.Token `[get],[set]`

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

8.4.4.5 string Photon.Chat.AuthenticationValues.UserId `[get],[set]`

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

8.5 Photon.Pun.UtilityScripts.ButtonInsideScrollList Class Reference

Button inside scroll list will stop scrolling ability of scrollRect container, so that when pressing down on a button and draggin up and down will not affect scrolling. this doesn't do anything if no scrollRect component found in Parent Hierarchy.

Inherits MonoBehaviour, IPointerDownHandler, and IPointerUpHandler.

8.5.1 Detailed Description

Button inside scroll list will stop scrolling ability of scrollRect container, so that when pressing down on a button and draggin up and down will not affect scrolling. this doesn't do anything if no scrollRect component found in Parent Hierarchy.

8.6 Photon.Pun.UtilityScripts.CellTree Class Reference

Represents the tree accessible from its root node.

Public Member Functions

- CellTree ()

    Default constructor.
- CellTree (CellTreeNode root)

    Constructor to define the root node.

Properties

- CellTreeNode RootNode `[get]`

    Represents the root node of the cell tree.

8.6.1 Detailed Description

Represents the tree accessible from its root node.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Photon.Pun.UtilityScripts.CellTree.CellTree ( )

Default constructor.

8.6.2.2 Photon.Pun.UtilityScripts.CellTree.CellTree (CellTreeNode *root*)

Constructor to define the root node.

Parameters

root	The root node of the tree.

8.6.3 Property Documentation

8.6.3.1 CellTreeNode Photon.Pun.UtilityScripts.CellTree.RootNode `[get]`

Represents the root node of the cell tree.

8.7 Photon.Pun.UtilityScripts.CellTreeNode Class Reference

Represents a single node of the tree.

Public Types

- enum **ENodeType**

Public Member Functions

- *CellTreeNode* ()

    Default constructor.
- *CellTreeNode* (byte id, ENodeType nodeType, *CellTreeNode* parent)

    Constructor to define the ID and the node type as well as setting a parent node.
- void *AddChild* (*CellTreeNode* child)

    Adds the given child to the node.
- void *Draw* ()

    Draws the cell in the editor.
- void *GetActiveCells* (List< byte > activeCells, bool yIsUpAxis, Vector3 position)

    Gathers all cell IDs the player is currently inside or nearby.
- bool *IsPointInsideCell* (bool yIsUpAxis, Vector3 point)

    Checks if the given point is inside the cell.
- bool *IsPointNearCell* (bool yIsUpAxis, Vector3 point)

    Checks if the given point is near the cell.

Public Attributes

- byte *Id*

    Represents the unique ID of the cell.
- Vector3 *Center*

    Represents the center, top-left or bottom-right position of the cell or the size of the cell.
- ENodeType *NodeType*

    Describes the current node type of the cell tree node.
- *CellTreeNode Parent*

    Reference to the parent node.
- List< *CellTreeNode* > *Childs*

    A list containing all child nodes.

8.7.1 Detailed Description

Represents a single node of the tree.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 Photon.Pun.UtilityScripts.CellTreeNode.CellTreeNode (   )

Default constructor.

8.7.2.2 Photon.Pun.UtilityScripts.CellTreeNode.CellTreeNode (byte *id,* ENodeType *nodeType,* CellTreeNode *parent*)

Constructor to define the ID and the node type as well as setting a parent node.

Parameters

id	The ID of the cell is used as the interest group.
nodeType	The node type of the cell tree node.

parent	The parent node of the cell tree node.

8.7.3 Member Function Documentation

8.7.3.1 void Photon.Pun.UtilityScripts.CellTreeNode.AddChild (**CellTreeNode** *child*)

Adds the given child to the node.

Parameters

child	The child which is added to the node.

8.7.3.2 void Photon.Pun.UtilityScripts.CellTreeNode.Draw ( )

Draws the cell in the editor.

8.7.3.3 void Photon.Pun.UtilityScripts.CellTreeNode.GetActiveCells (List< byte > *activeCells,* bool *yIsUpAxis,* Vector3 *position*)

Gathers all cell IDs the player is currently inside or nearby.

Parameters

activeCells	The list to add all cell IDs to the player is currently inside or nearby.
yIsUpAxis	Describes if the y-axis is used as up-axis.
position	The current position of the player.

8.7.3.4 bool Photon.Pun.UtilityScripts.CellTreeNode.IsPointInsideCell (bool *yIsUpAxis,* Vector3 *point*)

Checks if the given point is inside the cell.

Parameters

yIsUpAxis	Describes if the y-axis is used as up-axis.
point	The point to check.

Returns

True if the point is inside the cell, false if the point is not inside the cell.

8.7.3.5 bool Photon.Pun.UtilityScripts.CellTreeNode.IsPointNearCell (bool *yIsUpAxis,* Vector3 *point*)

Checks if the given point is near the cell.

Parameters

yIsUpAxis	Describes if the y-axis is used as up-axis.
point	The point to check.

Returns

True if the point is near the cell, false if the point is too far away.

8.7.4 Member Data Documentation

8.7.4.1 Vector3 Photon.Pun.UtilityScripts.CellTreeNode.Center

Represents the center, top-left or bottom-right position of the cell or the size of the cell.

8.7.4.2 List<**CellTreeNode**> Photon.Pun.UtilityScripts.CellTreeNode.Childs

A list containing all child nodes.

8.7.4.3 byte Photon.Pun.UtilityScripts.CellTreeNode.Id

Represents the unique ID of the cell.

8.7.4.4 ENodeType Photon.Pun.UtilityScripts.CellTreeNode.NodeType

Describes the current node type of the cell tree node.

8.7.4.5 CellTreeNode Photon.Pun.UtilityScripts.CellTreeNode.Parent

Reference to the parent node.

8.8 Photon.Chat.ChatChannel Class Reference

A channel of communication in Photon Chat, updated by ChatClient and provided as READ ONLY.

Public Member Functions

- ChatChannel (string name)

    Used internally to create new channels. This does NOT create a channel on the server! Use ChatClient.Subscribe.
- void Add (string sender, object message, int msgId)

    Used internally to add messages to this channel.
- void Add (string[ ] senders, object[ ] messages, int lastMsgId)

    Used internally to add messages to this channel.
- void TruncateMessages ()

    Reduces the number of locally cached messages in this channel to the MessageLimit (if set).
- void ClearMessages ()

    Clear the local cache of messages currently stored. This frees memory but doesn't affect the server.
- string ToStringMessages ()

    Provides a string-representation of all messages in this channel.

Public Attributes

- readonly string Name

    Name of the channel (used to subscribe and unsubscribe).
- readonly List< string > Senders = new List<string>()

    Senders of messages in chronoligical order. Senders and Messages refer to each other by index. Senders[x] is the sender of Messages[x].
- readonly List< object > Messages = new List<object>()

Messages in chronological order. Senders and Messages refer to each other by index. Senders[x] is the sender of Messages[x].

- int MessageLimit

  If greater than 0, this channel will limit the number of messages, that it caches locally.

Properties

- bool IsPrivate `[get, protected set]`

  Is this a private 1:1 channel?

- int MessageCount `[get]`

  Count of messages this client still buffers/knows for this channel.

- int LastMsgId `[get, protected set]`

  ID of the last message received.

8.8.1 Detailed Description

A channel of communication in Photon Chat, updated by ChatClient and provided as READ ONLY.

Contains messages and senders to use (read!) and display by your GUI. Access these by: ChatClient.Public↩ Channels ChatClient.PrivateChannels

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Photon.Chat.ChatChannel.ChatChannel (string *name*)

Used internally to create new channels. This does NOT create a channel on the server! Use ChatClient.Subscribe.

8.8.3 Member Function Documentation

8.8.3.1 void Photon.Chat.ChatChannel.Add (string *sender,* object *message,* int *msgId*)

Used internally to add messages to this channel.

8.8.3.2 void Photon.Chat.ChatChannel.Add (string[ ] *senders,* object[ ] *messages,* int *lastMsgId*)

Used internally to add messages to this channel.

8.8.3.3 void Photon.Chat.ChatChannel.ClearMessages ( )

Clear the local cache of messages currently stored. This frees memory but doesn't affect the server.

8.8.3.4 string Photon.Chat.ChatChannel.ToStringMessages ( )

Provides a string-representation of all messages in this channel.

Returns

All known messages in format "Sender: Message", line by line.

8.8.3.5 void Photon.Chat.ChatChannel.TruncateMessages ( )

Reduces the number of locally cached messages in this channel to the MessageLimit (if set).

8.8.4 Member Data Documentation

8.8.4.1 int Photon.Chat.ChatChannel.MessageLimit

If greater than 0, this channel will limit the number of messages, that it caches locally.

8.8.4.2 readonly List<object> Photon.Chat.ChatChannel.Messages = new List<object>()

Messages in chronological order. Senders and Messages refer to each other by index. Senders[x] is the sender of Messages[x].

8.8.4.3 readonly string Photon.Chat.ChatChannel.Name

Name of the channel (used to subscribe and unsubscribe).

8.8.4.4 readonly List<string> Photon.Chat.ChatChannel.Senders = new List<string>()

Senders of messages in chronoligical order. Senders and Messages refer to each other by index. Senders[x] is the sender of Messages[x].

8.8.5 Property Documentation

8.8.5.1 bool Photon.Chat.ChatChannel.IsPrivate `[get]`,`[protected set]`

Is this a private 1:1 channel?

8.8.5.2 int Photon.Chat.ChatChannel.LastMsgId `[get]`,`[protected set]`

ID of the last message received.

8.8.5.3 int Photon.Chat.ChatChannel.MessageCount `[get]`

Count of messages this client still buffers/knows for this channel.

8.9 Photon.Chat.ChatClient Class Reference

Central class of the Photon Chat API to connect, handle channels and messages.

Inherits IPhotonPeerListener.

Public Member Functions

- bool CanChatInChannel (string channelName)
    Checks if this client is ready to publish messages inside a public channel.
- ChatClient (IChatClientListener listener, ConnectionProtocol protocol=ConnectionProtocol.Udp)
    Chat client constructor.

- bool Connect (string appId, string appVersion, AuthenticationValues authValues)

  Connects this client to the Photon Chat Cloud service, which will also authenticate the user (and set a UserId).

- bool ConnectAndSetStatus (string appId, string appVersion, AuthenticationValues authValues, int status=ChatUserStatus.Online, object message=null)

  Connects this client to the Photon Chat Cloud service, which will also authenticate the user (and set a UserId). This also sets an online status once connected. By default it will set user status to ChatUserStatus.Online. See SetOnlineStatus(int,object) for more information.

- void Service ()

  Must be called regularly to keep connection between client and server alive and to process incoming messages.

- void SendAcksOnly ()

  Obsolete: Better use UseBackgroundWorkerForSending and Service().

- void Disconnect ()

  Disconnects from the Chat Server by sending a "disconnect command", which prevents a timeout server-side.

- void StopThread ()

  Locally shuts down the connection to the Chat Server. This resets states locally but the server will have to timeout this peer.

- bool Subscribe (string[ ] channels)

  Sends operation to subscribe to a list of channels by name.

- bool Subscribe (string[ ] channels, int[ ] lastMsgIds)

  Sends operation to subscribe to a list of channels by name and possibly retrieve messages we did not receive while unsubscribed.

- bool Subscribe (string[ ] channels, int messagesFromHistory)

  Sends operation to subscribe client to channels, optionally fetching a number of messages from the cache.

- bool Unsubscribe (string[ ] channels)

  Unsubscribes from a list of channels, which stops getting messages from those.

- bool PublishMessage (string channelName, object message, bool forwardAsWebhook=false)

  Sends a message to a public channel which this client subscribed to.

- bool SendPrivateMessage (string target, object message, bool forwardAsWebhook=false)

  Sends a private message to a single target user. Calls OnPrivateMessage on the receiving client.

- bool SendPrivateMessage (string target, object message, bool encrypt, bool forwardAsWebhook)

  Sends a private message to a single target user. Calls OnPrivateMessage on the receiving client.

- bool SetOnlineStatus (int status)

  Sets the user's status without changing your status-message.

- bool SetOnlineStatus (int status, object message)

  Sets the user's status without changing your status-message.

- bool AddFriends (string[ ] friends)

  Adds friends to a list on the Chat Server which will send you status updates for those.

- bool RemoveFriends (string[ ] friends)

  Removes the provided entries from the list on the Chat Server and stops their status updates.

- string GetPrivateChannelNameByUser (string userName)

  Get you the (locally used) channel name for the chat between this client and another user.

- bool TryGetChannel (string channelName, bool isPrivate, out ChatChannel channel)

  Simplified access to either private or public channels by name.

- bool TryGetChannel (string channelName, out ChatChannel channel)

  Simplified access to all channels by name. Checks public channels first, then private ones.

Public Attributes

- int MessageLimit

    If greater than 0, new channels will limit the number of messages they cache locally.

- readonly Dictionary< string, ChatChannel > PublicChannels

    Public channels this client is subscribed to.

- readonly Dictionary< string, ChatChannel > PrivateChannels

    Private channels in which this client has exchanged messages.

- ChatPeer chatPeer = null

    The Chat Peer used by this client.

Properties

- string NameServerAddress `[get]`

    The address of last connected Name Server.

- string FrontendAddress `[get]`

    The address of the actual chat server assigned from NameServer. Public for read only.

- string ChatRegion `[get, set]`

    Settable only before you connect! Defaults to "EU".

- ChatState State `[get]`

    Current state of the ChatClient. Also use CanChat.

- ChatDisconnectCause DisconnectedCause `[get]`

    Disconnection cause. Check this inside IChatClientListener.OnDisconnected.

- bool CanChat `[get]`

    Checks if this client is ready to send messages.

- string AppVersion `[get]`

    The version of your client. A new version also creates a new "virtual app" to separate players from older client versions.

- string AppId `[get]`

    The AppID as assigned from the Photon Cloud.

- AuthenticationValues AuthValues `[get, set]`

    Settable only before you connect!

- string UserId `[get]`

    The unique ID of a user/person, stored in AuthValues.UserId. Set it before you connect.

- bool UseBackgroundWorkerForSending `[get, set]`

    Defines if a background thread will call SendOutgoingCommands, while your code calls Service to dispatch received messages.

- ConnectionProtocol TransportProtocol `[get, set]`

    Exposes the TransportProtocol of the used PhotonPeer. Settable while not connected.

- Dictionary< ConnectionProtocol, Type > SocketImplementationConfig `[get]`

    Defines which IPhotonSocket class to use per ConnectionProtocol.

- DebugLevel DebugOut `[get, set]`

    Sets the level (and amount) of debug output provided by the library.

8.9.1 Detailed Description

Central class of the Photon Chat API to connect, handle channels and messages.

This class must be instantiated with a IChatClientListener instance to get the callbacks. Integrate it into your game loop by calling Service regularly. If the target platform supports Threads/Tasks, set UseBackgroundWorkerFor↩ Sending = true, to let the ChatClient keep the connection by sending from an independent thread.

Call Connect with an AppId that is setup as Photon Chat application. Note: Connect covers multiple messages between this client and the servers. A short workflow will connect you to a chat server.

Each ChatClient resembles a user in chat (set in Connect). Each user automatically subscribes a channel for incoming private messages and can message any other user privately. Before you publish messages in any non-private channel, that channel must be subscribed.

PublicChannels is a list of subscribed channels, containing messages and senders. PrivateChannels contains all incoming and sent private messages.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Photon.Chat.ChatClient.ChatClient (IChatClientListener *listener,* ConnectionProtocol *protocol* = ConnectionProtocol.Udp)

Chat client constructor.

Parameters

listener	The chat listener implementation.
protocol	Connection protocol to be used by this client. Default is ConnectionProtocol.Udp.

8.9.3 Member Function Documentation

8.9.3.1 bool Photon.Chat.ChatClient.AddFriends (string[ ] *friends*)

Adds friends to a list on the Chat Server which will send you status updates for those.

AddFriends and RemoveFriends enable clients to handle their friend list in the Photon Chat server. Having users on your friends list gives you access to their current online status (and whatever info your client sets in it).

Each user can set an online status consisting of an integer and an arbitratry (serializable) object. The object can be null, Hashtable, object[] or anything else Photon can serialize.

The status is published automatically to friends (anyone who set your user ID with AddFriends).

Photon flushes friends-list when a chat client disconnects, so it has to be set each time. If your community API gives you access to online status already, you could filter and set online friends in AddFriends.

Actual friend relations are not persistent and have to be stored outside of Photon.

Parameters

friends	Array of friend userIds.

Returns

If the operation could be sent.

8.9.3.2 bool Photon.Chat.ChatClient.CanChatInChannel (string *channelName*)

Checks if this client is ready to publish messages inside a public channel.

Parameters

channelName	The channel to do the check with.

Returns

Whether or not this client is ready to publish messages inside the public channel with the specified channel↩
Name.

8.9.3.3   bool Photon.Chat.ChatClient.Connect (string *appId,* string *appVersion,* AuthenticationValues *authValues*)

Connects this client to the Photon Chat Cloud service, which will also authenticate the user (and set a UserId).

Parameters

appId	Get your Photon Chat AppId from the `Dashboard`.
appVersion	Any version string you make up. Used to separate users and variants of your clients, which might be incompatible.
authValues	Values for authentication. You can leave this null, if you set a UserId before. If you set authValues, they will override any UserId set before.

Returns

8.9.3.4   bool Photon.Chat.ChatClient.ConnectAndSetStatus (string *appId,* string *appVersion,* AuthenticationValues *authValues,* int *status = ***ChatUserStatus.Online*,* object *message = *`null`)**

Connects this client to the Photon Chat Cloud service, which will also authenticate the user (and set a UserId). This also sets an online status once connected. By default it will set user status to ChatUserStatus.Online. See SetOnlineStatus(int,object) for more information.

Parameters

appId	Get your Photon Chat AppId from the `Dashboard`.
appVersion	Any version string you make up. Used to separate users and variants of your clients, which might be incompatible.
authValues	Values for authentication. You can leave this null, if you set a UserId before. If you set authValues, they will override any UserId set before.
status	User status to set when connected. Predefined states are in class ChatUserStatus. Other values can be used at will.
message	Optional status Also sets a status-message which your friends can get.

Returns

If the connection attempt could be sent at all.

8.9.3.5   void Photon.Chat.ChatClient.Disconnect (   )

Disconnects from the Chat Server by sending a "disconnect command", which prevents a timeout server-side.

8.9.3.6   string Photon.Chat.ChatClient.GetPrivateChannelNameByUser (string *userName*)

Get you the (locally used) channel name for the chat between this client and another user.

Parameters

userName	Remote user's name or UserId.

Returns

> The (locally used) channel name for a private channel.

8.9.3.7 bool Photon.Chat.ChatClient.PublishMessage (string *channelName,* object *message,* bool *forwardAsWebhook =* `false`)

Sends a message to a public channel which this client subscribed to.

Before you publish to a channel, you have to subscribe it. Everyone in that channel will get the message.

Parameters

channelName	Name of the channel to publish to.
message	Your message (string or any serializable data).
forwardAs←↩ Webhook	Optionally, public messages can be forwarded as webhooks. Configure webhooks for your Chat app to use this.

Returns

> False if the client is not yet ready to send messages.

8.9.3.8 bool Photon.Chat.ChatClient.RemoveFriends (string[ ] *friends*)

Removes the provided entries from the list on the Chat Server and stops their status updates.

Photon flushes friends-list when a chat client disconnects. Unless you want to remove individual entries, you don't have to RemoveFriends.

AddFriends and RemoveFriends enable clients to handle their friend list in the Photon Chat server. Having users on your friends list gives you access to their current online status (and whatever info your client sets in it).

Each user can set an online status consisting of an integer and an arbitratry (serializable) object. The object can be null, Hashtable, object[] or anything else Photon can serialize.

The status is published automatically to friends (anyone who set your user ID with AddFriends).

Photon flushes friends-list when a chat client disconnects, so it has to be set each time. If your community API gives you access to online status already, you could filter and set online friends in AddFriends.

Actual friend relations are not persistent and have to be stored outside of Photon.

AddFriends and RemoveFriends enable clients to handle their friend list in the Photon Chat server. Having users on your friends list gives you access to their current online status (and whatever info your client sets in it).

Each user can set an online status consisting of an integer and an arbitratry (serializable) object. The object can be null, Hashtable, object[] or anything else Photon can serialize.

The status is published automatically to friends (anyone who set your user ID with AddFriends).

Actual friend relations are not persistent and have to be stored outside of Photon.

Parameters

friends	Array of friend userIds.

Returns

> If the operation could be sent.

8.9.3.9 void Photon.Chat.ChatClient.SendAcksOnly ( )

Obsolete: Better use UseBackgroundWorkerForSending and Service().

8.9.3.10 bool Photon.Chat.ChatClient.SendPrivateMessage (string *target,* object *message,* bool *forwardAsWebhook =* `false`)

Sends a private message to a single target user. Calls OnPrivateMessage on the receiving client.

Parameters

target	Username to send this message to.
message	The message you want to send. Can be a simple string or anything serializable.
forwardAs← Webhook	Optionally, private messages can be forwarded as webhooks. Configure webhooks for your Chat app to use this.

Returns

True if this clients can send the message to the server.

8.9.3.11 bool Photon.Chat.ChatClient.SendPrivateMessage (string *target,* object *message,* bool *encrypt,* bool *forwardAsWebhook*)

Sends a private message to a single target user. Calls OnPrivateMessage on the receiving client.

Parameters

target	Username to send this message to.
message	The message you want to send. Can be a simple string or anything serializable.
encrypt	Optionally, private messages can be encrypted. Encryption is not end-to-end as the server decrypts the message.
forwardAs← Webhook	Optionally, private messages can be forwarded as webhooks. Configure webhooks for your Chat app to use this.

Returns

True if this clients can send the message to the server.

8.9.3.12 void Photon.Chat.ChatClient.Service ( )

Must be called regularly to keep connection between client and server alive and to process incoming messages.

This method limits the effort it does automatically using the private variable msDeltaForServiceCalls. That value is lower for connect and multiplied by 4 when chat-server connection is ready.

8.9.3.13 bool Photon.Chat.ChatClient.SetOnlineStatus (int *status*)

Sets the user's status without changing your status-message.

The predefined status values can be found in class ChatUserStatus. State ChatUserStatus.Invisible will make you offline for everyone and send no message.

You can set custom values in the status integer. Aside from the pre-configured ones, all states will be considered visible and online. Else, no one would see the custom state.

This overload does not change the set message.

Parameters

status	Predefined states are in class ChatUserStatus. Other values can be used at will.

Returns

> True if the operation gets called on the server.

8.9.3.14 bool Photon.Chat.ChatClient.SetOnlineStatus (int *status,* object *message*)

Sets the user's status without changing your status-message.

The predefined status values can be found in class ChatUserStatus. State ChatUserStatus.Invisible will make you offline for everyone and send no message.

You can set custom values in the status integer. Aside from the pre-configured ones, all states will be considered visible and online. Else, no one would see the custom state.

The message object can be anything that Photon can serialize, including (but not limited to) Hashtable, object[] and string. This value is defined by your own conventions.

Parameters

status	Predefined states are in class ChatUserStatus. Other values can be used at will.
message	Also sets a status-message which your friends can get.

Returns

> True if the operation gets called on the server.

8.9.3.15 void Photon.Chat.ChatClient.StopThread ( )

Locally shuts down the connection to the Chat Server. This resets states locally but the server will have to timeout this peer.

8.9.3.16 bool Photon.Chat.ChatClient.Subscribe (string[ ] *channels*)

Sends operation to subscribe to a list of channels by name.

Parameters

channels	List of channels to subscribe to. Avoid null or empty values.

Returns

> If the operation could be sent at all (Example: Fails if not connected to Chat Server).

8.9.3.17 bool Photon.Chat.ChatClient.Subscribe (string[ ] *channels,* int[ ] *lastMsgIds*)

Sends operation to subscribe to a list of channels by name and possibly retrieve messages we did not receive while unsubscribed.

Parameters

channels	List of channels to subscribe to. Avoid null or empty values.
lastMsgIds	ID of last message received per channel. Useful when re subscribing to receive only messages we missed.

Returns

If the operation could be sent at all (Example: Fails if not connected to Chat Server).

8.9.3.18   bool Photon.Chat.ChatClient.Subscribe (string[ ] *channels,* int *messagesFromHistory*)

Sends operation to subscribe client to channels, optionally fetching a number of messages from the cache.

Subscribes channels will forward new messages to this user. Use PublishMessage to do so. The messages cache is limited but can be useful to get into ongoing conversations, if that's needed.

Parameters

channels	List of channels to subscribe to. Avoid null or empty values.
messages← FromHistory	0: no history. 1 and higher: number of messages in history. -1: all available history.

Returns

If the operation could be sent at all (Example: Fails if not connected to Chat Server).

8.9.3.19   bool Photon.Chat.ChatClient.TryGetChannel (string *channelName,* bool *isPrivate,* out ChatChannel *channel*)

Simplified access to either private or public channels by name.

Parameters

channelName	Name of the channel to get. For private channels, the channel-name is composed of both user's names.
isPrivate	Define if you expect a private or public channel.
channel	Out parameter gives you the found channel, if any.

Returns

True if the channel was found.

8.9.3.20   bool Photon.Chat.ChatClient.TryGetChannel (string *channelName,* out ChatChannel *channel*)

Simplified access to all channels by name. Checks public channels first, then private ones.

Parameters

channelName	Name of the channel to get.
channel	Out parameter gives you the found channel, if any.

Returns

True if the channel was found.

8.9.3.21 bool Photon.Chat.ChatClient.Unsubscribe (string[ ] *channels*)

Unsubscribes from a list of channels, which stops getting messages from those.

The client will remove these channels from the PublicChannels dictionary once the server sent a response to this request.

The request will be sent to the server and IChatClientListener.OnUnsubscribed gets called when the server actually removed the channel subscriptions.

Unsubscribe will fail if you include null or empty channel names.

Parameters

channels	Names of channels to unsubscribe.

Returns

False, if not connected to a chat server.

8.9.4 Member Data Documentation

8.9.4.1 ChatPeer Photon.Chat.ChatClient.chatPeer = null

The Chat Peer used by this client.

8.9.4.2 int Photon.Chat.ChatClient.MessageLimit

If greater than 0, new channels will limit the number of messages they cache locally.

This can be useful to limit the amount of memory used by chats. You can set a MessageLimit per channel but this value gets applied to new ones.

Note: Changing this value, does not affect ChatChannels that are already in use!

8.9.4.3 readonly Dictionary<string, ChatChannel> Photon.Chat.ChatClient.PrivateChannels

Private channels in which this client has exchanged messages.

8.9.4.4 readonly Dictionary<string, ChatChannel> Photon.Chat.ChatClient.PublicChannels

Public channels this client is subscribed to.

8.9.5 Property Documentation

8.9.5.1 string Photon.Chat.ChatClient.AppId `[get]`

The AppID as assigned from the Photon Cloud.

8.9.5.2 string Photon.Chat.ChatClient.AppVersion `[get]`

The version of your client. A new version also creates a new "virtual app" to separate players from older client versions.

8.9.5.3 **AuthenticationValues Photon.Chat.ChatClient.AuthValues** `[get]`,`[set]`

Settable only before you connect!

8.9.5.4 **bool Photon.Chat.ChatClient.CanChat** `[get]`

Checks if this client is ready to send messages.

8.9.5.5 **string Photon.Chat.ChatClient.ChatRegion** `[get]`,`[set]`

Settable only before you connect! Defaults to "EU".

8.9.5.6 **DebugLevel Photon.Chat.ChatClient.DebugOut** `[get]`,`[set]`

Sets the level (and amount) of debug output provided by the library.

This affects the callbacks to IChatClientListener.DebugReturn. Default Level: Error.

8.9.5.7 **ChatDisconnectCause Photon.Chat.ChatClient.DisconnectedCause** `[get]`

Disconnection cause. Check this inside IChatClientListener.OnDisconnected.

8.9.5.8 **string Photon.Chat.ChatClient.FrontendAddress** `[get]`

The address of the actual chat server assigned from NameServer. Public for read only.

8.9.5.9 **string Photon.Chat.ChatClient.NameServerAddress** `[get]`

The address of last connected Name Server.

8.9.5.10 **Dictionary<ConnectionProtocol, Type> Photon.Chat.ChatClient.SocketImplementationConfig** `[get]`

Defines which IPhotonSocket class to use per ConnectionProtocol.

Several platforms have special Socket implementations and slightly different APIs. To accomodate this, switching the socket implementation for a network protocol was made available. By default, UDP and TCP have socket implementations assigned.

You only need to set the SocketImplementationConfig once, after creating a PhotonPeer and before connecting. If you switch the TransportProtocol, the correct implementation is being used.

8.9.5.11 **ChatState Photon.Chat.ChatClient.State** `[get]`

Current state of the ChatClient. Also use CanChat.

8.9.5.12 **ConnectionProtocol Photon.Chat.ChatClient.TransportProtocol** `[get]`,`[set]`

Exposes the TransportProtocol of the used PhotonPeer. Settable while not connected.

8.9.5.13 bool Photon.Chat.ChatClient.UseBackgroundWorkerForSending `[get],[set]`

Defines if a background thread will call SendOutgoingCommands, while your code calls Service to dispatch received messages.

The benefit of using a background thread to call SendOutgoingCommands is this:

Even if your game logic is being paused, the background thread will keep the connection to the server up. On a lower level, acknowledgements and pings will prevent a server-side timeout while (e.g.) Unity loads assets.

Your game logic still has to call Service regularly, or else incoming messages are not dispatched. As this typicalls triggers UI updates, it's easier to call Service from the main/UI thread.

8.9.5.14 string Photon.Chat.ChatClient.UserId `[get]`

The unique ID of a user/person, stored in AuthValues.UserId. Set it before you connect.

This value wraps AuthValues.UserId. It's not a nickname and we assume users with the same userID are the same person.

8.10 Photon.Chat.ChatEventCode Class Reference

Wraps up internally used constants in Photon Chat events. You don't have to use them directly usually.

Public Attributes

- const byte ChatMessages = 0

  (0) Event code for messages published in public channels.
- const byte Users = 1

  (1) Not Used.
- const byte PrivateMessage = 2

  (2) Event code for messages published in private channels
- const byte FriendsList = 3

  (3) Not Used.
- const byte StatusUpdate = 4

  (4) Event code for status updates.
- const byte Subscribe = 5

  (5) Event code for subscription acks.
- const byte Unsubscribe = 6

  (6) Event code for unsubscribe acks.

8.10.1 Detailed Description

Wraps up internally used constants in Photon Chat events. You don't have to use them directly usually.

8.10.2 Member Data Documentation

8.10.2.1 const byte Photon.Chat.ChatEventCode.ChatMessages = 0

(0) Event code for messages published in public channels.

8.10.2.2   const byte Photon.Chat.ChatEventCode.FriendsList = 3

(3) Not Used.

8.10.2.3   const byte Photon.Chat.ChatEventCode.PrivateMessage = 2

(2) Event code for messages published in private channels

8.10.2.4   const byte Photon.Chat.ChatEventCode.StatusUpdate = 4

(4) Event code for status updates.

8.10.2.5   const byte Photon.Chat.ChatEventCode.Subscribe = 5

(5) Event code for subscription acks.

8.10.2.6   const byte Photon.Chat.ChatEventCode.Unsubscribe = 6

(6) Event code for unsubscribe acks.

8.10.2.7   const byte Photon.Chat.ChatEventCode.Users = 1

(1) Not Used.

8.11   Photon.Chat.ChatOperationCode Class Reference

Wraps up codes for operations used internally in Photon Chat. You don't have to use them directly usually.

Public Attributes

- const byte Authenticate = 230

    (230) Operation Authenticate.
- const byte Subscribe = 0

    (0) Operation to subscribe to chat channels.
- const byte Unsubscribe = 1

    (1) Operation to unsubscribe from chat channels.
- const byte Publish = 2

    (2) Operation to publish a message in a chat channel.
- const byte SendPrivate = 3

    (3) Operation to send a private message to some other user.
- const byte ChannelHistory = 4

    (4) Not used yet.
- const byte UpdateStatus = 5

    (5) Set your (client's) status.
- const byte AddFriends = 6

    (6) Add friends the list of friends that should update you of their status.
- const byte RemoveFriends = 7

    (7) Remove friends from list of friends that should update you of their status.

8.11.1 Detailed Description

Wraps up codes for operations used internally in Photon Chat. You don't have to use them directly usually.

8.11.2 Member Data Documentation

8.11.2.1 const byte Photon.Chat.ChatOperationCode.AddFriends = 6

(6) Add friends the list of friends that should update you of their status.

8.11.2.2 const byte Photon.Chat.ChatOperationCode.Authenticate = 230

(230) Operation Authenticate.

8.11.2.3 const byte Photon.Chat.ChatOperationCode.ChannelHistory = 4

(4) Not used yet.

8.11.2.4 const byte Photon.Chat.ChatOperationCode.Publish = 2

(2) Operation to publish a message in a chat channel.

8.11.2.5 const byte Photon.Chat.ChatOperationCode.RemoveFriends = 7

(7) Remove friends from list of friends that should update you of their status.

8.11.2.6 const byte Photon.Chat.ChatOperationCode.SendPrivate = 3

(3) Operation to send a private message to some other user.

8.11.2.7 const byte Photon.Chat.ChatOperationCode.Subscribe = 0

(0) Operation to subscribe to chat channels.

8.11.2.8 const byte Photon.Chat.ChatOperationCode.Unsubscribe = 1

(1) Operation to unsubscribe from chat channels.

8.11.2.9 const byte Photon.Chat.ChatOperationCode.UpdateStatus = 5

(5) Set your (client's) status.

8.12 Photon.Chat.ChatParameterCode Class Reference

Wraps up codes for parameters (in operations and events) used internally in Photon Chat. You don't have to use them directly usually.

Public Attributes

- const byte Channels = 0

    (0) Array of chat channels.
- const byte Channel = 1

    (1) Name of a single chat channel.
- const byte Messages = 2

    (2) Array of chat messages.
- const byte Message = 3

    (3) A single chat message.
- const byte Senders = 4

    (4) Array of names of the users who sent the array of chat mesages.
- const byte Sender = 5

    (5) Name of a the user who sent a chat message.
- const byte ChannelUserCount = 6

    (6) Not used.
- const byte UserId = 225

    (225) Name of user to send a (private) message to.
- const byte MsgId = 8

    (8) Id of a message.
- const byte MsgIds = 9

    (9) Not used.
- const byte Secret = 221

    (221) Secret token to identify an authorized user.
- const byte SubscribeResults = 15

    (15) Subscribe operation result parameter. A bool[] with result per channel.
- const byte Status = 10

    (10) Status
- const byte Friends = 11

    (11) Friends
- const byte SkipMessage = 12

    (12) SkipMessage is used in SetOnlineStatus and if true, the message is not being broadcast.
- const byte HistoryLength = 14

    (14) Number of message to fetch from history. 0: no history. 1 and higher: number of messages in history. -1: all history.
- const byte WebFlags = 21

    (21) WebFlags object for changing behaviour of webhooks from client.

8.12.1 Detailed Description

Wraps up codes for parameters (in operations and events) used internally in Photon Chat. You don't have to use them directly usually.

8.12.2 Member Data Documentation

8.12.2.1 const byte Photon.Chat.ChatParameterCode.Channel = 1

(1) Name of a single chat channel.

8.12.2.2 const byte Photon.Chat.ChatParameterCode.Channels = 0

(0) Array of chat channels.

8.12.2.3 const byte Photon.Chat.ChatParameterCode.ChannelUserCount = 6

(6) Not used.

8.12.2.4 const byte Photon.Chat.ChatParameterCode.Friends = 11

(11) Friends

8.12.2.5 const byte Photon.Chat.ChatParameterCode.HistoryLength = 14

(14) Number of message to fetch from history. 0: no history. 1 and higher: number of messages in history. -1: all history.

8.12.2.6 const byte Photon.Chat.ChatParameterCode.Message = 3

(3) A single chat message.

8.12.2.7 const byte Photon.Chat.ChatParameterCode.Messages = 2

(2) Array of chat messages.

8.12.2.8 const byte Photon.Chat.ChatParameterCode.MsgId = 8

(8) Id of a message.

8.12.2.9 const byte Photon.Chat.ChatParameterCode.MsgIds = 9

(9) Not used.

8.12.2.10 const byte Photon.Chat.ChatParameterCode.Secret = 221

(221) Secret token to identify an authorized user.

The code is used in LoadBalancing and copied over here.

8.12.2.11 const byte Photon.Chat.ChatParameterCode.Sender = 5

(5) Name of a the user who sent a chat message.

8.12.2.12 const byte Photon.Chat.ChatParameterCode.Senders = 4

(4) Array of names of the users who sent the array of chat mesages.

8.12.2.13 const byte Photon.Chat.ChatParameterCode.SkipMessage = 12

(12) SkipMessage is used in SetOnlineStatus and if true, the message is not being broadcast.

8.12.2.14   const byte Photon.Chat.ChatParameterCode.Status = 10

(10) Status

8.12.2.15   const byte Photon.Chat.ChatParameterCode.SubscribeResults = 15

(15) Subscribe operation result parameter. A bool[] with result per channel.

8.12.2.16   const byte Photon.Chat.ChatParameterCode.UserId = 225

(225) Name of user to send a (private) message to.

The code is used in LoadBalancing and copied over here.

8.12.2.17   const byte Photon.Chat.ChatParameterCode.WebFlags = 21

(21) WebFlags object for changing behaviour of webhooks from client.

8.13   Photon.Chat.ChatPeer Class Reference

Provides basic operations of the Photon Chat server. This internal class is used by public ChatClient.

Inherits PhotonPeer.

Public Member Functions

- ChatPeer (IPhotonPeerListener listener, ConnectionProtocol protocol)

  Chat Peer constructor.
- bool Connect ()

  Connects to NameServer.
- bool AuthenticateOnNameServer (string appId, string appVersion, string region, AuthenticationValues auth←
  Values)

  Authenticates on NameServer.

Public Attributes

- const string NameServerHost = "ns.exitgames.com"

  Name Server Host Name for Photon Cloud. Without port and without any prefix.
- const string NameServerHttp = "http://ns.exitgamescloud.com:80/photon/n"

  Name Server for HTTP connections to the Photon Cloud. Includes prefix and port.

Properties

- string NameServerAddress `[get]`

  *Name Server Address for Photon Cloud (based on current protocol). You can use the default values and usually won't
  have to set this value.*

8.13.1   Detailed Description

Provides basic operations of the Photon Chat server. This internal class is used by public ChatClient.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Photon.Chat.ChatPeer.ChatPeer (IPhotonPeerListener *listener,* ConnectionProtocol *protocol*)

Chat Peer constructor.

Parameters

listener	Chat listener implementation.
protocol	Protocol to be used by the peer.

8.13.3 Member Function Documentation

8.13.3.1 bool Photon.Chat.ChatPeer.AuthenticateOnNameServer (string *appId,* string *appVersion,* string *region,* AuthenticationValues *authValues*)

Authenticates on NameServer.

Returns

If the authentication operation request could be sent.

8.13.3.2 bool Photon.Chat.ChatPeer.Connect ( )

Connects to NameServer.

Returns

If the connection attempt could be sent.

8.13.4 Member Data Documentation

8.13.4.1 const string Photon.Chat.ChatPeer.NameServerHost = "ns.exitgames.com"

Name Server Host Name for Photon Cloud. Without port and without any prefix.

8.13.4.2 const string Photon.Chat.ChatPeer.NameServerHttp = "http://ns.exitgamescloud.com:80/photon/n"

Name Server for HTTP connections to the Photon Cloud. Includes prefix and port.

8.13.5 Property Documentation

8.13.5.1 string Photon.Chat.ChatPeer.NameServerAddress `[get]`

Name Server Address for Photon Cloud (based on current protocol). You can use the default values and usually won't have to set this value.

8.14 Photon.Chat.ChatUserStatus Class Reference

Contains commonly used status values for SetOnlineStatus. You can define your own.

Public Attributes

- const int Offline = 0

    (0) Offline.
- const int Invisible = 1

    (1) Be invisible to everyone. Sends no message.
- const int Online = 2

    (2) Online and available.
- const int Away = 3

    (3) Online but not available.
- const int DND = 4

    (4) Do not disturb.
- const int LFG = 5

    (5) Looking For Game/Group. Could be used when you want to be invited or do matchmaking.
- const int Playing = 6

    (6) Could be used when in a room, playing.

8.14.1 Detailed Description

Contains commonly used status values for SetOnlineStatus. You can define your own.

While "online" (value 2 and up), the status message will be sent to anyone who has you on his friend list.

Define custom online status values as you like with these rules: 0: Means "offline". It will be used when you are not connected. In this status, there is no status message. 1: Means "invisible" and is sent to friends as "offline". They see status 0, no message but you can chat. 2: And any higher value will be treated as "online". Status can be set.

8.14.2 Member Data Documentation

8.14.2.1 const int Photon.Chat.ChatUserStatus.Away = 3

(3) Online but not available.

8.14.2.2 const int Photon.Chat.ChatUserStatus.DND = 4

(4) Do not disturb.

8.14.2.3 const int Photon.Chat.ChatUserStatus.Invisible = 1

(1) Be invisible to everyone. Sends no message.

8.14.2.4 const int Photon.Chat.ChatUserStatus.LFG = 5

(5) Looking For Game/Group. Could be used when you want to be invited or do matchmaking.

8.14.2.5 const int Photon.Chat.ChatUserStatus.Offline = 0

(0) Offline.

8.14.2.6 const int Photon.Chat.ChatUserStatus.Online = 2

(2) Online and available.

8.14.2.7   const int Photon.Chat.ChatUserStatus.Playing = 6

(6) Could be used when in a room, playing.

8.15   Photon.Pun.UtilityScripts.ConnectAndJoinRandom Class Reference

Simple component to call ConnectUsingSettings and to get into a PUN room easily.

Inherits Photon.Pun.MonoBehaviourPunCallbacks.

Public Member Functions

- void **Start** ()
- void **ConnectNow** ()
- override void OnConnectedToMaster ()

  Called when the client is connected to the Master Server and ready for matchmaking and other tasks.
- override void OnJoinedLobby ()

  Called on entering a lobby on the Master Server. The actual room-list updates will call OnRoomListUpdate.
- override void OnJoinRandomFailed (short returnCode, string message)

  Called when a previous OpJoinRandom call failed on the server.
- override void OnDisconnected (DisconnectCause cause)

  Called after disconnecting from the Photon server. It could be a failure or intentional
- override void OnJoinedRoom ()

  Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

Public Attributes

- bool AutoConnect = true

  Connect automatically? If false you can set this to true later on or call ConnectUsingSettings in your own scripts.
- byte Version = 1

  Used as PhotonNetwork.GameVersion.

Additional Inherited Members

8.15.1   Detailed Description

Simple component to call ConnectUsingSettings and to get into a PUN room easily.

A custom inspector provides a button to connect in PlayMode, should AutoConnect be false.

8.15.2   Member Function Documentation

8.15.2.1   override void Photon.Pun.UtilityScripts.ConnectAndJoinRandom.OnConnectedToMaster ( ) `[virtual]`

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

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

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.15.2.2 override void Photon.Pun.UtilityScripts.ConnectAndJoinRandom.OnDisconnected (DisconnectCause *cause*) `[virtual]`

Called after disconnecting from the Photon server. It could be a failure or intentional

The reason for this disconnect is provided as DisconnectCause.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.15.2.3 override void Photon.Pun.UtilityScripts.ConnectAndJoinRandom.OnJoinedLobby ( ) `[virtual]`

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

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify in the public cloud). The room list gets available via OnRoomListUpdate.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.15.2.4 override void Photon.Pun.UtilityScripts.ConnectAndJoinRandom.OnJoinedRoom ( ) `[virtual]`

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↩ CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom Property).

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.15.2.5 override void Photon.Pun.UtilityScripts.ConnectAndJoinRandom.OnJoinRandomFailed (short *returnCode,* string *message*) `[virtual]`

Called when a previous OpJoinRandom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

When using multiple lobbies (via OpJoinLobby or a TypedLobby parameter), another lobby might have more/fitting rooms.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.15.3 Member Data Documentation

8.15.3.1 bool Photon.Pun.UtilityScripts.ConnectAndJoinRandom.AutoConnect = true

Connect automatically? If false you can set this to true later on or call ConnectUsingSettings in your own scripts.

8.15.3.2 byte Photon.Pun.UtilityScripts.ConnectAndJoinRandom.Version = 1

Used as PhotonNetwork.GameVersion.

8.16 Photon.Realtime.ConnectionHandler Class Reference

Inherited by Photon.Pun.PhotonHandler.

Public Member Functions

- void **StartFallbackSendAckThread** ()
- void **StopFallbackSendAckThread** ()
- bool RealtimeFallbackThread ()

  A thread which runs independent from the Update() calls. Keeps connections online while loading or in background. See PhotonNetwork.BackgroundTimeout.

Public Attributes

- int KeepAliveInBackground = 5000

  Defines for how long the Fallback Thread should keep the connection, before it may time out as usual.

Properties

- LoadBalancingClient Client `[get, set]`

  Photon client to log information and statistics from.

- int CountSendAcksOnly `[get]`

  Counts how often the Fallback Thread called SendAcksOnly, which is purely of interest to monitor if the game logic called SendOutgoingCommands as intended.

- bool **FallbackThreadRunning** `[get]`

8.16.1 Member Function Documentation

8.16.1.1 bool Photon.Realtime.ConnectionHandler.RealtimeFallbackThread ( )

A thread which runs independent from the Update() calls. Keeps connections online while loading or in background. See PhotonNetwork.BackgroundTimeout.

8.16.2 Member Data Documentation

8.16.2.1 int Photon.Realtime.ConnectionHandler.KeepAliveInBackground = 5000

Defines for how long the Fallback Thread should keep the connection, before it may time out as usual.

We want to the Client to keep it's connection when an app is in the background (and doesn't call Update / Service Clients should not keep their connection indefinitely in the background, so after some milliseconds, the Fallback Thread should stop keeping it up.

8.16.3 Property Documentation

8.16.3.1 LoadBalancingClient Photon.Realtime.ConnectionHandler.Client `[get],[set]`

Photon client to log information and statistics from.

8.16.3.2 int Photon.Realtime.ConnectionHandler.CountSendAcksOnly `[get]`

Counts how often the Fallback Thread called SendAcksOnly, which is purely of interest to monitor if the game logic called SendOutgoingCommands as intended.

8.17 Photon.Pun.UtilityScripts.CountdownTimer Class Reference

This is a basic CountdownTimer. In order to start the timer, the MasterClient can add a certain entry to the Custom Room Properties, which contains the property's name 'StartTime' and the actual start time describing the moment, the timer has been started. To have a synchronized timer, the best practice is to use PhotonNetwork.Time. In order to subscribe to the CountdownTimerHasExpired event you can call CountdownTimer.OnCountdownTimer↩ HasExpired += OnCountdownTimerIsExpired; from Unity's OnEnable function for example. For unsubscribing simply call CountdownTimer.OnCountdownTimerHasExpired -= OnCountdownTimerIsExpired;. You can do this from Unity's OnDisable function for example.

Inherits Photon.Pun.MonoBehaviourPunCallbacks.

Public Member Functions

- delegate void CountdownTimerHasExpired ()

    OnCountdownTimerHasExpired delegate.

- void **Start** ()
- void **Update** ()
- override void OnRoomPropertiesUpdate (Hashtable propertiesThatChanged)

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

Public Attributes

- const string **CountdownStartTime** = "StartTime"
- Text **Text**
- float **Countdown** = 5.0f

Events

- static CountdownTimerHasExpired OnCountdownTimerHasExpired

    Called when the timer has expired.

Additional Inherited Members

8.17.1 Detailed Description

This is a basic CountdownTimer. In order to start the timer, the MasterClient can add a certain entry to the Custom Room Properties, which contains the property's name 'StartTime' and the actual start time describing the moment, the timer has been started. To have a synchronized timer, the best practice is to use PhotonNetwork.Time. In order to subscribe to the CountdownTimerHasExpired event you can call CountdownTimer.OnCountdownTimer↩ HasExpired += OnCountdownTimerIsExpired; from Unity's OnEnable function for example. For unsubscribing simply call CountdownTimer.OnCountdownTimerHasExpired -= OnCountdownTimerIsExpired;. You can do this from Unity's OnDisable function for example.

8.17.2 Member Function Documentation

8.17.2.1 delegate void Photon.Pun.UtilityScripts.CountdownTimer.CountdownTimerHasExpired ( )

OnCountdownTimerHasExpired delegate.

8.17.2.2 override void Photon.Pun.UtilityScripts.CountdownTimer.OnRoomPropertiesUpdate (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	

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.17.3 Event Documentation

8.17.3.1 CountdownTimerHasExpired Photon.Pun.UtilityScripts.CountdownTimer.OnCountdownTimerHasExpired `[static]`

Called when the timer has expired.

8.18 Photon.Pun.UtilityScripts.CullArea Class Reference

Represents the cull area used for network culling.

Inherits MonoBehaviour.

Public Member Functions

- void OnDrawGizmos ()

    Creates the cell hierarchy in editor and draws the cell view.
- List< byte > GetActiveCells (Vector3 position)

    Gets a list of all cell IDs the player is currently inside or nearby.

Public Attributes

- const int **MAX_NUMBER_OF_SUBDIVISIONS** = 3
- readonly byte FIRST_GROUP_ID = 1

    This represents the first ID which is assigned to the first created cell. If you already have some interest groups blocking this first ID, fell free to change it. However increasing the first group ID decreases the maximum amount of allowed cells. Allowed values are in range from 1 to 250.
- readonly int[ ] SUBDIVISION_FIRST_LEVEL_ORDER = new int[4] { 0, 1, 1, 1 }

    This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:
- readonly int[ ] SUBDIVISION_SECOND_LEVEL_ORDER = new int[8] { 0, 2, 1, 2, 0, 2, 1, 2 }

    This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:
- readonly int[ ] SUBDIVISION_THIRD_LEVEL_ORDER = new int[12] { 0, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3 }

> *This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:*

- Vector2 **Center**
- Vector2 **Size** = new Vector2(25.0f, 25.0f)
- Vector2[ ] **Subdivisions** = new Vector2[MAX_NUMBER_OF_SUBDIVISIONS]
- int **NumberOfSubdivisions**
- bool **YIsUpAxis** = false
- bool **RecreateCellHierarchy** = false

Properties

- int **CellCount** [get]
- CellTree **CellTree** [get]
- Dictionary< int, GameObject > **Map** [get]

8.18.1 Detailed Description

Represents the cull area used for network culling.

8.18.2 Member Function Documentation

8.18.2.1 List<byte> Photon.Pun.UtilityScripts.CullArea.GetActiveCells (Vector3 *position*)

Gets a list of all cell IDs the player is currently inside or nearby.

Parameters

position	The current position of the player.

Returns

A list containing all cell IDs the player is currently inside or nearby.

8.18.2.2 void Photon.Pun.UtilityScripts.CullArea.OnDrawGizmos ( )

Creates the cell hierarchy in editor and draws the cell view.

8.18.3 Member Data Documentation

8.18.3.1 readonly byte Photon.Pun.UtilityScripts.CullArea.FIRST_GROUP_ID = 1

This represents the first ID which is assigned to the first created cell. If you already have some interest groups blocking this first ID, fell free to change it. However increasing the first group ID decreases the maximum amount of allowed cells. Allowed values are in range from 1 to 250.

8.18.3.2 readonly int [ ] Photon.Pun.UtilityScripts.CullArea.SUBDIVISION_FIRST_LEVEL_ORDER = new int[4] { 0, 1, 1, 1 }

This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:

- 0: message is sent to all players

- 1: message is sent to players who are interested in the matching cell of the first subdivision If there is only one subdivision we are sending one update to all players before sending three consequent updates only to players who are in the same cell or interested in updates of the current cell.

8.18.3.3   readonly int [ ] Photon.Pun.UtilityScripts.CullArea.SUBDIVISION_SECOND_LEVEL_ORDER = new int[8] { 0, 2, 1, 2, 0, 2, 1, 2 }

This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:

- 0: message is sent to all players

- 1: message is sent to players who are interested in the matching cell of the first subdivision

- 2: message is sent to players who are interested in the matching cell of the second subdivision If there are two subdivisions we are sending every second update only to players who are in the same cell or interested in updates of the current cell.

8.18.3.4   readonly int [ ] Photon.Pun.UtilityScripts.CullArea.SUBDIVISION_THIRD_LEVEL_ORDER = new int[12] { 0, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3 }

This represents the order in which updates are sent. The number represents the subdivision of the cell hierarchy:

- 0: message is sent to all players

- 1: message is sent to players who are interested in the matching cell of the first subdivision

- 2: message is sent to players who are interested in the matching cell of the second subdivision

- 3: message is sent to players who are interested in the matching cell of the third subdivision If there are two subdivisions we are sending every second update only to players who are in the same cell or interested in updates of the current cell.

8.19   Photon.Pun.UtilityScripts.CullingHandler Class Reference

Handles the network culling.

Inherits MonoBehaviour, and Photon.Pun.IPunObservable.

Public Member Functions

- void OnPhotonSerializeView (PhotonStream stream, PhotonMessageInfo info)

    This time OnPhotonSerializeView is not used to send or receive any kind of data. It is used to change the currently active group of the PhotonView component, making it work together with PUN more directly. Keep in mind that this function is only executed, when there is at least one more player in the room.

8.19.1   Detailed Description

Handles the network culling.

8.19.2   Member Function Documentation

8.19.2.1   void Photon.Pun.UtilityScripts.CullingHandler.OnPhotonSerializeView (PhotonStream *stream,* PhotonMessageInfo *info*)

This time OnPhotonSerializeView is not used to send or receive any kind of data. It is used to change the currently active group of the PhotonView component, making it work together with PUN more directly. Keep in mind that this function is only executed, when there is at least one more player in the room.

Implements Photon.Pun.IPunObservable.

8.20 Photon.Realtime.EncryptionDataParameters Class Reference

Public Attributes

- const byte Mode = 0

  Key for encryption mode
- const byte Secret1 = 1

  Key for first secret
- const byte Secret2 = 2

  Key for second secret

8.20.1 Member Data Documentation

8.20.1.1 const byte Photon.Realtime.EncryptionDataParameters.Mode = 0

Key for encryption mode

8.20.1.2 const byte Photon.Realtime.EncryptionDataParameters.Secret1 = 1

Key for first secret

8.20.1.3 const byte Photon.Realtime.EncryptionDataParameters.Secret2 = 2

Key for second secret

8.21 Photon.Realtime.EnterRoomParams Class Reference

Public Attributes

- string **RoomName**
- RoomOptions **RoomOptions**
- TypedLobby **Lobby**
- Hashtable **PlayerProperties**
- bool **CreateIfNotExists**
- bool **RejoinOnly**
- string[ ] **ExpectedUsers**

8.22 Photon.Realtime.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.

- const int InvalidEncryptionParameters = 32741

  (32741) Server will react with this error if invalid encryption parameters provided by token

8.22.1 Detailed Description

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

8.22.2 Member Data Documentation

8.22.2.1 const int Photon.Realtime.ErrorCode.AuthenticationTicketExpired = 0x7FF1

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

8.22.2.2 const int Photon.Realtime.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.

8.22.2.3 const int Photon.Realtime.ErrorCode.ExternalHttpCallFailed = 32744

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

8.22.2.4 const int Photon.Realtime.ErrorCode.GameClosed = 0x7FFF - 3

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

8.22.2.5 const int Photon.Realtime.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.

8.22.2.6 const int Photon.Realtime.ErrorCode.GameFull = 0x7FFF - 2

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

8.22.2.7 const int Photon.Realtime.ErrorCode.GameIdAlreadyExists = 0x7FFF - 1

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

8.22.2.8 const int Photon.Realtime.ErrorCode.HttpLimitReached = 32745

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

8.22.2.9 const int Photon.Realtime.ErrorCode.InternalServerError = -1

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

8.22.2.10 const int Photon.Realtime.ErrorCode.InvalidAuthentication = 0x7FFF

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

8.22.2.11 const int Photon.Realtime.ErrorCode.InvalidEncryptionParameters = 32741

(32741) Server will react with this error if invalid encryption parameters provided by token

8.22.2.12 const int Photon.Realtime.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.

8.22.2.13 const int Photon.Realtime.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.

8.22.2.14 const int Photon.Realtime.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://dashboard.↩ photonengine.com`

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.

8.22.2.15 const int Photon.Realtime.ErrorCode.JoinFailedFoundActiveJoiner = 32746

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

8.22.2.16 const int Photon.Realtime.ErrorCode.JoinFailedFoundExcludedUserId = 32747

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

8.22.2.17   const int Photon.Realtime.ErrorCode.JoinFailedFoundInactiveJoiner = 32749

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

8.22.2.18   const int Photon.Realtime.ErrorCode.JoinFailedPeerAlreadyJoined = 32750

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

8.22.2.19   const int Photon.Realtime.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.

8.22.2.20   const int Photon.Realtime.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.

8.22.2.21   const int Photon.Realtime.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.

8.22.2.22   const int Photon.Realtime.ErrorCode.Ok = 0

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

8.22.2.23   const int Photon.Realtime.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 or ConnectedToMasterserver

8.22.2.24   const int Photon.Realtime.ErrorCode.PluginMismatch = 0x7FFF - 16

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

8.22.2.25   const int Photon.Realtime.ErrorCode.PluginReportedError = 0x7FFF - 15

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

8.22.2.26 const int Photon.Realtime.ErrorCode.ServerFull = 0x7FFF - 5

(32762) Not in use currently.

8.22.2.27 const int Photon.Realtime.ErrorCode.SlotError = 32742

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

8.22.2.28 const int Photon.Realtime.ErrorCode.UserBlocked = 0x7FFF - 6

(32761) Not in use currently.

8.23 Photon.Chat.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 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 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.

8.23.1 Detailed Description

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

8.23.2 Member Data Documentation

8.23.2.1 const int Photon.Chat.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.

8.23.2.2 const int Photon.Chat.ErrorCode.GameClosed = 0x7FFF - 3

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

8.23.2.3 const int Photon.Chat.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.

8.23.2.4 const int Photon.Chat.ErrorCode.GameFull = 0x7FFF - 2

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

8.23.2.5 const int Photon.Chat.ErrorCode.GameIdAlreadyExists = 0x7FFF - 1

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

8.23.2.6 const int Photon.Chat.ErrorCode.InternalServerError = -1

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

8.23.2.7 const int Photon.Chat.ErrorCode.InvalidAuthentication = 0x7FFF

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

8.23.2.8 const int Photon.Chat.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.

8.23.2.9 const int Photon.Chat.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://cloud.↩ 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.

8.23.2.10 const int Photon.Chat.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.

8.23.2.11 const int Photon.Chat.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.

8.23.2.12 const int Photon.Chat.ErrorCode.Ok = 0

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

8.23.2.13 const int Photon.Chat.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 or ConnectedToMaster

8.23.2.14 const int Photon.Chat.ErrorCode.ServerFull = 0x7FFF - 5

(32762) Not in use currently.

8.23.2.15 const int Photon.Chat.ErrorCode.UserBlocked = 0x7FFF - 6

(32761) Not in use currently.

8.24 Photon.Realtime.EventCode Class Reference

Class for constants. These values are for events defined by Photon Loadbalancing.

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.

- const byte AuthEvent = 223

    (223) Sent by Photon to update a token before it times out.

8.24.1 Detailed Description

Class for constants. These values are for events defined by Photon Loadbalancing.

They start at 255 and go DOWN. Your own in-game events can start at 0. Pun uses these constants internally.

8.24.2 Member Data Documentation

8.24.2.1 const byte Photon.Realtime.EventCode.AppStats = 226

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

8.24.2.2 const byte Photon.Realtime.EventCode.AuthEvent = 223

(223) Sent by Photon to update a token before it times out.

8.24.2.3   const byte Photon.Realtime.EventCode.AzureNodeInfo = 210

(210) Internally used in case of hosting by Azure

8.24.2.4   const byte Photon.Realtime.EventCode.CacheSliceChanged = 250

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

8.24.2.5   const byte Photon.Realtime.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-us/realtime/current/reference/webhooks::options

8.24.2.6   const byte Photon.Realtime.EventCode.GameList = 230

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

8.24.2.7   const byte Photon.Realtime.EventCode.GameListUpdate = 229

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

8.24.2.8   const byte Photon.Realtime.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).

8.24.2.9   const byte Photon.Realtime.EventCode.Leave = (byte)254

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

8.24.2.10   const byte Photon.Realtime.EventCode.LobbyStats = 224

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

8.24.2.11   const byte Photon.Realtime.EventCode.Match = 227

(227) Currently not used. Event for matchmaking

8.24.2.12   const byte Photon.Realtime.EventCode.PropertiesChanged = (byte)253

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

8.24.2.13   const byte Photon.Realtime.EventCode.QueueState = 228

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

8.24.2.14   const byte Photon.Realtime.EventCode.SetProperties = (byte)253

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

8.25   Photon.Pun.UtilityScripts.EventSystemSpawner Class Reference

Event system spawner.   Will add an EventSystem GameObject with an EventSystem component and a StandaloneInputModule component Use this in additive scene loading context where you would otherwise get a "Multiple eventsystem in scene... this is not supported" error from Unity

Inherits MonoBehaviour.

8.25.1   Detailed Description

Event system spawner.   Will add an EventSystem GameObject with an EventSystem component and a StandaloneInputModule component Use this in additive scene loading context where you would otherwise get a "Multiple eventsystem in scene... this is not supported" error from Unity

8.26   Photon.Realtime.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 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)

    Helper method for debugging of IDictionary content, inlcuding type-information. Using this is not performant.
- static string ToStringFull< T > (this List< T > data)

    Helper method for debugging of List<T> content. Using this is not performant.
- static string ToStringFull (this object[ ] data)

    Helper method for debugging of object[] content. Using this is not performant.
- 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.

8.26.1   Detailed Description

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

8.26.2 Member Function Documentation

8.26.2.1 static bool Photon.Realtime.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.

8.26.2.2 static void Photon.Realtime.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.

8.26.2.3 static void Photon.Realtime.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.

8.26.2.4 static void Photon.Realtime.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.

8.26.2.5 static Hashtable Photon.Realtime.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.

8.26.2.6   static string Photon.Realtime.Extensions.ToStringFull (this IDictionary *origin*) `[static]`

Helper method for debugging of IDictionary content, inlcuding type-information. Using this is not performant.

Should only be used for debugging as necessary.

Parameters

origin	Some Dictionary or Hashtable.

Returns

String of the content of the IDictionary.

8.26.2.7   static string Photon.Realtime.Extensions.ToStringFull (this object[ ] *data*) `[static]`

Helper method for debugging of object[] content. Using this is not performant.

Should only be used for debugging as necessary.

Parameters

data	Any object[].

Returns

A comma-separated string containing each value's ToString().

8.26.2.8   static string Photon.Realtime.Extensions.ToStringFull< T > (this List< T > *data*) `[static]`

Helper method for debugging of List<T> content. Using this is not performant.

Should only be used for debugging as necessary.

Parameters

data	Any List<T> where T implements .ToString().

Returns

A comma-separated string containing each value's ToString().

8.27   Photon.Realtime.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]`
- string **UserId** `[get, protected set]`
- bool **IsOnline** `[get, protected set]`
- string **Room** `[get, protected set]`
- bool **IsInRoom** `[get]`

8.27.1 Detailed Description

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

8.28 Photon.Realtime.GamePropertyKey Class Reference

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

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.
- const byte PlayerTtl = (byte)246

    (246) Player Time To Live. How long any player can be inactive (due to disconnect or leave) before the user gets removed from the playerlist (freeing a slot).
- const byte EmptyRoomTtl = (byte)245

    (245) Room Time To Live. How long a room stays available (and in server-memory), after the last player becomes inactive. After this time, the room gets persisted or destroyed.

8.28.1 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.

8.28.2 Member Data Documentation

8.28.2.1 const byte Photon.Realtime.GamePropertyKey.CleanupCacheOnLeave = 249

(249) Equivalent of Operation Join parameter CleanupCacheOnLeave.

8.28.2.2 const byte Photon.Realtime.GamePropertyKey.EmptyRoomTtl = (byte)245

(245) Room Time To Live. How long a room stays available (and in server-memory), after the last player becomes inactive. After this time, the room gets persisted or destroyed.

8.28.2.3 const byte Photon.Realtime.GamePropertyKey.ExpectedUsers = (byte)247

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

8.28.2.4 const byte Photon.Realtime.GamePropertyKey.IsOpen = 253

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

8.28.2.5 const byte Photon.Realtime.GamePropertyKey.IsVisible = 254

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

8.28.2.6 const byte Photon.Realtime.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.

8.28.2.7 const byte Photon.Realtime.GamePropertyKey.MaxPlayers = 255

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

8.28.2.8 const byte Photon.Realtime.GamePropertyKey.PlayerCount = 252

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

8.28.2.9 const byte Photon.Realtime.GamePropertyKey.PlayerTtl = (byte)246

(246) Player Time To Live. How long any player can be inactive (due to disconnect or leave) before the user gets removed from the playerlist (freeing a slot).

8.28.2.10 const byte Photon.Realtime.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.

8.28.2.11 const byte Photon.Realtime.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)

8.29   Photon.Pun.UtilityScripts.GraphicToggleIsOnTransition Class Reference

Use this on toggles texts to have some color transition on the text depending on the isOn State.

Inherits MonoBehaviour, IPointerEnterHandler, and IPointerExitHandler.

Public Member Functions

- • void **OnPointerEnter** (PointerEventData eventData)
- • void **OnPointerExit** (PointerEventData eventData)
- • void **OnEnable** ()
- • void **OnDisable** ()
- • void **OnValueChanged** (bool isOn)

Public Attributes

- • Toggle **toggle**
- • Color **NormalOnColor** = Color.white
- • Color **NormalOffColor** = Color.black
- • Color **HoverOnColor** = Color.black
- • Color **HoverOffColor** = Color.black

8.29.1   Detailed Description

Use this on toggles texts to have some color transition on the text depending on the isOn State.

8.30   Photon.Chat.IChatClientListener Interface Reference

Callback interface for Chat client side. Contains callback methods to notify your app about updates. Must be provided to new ChatClient in constructor

Public Member Functions

- • void DebugReturn (DebugLevel level, string message)

     All debug output of the library will be reported through this method. Print it or put it in a buffer to use it on-screen.
- • void OnDisconnected ()

     Disconnection happened.
- • void OnConnected ()

     Client is connected now.
- • void OnChatStateChange (ChatState state)

     The ChatClient's state changed. Usually, OnConnected and OnDisconnected are the callbacks to react to.
- • void OnGetMessages (string channelName, string[ ] senders, object[ ] messages)

     Notifies app that client got new messages from server Number of senders is equal to number of messages in 'messages'. Sender with number '0' corresponds to message with number '0', sender with number '1' corresponds to message with number '1' and so on
- • void OnPrivateMessage (string sender, object message, string channelName)

     Notifies client about private message
- • void OnSubscribed (string[ ] channels, bool[ ] results)

     Result of Subscribe operation. Returns subscription result for every requested channel name.
- • void OnUnsubscribed (string[ ] channels)

     Result of Unsubscribe operation. Returns for channel name if the channel is now unsubscribed.

- void OnStatusUpdate (string user, int status, bool gotMessage, object message)

    New status of another user (you get updates for users set in your friends list).

8.30.1 Detailed Description

Callback interface for Chat client side. Contains callback methods to notify your app about updates. Must be provided to new ChatClient in constructor

8.30.2 Member Function Documentation

8.30.2.1 void Photon.Chat.IChatClientListener.DebugReturn (DebugLevel *level,* string *message*)

All debug output of the library will be reported through this method. Print it or put it in a buffer to use it on-screen.

Parameters

level	DebugLevel (severity) of the message.
message	Debug text. Print to System.Console or screen.

8.30.2.2 void Photon.Chat.IChatClientListener.OnChatStateChange (ChatState *state*)

The ChatClient's state changed. Usually, OnConnected and OnDisconnected are the callbacks to react to.

Parameters

state	The new state.

8.30.2.3 void Photon.Chat.IChatClientListener.OnConnected ( )

Client is connected now.

Clients have to be connected before they can send their state, subscribe to channels and send any messages.

8.30.2.4 void Photon.Chat.IChatClientListener.OnDisconnected ( )

Disconnection happened.

8.30.2.5 void Photon.Chat.IChatClientListener.OnGetMessages (string *channelName,* string[ ] *senders,* object[ ] *messages*)

Notifies app that client got new messages from server Number of senders is equal to number of messages in 'messages'. Sender with number '0' corresponds to message with number '0', sender with number '1' corresponds to message with number '1' and so on

Parameters

channelName	channel from where messages came
senders	list of users who sent messages
messages	list of messages it self

8.30.2.6 void Photon.Chat.IChatClientListener.OnPrivateMessage (string *sender,* object *message,* string *channelName*)

Notifies client about private message

Parameters

sender	user who sent this message
message	message it self
channelName	channelName for private messages (messages you sent yourself get added to a channel per target username)

8.30.2.7   void Photon.Chat.IChatClientListener.OnStatusUpdate (string *user,* int *status,* bool *gotMessage,* object *message*)

New status of another user (you get updates for users set in your friends list).

Parameters

user	Name of the user.
status	New status of that user.
gotMessage	True if the status contains a message you should cache locally. False: This status update does not include a message (keep any you have).
message	Message that user set.

8.30.2.8   void Photon.Chat.IChatClientListener.OnSubscribed (string[ ] *channels,* bool[ ] *results*)

Result of Subscribe operation. Returns subscription result for every requested channel name.

If multiple channels sent in Subscribe operation, OnSubscribed may be called several times, each call with part of sent array or with single channel in "channels" parameter. Calls order and order of channels in "channels" parameter may differ from order of channels in "channels" parameter of Subscribe operation.

Parameters

channels	Array of channel names.
results	Per channel result if subscribed.

8.30.2.9   void Photon.Chat.IChatClientListener.OnUnsubscribed (string[ ] *channels*)

Result of Unsubscribe operation. Returns for channel name if the channel is now unsubscribed.

If multiple channels sent in Unsubscribe operation, OnUnsubscribed may be called several times, each call with part of sent array or with single channel in "channels" parameter. Calls order and order of channels in "channels" parameter may differ from order of channels in "channels" parameter of Unsubscribe operation.

Parameters

channels	Array of channel names that are no longer subscribed.

8.31   Photon.Realtime.IConnectionCallbacks Interface Reference

Collection of "organizational" callbacks for the Realtime Api to cover: Connection and Regions.

Inherited by Photon.Pun.MonoBehaviourPunCallbacks, Photon.Pun.UtilityScripts.OnJoinedInstantiate, Photon.↵
Realtime.ConnectionCallbacksContainer, and Photon.Realtime.SupportLogger.

Public Member Functions

- void OnConnected ()

  Called to signal that the "low level connection" got established but before the client can call operation on the server.

- void OnConnectedToMaster ()

    Called when the client is connected to the Master Server and ready for matchmaking and other tasks.
- void OnDisconnected (DisconnectCause cause)

    Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call
- void OnRegionListReceived (RegionHandler regionHandler)

    Called when the Name Server provided a list of regions for your title.
- void OnCustomAuthenticationResponse (Dictionary< string, object > data)

    Called when your Custom Authentication service responds with additional data.
- void OnCustomAuthenticationFailed (string debugMessage)

    Called when the custom authentication failed. Followed by disconnect!

8.31.1 Detailed Description

Collection of "organizational" callbacks for the Realtime Api to cover: Connection and Regions.

Classes that implement this interface must be registered to get callbacks for various situations.

To register for callbacks, PhotonNetwork.AddCallbackTarget(<Your Component implementing this interface>); To stop getting callbacks, PhotonNetwork.RemoveCallbackTarget(<Your Component implementing this interface>);

You can also simply override MonoBehaviourPunCallbacks which will provide you with Magic Callbacks (like Unity would call Start(), Update() on a MonoBehaviour)

8.31.2 Member Function Documentation

8.31.2.1 void Photon.Realtime.IConnectionCallbacks.OnConnected ( )

Called to signal that the "low level connection" got established but before the client can call operation on the server.

After the (low level transport) connection is established, the client will automatically send the Authentication operation, which needs to get a response before the client can call other operations.

Your logic should wait for either: OnRegionListReceived or OnConnectedToMaster.

This callback is useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnDisconnected(DisconnectCause cause) and check for the cause.

This is not called for transitions from the masterserver to game servers.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↩ UtilityScripts.OnJoinedInstantiate.

8.31.2.2 void Photon.Realtime.IConnectionCallbacks.OnConnectedToMaster ( )

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

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

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩ Scripts.OnJoinedInstantiate, and Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.31.2.3 void Photon.Realtime.IConnectionCallbacks.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.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↵UtilityScripts.OnJoinedInstantiate.

8.31.2.4   void Photon.Realtime.IConnectionCallbacks.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-us/realtime/current/reference/custom-authentication

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↵UtilityScripts.OnJoinedInstantiate.

8.31.2.5   void Photon.Realtime.IConnectionCallbacks.OnDisconnected (DisconnectCause *cause*)

Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call

The reason for this disconnect is provided as DisconnectCause.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↵Scripts.OnJoinedInstantiate, and Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.31.2.6   void Photon.Realtime.IConnectionCallbacks.OnRegionListReceived (RegionHandler *regionHandler*)

Called when the Name Server provided a list of regions for your title.

Check the RegionHandler class description, to make use of the provided values.

Parameters

regionHandler	The currently used RegionHandler.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↵UtilityScripts.OnJoinedInstantiate.

8.32   Photon.Realtime.IInRoomCallbacks Interface Reference

Collection of "in room" callbacks for the Realtime Api to cover: Players entering or leaving, property updates and Master Client switching.

Inherited by Photon.Pun.MonoBehaviourPunCallbacks, Photon.Pun.PhotonHandler, Photon.Realtime.InRoom↵CallbacksContainer, and Photon.Realtime.SupportLogger.

Public Member Functions

- void OnPlayerEnteredRoom (Player newPlayer)

  Called when a remote player entered the room. This Player is already added to the playerlist.
- void OnPlayerLeftRoom (Player otherPlayer)

  Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.
- void OnRoomPropertiesUpdate (Hashtable propertiesThatChanged)

  Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.↩ SetCustomProperties.
- void OnPlayerPropertiesUpdate (Player targetPlayer, Hashtable changedProps)

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

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

8.32.1 Detailed Description

Collection of "in room" callbacks for the Realtime Api to cover: Players entering or leaving, property updates and Master Client switching.

The callback to get events is in a separate interface: IOnEventCallback.

To register for callbacks, PhotonNetwork.AddCallbackTarget(<Your Component implementing this interface>); To stop getting callbacks, PhotonNetwork.RemoveCallbackTarget(<Your Component implementing this interface>);

You can also simply override MonoBehaviourPunCallbacks which will provide you with Magic Callbacks (like Unity would call Start(), Update() on a MonoBehaviour)

8.32.2 Member Function Documentation

8.32.2.1 void Photon.Realtime.IInRoomCallbacks.OnMasterClientSwitched (Player *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.Pun.MonoBehaviourPunCallbacks, and Photon.Realtime.SupportLogger.

8.32.2.2 void Photon.Realtime.IInRoomCallbacks.OnPlayerEnteredRoom (Player *newPlayer*)

Called when a remote player entered the room. This Player is already added to the playerlist.

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.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩ Scripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.32.2.3 void Photon.Realtime.IInRoomCallbacks.OnPlayerLeftRoom (Player *otherPlayer*)

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

If another player leaves the room or if the server detects a lost connection, this callback will be used to notify your game logic.

Depending on the room's setup, players may become inactive, which means they may return and retake their spot in the room. In such cases, the Player stays in the Room.Players dictionary.

If the player is not just inactive, it gets removed from the Room.Players dictionary, before the callback is called.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩
Scripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.32.2.4   void Photon.Realtime.IInRoomCallbacks.OnPlayerPropertiesUpdate (Player *targetPlayer,* Hashtable *changedProps*)

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

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

Parameters

targetPlayer	Contains Player that changed.
changedProps	Contains the properties that changed.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩
Scripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.32.2.5   void Photon.Realtime.IInRoomCallbacks.OnRoomPropertiesUpdate (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.Pun.MonoBehaviourPunCallbacks, Photon.Pun.UtilityScripts.PunTurnManager, Photon.↩
Realtime.SupportLogger, and Photon.Pun.UtilityScripts.CountdownTimer.

8.33   Photon.Realtime.ILobbyCallbacks Interface Reference

Collection of "organizational" callbacks for the Realtime Api to cover the Lobby.

Inherited by Photon.Pun.MonoBehaviourPunCallbacks, Photon.Pun.UtilityScripts.OnJoinedInstantiate, Photon.↩
Realtime.LobbyCallbacksContainer, and Photon.Realtime.SupportLogger.

Public Member Functions

- void OnJoinedLobby ()

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

- void OnLeftLobby ()

    Called after leaving a lobby.

- void OnRoomListUpdate (List< RoomInfo > roomList)

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

- void OnLobbyStatisticsUpdate (List< TypedLobbyInfo > lobbyStatistics)

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

8.33.1   Detailed Description

Collection of "organizational" callbacks for the Realtime Api to cover the Lobby.

Classes that implement this interface must be registered to get callbacks for various situations.

To register for callbacks, PhotonNetwork.AddCallbackTarget(<Your Component implementing this interface>); To stop getting callbacks, PhotonNetwork.RemoveCallbackTarget(<Your Component implementing this interface>);

You can also simply override MonoBehaviourPunCallbacks which will provide you with Magic Callbacks (like Unity would call Start(), Update() on a MonoBehaviour)

8.33.2 Member Function Documentation

8.33.2.1 void Photon.Realtime.ILobbyCallbacks.OnJoinedLobby ( )

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

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify in the public cloud). The room list gets available via OnRoomListUpdate.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility← Scripts.OnJoinedInstantiate, and Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.33.2.2 void Photon.Realtime.ILobbyCallbacks.OnLeftLobby ( )

Called after leaving a lobby.

When you leave a lobby, OpCreateRoom and OpJoinRandomRoom automatically refer to the default lobby.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.← UtilityScripts.OnJoinedInstantiate.

8.33.2.3 void Photon.Realtime.ILobbyCallbacks.OnLobbyStatisticsUpdate (List< **TypedLobbyInfo** > *lobbyStatistics*)

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.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.← UtilityScripts.OnJoinedInstantiate.

8.33.2.4 void Photon.Realtime.ILobbyCallbacks.OnRoomListUpdate (List< **RoomInfo** > *roomList*)

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

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.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.← UtilityScripts.OnJoinedInstantiate.

8.34 Photon.Realtime.IMatchmakingCallbacks Interface Reference

Collection of "organizational" callbacks for the Realtime Api to cover Matchmaking.

Inherited by Photon.Pun.MonoBehaviourPunCallbacks, Photon.Pun.PhotonHandler, Photon.Pun.UtilityScripts.← OnJoinedInstantiate, Photon.Realtime.MatchMakingCallbacksContainer, and Photon.Realtime.SupportLogger.

Public Member Functions

- void OnFriendListUpdate (List< FriendInfo > friendList)

Called when the server sent the response to a FindFriends request.

- void OnCreatedRoom ()

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

- void OnCreateRoomFailed (short returnCode, string message)

    Called when the server couldn't create a room (OpCreateRoom failed).

- void OnJoinedRoom ()

    Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

- void OnJoinRoomFailed (short returnCode, string message)

    Called when a previous OpJoinRoom call failed on the server.

- void OnJoinRandomFailed (short returnCode, string message)

    Called when a previous OpJoinRandom call failed on the server.

- void OnLeftRoom ()

    Called when the local user/client left a room, so the game's logic can clean up it's internal state.

8.34.1 Detailed Description

Collection of "organizational" callbacks for the Realtime Api to cover Matchmaking.

Classes that implement this interface must be registered to get callbacks for various situations.

To register for callbacks, PhotonNetwork.AddCallbackTarget(<Your Component implementing this interface>); To stop getting callbacks, PhotonNetwork.RemoveCallbackTarget(<Your Component implementing this interface>);

You can also simply override MonoBehaviourPunCallbacks which will provide you with Magic Callbacks (like Unity would call Start(), Update() on a MonoBehaviour)

8.34.2 Member Function Documentation

8.34.2.1 void Photon.Realtime.IMatchmakingCallbacks.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 OpCreateRoom).

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", implement OnMasterClientSwitched() and make each new MasterClient check the room's state.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↩ UtilityScripts.OnJoinedInstantiate.

8.34.2.2 void Photon.Realtime.IMatchmakingCallbacks.OnCreateRoomFailed (short *returnCode,* string *message*)

Called when the server couldn't create a room (OpCreateRoom failed).

Creating a room may fail for various reasons. Most often, the room already exists (roomname in use) or the Room↩ Options clash and it's impossible to create the room.

When creating a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↩
UtilityScripts.OnJoinedInstantiate.

8.34.2.3   void Photon.Realtime.IMatchmakingCallbacks.OnFriendListUpdate (List< FriendInfo > *friendList*)

Called when the server sent the response to a FindFriends request.

After calling OpFindFriends, the Master Server will cache the friend list and send updates to the friend list. The friends includes the name, userId, online state and the room (if any) for each requested user/friend.

Use the friendList to update your UI and store it, if the UI should highlight changes.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↩
UtilityScripts.OnJoinedInstantiate.

8.34.2.4   void Photon.Realtime.IMatchmakingCallbacks.OnJoinedRoom ( )

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↩
CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom Property).

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩
Scripts.PlayerNumbering,   Photon.Pun.UtilityScripts.ConnectAndJoinRandom,   Photon.Pun.UtilityScripts.Pun↩
Teams, and Photon.Pun.UtilityScripts.OnJoinedInstantiate.

8.34.2.5   void Photon.Realtime.IMatchmakingCallbacks.OnJoinRandomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRandom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

This operation is only ever sent to the Master Server. Once a room is found by the Master Server, the client will head off to the designated Game Server and use the operation Join on the Game Server.

When using multiple lobbies (via OpJoinLobby or a TypedLobby parameter), another lobby might have more/fitting rooms.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↩
Scripts.OnJoinedInstantiate, and Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.34.2.6   void Photon.Realtime.IMatchmakingCallbacks.OnJoinRoomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRoom call failed on the server.

Joining a room may fail for various reasons. Most often, the room is full or does not exist anymore (due to someone else being faster or closing the room).

When joining a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, and Photon.Pun.↵
UtilityScripts.OnJoinedInstantiate.

8.34.2.7 void Photon.Realtime.IMatchmakingCallbacks.OnLeftRoom ( )

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Implemented in Photon.Pun.MonoBehaviourPunCallbacks, Photon.Realtime.SupportLogger, Photon.Pun.Utility↵
Scripts.OnJoinedInstantiate, Photon.Pun.UtilityScripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.35 Photon.Realtime.IOnEventCallback Interface Reference

Event callback for the Realtime Api. Covers events from the server and those sent by clients via OpRaiseEvent.

Public Member Functions

- void OnEvent (EventData photonEvent)

  Called for any incoming events.

8.35.1 Detailed Description

Event callback for the Realtime Api. Covers events from the server and those sent by clients via OpRaiseEvent.

Classes that implement this interface must be registered to get callbacks for various situations.

To register for callbacks, register the instance via: LoadBalancingClient.EventReceived += instance. To stop getting callbacks, remove the instance via: -=.

8.35.2 Member Function Documentation

8.35.2.1 void Photon.Realtime.IOnEventCallback.OnEvent (EventData *photonEvent*)

Called for any incoming events.

To receive events, implement IOnEventCallback in any class and register it via AddCallbackTarget (either in Load↵
BalancingClient or PhotonNetwork).

With the EventData.Sender you can look up the Player who sent the event.

It is best practice to assign an eventCode for each different type of content and action, so the Code will be essential to read the incoming events.

8.36 Photon.Pun.IPunInstantiateMagicCallback Interface Reference

Public Member Functions

- void **OnPhotonInstantiate** (PhotonMessageInfo info)

8.37 Photon.Pun.IPunObservable Interface Reference

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

Inherited by Photon.Pun.PhotonAnimatorView, Photon.Pun.PhotonRigidbody2DView, Photon.Pun.Photon↩
RigidbodyView, Photon.Pun.PhotonTransformView, Photon.Pun.UtilityScripts.CullingHandler, and Photon.Pun.↩
UtilityScripts.SmoothSyncMovement.

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.*

8.37.1 Detailed Description

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

8.38 Photon.Pun.IPunOwnershipCallbacks Interface Reference

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

Public Member Functions

- void OnOwnershipRequest (PhotonView targetView, Player requestingPlayer)

    Called when another player requests ownership of a PhotonView from you (the current owner).
- void OnOwnershipTransfered (PhotonView targetView, Player previousOwner)

    Called when ownership of a PhotonView is transfered to another player.

8.38.1 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: IPun↩
Callbacks.OnConnected 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.

8.38.2 Member Function Documentation

8.38.2.1 void Photon.Pun.IPunOwnershipCallbacks.OnOwnershipRequest (PhotonView *targetView,* Player *requestingPlayer*)

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

The parameter viewAndPlayer contains:

PhotonView view = viewAndPlayer[0] as PhotonView;

Player requestingPlayer = viewAndPlayer[1] as Player;

Parameters

targetView	PhotonView for which ownership gets requested.
requestingPlayer	Player who requests ownership.

8.38.2.2 void Photon.Pun.IPunOwnershipCallbacks.OnOwnershipTransfered (PhotonView *targetView,* Player *previousOwner*)

Called when ownership of a PhotonView is transfered to another player.

The parameter viewAndPlayers contains:

PhotonView view = viewAndPlayers[0] as PhotonView;

Player newOwner = viewAndPlayers[1] as Player;

Player oldOwner = viewAndPlayers[2] as Player;

void OnOwnershipTransfered(object[ ] viewAndPlayers) {} //

Parameters

targetView	PhotonView for which ownership changed.
previousOwner	Player who was the previous owner (or null, if none).

8.39 Photon.Pun.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.

8.39.1 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.

8.39.2 Member Function Documentation

8.39.2.1 void Photon.Pun.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.

8.39.2.2 GameObject Photon.Pun.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.

8.40 Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks Interface Reference

Public Member Functions

- void OnTurnBegins (int turn)

    Called the turn begins event.
- void OnTurnCompleted (int turn)

    Called when a turn is completed (finished by all players)
- void OnPlayerMove (Player player, int turn, object move)

    Called when a player moved (but did not finish the turn)
- void OnPlayerFinished (Player player, int turn, object move)

    When a player finishes a turn (includes the action/move of that player)
- void OnTurnTimeEnds (int turn)

    Called when a turn completes due to a time constraint (timeout for a turn)

8.40.1 Member Function Documentation

8.40.1.1 void Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks.OnPlayerFinished (Player *player,* int *turn,* object *move*)

When a player finishes a turn (includes the action/move of that player)

Parameters

player	Player reference
turn	Turn index
move	Move Object data

8.40.1.2   void Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks.OnPlayerMove (Player *player,* int *turn,* object *move*)

Called when a player moved (but did not finish the turn)

Parameters

player	Player reference
turn	Turn Index
move	Move Object data

8.40.1.3   void Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks.OnTurnBegins (int *turn*)

Called the turn begins event.

Parameters

turn	Turn Index

8.40.1.4   void Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks.OnTurnCompleted (int *turn*)

Called when a turn is completed (finished by all players)

Parameters

turn	Turn Index

8.40.1.5   void Photon.Pun.UtilityScripts.IPunTurnManagerCallbacks.OnTurnTimeEnds (int *turn*)

Called when a turn completes due to a time constraint (timeout for a turn)

Parameters

turn	Turn index

8.41   Photon.Realtime.IWebRpcCallback Interface Reference

Interface for "WebRpc" callbacks for the Realtime Api. Currently includes only responses for Web RPCs.

Inherited by Photon.Realtime.WebRpcCallbacksContainer.

Public Member Functions

- void OnWebRpcResponse (OperationResponse response)

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

8.41.1 Detailed Description

Interface for "WebRpc" callbacks for the Realtime Api. Currently includes only responses for Web RPCs.

Classes that implement this interface must be registered to get callbacks for various situations.

To register for callbacks, use the LoadBalancingClient.WebRpcCallbackTargets and Add() the instance. To stop getting callbacks, Remove() the instance.

8.41.2 Member Function Documentation

8.41.2.1 void Photon.Realtime.IWebRpcCallback.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)

8.42 Photon.Realtime.LoadBalancingClient Class Reference

This class implements the Photon LoadBalancing workflow by using a LoadBalancingPeer. It keeps a state and will automatically execute transitions between the Master and Game Servers.

Inherits IPhotonPeerListener.

Public Member Functions

- LoadBalancingClient (ConnectionProtocol protocol=ConnectionProtocol.Udp)

    Creates a LoadBalancingClient with UDP protocol or the one specified.
- LoadBalancingClient (string masterAddress, string appId, string gameVersion, ConnectionProtocol protocol=ConnectionProtocol.Udp)

    Creates a LoadBalancingClient, setting various values needed before connecting.
- virtual bool Connect ()

    Starts the "process" to connect to a Master Server, using MasterServerAddress and AppId properties.
- bool ConnectToNameServer ()

    Connects to the NameServer for Photon Cloud, where a region and server list can be obtained.
- bool ConnectToRegionMaster (string region)

    Connects you to a specific region's Master Server, using the Name Server to find the IP.
- bool ReconnectToMaster ()

    Can be used to reconnect to the master server after a disconnect.
- bool ReconnectAndRejoin ()

    Can be used to return to a room quickly, by directly reconnecting to a game server to rejoin a room.
- void Disconnect ()

    Disconnects this client from any server and sets this.State if the connection is successfuly closed.
- void Service ()

    This method dispatches all available incoming commands and then sends this client's outgoing commands. It uses DispatchIncomingCommands and SendOutgoingCommands to do that.
- bool OpFindFriends (string[ ] friendsToFind)

    Request the rooms and online status for a list of friends. All clients should set a unique UserId before connecting. The result is available in this.FriendList.

- bool OpJoinLobby (TypedLobby lobby)

  If already connected to a Master Server, this joins the specified lobby. This request triggers an OnOperation↩ Response() call and the callback OnJoinedLobby().

- bool OpLeaveLobby ()

  Opposite of joining a lobby. You don't have to explicitly leave a lobby to join another (client can be in one max, at any time).

- bool OpJoinRandomRoom (OpJoinRandomRoomParams opJoinRandomRoomParams=null)

  Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.

- bool OpCreateRoom (EnterRoomParams enterRoomParams)

  Creates a new room. Will callback: OnCreatedRoom and OnJoinedRoom or OnCreateRoomFailed.

- bool OpJoinOrCreateRoom (EnterRoomParams enterRoomParams)

  Joins a specific room by name and creates it on demand. Will callback: OnJoinedRoom or OnJoinRoomFailed.

- bool OpJoinRoom (EnterRoomParams enterRoomParams)

  Joins a room by name. Will callback: OnJoinedRoom or OnJoinRoomFailed.

- bool OpRejoinRoom (string roomName)

  Rejoins a room by roomName (using the userID internally to return). Will callback: OnJoinedRoom or OnJoinRoom↩ Failed.

- bool OpLeaveRoom (bool becomeInactive, bool sendAuthCookie=false)

  Leaves the current room, optionally telling the server that the user is just becoming inactive. Will callback: OnLeft↩ Room.

- bool OpGetGameList (TypedLobby typedLobby, string sqlLobbyFilter)

  Gets a list of games matching a SQL-like where clause.

- bool OpSetCustomPropertiesOfActor (int actorNr, Hashtable propertiesToSet, Hashtable expected↩ Properties=null, WebFlags webFlags=null)

  Updates and synchronizes a Player's Custom Properties. Optionally, expectedProperties can be provided as condition.

- bool OpSetCustomPropertiesOfRoom (Hashtable propertiesToSet, Hashtable expectedProperties=null, WebFlags webFlags=null)

  Updates and synchronizes this Room's Custom Properties. Optionally, expectedProperties can be provided as condition.

- virtual bool OpRaiseEvent (byte eventCode, object customEventContent, RaiseEventOptions raiseEvent↩ Options, SendOptions sendOptions)

  Send an event with custom code/type and any content to the other players in the same room.

- virtual bool OpChangeGroups (byte[ ] groupsToRemove, byte[ ] groupsToAdd)

  Operation to handle this client's interest groups (for events in room).

- virtual void DebugReturn (DebugLevel level, string message)

  Debug output of low level api (and this client).

- virtual void OnOperationResponse (OperationResponse operationResponse)

  Uses the OperationResponses provided by the server to advance the internal state and call ops as needed.

- virtual void OnStatusChanged (StatusCode statusCode)

  Uses the connection's statusCodes to advance the internal state and call operations as needed.

- virtual void OnEvent (EventData photonEvent)

  Uses the photonEvent's provided by the server to advance the internal state and call ops as needed.

- virtual void OnMessage (object message)

  In Photon 4, "raw messages" will get their own callback method in the interface. Not used yet.

- bool OpWebRpc (string uriPath, object parameters, bool sendAuthCookie=false)

  This operation makes Photon call your custom web-service by path/name with the given parameters (converted into Json).

- void **AddCallbackTarget** (object target)

- void **RemoveCallbackTarget** (object target)

Public Attributes

- **AuthModeOption AuthMode** = AuthModeOption.Auth

  Enables the new Authentication workflow.
- **EncryptionMode EncryptionMode** = **EncryptionMode.PayloadEncryption**

  Defines how the communication gets encrypted.
- ConnectionProtocol **ExpectedProtocol** = ConnectionProtocol.Udp

  The protocol which will be used on Master- and Gameserver.
- string **NameServerHost** = "ns.exitgames.com"

  Name Server Host Name for Photon Cloud. Without port and without any prefix.
- string **NameServerHttp** = "http://ns.exitgames.com:80/photon/n"

  Name Server for HTTP connections to the Photon Cloud. Includes prefix and port.
- bool **EnableLobbyStatistics**

  If enabled, the client will get a list of available lobbies from the Master Server.
- **RegionHandler RegionHandler**

  Contains the list if enabled regions this client may use. Null, unless the client got a response to OpGetRegions.

Properties

- **LoadBalancingPeer LoadBalancingPeer** `[get]`

  The client uses a LoadBalancingPeer as API to communicate with the server. This is public for ease-of-use: Some methods like OpRaiseEvent are not relevant for the connection state and don't need a override.
- string **AppVersion** `[get, set]`

  The version of your client. A new version also creates a new "virtual app" to separate players from older client versions.
- string **AppId** `[get, set]`

  The AppID as assigned from the Photon Cloud. If you host yourself, this is the "regular" Photon Server Application Name (most likely: "LoadBalancing").
- **AuthenticationValues AuthValues** `[get, set]`

  User authentication values to be sent to the Photon server right after connecting.
- bool **IsUsingNameServer** `[get, set]`

  True if this client uses a NameServer to get the Master Server address.
- string **NameServerAddress** `[get]`

  Name Server Address for Photon Cloud (based on current protocol). You can use the default values and usually won't have to set this value.
- bool **UseAlternativeUdpPorts** `[get, set]`

  Use the alternative ports for UDP connections in the Public Cloud (27000 to 27003).
- string **CurrentServerAddress** `[get]`

  The currently used server address (if any). The type of server is define by Server property.
- string **MasterServerAddress** `[get, set]`

  Your Master Server address. In PhotonCloud, call ConnectToRegionMaster() to find your Master Server.
- string **GameServerAddress** `[get, set]`

  The game server's address for a particular room. In use temporarily, as assigned by master.
- **ServerConnection Server** `[get]`

  The server this client is currently connected or connecting to.
- **ClientState State** `[get, set]`

  Current state this client is in. Careful: several states are "transitions" that lead to other states.
- bool **IsConnected** `[get]`

  Returns if this client is currently connected or connecting to some type of server.
- bool **IsConnectedAndReady** `[get]`

  A refined version of IsConnected which is true only if your connection is ready to send operations.
- **DisconnectCause DisconnectedCause** `[get, protected set]`

Summarizes (aggregates) the different causes for disconnects of a client.

- bool InLobby [get]

    Internal value if the client is in a lobby.
- TypedLobby CurrentLobby [get, set]

    The lobby this client currently uses.
- Player LocalPlayer [get, set]

    The local player is never null but not valid unless the client is in a room, too. The ID will be -1 outside of rooms.
- string NickName [get, set]

    The nickname of the player (synced with others). Same as client.LocalPlayer.NickName.
- string UserId [get, set]

    An ID for this user. Sent in OpAuthenticate when you connect. If not set, the PlayerName is applied during connect.
- Room CurrentRoom [get]

    The current room this client is connected to (null if none available).
- bool InRoom [get]

    Is true while being in a room (this.state == ClientState.Joined).
- int PlayersOnMasterCount [get, set]

    Statistic value available on master server: Players on master (looking for games).
- int PlayersInRoomsCount [get, set]

    Statistic value available on master server: Players in rooms (playing).
- int RoomsCount [get, set]

    Statistic value available on master server: Rooms currently created.
- bool IsFetchingFriendList [get]

    Internal flag to know if the client currently fetches a friend list.
- string CloudRegion [get]

    The cloud region this client connects to. Set by ConnectToRegionMaster(). Not set if you don't use a NameServer!

Events

- Action< ClientState, ClientState > StateChanged

    Register a method to be called when this client's ClientState gets set.
- Action< EventData > EventReceived

    Register a method to be called when an event got dispatched. Gets called after the LoadBalancingClient handled the internal events first.
- Action< OperationResponse > OpResponseReceived

    Register a method to be called when an operation response is received.

8.42.1 Detailed Description

This class implements the Photon LoadBalancing workflow by using a LoadBalancingPeer. It keeps a state and will automatically execute transitions between the Master and Game Servers.

This class (and the Player class) should be extended to implement your own game logic. You can override Create↩ Player as "factory" method for Players and return your own Player instances. The State of this class is essential to know when a client is in a lobby (or just on the master) and when in a game where the actual gameplay should take place. Extension notes: An extension of this class should override the methods of the IPhotonPeerListener, as they are called when the state changes. Call base.method first, then pick the operation or state you want to react to and put it in a switch-case. We try to provide demo to each platform where this api can be used, so lookout for those.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 Photon.Realtime.LoadBalancingClient.LoadBalancingClient (ConnectionProtocol *protocol =* ConnectionProtocol.Udp)

Creates a LoadBalancingClient with UDP protocol or the one specified.

Parameters

protocol	Specifies the network protocol to use for connections.

8.42.2.2 Photon.Realtime.LoadBalancingClient.LoadBalancingClient (string *masterAddress,* string *appId,* string *gameVersion,* ConnectionProtocol *protocol =* `ConnectionProtocol.Udp`)

Creates a LoadBalancingClient, setting various values needed before connecting.

Parameters

masterAddress	The Master Server's address to connect to. Used in Connect.
appId	The AppId of this title. Needed for the Photon Cloud. Find it in the Dashboard.
gameVersion	A version for this client/build. In the Photon Cloud, players are separated by AppId, Game↩ Version and Region.
protocol	Specifies the network protocol to use for connections.

8.42.3 Member Function Documentation

8.42.3.1 virtual bool Photon.Realtime.LoadBalancingClient.Connect ( ) `[virtual]`

Starts the "process" to connect to a Master Server, using MasterServerAddress and AppId properties.

To connect to the Photon Cloud, use ConnectToRegionMaster().

The process to connect includes several steps: the actual connecting, establishing encryption, authentification (of app and optionally the user) and connecting to the MasterServer

Users can connect either anonymously or use "Custom Authentication" to verify each individual player's login. Custom Authentication in Photon uses external services and communities to verify users. While the client provides a user's info, the service setup is done in the Photon Cloud Dashboard. The parameter authValues will set this.↩ AuthValues and use them in the connect process.

Connecting to the Photon Cloud might fail due to:

- Network issues (OnStatusChanged() StatusCode.ExceptionOnConnect)

- Region not available (OnOperationResponse() for OpAuthenticate with ReturnCode == ErrorCode.Invalid↩ Region)

- Subscription CCU limit reached (OnOperationResponse() for OpAuthenticate with ReturnCode == Error↩ Code.MaxCcuReached)

8.42.3.2 bool Photon.Realtime.LoadBalancingClient.ConnectToNameServer (   )

Connects to the NameServer for Photon Cloud, where a region and server list can be obtained.

OpGetRegions

Returns

If the workflow was started or failed right away.

8.42.3.3 bool Photon.Realtime.LoadBalancingClient.ConnectToRegionMaster (string *region*)

Connects you to a specific region's Master Server, using the Name Server to find the IP.

Returns

If the operation could be sent. If false, no operation was sent.

8.42.3.4 virtual void Photon.Realtime.LoadBalancingClient.DebugReturn (DebugLevel *level,* string *message*) `[virtual]`

Debug output of low level api (and this client).

This method is not responsible to keep up the state of a LoadBalancingClient. Calling base.DebugReturn on overrides is optional.

8.42.3.5 void Photon.Realtime.LoadBalancingClient.Disconnect ( )

Disconnects this client from any server and sets this.State if the connection is successfuly closed.

8.42.3.6 virtual void Photon.Realtime.LoadBalancingClient.OnEvent (EventData *photonEvent*) `[virtual]`

Uses the photonEvent's provided by the server to advance the internal state and call ops as needed.

This method is essential to update the internal state of a LoadBalancingClient. Overriding methods must call base.OnEvent.

8.42.3.7 virtual void Photon.Realtime.LoadBalancingClient.OnMessage (object *message*) `[virtual]`

In Photon 4, "raw messages" will get their own callback method in the interface. Not used yet.

8.42.3.8 virtual void Photon.Realtime.LoadBalancingClient.OnOperationResponse (OperationResponse *operationResponse*) `[virtual]`

Uses the OperationResponses provided by the server to advance the internal state and call ops as needed.

When this method finishes, it will call your OnOpResponseAction (if any). This way, you can get any operation response without overriding this class.

To implement a more complex game/app logic, you should implement your own class that inherits the Load↩ BalancingClient. Override this method to use your own operation-responses easily.

This method is essential to update the internal state of a LoadBalancingClient, so overriding methods must call base.OnOperationResponse().

Parameters

operation↩ Response	Contains the server's response for an operation called by this peer.

8.42.3.9 virtual void Photon.Realtime.LoadBalancingClient.OnStatusChanged (StatusCode *statusCode*) `[virtual]`

Uses the connection's statusCodes to advance the internal state and call operations as needed.

This method is essential to update the internal state of a LoadBalancingClient. Overriding methods must call base.OnStatusChanged.

8.42.3.10 virtual bool Photon.Realtime.LoadBalancingClient.OpChangeGroups (byte[ ] *groupsToRemove,* byte[ ] *groupsToAdd*) `[virtual]`

Operation to handle this client's interest groups (for events in room).

Note the difference between passing null and byte[0]: null won't add/remove any groups. byte[0] will add/remove all (existing) groups. First, removing groups is executed. This way, you could leave all groups and join only the ones provided.

Changes become active not immediately but when the server executes this operation (approximately RTT/2).

Parameters

groupsTo↩ Remove	Groups to remove from interest. Null will not remove any. A byte[0] will remove all.
groupsToAdd	Groups to add to interest. Null will not add any. A byte[0] will add all current.

Returns

If operation could be enqueued for sending. Sent when calling: Service or SendOutgoingCommands.

8.42.3.11   bool Photon.Realtime.LoadBalancingClient.OpCreateRoom (EnterRoomParams *enterRoomParams*)

Creates a new room. Will callback: OnCreatedRoom and OnJoinedRoom or OnCreateRoomFailed.

When successful, the client will enter the specified room and callback both OnCreatedRoom and OnJoinedRoom. In all error cases, OnCreateRoomFailed gets called.

Creating a room will fail if the room name is already in use or when the RoomOptions clashing with one another. Check the EnterRoomParams reference for the various room creation options.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

When you're in the room, this client's State will become ClientState.Joined.

When entering a room, this client's Player Custom Properties will be sent to the room. Use LocalPlayer.Set↩ CustomProperties to set them, even while not yet in the room. Note that the player properties will be cached locally and are not wiped when leaving a 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

enterRoom↩ Params	Definition of properties for the room to create.

Returns

If the operation could be sent currently (requires connection to Master Server).

8.42.3.12   bool Photon.Realtime.LoadBalancingClient.OpFindFriends (string[ ] *friendsToFind*)

Request the rooms and online status for a list of friends. All clients should set a unique UserId before connecting. The result is available in this.FriendList.

Used on Master Server to find the rooms played by a selected list of users. The result will be stored in Load↩ BalancingClient.FriendList, which is null before the first server response.

Users identify themselves by setting a UserId in the LoadBalancingClient instance. This will send the ID in Op↩ Authenticate during connect (to master and game servers). Note: Changing a player's name doesn't make sense when using a friend list.

The list of usernames 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): Parameter↩
Code.FindFriendsResponseOnlineList = bool[] of online states ParameterCode.FindFriendsResponseRoomIdList =
string[] of room names (empty string if not in a room)

Parameters

friendsToFind	Array of friend's names (make sure they are unique).

Returns

If the operation could be sent (requires connection).

8.42.3.13   bool Photon.Realtime.LoadBalancingClient.OpGetGameList (TypedLobby *typedLobby,* **string** *sqlLobbyFilter* **)**

Gets a list of games matching a SQL-like where clause.

Operation is only available for lobbies of type SqlLobby. This is an async request which triggers a OnOperation↩
Response() call.

https://doc.photonengine.com/en-us/realtime/current/reference/matchmaking-and-lobby::sql_lobby_type

Parameters

typedLobby	The lobby to query. Has to be of type SqlLobby.
sqlLobbyFilter	The sql query statement.

Returns

If the operation could be sent (has to be connected).

8.42.3.14   bool Photon.Realtime.LoadBalancingClient.OpJoinLobby (TypedLobby *lobby* **)**

If already connected to a Master Server, this joins the specified lobby. This request triggers an OnOperation↩
Response() call and the callback OnJoinedLobby().

Parameters

lobby	The lobby to join. Use null for default lobby.

Returns

If the operation could be sent. False, if the client is not IsConnectedAndReady or when it's not connected to a
Master Server.

8.42.3.15   bool Photon.Realtime.LoadBalancingClient.OpJoinOrCreateRoom (EnterRoomParams *enterRoomParams* **)**

Joins a specific room by name and creates it on demand. Will callback: OnJoinedRoom or OnJoinRoomFailed.

Useful when players make up a room name to meet in: All involved clients call the same method and whoever is
first, also creates the room.

When successful, the client will enter the specified room. The client which creates the room, will callback both
OnCreatedRoom and OnJoinedRoom. Clients that join an existing room will only callback OnJoinedRoom. In all
error cases, OnJoinRoomFailed gets called.

Joining a room will fail, if the room is full, closed or when the user already is present in the room (checked by userId).

To return to a room, use OpRejoinRoom.

This method can only be called while the client is connected to a Master Server so you should implement the
callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server.
Note: There will be no callbacks if this method returned false.

This client's State is set to ClientState.Joining immediately, when the operation could be called. In the background,
the client will switch servers and call various related operations.

When you're in the room, this client's State will become ClientState.Joined.

If you set room properties in roomOptions, they get ignored when the room is existing already. This avoids changing the room properties by late joining players.

When entering a room, this client's Player Custom Properties will be sent to the room. Use LocalPlayer.Set↵ CustomProperties to set them, even while not yet in the room. Note that the player properties will be cached locally and are not wiped when leaving a 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

enterRoom↵ Params	Definition of properties for the room to create or join.

Returns

> If the operation could be sent currently (requires connection to Master Server).

8.42.3.16 bool Photon.Realtime.LoadBalancingClient.OpJoinRandomRoom (OpJoinRandomRoomParams *opJoinRandomRoomParams =* null **)**

Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.

Used for random matchmaking. You can join any room or one with specific properties defined in opJoinRandom↵ RoomParams.

You can use expectedCustomRoomProperties and expectedMaxPlayers as filters for accepting rooms. If you set expectedCustomRoomProperties, a room must have the exact same key values set at Custom Properties. You need to define which Custom Room Properties will be available for matchmaking when you create a room. See: OpCreateRoom(string roomName, RoomOptions roomOptions, TypedLobby lobby)

This operation fails if no rooms are fitting or available (all full, closed or not visible). It may also fail when actually joining the room which was found. Rooms may close, become full or empty anytime.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

This client's State is set to ClientState.Joining immediately, when the operation could be called. In the background, the client will switch servers and call various related operations.

When you're in the room, this client's State will become ClientState.Joined.

When entering a room, this client's Player Custom Properties will be sent to the room. Use LocalPlayer.Set↵ CustomProperties to set them, even while not yet in the room. Note that the player properties will be cached locally and are not wiped when leaving a room.

More about matchmaking: https://doc.photonengine.com/en-us/realtime/current/reference/matchmak

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

opJoin↵ RandomRoom↵ Params	Optional definition of properties to filter rooms in random matchmaking.

Returns

> If the operation could be sent currently (requires connection to Master Server).

8.42.3.17 bool Photon.Realtime.LoadBalancingClient.OpJoinRoom (EnterRoomParams *enterRoomParams*)

Joins a room by name. Will callback: OnJoinedRoom or OnJoinRoomFailed.

Useful when using lobbies or when players follow friends or invite each other.

When successful, the client will enter the specified room and callback via OnJoinedRoom. In all error cases, On↩
JoinRoomFailed gets called.

Joining a room will fail if the room is full, closed, not existing or when the user already is present in the room (checked by userId).

To return to a room, use OpRejoinRoom. When players invite each other and it's unclear who's first to respond, use OpJoinOrCreateRoom instead.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

A room's name has to be unique (per region, appid and gameversion). When your title uses a global matchmaking or invitations (e.g. an external solution), keep regions and the game versions in mind to join a room.

This client's State is set to ClientState.Joining immediately, when the operation could be called. In the background, the client will switch servers and call various related operations.

When you're in the room, this client's State will become ClientState.Joined.

When entering a room, this client's Player Custom Properties will be sent to the room. Use LocalPlayer.Set↩
CustomProperties to set them, even while not yet in the room. Note that the player properties will be cached locally and are not wiped when leaving a room.

You can define an array of expectedUsers, to reserve player slots in the room for friends or party members. The corresponding feature in Photon is called "Slot Reservation" and can be found in the doc pages.

Parameters

enterRoom↩ Params	Definition of properties for the room to join.

Returns

If the operation could be sent currently (requires connection to Master Server).

8.42.3.18 bool Photon.Realtime.LoadBalancingClient.OpLeaveLobby ( )

Opposite of joining a lobby. You don't have to explicitly leave a lobby to join another (client can be in one max, at any time).

Returns

If the operation could be sent (has to be connected).

8.42.3.19 bool Photon.Realtime.LoadBalancingClient.OpLeaveRoom (bool *becomeInactive,* bool *sendAuthCookie =* `false`)

Leaves the current room, optionally telling the server that the user is just becoming inactive. Will callback: OnLeft↩
Room.

OpLeaveRoom skips execution when the room is null or the server is not GameServer or the client is disconnecting from GS already. OpLeaveRoom returns false in those cases and won't change the state, so check return of this method.

In some cases, this method will skip the OpLeave call and just call Disconnect(), which not only leaves the room but also the server. Disconnect also triggers a leave and so that workflow is is quicker.

Parameters

becomeInactive	If true, this player becomes inactive in the game and can return later (if PlayerTTL of the room is != 0).
sendAuthCookie	WebFlag: Securely transmit the encrypted object AuthCookie to the web service in Path↩ Leave webhook when available

Returns

If the current room could be left (impossible while not in a room).

8.42.3.20 virtual bool Photon.Realtime.LoadBalancingClient.OpRaiseEvent (byte *eventCode,* object *customEventContent,* RaiseEventOptions *raiseEventOptions,* SendOptions *sendOptions*) `[virtual]`

Send an event with custom code/type and any content to the other players in the same room.

Parameters

eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
customEvent↩ Content	Any serializable datatype (including Hashtable like the other OpRaiseEvent overloads).
raiseEvent↩ Options	Contains used send options. If you pass null, the default options will be used.
sendOptions	Send options for reliable, encryption etc

Returns

If operation could be enqueued for sending. Sent when calling: Service or SendOutgoingCommands.

8.42.3.21 bool Photon.Realtime.LoadBalancingClient.OpRejoinRoom (string *roomName*)

Rejoins a room by roomName (using the userID internally to return). Will callback: OnJoinedRoom or OnJoin↩ RoomFailed.

Used to return to a room, before this user was removed from the players list. Internally, the userID will be checked by the server, to make sure this user is in the room (active or inactice).

In contrast to join, this operation never adds a players to a room. It will attempt to retake an existing spot in the playerlist or fail. This makes sure the client doean't accidentally join a room when the game logic meant to re-activate an existing actor in an existing room.

This method will fail on the server, when the room does not exist, can't be loaded (persistent rooms) or when the userId is not in the player list of this room. This will lead to a callback OnJoinRoomFailed.

8.42.3.22 bool Photon.Realtime.LoadBalancingClient.OpSetCustomPropertiesOfActor (int *actorNr,* Hashtable *propertiesToSet,* Hashtable *expectedProperties =* `null`, WebFlags *webFlags =* `null`)

Updates and synchronizes a Player's Custom Properties. Optionally, expectedProperties can be provided as condition.

Custom Properties are a set of string keys and arbitrary values which is synchronized for the players in a Room. They are available when the client enters the room, as they are in the response of OpJoin and OpCreate.

Custom Properties either relate to the (current) Room or a Player (in that Room).

Both classes locally cache the current key/values and make them available as property: CustomProperties. This is provided only to read them. You must use the method SetCustomProperties to set/modify them.

Any client can set any Custom Properties anytime (when in a room). It's up to the game logic to organize how they are best used.

You should call SetCustomProperties only with key/values that are new or changed. This reduces traffic and performance.

Unless you define some expectedProperties, setting key/values is always permitted. In this case, the property-setting client will not receive the new values from the server but instead update its local cache in SetCustom↩ Properties.

If you define expectedProperties, the server will skip updates if the server property-cache does not contain all expectedProperties with the same values. In this case, the property-setting client will get an update from the server and update it's cached key/values at about the same time as everyone else.

The benefit of using expectedProperties can be only one client successfully sets a key from one known value to another. As example: Store who owns an item in a Custom Property "ownedBy". It's 0 initally. When multiple players reach the item, they all attempt to change "ownedBy" from 0 to their actorNumber. If you use expectedProperties {"ownedBy", 0} as condition, the first player to take the item will have it (and the others fail to set the ownership).

Properties get saved with the game state for Turnbased games (which use IsPersistent = true).

Parameters

actorNr	Defines which player the Custom Properties belong to. ActorID of a player.
propertiesToSet	Hashtable of Custom Properties that changes.
expected↩ Properties	Provide some keys/values to use as condition for setting the new values. Client must be in room.
webFlags	Defines if the set properties should be forwarded to a WebHook. Client must be in room.

8.42.3.23 bool Photon.Realtime.LoadBalancingClient.OpSetCustomPropertiesOfRoom (Hashtable *propertiesToSet,* Hashtable *expectedProperties* = `null`*,* **WebFlags *webFlags* =** `null` **)**

Updates and synchronizes this Room's Custom Properties. Optionally, expectedProperties can be provided as condition.

Custom Properties are a set of string keys and arbitrary values which is synchronized for the players in a Room. They are available when the client enters the room, as they are in the response of OpJoin and OpCreate.

Custom Properties either relate to the (current) Room or a Player (in that Room).

Both classes locally cache the current key/values and make them available as property: CustomProperties. This is provided only to read them. You must use the method SetCustomProperties to set/modify them.

Any client can set any Custom Properties anytime (when in a room). It's up to the game logic to organize how they are best used.

You should call SetCustomProperties only with key/values that are new or changed. This reduces traffic and performance.

Unless you define some expectedProperties, setting key/values is always permitted. In this case, the property-setting client will not receive the new values from the server but instead update its local cache in SetCustom↩ Properties.

If you define expectedProperties, the server will skip updates if the server property-cache does not contain all expectedProperties with the same values. In this case, the property-setting client will get an update from the server and update it's cached key/values at about the same time as everyone else.

The benefit of using expectedProperties can be only one client successfully sets a key from one known value to another. As example: Store who owns an item in a Custom Property "ownedBy". It's 0 initally. When multiple players reach the item, they all attempt to change "ownedBy" from 0 to their actorNumber. If you use expectedProperties {"ownedBy", 0} as condition, the first player to take the item will have it (and the others fail to set the ownership).

Properties get saved with the game state for Turnbased games (which use IsPersistent = true).

Parameters

propertiesToSet	Hashtable of Custom Properties that changes.
expected← *Properties*	Provide some keys/values to use as condition for setting the new values.
webFlags	Defines web flags for an optional PathProperties webhook.

8.42.3.24 bool Photon.Realtime.LoadBalancingClient.OpWebRpc (string *uriPath,* object *parameters,* bool *sendAuthCookie =* `false`)

This operation makes Photon call your custom web-service by path/name with the given parameters (converted into Json).

A WebRPC calls a custom, http-based function on a server you provide. The uriPath is relative to a "base path" which is configured server-side. The sent parameters get converted from C# types to Json. Vice versa, the response of the web-service will be converted to C# types and sent back as normal operation response.

To use this feature, you have to setup your server:

For a Photon Cloud application, `visit the Dashboard` and setup "WebHooks". The BaseUrl is used for WebRPCs as well.

The response by Photon will call OnOperationResponse() with Code: OperationCode.WebRpc. To get this response, you can derive the LoadBalancingClient, or (much easier) you set a suitable OnOpResponseAction to be called.

It's important to understand that the OperationResponse tells you if the WebRPC could be called or not but the content of the response will contain the values the web-service sent (if any). If the web-service could not execute the request, it might return another error and a message. This is inside the OperationResponse.

The class WebRpcResponse is a helper-class that extracts the most valuable content from the WebRPC response.

To get a WebRPC response, set a OnOpResponseAction:

```
this.OnOpResponseAction = this.OpResponseHandler;
```

It could look like this:

```
public void OpResponseHandler(OperationResponse operationResponse)
{
    if (operationResponse.OperationCode == OperationCode.WebRpc)
    {
        if (operationResponse.ReturnCode != 0)
        {
            Console.WriteLine("WebRpc failed. Response: " + operationResponse.ToStringFull());
        }
        else
        {
            WebRpcResponse webResponse = new WebRpcResponse(operationResponse);
            Console.WriteLine(webResponse.DebugMessage);    // message from the webserver

            // do something with the response...
        }
    }
}
```

Parameters

uriPath	The url path to call, relative to the baseUrl configured on Photon's server-side.
parameters	The parameters to send to the web-service method.

sendAuthCookie	Defines if the authentication cookie gets sent to a WebHook (if setup).

8.42.3.25 bool Photon.Realtime.LoadBalancingClient.ReconnectAndRejoin ( )

Can be used to return to a room quickly, by directly reconnecting to a game server to rejoin a room.

Returns

False, if the conditions are not met. Then, this client does not attempt the ReconnectAndRejoin.

8.42.3.26 bool Photon.Realtime.LoadBalancingClient.ReconnectToMaster ( )

Can be used to reconnect to the master server after a disconnect.

Common use case: Press the Lock Button on a iOS device and you get disconnected immediately.

8.42.3.27 void Photon.Realtime.LoadBalancingClient.Service ( )

This method dispatches all available incoming commands and then sends this client's outgoing commands. It uses DispatchIncomingCommands and SendOutgoingCommands to do that.

The Photon client libraries are designed to fit easily into a game or application. The application is in control of the context (thread) in which incoming events and responses are executed and has full control of the creation of UDP/TCP packages.

Sending packages and dispatching received messages are two separate tasks. Service combines them into one method at the cost of control. It calls DispatchIncomingCommands and SendOutgoingCommands.

Call this method regularly (2..20 times a second).

This will Dispatch ANY received commands (unless a reliable command in-order is still missing) and events AND will send queued outgoing commands. Fewer calls might be more effective if a device cannot send many packets per second, as multiple operations might be combined into one package.

You could replace Service by:

```
while (DispatchIncomingCommands()); //Dispatch until everything is Dispatched...
SendOutgoingCommands(); //Send a UDP/TCP package with outgoing messages
```

See also

PhotonPeer.DispatchIncomingCommands, PhotonPeer.SendOutgoingCommands

8.42.4 Member Data Documentation

8.42.4.1 AuthModeOption Photon.Realtime.LoadBalancingClient.AuthMode = AuthModeOption.Auth

Enables the new Authentication workflow.

8.42.4.2 bool Photon.Realtime.LoadBalancingClient.EnableLobbyStatistics

If enabled, the client will get a list of available lobbies from the Master Server.

Set this value before the client connects to the Master Server. While connected to the Master Server, a change has no effect.

Implement OptionalInfoCallbacks.OnLobbyStatisticsUpdate, to get the list of used lobbies.

The lobby statistics can be useful if your title dynamically uses lobbies, depending (e.g.) on current player activity or such. In this case, getting a list of available lobbies, their room-count and player-count can be useful info.

ConnectUsingSettings sets this to the PhotonServerSettings value.

8.42.4.3 EncryptionMode Photon.Realtime.LoadBalancingClient.EncryptionMode = EncryptionMode.Payload↩
Encryption

Defines how the communication gets encrypted.

8.42.4.4 ConnectionProtocol Photon.Realtime.LoadBalancingClient.ExpectedProtocol = ConnectionProtocol.Udp

The protocol which will be used on Master- and Gameserver.

When using AuthMode = AuthModeOption.AuthOnceWss, the client uses a wss-connection on the Nameserver but another protocol on the other servers. As the Nameserver sends an address, which is different per protocol, it needs to know the expected protocol.

summary>Simplifies getting the token for connect/init requests, if this feature is enabled.

8.42.4.5 string Photon.Realtime.LoadBalancingClient.NameServerHost = "ns.exitgames.com"

Name Server Host Name for Photon Cloud. Without port and without any prefix.

8.42.4.6 string Photon.Realtime.LoadBalancingClient.NameServerHttp = "http://ns.exitgames.com:80/photon/n"

Name Server for HTTP connections to the Photon Cloud. Includes prefix and port.

8.42.4.7 RegionHandler Photon.Realtime.LoadBalancingClient.RegionHandler

Contains the list if enabled regions this client may use. Null, unless the client got a response to OpGetRegions.

8.42.5 Property Documentation

8.42.5.1 string Photon.Realtime.LoadBalancingClient.AppId `[get]`,`[set]`

The AppID as assigned from the Photon Cloud. If you host yourself, this is the "regular" Photon Server Application Name (most likely: "LoadBalancing").

8.42.5.2 string Photon.Realtime.LoadBalancingClient.AppVersion `[get]`,`[set]`

The version of your client. A new version also creates a new "virtual app" to separate players from older client versions.

8.42.5.3 AuthenticationValues Photon.Realtime.LoadBalancingClient.AuthValues `[get]`,`[set]`

User authentication values to be sent to the Photon server right after connecting.

Set this property or pass AuthenticationValues by Connect(..., authValues).

8.42.5.4 string Photon.Realtime.LoadBalancingClient.CloudRegion `[get]`

The cloud region this client connects to. Set by ConnectToRegionMaster(). Not set if you don't use a NameServer!

8.42.5.5 **TypedLobby Photon.Realtime.LoadBalancingClient.CurrentLobby** `[get]`,`[set]`

The lobby this client currently uses.

8.42.5.6 **Room Photon.Realtime.LoadBalancingClient.CurrentRoom** `[get]`

The current room this client is connected to (null if none available).

8.42.5.7 **string Photon.Realtime.LoadBalancingClient.CurrentServerAddress** `[get]`

The currently used server address (if any). The type of server is define by Server property.

8.42.5.8 **DisconnectCause Photon.Realtime.LoadBalancingClient.DisconnectedCause** `[get]`,`[protected set]`

Summarizes (aggregates) the different causes for disconnects of a client.

A disconnect can be caused by: errors in the network connection or some vital operation failing (which is considered "high level"). While operations always trigger a call to OnOperationResponse, connection related changes are treated in OnStatusChanged. The DisconnectCause is set in either case and summarizes the causes for any disconnect in a single state value which can be used to display (or debug) the cause for disconnection.

8.42.5.9 **string Photon.Realtime.LoadBalancingClient.GameServerAddress** `[get]`,`[set]`

The game server's address for a particular room. In use temporarily, as assigned by master.

8.42.5.10 **bool Photon.Realtime.LoadBalancingClient.InLobby** `[get]`

Internal value if the client is in a lobby.

This is used to re-set this.State, when joining/creating a room fails.

8.42.5.11 **bool Photon.Realtime.LoadBalancingClient.InRoom** `[get]`

Is true while being in a room (this.state == ClientState.Joined).

Aside from polling this value, game logic should implement IMatchmakingCallbacks in some class and react when that gets called.
OpRaiseEvent, OpLeave and some other operations can only be used (successfully) when the client is in a room.
PUN's PhotonNetwork.InRoom will support being InRoom in it's offline mode (by providing it's own property).

8.42.5.12 **bool Photon.Realtime.LoadBalancingClient.IsConnected** `[get]`

Returns if this client is currently connected or connecting to some type of server.

This is even true while switching servers. Use IsConnectedAndReady to check only for those states that enable you to send Operations.

8.42.5.13 **bool Photon.Realtime.LoadBalancingClient.IsConnectedAndReady** `[get]`

A refined version of IsConnected which is true only if your connection is ready to send operations.

Not all operations can be called on all types of servers. If an operation is unavailable on the currently connected server, this will result in a OperationResponse with ErrorCode != 0.

Examples: The NameServer allows OpGetRegions which is not available anywhere else. The MasterServer does not allow you to send events (OpRaiseEvent) and on the GameServer you are unable to join a lobby (OpJoinLobby).

To check which server you are on, use: PhotonNetwork.Server.

8.42.5.14   bool Photon.Realtime.LoadBalancingClient.IsFetchingFriendList `[get]`

Internal flag to know if the client currently fetches a friend list.

8.42.5.15   bool Photon.Realtime.LoadBalancingClient.IsUsingNameServer `[get],[set]`

True if this client uses a NameServer to get the Master Server address.

8.42.5.16   LoadBalancingPeer Photon.Realtime.LoadBalancingClient.LoadBalancingPeer `[get]`

The client uses a LoadBalancingPeer as API to communicate with the server. This is public for ease-of-use: Some methods like OpRaiseEvent are not relevant for the connection state and don't need a override.

8.42.5.17   Player Photon.Realtime.LoadBalancingClient.LocalPlayer `[get],[set]`

The local player is never null but not valid unless the client is in a room, too. The ID will be -1 outside of rooms.

8.42.5.18   string Photon.Realtime.LoadBalancingClient.MasterServerAddress `[get],[set]`

Your Master Server address. In PhotonCloud, call ConnectToRegionMaster() to find your Master Server.

In the Photon Cloud, explicit definition of a Master Server Address is not best practice. The Photon Cloud has a "Name Server" which redirects clients to a specific Master Server (per Region and AppId).

8.42.5.19   string Photon.Realtime.LoadBalancingClient.NameServerAddress `[get]`

Name Server Address for Photon Cloud (based on current protocol). You can use the default values and usually won't have to set this value.

8.42.5.20   string Photon.Realtime.LoadBalancingClient.NickName `[get],[set]`

The nickname of the player (synced with others). Same as client.LocalPlayer.NickName.

8.42.5.21   int Photon.Realtime.LoadBalancingClient.PlayersInRoomsCount `[get],[set]`

Statistic value available on master server: Players in rooms (playing).

8.42.5.22   int Photon.Realtime.LoadBalancingClient.PlayersOnMasterCount `[get],[set]`

Statistic value available on master server: Players on master (looking for games).

8.42.5.23   int Photon.Realtime.LoadBalancingClient.RoomsCount `[get],[set]`

Statistic value available on master server: Rooms currently created.

8.42.5.24 ServerConnection Photon.Realtime.LoadBalancingClient.Server `[get]`

The server this client is currently connected or connecting to.

Each server (NameServer, MasterServer, GameServer) allow some operations and reject others.

8.42.5.25 ClientState Photon.Realtime.LoadBalancingClient.State `[get]`,`[set]`

Current state this client is in. Careful: several states are "transitions" that lead to other states.

8.42.5.26 bool Photon.Realtime.LoadBalancingClient.UseAlternativeUdpPorts `[get]`,`[set]`

Use the alternative ports for UDP connections in the Public Cloud (27000 to 27003).

This should be used when players have issues with connection stability. Some players reported better connectivity for Steam games. The effect might vary, which is why the alternative ports are not the new default.

The alternative (server) ports are 27000 up to 27003.

The values are appplied by replacing any incoming server-address string accordingly. You only need to set this to true though.

This value does not affect TCP or WebSocket connections.

8.42.5.27 string Photon.Realtime.LoadBalancingClient.UserId `[get]`,`[set]`

An ID for this user. Sent in OpAuthenticate when you connect. If not set, the PlayerName is applied during connect.

On connect, if the UserId is null or empty, the client will copy the PlayName to UserId. If PlayerName is not set either (before connect), the server applies a temporary ID which stays unknown to this client and other clients.

The UserId is what's used in FindFriends and for fetching data for your account (with WebHooks e.g.).

By convention, set this ID before you connect, not while being connected. There is no error but the ID won't change while being connected.

8.42.6 Event Documentation

8.42.6.1 Action<EventData> Photon.Realtime.LoadBalancingClient.EventReceived

Register a method to be called when an event got dispatched. Gets called after the LoadBalancingClient handled the internal events first.

This is an alternative to extending LoadBalancingClient to override OnEvent().

Note that OnEvent is calling EventReceived after it handled internal events first. That means for example: Joining players will already be in the player list but leaving players will already be removed from the room.

8.42.6.2 Action<OperationResponse> Photon.Realtime.LoadBalancingClient.OpResponseReceived

Register a method to be called when an operation response is received.

This is an alternative to extending LoadBalancingClient to override OnOperationResponse().

Note that OnOperationResponse gets executed before your Action is called. That means for example: The Op↩
JoinLobby response already set the state to "JoinedLobby" and the response to OpLeave already triggered the Disconnect before this is called.

8.42.6.3 **Action**<**ClientState, ClientState**> **Photon.Realtime.LoadBalancingClient.StateChanged**

Register a method to be called when this client's ClientState gets set.

This can be useful to react to being connected, joined into a room, etc.

8.43 Photon.Realtime.LoadBalancingPeer Class Reference

A LoadbalancingPeer provides the operations and enum definitions needed to use the loadbalancing server application which is also used in Photon Cloud.

Inherits PhotonPeer.

Public Member Functions

- LoadBalancingPeer (ConnectionProtocol protocolType)

  Creates a Peer with specified connection protocol. You need to set the Listener before using the peer.
- LoadBalancingPeer (IPhotonPeerListener listener, ConnectionProtocol protocolType)

  Creates a Peer with specified connection protocol and a Listener for callbacks.
- virtual bool **OpGetRegions** (string appId)
- virtual bool OpJoinLobby (TypedLobby lobby=null)

  Joins the lobby on the Master Server, where you get a list of RoomInfos of currently open rooms. This is an async request which triggers a OnOperationResponse() call.
- virtual bool OpLeaveLobby ()

  Leaves the lobby on the Master Server. This is an async request which triggers a OnOperationResponse() call.
- virtual bool OpCreateRoom (EnterRoomParams opParams)

  Creates a room (on either Master or Game Server). The OperationResponse depends on the server the peer is connected to: Master will return a Game Server to connect to. Game Server will return the joined Room's data. This is an async request which triggers a OnOperationResponse() call.
- virtual bool OpJoinRoom (EnterRoomParams opParams)

  Joins a room by name or creates new room if room with given name not exists. The OperationResponse depends on the server the peer is connected to: Master will return a Game Server to connect to. Game Server will return the joined Room's data. This is an async request which triggers a OnOperationResponse() call.
- virtual bool OpJoinRandomRoom (OpJoinRandomRoomParams opJoinRandomRoomParams)

  Operation to join a random, available room. Overloads take additional player properties. This is an async request which triggers a OnOperationResponse() call. If all rooms are closed or full, the OperationResponse will have a returnCode of ErrorCode.NoRandomMatchFound. If successful, the OperationResponse contains a gameserver address and the name of some room.
- virtual bool OpLeaveRoom (bool becomeInactive, bool sendAuthCookie=false)

  Leaves a room with option to come back later or "for good".
- virtual bool OpGetGameList (TypedLobby lobby, string queryData)

  Gets a list of games matching a SQL-like where clause.
- virtual bool OpFindFriends (string[ ] friendsToFind)

  Request the rooms and online status for a list of friends (each client must set a unique username via OpAuthenticate).
- bool **OpSetCustomPropertiesOfActor** (int actorNr, Hashtable actorProperties)
- bool **OpSetCustomPropertiesOfRoom** (Hashtable gameProperties)
- virtual bool OpAuthenticate (string appId, string appVersion, AuthenticationValues authValues, string region↩ Code, bool getLobbyStatistics)

  Sends this app's appId and appVersion to identify this application server side. This is an async request which triggers a OnOperationResponse() call.
- virtual bool OpAuthenticateOnce (string appId, string appVersion, AuthenticationValues authValues, string regionCode, EncryptionMode encryptionMode, ConnectionProtocol expectedProtocol)

  Sends this app's appId and appVersion to identify this application server side. This is an async request which triggers a OnOperationResponse() call.

- virtual bool OpChangeGroups (byte[ ] groupsToRemove, byte[ ] groupsToAdd)

    Operation to handle this client's interest groups (for events in room).
- virtual bool OpRaiseEvent (byte eventCode, object customEventContent, RaiseEventOptions raiseEvent↩
Options, SendOptions sendOptions)

    Send an event with custom code/type and any content to the other players in the same room.
- virtual bool OpSettings (bool receiveLobbyStats)

    Internally used operation to set some "per server" settings. This is for the Master Server.

Protected Member Functions

- void **OpSetPropertyOfRoom** (byte propCode, object value)

8.43.1 Detailed Description

A LoadbalancingPeer provides the operations and enum definitions needed to use the loadbalancing server application which is also used in Photon Cloud.

Internally used by PUN. The LoadBalancingPeer does not keep a state, instead this is done by a LoadBalancing↩
Client.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 Photon.Realtime.LoadBalancingPeer.LoadBalancingPeer (ConnectionProtocol *protocolType*)

Creates a Peer with specified connection protocol. You need to set the Listener before using the peer.

Each connection protocol has it's own default networking ports for Photon.

Parameters

protocolType	The preferred option is UDP.

8.43.2.2 Photon.Realtime.LoadBalancingPeer.LoadBalancingPeer (IPhotonPeerListener *listener,* ConnectionProtocol *protocolType*)

Creates a Peer with specified connection protocol and a Listener for callbacks.

8.43.3 Member Function Documentation

8.43.3.1 virtual bool Photon.Realtime.LoadBalancingPeer.OpAuthenticate (string *appId,* string *appVersion,* AuthenticationValues *authValues,* string *regionCode,* bool *getLobbyStatistics*) `[virtual]`

Sends this app's appId and appVersion to identify this application server side. This is an async request which triggers a OnOperationResponse() call.

This operation makes use of encryption, if that is established before. See: EstablishEncryption(). Check encryption with IsEncryptionAvailable. This operation is allowed only once per connection (multiple calls will have ErrorCode != Ok).

Parameters

appId	Your application's name or ID to authenticate. This is assigned by Photon Cloud (webpage).
appVersion	The client's version (clients with differing client appVersions are separated and players don't meet).
authValues	Contains all values relevant for authentication. Even without account system (external Custom Auth), the clients are allowed to identify themselves.
regionCode	Optional region code, if the client should connect to a specific Photon Cloud Region.
getLobby↩ Statistics	Set to true on Master Server to receive "Lobby Statistics" events.

Returns

If the operation could be sent (has to be connected).

8.43.3.2 virtual bool Photon.Realtime.LoadBalancingPeer.OpAuthenticateOnce (string *appId,* string *appVersion,* AuthenticationValues *authValues,* string *regionCode,* EncryptionMode *encryptionMode,* ConnectionProtocol *expectedProtocol*) `[virtual]`

Sends this app's appId and appVersion to identify this application server side. This is an async request which triggers a OnOperationResponse() call.

This operation makes use of encryption, if that is established before. See: EstablishEncryption(). Check encryption with IsEncryptionAvailable. This operation is allowed only once per connection (multiple calls will have ErrorCode != Ok).

Parameters

appId	Your application's name or ID to authenticate. This is assigned by Photon Cloud (webpage).
appVersion	The client's version (clients with differing client appVersions are separated and players don't meet).
authValues	Optional authentication values. The client can set no values or a UserId or some parameters for Custom Authentication by a server.
regionCode	Optional region code, if the client should connect to a specific Photon Cloud Region.
encryptionMode	
expected↩ Protocol	

Returns

If the operation could be sent (has to be connected).

8.43.3.3 virtual bool Photon.Realtime.LoadBalancingPeer.OpChangeGroups (byte[ ] *groupsToRemove,* byte[ ] *groupsToAdd*) `[virtual]`

Operation to handle this client's interest groups (for events in room).

Note the difference between passing null and byte[0]: null won't add/remove any groups. byte[0] will add/remove all (existing) groups. First, removing groups is executed. This way, you could leave all groups and join only the ones provided.

Changes become active not immediately but when the server executes this operation (approximately RTT/2).

Parameters

| groupsTo↩ Remove | Groups to remove from interest. Null will not remove any. A byte[0] will remove all. |

groupsToAdd	Groups to add to interest. Null will not add any. A byte[0] will add all current.

Returns

If operation could be enqueued for sending. Sent when calling: Service or SendOutgoingCommands.

8.43.3.4 virtual bool Photon.Realtime.LoadBalancingPeer.OpCreateRoom (EnterRoomParams *opParams*) `[virtual]`

Creates a room (on either Master or Game Server). The OperationResponse depends on the server the peer is connected to: Master will return a Game Server to connect to. Game Server will return the joined Room's data. This is an async request which triggers a OnOperationResponse() call.

If the room is already existing, the OperationResponse will have a returnCode of ErrorCode.GameAlreadyExists.

8.43.3.5 virtual bool Photon.Realtime.LoadBalancingPeer.OpFindFriends (string[ ] *friendsToFind*) `[virtual]`

Request the rooms and online status for a list of friends (each client must set a unique username via Op←Authenticate).

Used on Master Server to find the rooms played by a selected list of users. Users identify themselves by using Op←Authenticate with a unique username. The list of usernames must be fetched from some other source (not provided by Photon).

The server response includes 2 arrays of info (each index matching a friend from the request): Parameter←Code.FindFriendsResponseOnlineList = bool[] of online states ParameterCode.FindFriendsResponseRoomIdList = string[] of room names (empty string if not in a room)

Parameters

friendsToFind	Array of friend's names (make sure they are unique).

Returns

If the operation could be sent (requires connection).

8.43.3.6 virtual bool Photon.Realtime.LoadBalancingPeer.OpGetGameList (TypedLobby *lobby,* string *queryData*) `[virtual]`

Gets a list of games matching a SQL-like where clause.

Operation is only available in lobbies of type SqlLobby. This is an async request which triggers a OnOperation←Response() call. Returned game list is stored in RoomInfoList.

https://doc.photonengine.com/en-us/realtime/current/reference/matchmaking-and-lobby::sql_lobby_type

Parameters

lobby	The lobby to query. Has to be of type SqlLobby.
queryData	The sql query statement.

Returns

If the operation could be sent (has to be connected).

8.43.3.7 virtual bool Photon.Realtime.LoadBalancingPeer.OpJoinLobby (TypedLobby *lobby =* null) `[virtual]`

Joins the lobby on the Master Server, where you get a list of RoomInfos of currently open rooms. This is an async request which triggers a OnOperationResponse() call.

Parameters

lobby	The lobby join to.

Returns

If the operation could be sent (has to be connected).

8.43.3.8   virtual bool Photon.Realtime.LoadBalancingPeer.OpJoinRandomRoom (OpJoinRandomRoomParams *opJoinRandomRoomParams* **)**   `[virtual]`

Operation to join a random, available room. Overloads take additional player properties. This is an async request which triggers a OnOperationResponse() call. If all rooms are closed or full, the OperationResponse will have a returnCode of ErrorCode.NoRandomMatchFound. If successful, the OperationResponse contains a gameserver address and the name of some room.

Returns

If the operation could be sent currently (requires connection).

8.43.3.9   virtual bool Photon.Realtime.LoadBalancingPeer.OpJoinRoom (EnterRoomParams *opParams* **)**   `[virtual]`

Joins a room by name or creates new room if room with given name not exists. The OperationResponse depends on the server the peer is connected to: Master will return a Game Server to connect to. Game Server will return the joined Room's data. This is an async request which triggers a OnOperationResponse() call.

If the room is not existing (anymore), the OperationResponse will have a returnCode of ErrorCode.GameDoes←
NotExist. Other possible ErrorCodes are: GameClosed, GameFull.

Returns

If the operation could be sent (requires connection).

8.43.3.10   virtual bool Photon.Realtime.LoadBalancingPeer.OpLeaveLobby ( )   `[virtual]`

Leaves the lobby on the Master Server. This is an async request which triggers a OnOperationResponse() call.

Returns

If the operation could be sent (requires connection).

8.43.3.11   virtual bool Photon.Realtime.LoadBalancingPeer.OpLeaveRoom (bool *becomeInactive,* **bool** *sendAuthCookie =*   `false` **)**   `[virtual]`

Leaves a room with option to come back later or "for good".

Parameters

becomeInactive	Async games can be re-joined (loaded) later on. Set to false, if you want to abandon a game entirely.

| *sendAuthCookie* | WebFlag: Securely transmit the encrypted object AuthCookie to the web service in Path↩ |
| | Leave webhook when available |

Returns

If the opteration can be send currently.

8.43.3.12 virtual bool Photon.Realtime.LoadBalancingPeer.OpRaiseEvent (byte *eventCode,* object *customEventContent,* RaiseEventOptions *raiseEventOptions,* SendOptions *sendOptions*) `[virtual]`

Send an event with custom code/type and any content to the other players in the same room.

This override explicitly uses another parameter order to not mix it up with the implementation for Hashtable only.

Parameters

eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
customEvent↩ *Content*	Any serializable datatype (including Hashtable like the other OpRaiseEvent overloads).
raiseEvent↩ *Options*	Contains (slightly) less often used options. If you pass null, the default options will be used.
sendOptions	Send options for reliable, encryption etc

Returns

If operation could be enqueued for sending. Sent when calling: Service or SendOutgoingCommands.

8.43.3.13 virtual bool Photon.Realtime.LoadBalancingPeer.OpSettings (bool *receiveLobbyStats*) `[virtual]`

Internally used operation to set some "per server" settings. This is for the Master Server.

Parameters

receiveLobby↩ *Stats*	Set to true, to get Lobby Statistics (lists of existing lobbies).

Returns

False if the operation could not be sent.

8.44 Photon.Pun.MonoBehaviourPun Class Reference

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

Inherits MonoBehaviour.

Inherited by Photon.Pun.MonoBehaviourPunCallbacks, Photon.Pun.UtilityScripts.MoveByKeys, Photon.Pun.↩ UtilityScripts.OnClickDestroy, and Photon.Pun.UtilityScripts.SmoothSyncMovement.

Properties

- PhotonView photonView `[get]`

    A cached reference to a PhotonView on this GameObject.

8.44.1 Detailed Description

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

8.44.2 Property Documentation

8.44.2.1 PhotonView Photon.Pun.MonoBehaviourPun.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.

8.45 Photon.Pun.MonoBehaviourPunCallbacks 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.Pun.MonoBehaviourPun, Photon.Realtime.IConnectionCallbacks, Photon.Realtime.IMatchmaking↩ Callbacks, Photon.Realtime.IInRoomCallbacks, and Photon.Realtime.ILobbyCallbacks.

Inherited by Photon.Pun.UtilityScripts.ConnectAndJoinRandom, Photon.Pun.UtilityScripts.CountdownTimer, Photon.Pun.UtilityScripts.PlayerNumbering, Photon.Pun.UtilityScripts.PunTeams, and Photon.Pun.UtilityScripts.↩ PunTurnManager.

Public Member Functions

- virtual void **OnEnable** ()
- virtual void **OnDisable** ()
- virtual void OnConnected ()

    Called to signal that the raw connection got established but before the client can call operation on the server.
- virtual void OnLeftRoom ()

    Called when the local user/client left a room, so the game's logic can clean up it's internal state.
- virtual void OnMasterClientSwitched (Player newMasterClient)

    Called after switching to a new MasterClient when the current one leaves.
- virtual void OnCreateRoomFailed (short returnCode, string message)

    Called when the server couldn't create a room (OpCreateRoom failed).
- virtual void OnJoinRoomFailed (short returnCode, string message)

    Called when a previous OpJoinRoom call failed on the server.
- 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 OnRoomListUpdate.
- virtual void OnLeftLobby ()

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

    Called after disconnecting from the Photon server. It could be a failure or intentional
- virtual void OnRegionListReceived (RegionHandler regionHandler)

    Called when the Name Server provided a list of regions for your title.
- virtual void OnRoomListUpdate (List< RoomInfo > roomList)

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

- virtual void OnJoinedRoom ()

    Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.
- virtual void OnPlayerEnteredRoom (Player newPlayer)

    Called when a remote player entered the room. This Player is already added to the playerlist.
- virtual void OnPlayerLeftRoom (Player otherPlayer)

    Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.
- virtual void OnJoinRandomFailed (short returnCode, string message)

    Called when a previous OpJoinRandom call failed on the server.
- virtual void OnConnectedToMaster ()

    Called when the client is connected to the Master Server and ready for matchmaking and other tasks.
- virtual void OnRoomPropertiesUpdate (Hashtable propertiesThatChanged)

    Called when a room's custom properties changed. The propertiesThatChanged contains all that was set via Room.↩
    SetCustomProperties.
- virtual void OnPlayerPropertiesUpdate (Player target, Hashtable changedProps)

    Called when custom player-properties are changed. Player and the changed properties are passed as object[].
- virtual void OnFriendListUpdate (List< FriendInfo > friendList)

    Called when the server sent the response to a FindFriends request.
- virtual void OnCustomAuthenticationResponse (Dictionary< string, object > data)

    Called when your Custom Authentication service responds with additional data.
- void OnCustomAuthenticationFailed (string debugMessage)

    Called when the custom authentication failed. Followed by disconnect!
- virtual void **OnWebRpcResponse** (OperationResponse response)
- virtual void OnLobbyStatisticsUpdate (List< TypedLobbyInfo > lobbyStatistics)

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

Additional Inherited Members

8.45.1 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 all callback interfaces and extends Photon.Pun.MonoBehaviourPun.

8.45.2 Member Function Documentation

8.45.2.1 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnConnected ( ) `[virtual]`

Called to signal that the raw connection got established but before the client can call operation on the server.

After the (low level transport) connection is established, the client will automatically send the Authentication operation, which needs to get a response before the client can call other operations.

Your logic should wait for either: OnRegionListReceived or OnConnectedToMaster.

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

This is not called for transitions from the masterserver to game servers.

Implements Photon.Realtime.IConnectionCallbacks.

8.45.2.2 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnConnectedToMaster ( ) `[virtual]`

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

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

Implements Photon.Realtime.IConnectionCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.45.2.3 virtual void Photon.Pun.MonoBehaviourPunCallbacks.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 OpCreateRoom).

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", implement OnMasterClientSwitched() and make each new MasterClient check the room's state.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.45.2.4 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnCreateRoomFailed (short *returnCode,* string *message*) `[virtual]`

Called when the server couldn't create a room (OpCreateRoom failed).

The most common cause to fail creating a room, is when a title relies on fixed room-names and the room already exists.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.45.2.5 void Photon.Pun.MonoBehaviourPunCallbacks.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.

Implements Photon.Realtime.IConnectionCallbacks.

8.45.2.6 virtual void Photon.Pun.MonoBehaviourPunCallbacks.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-us/realtime/current/reference/custom-authentication

Implements Photon.Realtime.IConnectionCallbacks.

8.45.2.7   virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnDisconnected (DisconnectCause *cause*)
        `[virtual]`

Called after disconnecting from the Photon server. It could be a failure or intentional

The reason for this disconnect is provided as DisconnectCause.

Implements Photon.Realtime.IConnectionCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.45.2.8   virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnFriendListUpdate (List< FriendInfo > *friendList*)
        `[virtual]`

Called when the server sent the response to a FindFriends request.

After calling OpFindFriends, the Master Server will cache the friend list and send updates to the friend list. The friends includes the name, userId, online state and the room (if any) for each requested user/friend.

Use the friendList to update your UI and store it, if the UI should highlight changes.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.45.2.9   virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnJoinedLobby ( ) `[virtual]`

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

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify in the public cloud). The room list gets available via OnRoomListUpdate.

Implements Photon.Realtime.ILobbyCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.45.2.10   virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnJoinedRoom ( ) `[virtual]`

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↩ CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom Property).

Implements Photon.Realtime.IMatchmakingCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PlayerNumbering, Photon.Pun.UtilityScripts.ConnectAndJoinRandom, and Photon.Pun.UtilityScripts.PunTeams.

8.45.2.11 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnJoinRandomFailed (short *returnCode,* string *message*) `[virtual]`

Called when a previous OpJoinRandom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

When using multiple lobbies (via OpJoinLobby or a TypedLobby parameter), another lobby might have more/fitting rooms.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.ConnectAndJoinRandom.

8.45.2.12 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnJoinRoomFailed (short *returnCode,* string *message*) `[virtual]`

Called when a previous OpJoinRoom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.45.2.13 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnLeftLobby ( ) `[virtual]`

Called after leaving a lobby.

When you leave a lobby, OpCreateRoom and OpJoinRandomRoom automatically refer to the default lobby.

Implements Photon.Realtime.ILobbyCallbacks.

8.45.2.14 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnLeftRoom ( ) `[virtual]`

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Implements Photon.Realtime.IMatchmakingCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.45.2.15 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnLobbyStatisticsUpdate (List< TypedLobbyInfo > ***lobbyStatistics*)** `[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 Photon.Realtime.ILobbyCallbacks.

8.45.2.16 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnMasterClientSwitched (Player *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 Photon.Realtime.IInRoomCallbacks.

8.45.2.17 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnPlayerEnteredRoom (Player *newPlayer*) `[virtual]`

Called when a remote player entered the room. This Player is already added to the playerlist.

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 Photon.Realtime.IInRoomCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.45.2.18 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnPlayerLeftRoom (Player *otherPlayer*) `[virtual]`

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

If another player leaves the room or if the server detects a lost connection, this callback will be used to notify your game logic.

Depending on the room's setup, players may become inactive, which means they may return and retake their spot in the room. In such cases, the Player stays in the Room.Players dictionary.

If the player is not just inactive, it gets removed from the Room.Players dictionary, before the callback is called.

Implements Photon.Realtime.IInRoomCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.45.2.19 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnPlayerPropertiesUpdate (Player *target,* Hashtable *changedProps*) `[virtual]`

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

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

Parameters

targetPlayer	Contains Player that changed.
changedProps	Contains the properties that changed.

Implements Photon.Realtime.IInRoomCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PlayerNumbering, and Photon.Pun.UtilityScripts.PunTeams.

8.45.2.20 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnRegionListReceived (RegionHandler *regionHandler*) `[virtual]`

Called when the Name Server provided a list of regions for your title.

Check the RegionHandler class description, to make use of the provided values.

Parameters

regionHandler	The currently used RegionHandler.

Implements Photon.Realtime.IConnectionCallbacks.

8.45.2.21 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnRoomListUpdate (List< RoomInfo > *roomList*)
`[virtual]`

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

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 Photon.Realtime.ILobbyCallbacks.

8.45.2.22 virtual void Photon.Pun.MonoBehaviourPunCallbacks.OnRoomPropertiesUpdate (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 Photon.Realtime.IInRoomCallbacks.

Reimplemented in Photon.Pun.UtilityScripts.PunTurnManager, and Photon.Pun.UtilityScripts.CountdownTimer.

8.46 Photon.Pun.UtilityScripts.MoveByKeys Class Reference

Very basic component to move a GameObject by WASD and Space.

Inherits Photon.Pun.MonoBehaviourPun.

Public Member Functions

- void **Start** ()
- void **FixedUpdate** ()

Public Attributes

- float **Speed** = 10f
- float **JumpForce** = 200f
- float **JumpTimeout** = 0.5f

Additional Inherited Members

8.46.1 Detailed Description

Very basic component to move a GameObject by WASD and Space.

Requires a [PhotonView](#). Disables itself on GameObjects that are not owned on Start.

Speed affects movement-speed. JumpForce defines how high the object "jumps". JumpTimeout defines after how many seconds you can jump again.

8.47 Photon.Pun.UtilityScripts.OnClickDestroy Class Reference

Implements OnClick to destroy the GameObject it's attached to. Optionally a RPC is sent to do this.

Inherits [Photon.Pun.MonoBehaviourPun](#).

Public Member Functions

- void **OnClick** ()
- IEnumerator **DestroyRpc** ()

Public Attributes

- bool **DestroyByRpc**

Additional Inherited Members

8.47.1 Detailed Description

Implements OnClick to destroy the GameObject it's attached to. Optionally a RPC is sent to do this.

Using an RPC to Destroy a GameObject allows any player to Destroy a GameObject. But it might cause errors. RPC and the Instantiated GameObject are not fully linked on the server. One might stick in the server witout the other.

A buffered RPC gets cleaned up when the sending player leaves the room. This means, the RPC gets lost.

Vice versus, a GameObject Instantiate might get cleaned up when the creating player leaves a room. This way, the GameObject that a RPC targets might become lost.

It makes sense to test those cases. Many are not breaking errors and you just have to be aware of them.

Gets OnClick() calls by InputToEvent class attached to a camera.

8.48 Photon.Pun.UtilityScripts.OnClickInstantiate Class Reference

This component will instantiate a network GameObject when in a room and the user click on that component's Gameobject Uses PhysicsRaycaster for positioning

Inherits MonoBehaviour, and IPointerClickHandler.

Public Attributes

- GameObject **Prefab**
- int **InstantiateType**
- bool **showGui**

8.48.1 Detailed Description

This component will instantiate a network GameObject when in a room and the user click on that component's Gameobject Uses PhysicsRaycaster for positioning

8.49 Photon.Pun.UtilityScripts.OnEscapeQuit Class Reference

This component will quit the application when escape key is pressed

Inherits MonoBehaviour.

Public Member Functions

- void **Update** ()

8.49.1 Detailed Description

This component will quit the application when escape key is pressed

8.50 Photon.Pun.UtilityScripts.OnJoinedInstantiate Class Reference

This component will instantiate a network GameObject when a room is joined

Inherits MonoBehaviour, Photon.Realtime.IConnectionCallbacks, Photon.Realtime.IMatchmakingCallbacks, and Photon.Realtime.ILobbyCallbacks.

Public Member Functions

- virtual void **OnEnable** ()
- virtual void **OnDisable** ()
- void OnJoinedRoom ()

    Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.
- void OnConnected ()

    Called to signal that the "low level connection" got established but before the client can call operation on the server.
- void OnCustomAuthenticationResponse (Dictionary< string, object > data)

    Called when your Custom Authentication service responds with additional data.
- void OnCustomAuthenticationFailed (string debugMessage)

    Called when the custom authentication failed. Followed by disconnect!
- void OnConnectedToMaster ()

    Called when the client is connected to the Master Server and ready for matchmaking and other tasks.
- void OnDisconnected (DisconnectCause cause)

    Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call
- void OnRegionListReceived (RegionHandler regionHandler)

    Called when the Name Server provided a list of regions for your title.
- void OnRoomListUpdate (List< RoomInfo > roomList)

    Called for any update of the room-listing while in a lobby (InLobby) on the Master Server.
- void OnFriendListUpdate (List< FriendInfo > friendList)

    Called when the server sent the response to a FindFriends request.
- void OnJoinedLobby ()

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

- void OnLeftLobby ()

    Called after leaving a lobby.

- void OnLobbyStatisticsUpdate (List< TypedLobbyInfo > lobbyStatistics)

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

- void OnCreatedRoom ()

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

- void OnCreateRoomFailed (short returnCode, string message)

    Called when the server couldn't create a room (OpCreateRoom failed).

- void OnJoinRoomFailed (short returnCode, string message)

    Called when a previous OpJoinRoom call failed on the server.

- void OnJoinRandomFailed (short returnCode, string message)

    Called when a previous OpJoinRandom call failed on the server.

- void OnLeftRoom ()

    Called when the local user/client left a room, so the game's logic can clean up it's internal state.

Public Attributes

- Transform **SpawnPosition**
- float **PositionOffset** = 2.0f
- GameObject[ ] **PrefabsToInstantiate**

8.50.1 Detailed Description

This component will instantiate a network GameObject when a room is joined

8.50.2 Member Function Documentation

8.50.2.1 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnConnected ( )

Called to signal that the "low level connection" got established but before the client can call operation on the server.

After the (low level transport) connection is established, the client will automatically send the Authentication operation, which needs to get a response before the client can call other operations.

Your logic should wait for either: OnRegionListReceived or OnConnectedToMaster.

This callback is useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnDisconnected(DisconnectCause cause) and check for the cause.

This is not called for transitions from the masterserver to game servers.

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.2 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnConnectedToMaster ( )

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

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

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.3 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.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 OpCreateRoom).

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", implement OnMasterClientSwitched() and make each new MasterClient check the room's state.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.4 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnCreateRoomFailed (short *returnCode,* string *message*)

Called when the server couldn't create a room (OpCreateRoom failed).

Creating a room may fail for various reasons. Most often, the room already exists (roomname in use) or the Room↩
Options clash and it's impossible to create the room.

When creating a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.5 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.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 develop-ment.

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.6 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.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-us/realtime/current/reference/custom-authentication

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.7 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnDisconnected (DisconnectCause *cause*)

Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call

The reason for this disconnect is provided as DisconnectCause.

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.8 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnFriendListUpdate (List< FriendInfo > *friendList*)

Called when the server sent the response to a FindFriends request.

After calling OpFindFriends, the Master Server will cache the friend list and send updates to the friend list. The friends includes the name, userId, online state and the room (if any) for each requested user/friend.

Use the friendList to update your UI and store it, if the UI should highlight changes.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.9 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnJoinedLobby ( )

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

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify in the public cloud). The room list gets available via OnRoomListUpdate.

Implements Photon.Realtime.ILobbyCallbacks.

8.50.2.10 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnJoinedRoom ( )

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↵ CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom Property).

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.11 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnJoinRandomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRandom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

This operation is only ever sent to the Master Server. Once a room is found by the Master Server, the client will head off to the designated Game Server and use the operation Join on the Game Server.

When using multiple lobbies (via OpJoinLobby or a TypedLobby parameter), another lobby might have more/fitting rooms.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.12 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnJoinRoomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRoom call failed on the server.

Joining a room may fail for various reasons. Most often, the room is full or does not exist anymore (due to someone else being faster or closing the room).

When joining a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.13 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnLeftLobby ( )

Called after leaving a lobby.

When you leave a lobby, OpCreateRoom and OpJoinRandomRoom automatically refer to the default lobby.

Implements Photon.Realtime.ILobbyCallbacks.

8.50.2.14 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnLeftRoom ( )

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.50.2.15 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnLobbyStatisticsUpdate (List< TypedLobbyInfo > *lobbyStatistics* **)**

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 Photon.Realtime.ILobbyCallbacks.

8.50.2.16 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnRegionListReceived (RegionHandler *regionHandler* **)**

Called when the Name Server provided a list of regions for your title.

Check the RegionHandler class description, to make use of the provided values.

Parameters

regionHandler	The currently used RegionHandler.

Implements Photon.Realtime.IConnectionCallbacks.

8.50.2.17 void Photon.Pun.UtilityScripts.OnJoinedInstantiate.OnRoomListUpdate (List< **RoomInfo** > *roomList* **)**

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

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 Photon.Realtime.ILobbyCallbacks.

8.51 Photon.Pun.UtilityScripts.OnPointerOverTooltip Class Reference

Set focus to a given photonView when pointed is over

Inherits MonoBehaviour, IPointerEnterHandler, and IPointerExitHandler.

8.51.1 Detailed Description

Set focus to a given photonView when pointed is over

8.52 Photon.Pun.UtilityScripts.OnStartDelete Class Reference

This component will destroy the GameObject it is attached to (in Start()).

Inherits MonoBehaviour.

8.52.1 Detailed Description

This component will destroy the GameObject it is attached to (in Start()).

8.53 Photon.Realtime.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 AuthenticateOnce = 231

  (231) Authenticates this peer and connects to a virtual application
- 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.
- const byte ServerSettings = 218

  (218) Operation to set some server settings. Used with different parameters on various servers.
- const byte GetGameList = 217

  (217) Get the game list matching a supplied sql filter (SqlListLobby only)

8.53.1 Detailed Description

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

8.53.2 Member Data Documentation

8.53.2.1 const byte Photon.Realtime.OperationCode.Authenticate = 230

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

8.53.2.2 const byte Photon.Realtime.OperationCode.AuthenticateOnce = 231

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

8.53.2.3 const byte Photon.Realtime.OperationCode.ChangeGroups = (byte)248

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

8.53.2.4 const byte Photon.Realtime.OperationCode.CreateGame = 227

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

8.53.2.5 const byte Photon.Realtime.OperationCode.FindFriends = 222

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

8.53.2.6 const byte Photon.Realtime.OperationCode.GetGameList = 217

(217) Get the game list matching a supplied sql filter (SqlListLobby only)

8.53.2.7 const byte Photon.Realtime.OperationCode.GetLobbyStats = 221

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

8.53.2.8 const byte Photon.Realtime.OperationCode.GetProperties = (byte)251

(251) Get Properties

8.53.2.9 const byte Photon.Realtime.OperationCode.GetRegions = 220

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

8.53.2.10   const byte Photon.Realtime.OperationCode.Join = 255

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

8.53.2.11   const byte Photon.Realtime.OperationCode.JoinGame = 226

(226) Join game (by name)

8.53.2.12   const byte Photon.Realtime.OperationCode.JoinLobby = 229

(229) Joins lobby (on master)

8.53.2.13   const byte Photon.Realtime.OperationCode.JoinRandomGame = 225

(225) Joins random game (on master)

8.53.2.14   const byte Photon.Realtime.OperationCode.Leave = (byte)254

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

8.53.2.15   const byte Photon.Realtime.OperationCode.LeaveLobby = 228

(228) Leaves lobby (on master)

8.53.2.16 const byte Photon.Realtime.OperationCode.RaiseEvent = (byte)253

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

8.53.2.17 const byte Photon.Realtime.OperationCode.ServerSettings = 218

(218) Operation to set some server settings. Used with different parameters on various servers.

8.53.2.18 const byte Photon.Realtime.OperationCode.SetProperties = (byte)252

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

8.53.2.19 const byte Photon.Realtime.OperationCode.WebRpc = 219

(219) WebRpc Operation.

8.54 Photon.Realtime.OpJoinRandomRoomParams Class Reference

Public Attributes

- Hashtable **ExpectedCustomRoomProperties**
- byte **ExpectedMaxPlayers**
- MatchmakingMode **MatchingType**
- TypedLobby **TypedLobby**
- string **SqlLobbyFilter**
- string[ ] **ExpectedUsers**

8.55 Photon.Realtime.ParameterCode Class Reference

Class for constants. Codes for parameters of Operations and Events.

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 Actor↩ Properties 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

  (204) Informs the server of the expected plugin setup.

- 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

- const byte ExpectedProtocol = 195

  (195) Protocol which will be used by client to connect master/game servers. Used for nameserver.

- const byte CustomInitData = 194

  (194) Set of custom parameters which are sent in auth request.

- const byte EncryptionMode = 193

  (193) How are we going to encrypt data.

- const byte EncryptionData = 192

  (192) Parameter of Authentication, which contains encryption keys (depends on AuthMode and EncryptionMode).

- const byte RoomOptionFlags = 191

  (191) An int parameter summarizing several boolean room-options with bit-flags.

8.55.1 Detailed Description

Class for constants. Codes for parameters of Operations and Events.

Pun uses these constants internally.

8.55.2 Member Data Documentation

8.55.2.1 const byte Photon.Realtime.ParameterCode.ActorList = (byte)252

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

8.55.2.2 const byte Photon.Realtime.ParameterCode.ActorNr = (byte)254

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

8.55.2.3 const byte Photon.Realtime.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.

8.55.2.4 const byte Photon.Realtime.ParameterCode.Address = 230

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

8.55.2.5 const byte Photon.Realtime.ParameterCode.ApplicationId = 224

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

8.55.2.6 const byte Photon.Realtime.ParameterCode.AppVersion = 220

(220) Version of your application

8.55.2.7 const byte Photon.Realtime.ParameterCode.AzureLocalNodeId = 209

(209) Internally used in case of hosting by Azure

8.55.2.8 const byte Photon.Realtime.ParameterCode.AzureMasterNodeId = 208

(208) Internally used in case of hosting by Azure

8.55.2.9 const byte Photon.Realtime.ParameterCode.AzureNodeInfo = 210

(210) Internally used in case of hosting by Azure

8.55.2.10 const byte Photon.Realtime.ParameterCode.Broadcast = (byte)250

(250) Code for broadcast parameter of OpSetProperties method.

8.55.2.11 const byte Photon.Realtime.ParameterCode.Cache = (byte)247

(247) Code for caching events while raising them.

8.55.2.12 const byte Photon.Realtime.ParameterCode.CacheSliceIndex = 205

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

8.55.2.13 const byte Photon.Realtime.ParameterCode.CheckUserOnJoin = (byte)232

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

8.55.2.14 const byte Photon.Realtime.ParameterCode.CleanupCacheOnLeave = (byte)241

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

8.55.2.15   const byte Photon.Realtime.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

8.55.2.16   const byte Photon.Realtime.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

8.55.2.17   const byte Photon.Realtime.ParameterCode.ClientAuthenticationType = 217

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

8.55.2.18   const byte Photon.Realtime.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.

8.55.2.19   const byte Photon.Realtime.ParameterCode.CustomEventContent = (byte)245

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

8.55.2.20   const byte Photon.Realtime.ParameterCode.CustomInitData = 194

(194) Set of custom parameters which are sent in auth request.

8.55.2.21   const byte Photon.Realtime.ParameterCode.Data = (byte)245

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

8.55.2.22   const byte Photon.Realtime.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.

8.55.2.23   const byte Photon.Realtime.ParameterCode.EncryptionData = 192

(192) Parameter of Authentication, which contains encryption keys (depends on AuthMode and EncryptionMode).

8.55.2.24   const byte Photon.Realtime.ParameterCode.EncryptionMode = 193

(193) How are we going to encrypt data.

8.55.2.25   const byte Photon.Realtime.ParameterCode.EventForward = 234

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

8.55.2.26   const byte Photon.Realtime.ParameterCode.ExpectedProtocol = 195

(195) Protocol which will be used by client to connect master/game servers. Used for nameserver.

8.55.2.27   const byte Photon.Realtime.ParameterCode.ExpectedValues = (byte)231

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

8.55.2.28   const byte Photon.Realtime.ParameterCode.FindFriendsRequestList = (byte)1

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

8.55.2.29   const byte Photon.Realtime.ParameterCode.FindFriendsResponseOnlineList = (byte)1

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

8.55.2.30   const byte Photon.Realtime.ParameterCode.FindFriendsResponseRoomIdList = (byte)2

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

8.55.2.31   const byte Photon.Realtime.ParameterCode.GameCount = 228

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

8.55.2.32   const byte Photon.Realtime.ParameterCode.GameList = 222

(222) List of RoomInfos about open / listed rooms

8.55.2.33   const byte Photon.Realtime.ParameterCode.GameProperties = (byte)248

(248) Code for property set (Hashtable).

8.55.2.34   const byte Photon.Realtime.ParameterCode.Group = 240

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

8.55.2.35   const byte Photon.Realtime.ParameterCode.Info = 218

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

8.55.2.36   const byte Photon.Realtime.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.

8.55.2.37   const byte Photon.Realtime.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.

8.55.2.38   const byte Photon.Realtime.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.

8.55.2.39   const byte Photon.Realtime.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).

8.55.2.40   const byte Photon.Realtime.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

8.55.2.41   const byte Photon.Realtime.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.

8.55.2.42   const byte Photon.Realtime.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.

8.55.2.43   const byte Photon.Realtime.ParameterCode.MasterPeerCount = 227

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

8.55.2.44   const byte Photon.Realtime.ParameterCode.MatchMakingType = 223

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

8.55.2.45   const byte Photon.Realtime.ParameterCode.NickName = 202

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

8.55.2.46   const byte Photon.Realtime.ParameterCode.PeerCount = 229

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

8.55.2.47   const byte Photon.Realtime.ParameterCode.PlayerProperties = (byte)249

(249) Code for property set (Hashtable).

8.55.2.48   const byte Photon.Realtime.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.

8.55.2.49   const byte Photon.Realtime.ParameterCode.PluginName = 201

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

8.55.2.50   const byte Photon.Realtime.ParameterCode.Plugins = 204

(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.

8.55.2.51   const byte Photon.Realtime.ParameterCode.PluginVersion = 200

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

8.55.2.52   const byte Photon.Realtime.ParameterCode.Position = 223

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

8.55.2.53   const byte Photon.Realtime.ParameterCode.Properties = (byte)251

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

8.55.2.54   const byte Photon.Realtime.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.

8.55.2.55   const byte Photon.Realtime.ParameterCode.ReceiverGroup = (byte)246

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

8.55.2.56   const byte Photon.Realtime.ParameterCode.Region = (byte)210

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

8.55.2.57   const byte Photon.Realtime.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.

8.55.2.58   const byte Photon.Realtime.ParameterCode.RoomName = (byte)255

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

8.55.2.59   const byte Photon.Realtime.ParameterCode.RoomOptionFlags = 191

(191) An int parameter summarizing several boolean room-options with bit-flags.

8.55.2.60   const byte Photon.Realtime.ParameterCode.Secret = 221

(221) Internally used to establish encryption

8.55.2.61   const byte Photon.Realtime.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).

8.55.2.62   const byte Photon.Realtime.ParameterCode.TargetActorNr = (byte)253

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

8.55.2.63   const byte Photon.Realtime.ParameterCode.UriPath = 209

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

8.55.2.64   const byte Photon.Realtime.ParameterCode.UserId = 225

(225) User's ID

8.55.2.65   const byte Photon.Realtime.ParameterCode.WebRpcParameters = 208

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

8.55.2.66   const byte Photon.Realtime.ParameterCode.WebRpcReturnCode = 207

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

8.55.2.67   const byte Photon.Realtime.ParameterCode.WebRpcReturnMessage = 206

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

8.56   Photon.Chat.ParameterCode Class Reference

Class for constants. Codes for parameters of Operations and Events.

Public Attributes

- const byte ApplicationId = 224

    (224) Your application's ID: a name on your own Photon or a GUID on the Photon Cloud
- const byte Secret = 221

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

(220) Version of your application

- 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 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 Region = 210

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

- const byte Address = 230

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

- const byte UserId = 225

    (225) User's ID

8.56.1 Detailed Description

Class for constants. Codes for parameters of Operations and Events.

8.56.2 Member Data Documentation

8.56.2.1 const byte Photon.Chat.ParameterCode.Address = 230

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

8.56.2.2 const byte Photon.Chat.ParameterCode.ApplicationId = 224

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

8.56.2.3 const byte Photon.Chat.ParameterCode.AppVersion = 220

(220) Version of your application

8.56.2.4 const byte Photon.Chat.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

8.56.2.5 const byte Photon.Chat.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

8.56.2.6 const byte Photon.Chat.ParameterCode.ClientAuthenticationType = 217

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

8.56.2.7   const byte Photon.Chat.ParameterCode.Region = 210

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

8.56.2.8   const byte Photon.Chat.ParameterCode.Secret = 221

(221) Internally used to establish encryption

8.56.2.9   const byte Photon.Chat.ParameterCode.UserId = 225

(225) User's ID

8.57   Photon.Pun.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, and Photon.Pun.IPunObservable.

Classes

- class SynchronizedLayer
- class SynchronizedParameter

Public Types

- enum **ParameterType**
- enum **SynchronizeType**

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

- 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.

8.57.1 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.

8.57.2 Member Function Documentation

8.57.2.1 void Photon.Pun.PhotonAnimatorView.CacheDiscreteTriggers ( )

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

8.57.2.2 bool Photon.Pun.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

8.57.2.3 bool Photon.Pun.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

8.57.2.4 SynchronizeType Photon.Pun.PhotonAnimatorView.GetLayerSynchronizeType (int *layerIndex*)

Gets the type how the layer is synchronized

Parameters

layerIndex	Index of the layer.

Returns

Disabled/Discrete/Continuous

8.57.2.5   SynchronizeType Photon.Pun.PhotonAnimatorView.GetParameterSynchronizeType (string *name*)

Gets the type how the parameter is synchronized

Parameters

name	The name of the parameter.

Returns

Disabled/Discrete/Continuous

8.57.2.6 List<**SynchronizedLayer**> **Photon.Pun.PhotonAnimatorView.GetSynchronizedLayers ( )**

Get a list of all synchronized layers

Returns

List of SynchronizedLayer objects

8.57.2.7 List<**SynchronizedParameter**> **Photon.Pun.PhotonAnimatorView.GetSynchronizedParameters ( )**

Get a list of all synchronized parameters

Returns

List of SynchronizedParameter objects

8.57.2.8 void Photon.Pun.PhotonAnimatorView.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.SerializationRate 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 Photon←
View.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 Photon.Pun.IPunObservable.

8.57.2.9 void Photon.Pun.PhotonAnimatorView.SetLayerSynchronized (int *layerIndex,* SynchronizeType *synchronizeType*)

Sets the how a layer should be synchronized

Parameters

layerIndex	Index of the layer.
synchronizeType	Disabled/Discrete/Continuous

8.57.2.10 void Photon.Pun.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

8.58 Photon.Pun.UtilityScripts.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).

8.58.1 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.

8.58.2 Member Data Documentation

8.58.2.1 bool Photon.Pun.UtilityScripts.PhotonLagSimulationGui.Visible = true

Shows or hides GUI (does not affect settings).

8.58.2.2 int Photon.Pun.UtilityScripts.PhotonLagSimulationGui.WindowId = 101

Unity GUI Window ID (must be unique or will cause issues).

8.58.2.3 Rect Photon.Pun.UtilityScripts.PhotonLagSimulationGui.WindowRect = new Rect(0, 100, 120, 100)

Positioning rect for window.

8.58.3 Property Documentation

8.58.3.1 PhotonPeer Photon.Pun.UtilityScripts.PhotonLagSimulationGui.Peer `[get]`,`[set]`

The peer currently in use (to set the network simulation).

8.59 Photon.Pun.PhotonMessageInfo Struct Reference

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

Public Member Functions

- **PhotonMessageInfo** (Player player, int timestamp, PhotonView view)
- override string **ToString** ()

Public Attributes

- readonly Player Sender

    The sender of a message / event. May be null.
- readonly PhotonView **photonView**

Properties

- double **timestamp** `[get]`

8.59.1 Detailed Description

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

8.59.2 Member Data Documentation

8.59.2.1 readonly Player Photon.Pun.PhotonMessageInfo.Sender

The sender of a message / event. May be null.

8.60 Photon.Pun.PhotonNetwork Class Reference

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

Static Public Member Functions

- static bool ConnectUsingSettings ()

  Connect to Photon as configured in the PhotonServerSettings file.
- static bool ConnectToMaster (string masterServerAddress, int port, string appID)

  Connect to a Photon Master Server by address, port, appID.
- static bool ConnectToBestCloudServer ()

  Connect to the Photon Cloud region with the lowest ping (on platforms that support Unity's Ping).
- static bool ConnectToRegion (string region)

  Connects to the Photon Cloud region of choice.
- static void Disconnect ()

  Makes this client disconnect from the photon server, a process that leaves any room and calls OnDisconnected on completion.
- static bool Reconnect ()

  Can be used to reconnect to the master server after a disconnect.
- 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 int GetPing ()

  The current roundtrip time to the photon server.
- static void FetchServerTimestamp ()

  Refreshes the server timestamp (async operation, takes a roundtrip).
- static void SendAllOutgoingCommands ()

  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 (Player kickPlayer)

  Request a client to disconnect (KICK). Only the master client can do this
- static bool SetMasterClient (Player masterClientPlayer)

  Asks the server to assign another player as Master Client of your current room.
- static bool JoinRandomRoom ()

  Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.
- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)

  Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.
- static bool JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType, TypedLobby typedLobby, string sqlLobbyFilter, string[ ] expected↩ Users=null)

  Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.
- static bool CreateRoom (string roomName, RoomOptions roomOptions=null, TypedLobby typedLobby=null, string[ ] expectedUsers=null)

  Creates a new room. Will callback: OnCreatedRoom and OnJoinedRoom or OnCreateRoomFailed.
- static bool JoinOrCreateRoom (string roomName, RoomOptions roomOptions, TypedLobby typedLobby, string[ ] expectedUsers=null)

  Joins a specific room by name and creates it on demand. Will callback: OnJoinedRoom or OnJoinRoomFailed.
- static bool JoinRoom (string roomName, string[ ] expectedUsers=null)

  Joins a room by name. Will callback: OnJoinedRoom or OnJoinRoomFailed.
- static bool RejoinRoom (string roomName)

  Rejoins a room by roomName (using the userID internally to return). Will callback: OnJoinedRoom or OnJoinRoom↩ Failed.
- static bool ReconnectAndRejoin ()

  When the client lost connection during gameplay, this method attempts to reconnect and rejoin the room.
- static bool LeaveRoom (bool becomeInactive=true)

  Leave the current room and return to the Master Server where you can join or create rooms (see remarks).

- 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 FindFriends (string[ ] friendsToFind)

    Requests the rooms and online status for a list of friends and saves the result in PhotonNetwork.Friends.

- static bool GetCustomRoomList (TypedLobby typedLobby, string sqlLobbyFilter)

    Fetches a custom list of games from the server, matching a SQL-like "where" clause, then triggers OnRoomListUpdate callback.

- 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, RaiseEventOptions raiseEventOptions, Send← Options sendOptions)

    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, byte group=0, object[ ] data=null)

    Instantiate a prefab over the network.

- static GameObject InstantiateSceneObject (string prefabName, Vector3 position, Quaternion rotation, byte group=0, object[ ] data=null)

    Instantiate a scene-owned prefab over the network.

- 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 (Player 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 (Player 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 HashSet< GameObject > FindGameObjectsWithComponent (Type type)

    Finds the GameObjects with Components of a specific type (using FindObjectsOfType).

- static void SetInterestGroups (byte group, bool enabled)

  Enable/disable receiving events from a given Interest Group.

- static void LoadLevel (int levelNumber)

  Wraps loading a level to pause the network message-queue. Optionally syncs the loaded level in a room.

- static void LoadLevel (string levelName)

  Wraps loading a level to pause the network message-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.

- static object[ ] **FetchInstantiationData** (int instantiationId)

- static void DestroyPlayerObjects (int playerId, bool localOnly)

  Destroys all Instantiates and RPCs locally and (if not localOnly) sends EvDestroy(player) and clears related events in the server buffer.

- static void **DestroyAll** (bool localOnly)

- static bool **LocalCleanPhotonView** (PhotonView view)

- static PhotonView **GetPhotonView** (int viewID)

- static void **RegisterPhotonView** (PhotonView netView)

- static void OpCleanRpcBuffer (int actorNumber)

  Removes the RPCs of someone else (to be used as master). This won't clean any local caches. It just tells the server to forget a player's RPCs and instantiates.

- static void OpRemoveCompleteCacheOfPlayer (int actorNumber)

  Instead removing RPCs or Instantiates, this removed everything cached by the actor.

- static void **OpRemoveCompleteCache** ()

- static void **CleanRpcBufferIfMine** (PhotonView view)

- static void OpCleanRpcBuffer (PhotonView view)

  Cleans server RPCs for PhotonView (without any further checks).

- static void RemoveRPCsInGroup (int group)

  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 SetLevelPrefix (short prefix)

  Sets level prefix for PhotonViews instantiated later on. Don't set it if you need only one!

- static void SetInterestGroups (byte[ ] disableGroups, byte[ ] enableGroups)

  Enable/disable receiving on given Interest Groups (applied to PhotonViews).

- static void SetSendingEnabled (byte group, bool enabled)

  Enable/disable sending on given group (applied to PhotonViews)

- static void SetSendingEnabled (byte[ ] disableGroups, byte[ ] enableGroups)

  Enable/disable sending on given groups (applied to PhotonViews)

Public Attributes

- const string PunVersion = "2.0"

  Version number of PUN. Used in the AppVersion, which separates your playerbase in matchmaking.

- const int **SyncViewId** = 0

- const int **SyncCompressed** = 1

- const int **SyncNullValues** = 2

- const int **SyncFirstValue** = 3

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.Server↩ SettingsFileName, typeof(ServerSettings))

    Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.

- static ConnectMethod ConnectMethod = ConnectMethod.NotCalled

    Tracks, which Connect method was called last.

- static bool InstantiateInRoomOnly = true

    If true, Instantiate methods will check if you are in a room and fail if you are not.

- static PunLogLevel LogLevel = PunLogLevel.ErrorsOnly

    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 float BackgroundTimeout = 60.0f

    Defines how many seconds PUN keeps the connection, after Unity's OnApplicationPause(true) call. Default: 60 seconds.

- static bool **UsePrefabCache** = true
- static Dictionary< string, GameObject > **PrefabCache** = new Dictionary<string, GameObject>()
- static bool UseRpcMonoBehaviourCache

    While enabled, the MonoBehaviours on which we call RPCs are cached, avoiding costly GetComponents<Mono↩ Behaviour>() calls.

- static int ObjectsInOneUpdate = 10

    Defines how many OnPhotonSerialize()-calls might get summarized in one message.

Properties

- static string GameVersion `[get, set]`

    Version number of your game. Setting this updates the AppVersion, which separates your playerbase in matchmaking.

- static string AppVersion `[get]`

    Sent to Photon Server to specifiy the "Virtual AppId".

- static string ServerAddress `[get]`

    Currently used server address (no matter if master or game server).

- static string CloudRegion `[get]`

    Currently used Cloud Region (if any). As long as the client is not on a Master Server or Game Server, the region is not yet defined.

- static string BestRegionSummaryInPreferences `[get, set]`

    Used to store and access the "Best Region Summary" in the Player Preferences.

- static bool IsConnected `[get]`

    False until you connected to Photon initially. True in offline mode, while connected to any server and even while switching servers.

- static bool IsConnectedAndReady `[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 ClientState NetworkClientState `[get]`

Directly provides the network-level client state, unless in OfflineMode.

- 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.

- static TypedLobby **CurrentLobby** `[get, set]`

  The lobby that will be used when PUN joins a lobby or creates a game.

- static Room **CurrentRoom** `[get]`

  Get the room we're currently in (also when in OfflineMode). Null if we aren't in any room.

- static Player **LocalPlayer** `[get]`

  This client's Player instance is always available, unless the app shuts down.

- static string **NickName** `[get, set]`

  Set to synchronize the player's nickname with everyone in the room(s) you enter. This sets PhotonNetwork.player.↵ NickName.

- static Player[] **PlayerList** `[get]`
- static Player[] **PlayerListOthers** `[get]`
- static bool **OfflineMode** `[get, set]`

  Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on Photon↵ Network 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 **EnableLobbyStatistics** `[get]`

  If enabled, the client will get a list of available lobbies from the Master Server.

- static bool **InLobby** `[get]`

  True while this client is in a lobby.

- 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 'SerializationRate'.

- static int **SerializationRate** `[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 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 Player **MasterClient** `[get]`

  The Master Client of the current room or null (outside of rooms).

- static bool **InRoom** `[get]`

  Is true while being in a room (NetworkClientState == ClientState.Joined).

- 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 PhotonNetwork.↵ PlayerList.Length or PhotonNetwork.CurrentRoom.PlayerCount 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.
- static bool UseAlternativeUdpPorts `[get, set]`

    Switch to alternative ports for a UDP connection to the Public Cloud.
- static PhotonView[] PhotonViews `[get]`

    Gets the photon views.
- 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 float LevelLoadingProgress `[get]`

    Gets the networked level loading progress. Value will be be zero until the first loading, and remain at one in between loadings Use PhotonNetwork.LoadLevel() to initiate a networked level Loading

8.60.1 Detailed Description

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

8.60.2 Member Function Documentation

8.60.2.1 static int Photon.Pun.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.

8.60.2.2 static int Photon.Pun.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.

8.60.2.3 static bool Photon.Pun.PhotonNetwork.CloseConnection (Player *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 Player to kick.

8.60.2.4   static bool Photon.Pun.PhotonNetwork.ConnectToBestCloudServer ( ) `[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↩ ://dashboard.photonengine.com`

Connecting to the Photon Cloud might fail due to:

- Invalid AppId

- Network issues

- Invalid region

- Subscription CCU limit reached

- etc.

In general check out the DisconnectCause from the IConnectionCallbacks.OnDisconnected callback.

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.

8.60.2.5   static bool Photon.Pun.PhotonNetwork.ConnectToMaster (string *masterServerAddress,* int *port,* string *appID*) `[static]`

Connect to a Photon Master Server by address, port, appID.

To connect to the Photon Cloud, a valid AppId must be in the settings file (shown in the Photon Cloud Dashboard). `https://dashboard.photonengine.com`

Connecting to the Photon Cloud might fail due to:

- Invalid AppId

- Network issues

- Invalid region

- Subscription CCU limit reached

- etc.

In general check out the DisconnectCause from the IConnectionCallbacks.OnDisconnected callback.

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).

8.60.2.6   static bool Photon.Pun.PhotonNetwork.ConnectToRegion (string *region*) `[static]`

Connects to the Photon Cloud region of choice.

8.60.2.7   static bool Photon.Pun.PhotonNetwork.ConnectUsingSettings ( ) `[static]`

Connect to Photon as configured in the PhotonServerSettings file.

Implement IConnectionCallbacks, to make your game logic aware of state changes. Especially, IConnection↩Callbacks.ConnectedToMasterserver is useful to react when the client can do matchmaking.

This method will disable OfflineMode (which won't destroy any instantiated GOs) and it will set IsMessageQueue↩Running to true.

Your Photon configuration is created by the PUN Wizard and contains the AppId, region for Photon Cloud games, the server address among other things.

To ignore the settings file, set the relevant values and connect by calling ConnectToMaster, ConnectToRegion.

To connect to the Photon Cloud, a valid AppId must be in the settings file (shown in the Photon Cloud Dashboard). `https://dashboard.photonengine.com`

Connecting to the Photon Cloud might fail due to:

- Invalid AppId

- Network issues

- Invalid region

- Subscription CCU limit reached

- etc.

In general check out the DisconnectCause from the IConnectionCallbacks.OnDisconnected callback.

**8.60.2.8   static bool Photon.Pun.PhotonNetwork.CreateRoom (string *roomName,* RoomOptions *roomOptions =* `null,`   TypedLobby *typedLobby =* `null,` string[ ] *expectedUsers =* `null )` `[static]`

Creates a new room. Will callback: OnCreatedRoom and OnJoinedRoom or OnCreateRoomFailed.

When successful, this calls the callbacks OnCreatedRoom and OnJoinedRoom (the latter, cause you join as first player). In all error cases, OnCreateRoomFailed gets called.

Creating a room will fail if the room name is already in use or when the RoomOptions clashing with one another. Check the EnterRoomParams reference for the various room creation options.

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).

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

More about PUN matchmaking: `https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat`

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.

8.60.2.9   static void Photon.Pun.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.

8.60.2.10   static void Photon.Pun.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.

8.60.2.11   static void Photon.Pun.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.

8.60.2.12   static void Photon.Pun.PhotonNetwork.DestroyPlayerObjects (int *playerId,* bool *localOnly*) `[static]`

Destroys all Instantiates and RPCs locally and (if not localOnly) sends EvDestroy(player) and clears related events in the server buffer.

8.60.2.13   static void Photon.Pun.PhotonNetwork.DestroyPlayerObjects (Player *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.

8.60.2.14   static void Photon.Pun.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.

8.60.2.15   static void Photon.Pun.PhotonNetwork.Disconnect ( ) `[static]`

Makes this client disconnect from the photon server, a process that leaves any room and calls OnDisconnected 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 OnDisconnected are immediate. Offline mode is set to false as well. Once disconnected, the client can connect again. Use ConnectUsingSettings.

8.60.2.16   static void Photon.Pun.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.

8.60.2.17   static bool Photon.Pun.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 OpFind↩Friends (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 userId in the PhotonNetwork.AuthValues.   See remarks of AuthenticationValues for info about how this is set and used.

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): Parameter↩Code.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 NickName or AuthValues).

Returns

If the operation could be sent (requires connection, only one request is allowed at any time). Always false in offline mode.

8.60.2.18 static HashSet<GameObject> Photon.Pun.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.

8.60.2.19 static bool Photon.Pun.PhotonNetwork.GetCustomRoomList (TypedLobby *typedLobby,* string *sqlLobbyFilter*) `[static]`

Fetches a custom list of games from the server, matching a SQL-like "where" clause, then triggers OnRoomList↩ Update callback.

Operation is only available for lobbies of type SqlLobby. Note: You don't have to join that lobby. This is an async request.

When done, OnRoomListUpdate gets called. Use GetRoomList() to access it.

https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/matchmaking-and-lobby\::sql_lobby_type

Parameters

typedLobby	The lobby to query. Has to be of type SqlLobby.
sqlLobbyFilter	The sql query statement.

Returns

If the operation could be sent (has to be connected).

8.60.2.20 static int Photon.Pun.PhotonNetwork.GetPing ( ) `[static]`

The current roundtrip time to the photon server.

Returns

Roundtrip time (to server and back).

8.60.2.21 static GameObject Photon.Pun.PhotonNetwork.Instantiate (string *prefabName,* Vector3 *position,* Quaternion *rotation,* byte *group =* `0`*,* object[ ] *data =* `null`) `[static]`

Instantiate a prefab over the network.

By default, a 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.

Parameters

prefabName	Name of the prefab to instantiate.
position	Position Vector3 to apply on instantiation.
rotation	Rotation Quaternion to apply on instantiation.
group	Optional Interest Group value 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.

**8.60.2.22   static GameObject Photon.Pun.PhotonNetwork.InstantiateSceneObject (string *prefabName,* Vector3 *position,*
Quaternion *rotation,* byte *group =* `0`*,* object[ ] *data =* `null`) [static]**

Instantiate a scene-owned prefab over the network.

The networked Game Object will be controllable by the MasterClient (via PhotonView). Only the master client can Instantiate scene objects.

By default, a 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.

Parameters

prefabName	Name of the prefab to instantiate.
position	Position Vector3 to apply on instantiation.
rotation	Rotation Quaternion to apply on instantiation.
group	Optional Interest Group value 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.

8.60.2.23   static bool Photon.Pun.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: `https://doc.photonengine.`↵
`com/en-us/pun/v2/lobby-and-matchmaking/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!

8.60.2.24   static bool Photon.Pun.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: `https://doc.photonengine.↵com/en-us/realtime/current/reference/matchmaking-and-lobby`

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!

Parameters

typedLobby	A typed lobby to join (must have name and type).

8.60.2.25 **static bool Photon.Pun.PhotonNetwork.JoinOrCreateRoom (string** *roomName,* **RoomOptions** *roomOptions,* **TypedLobby** *typedLobby,* **string[ ]** *expectedUsers =* `null` **)** `[static]`

Joins a specific room by name and creates it on demand. Will callback: OnJoinedRoom or OnJoinRoomFailed.

Useful when players make up a room name to meet in: All involved clients call the same method and whoever is first, also creates the room.

When successful, the client will enter the specified room. The client which creates the room, will callback both OnCreatedRoom and OnJoinedRoom. Clients that join an existing room will only callback OnJoinedRoom. In all error cases, OnJoinRoomFailed gets called.

Joining a room will fail, if the room is full, closed or when the user already is present in the room (checked by userId).

To return to a room, use OpRejoinRoom.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

If you set room properties in roomOptions, they get ignored when the room is existing already. This avoids changing the room properties by late joining players.

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.

More about PUN matchmaking: `https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat`

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.

8.60.2.26 static bool Photon.Pun.PhotonNetwork.JoinRandomRoom ( ) `[static]`

Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.

Used for random matchmaking. You can join any room or one with specific properties defined in opJoinRandom←
RoomParams.

This operation fails if no rooms are fitting or available (all full, closed, in another lobby or not visible). It may also fail when actually joining the room which was found. Rooms may close, become full or empty anytime.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

More about PUN matchmaking: https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat

8.60.2.27 static bool Photon.Pun.PhotonNetwork.JoinRandomRoom (Hashtable *expectedCustomRoomProperties,* byte *expectedMaxPlayers*) `[static]`

Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.

Used for random matchmaking. You can join any room or one with specific properties defined in opJoinRandom←
RoomParams.

This operation fails if no rooms are fitting or available (all full, closed, in another lobby or not visible). It may also fail when actually joining the room which was found. Rooms may close, become full or empty anytime.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

More about PUN matchmaking: https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat

Parameters

expected← *CustomRoom←* *Properties*	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
expectedMax← *Players*	Filters for a particular maxplayer setting. Use 0 to accept any maxPlayer value.

Returns

> If the operation got queued and will be sent.

8.60.2.28 static bool Photon.Pun.PhotonNetwork.JoinRandomRoom (Hashtable *expectedCustomRoomProperties,* byte *expectedMaxPlayers,* MatchmakingMode *matchingType,* TypedLobby *typedLobby,* string *sqlLobbyFilter,* string[ ] *expectedUsers =* ` null `) `[static]`

Joins a random room that matches the filter. Will callback: OnJoinedRoom or OnJoinRandomFailed.

Used for random matchmaking. You can join any room or one with specific properties defined in opJoinRandom←
RoomParams.

This operation fails if no rooms are fitting or available (all full, closed, in another lobby or not visible). It may also fail when actually joining the room which was found. Rooms may close, become full or empty anytime.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

More about PUN matchmaking: `https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat`

Parameters

expected↩ *CustomRoom*↩ *Properties*	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
expectedMax↩ *Players*	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.

8.60.2.29 static bool Photon.Pun.PhotonNetwork.JoinRoom (string *roomName,* string[ ] *expectedUsers =* null)
         `[static]`

Joins a room by name. Will callback: OnJoinedRoom or OnJoinRoomFailed.

Useful when using lobbies or when players follow friends or invite each other.

When successful, the client will enter the specified room and callback via OnJoinedRoom. In all error cases, On↩ JoinRoomFailed gets called.

Joining a room will fail if the room is full, closed, not existing or when the user already is present in the room (checked by userId).

To return to a room, use OpRejoinRoom. When players invite each other and it's unclear who's first to respond, use OpJoinOrCreateRoom instead.

This method can only be called while the client is connected to a Master Server so you should implement the callback OnConnectedToMaster. Check the return value to make sure the operation will be called on the server. Note: There will be no callbacks if this method returned false.

More about PUN matchmaking: `https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/mat`

OnJoinRoomFailed 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.

8.60.2.30   static bool Photon.Pun.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.

8.60.2.31   static bool Photon.Pun.PhotonNetwork.LeaveRoom (bool *becomeInactive =* `true`) `[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.

In a room with playerTTL < 0, LeaveRoom just turns a client inactive. The player stays in the room's player list and can return later on. Setting becomeInactive to false deliberately, means to "abandon" the room, despite the playerTTL allowing you to come back.

In a room with playerTTL == 0, become inactive has no effect (clients are removed from the room right away).

Parameters

becomeInactive	If this client becomes inactive in a room with playerTTL < 0. Defaults to true.

8.60.2.32   static void Photon.Pun.PhotonNetwork.LoadLevel (int *levelNumber*) `[static]`

Wraps loading a level to pause the network message-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 Game↩ Objects and PhotonViews). You can call this in OnJoinedRoom.

This uses SceneManager.LoadSceneAsync().

Check the progress of the LevelLoading using [PhotonNetwork.LevelLoadingProgress](#) (-1 means no loading, then it ranges from 0 to 1)

Parameters

levelNumber	Number of the level to load. When using level numbers, make sure they are identical on all clients.

8.60.2.33   static void Photon.Pun.PhotonNetwork.LoadLevel (string *levelName*) `[static]`

Wraps loading a level to pause the network message-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 Game↩
Objects and PhotonViews). You can call this in OnJoinedRoom.

This uses SceneManager.LoadSceneAsync().

Check the progress of the LevelLoading using PhotonNetwork.LevelLoadingProgress (-1 means no loading, then it
ranges from 0 to 1)

Parameters

levelName	Name of the level to load. Make sure it's available to all clients in the same room.

8.60.2.34 static void Photon.Pun.PhotonNetwork.NetworkStatisticsReset ( ) `[static]`

Resets the traffic stats and re-enables them.

8.60.2.35 static string Photon.Pun.PhotonNetwork.NetworkStatisticsToString ( ) `[static]`

Only available when NetworkStatisticsEnabled was used to gather some stats.

Returns

A string with vital networking statistics.

8.60.2.36 static void Photon.Pun.PhotonNetwork.OpCleanRpcBuffer (int *actorNumber*) `[static]`

Removes the RPCs of someone else (to be used as master). This won't clean any local caches. It just tells the
server to forget a player's RPCs and instantiates.

Parameters

actorNumber	

8.60.2.37 static void Photon.Pun.PhotonNetwork.OpCleanRpcBuffer (PhotonView *view*) `[static]`

Cleans server RPCs for PhotonView (without any further checks).

8.60.2.38 static void Photon.Pun.PhotonNetwork.OpRemoveCompleteCacheOfPlayer (int *actorNumber*) `[static]`

Instead removing RPCs or Instantiates, this removed everything cached by the actor.

Parameters

actorNumber	

8.60.2.39 static bool Photon.Pun.PhotonNetwork.RaiseEvent (byte *eventCode,* object *eventContent,* RaiseEventOptions **
        raiseEventOptions, SendOptions *sendOptions*)** `[static]`

Sends fully customizable events in a room. Events consist of at least an EventCode (0..199) and can have content.

To receive events, implement IOnEventCallback in any class and register it via PhotonNetwork.AddCallbackTarget.
See IOnEventCallback.OnEvent.

The eventContent is optional. If set, eventContent 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 are not supported.

You can turn a class into 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 Player.ID values), the receivers parameter gets ignored. When using event caching, the targetActors, receivers and interest↩ Group 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.
raiseEvent↩ Options	Allows more complex usage of events. If null, RaiseEventOptions.Default will be used (which is fine).
sendOptions	Send options for reliable, encryption etc..

Returns

> False if event could not be sent.

8.60.2.40 static bool Photon.Pun.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 RejoinRoom(roomname) to return to a game. Common use case: Press the Lock Button on a iOS device and you get disconnected immediately.

8.60.2.41 static bool Photon.Pun.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 OnJoinRoomFailed 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 Reconnect↩ AndRejoin returns false, you can still attempt a Reconnect and Rejoin.

Similar to PhotonNetwork.RejoinRoom, 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 Reconnect↩ AndRejoin.

8.60.2.42 static bool Photon.Pun.PhotonNetwork.RejoinRoom (string *roomName*) `[static]`

Rejoins a room by roomName (using the userID internally to return). Will callback: OnJoinedRoom or OnJoin↩ RoomFailed.

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 RejoinRoom(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, if you use PhotonViews. 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.

8.60.2.43   static void Photon.Pun.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.

8.60.2.44   static void Photon.Pun.PhotonNetwork.RemoveRPCs (Player *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 Player'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.

8.60.2.45   static void Photon.Pun.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.

8.60.2.46   static void Photon.Pun.PhotonNetwork.RemoveRPCsInGroup (int *group*) `[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

group	Interest group that gets all RPCs removed.

8.60.2.47   static void Photon.Pun.PhotonNetwork.SendAllOutgoingCommands ( ) `[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.

8.60.2.48   static void Photon.Pun.PhotonNetwork.SetInterestGroups (byte[ ] *disableGroups,* byte[ ] *enableGroups*) `[static]`

Enable/disable receiving on given Interest Groups (applied to PhotonViews).

A client can tell the server which Interest Groups it's interested in. The server will only forward events for those Interest Groups to that client (saving bandwidth and performance).

See: `https://doc.photonengine.com/en-us/pun/v2/gameplay/interestgroups`

See: `https://doc.photonengine.com/en-us/pun/v2/demos-and-tutorials/package-demos/culling-`
Parameters

disableGroups	The interest groups to disable (or null).
enableGroups	The interest groups to enable (or null).

8.60.2.49   static void Photon.Pun.PhotonNetwork.SetInterestGroups (byte *group,* bool *enabled*) `[static]`

Enable/disable receiving events from a given Interest Group.

A client can tell the server which Interest Groups it's interested in. The server will only forward events for those Interest Groups to that client (saving bandwidth and performance).

See: `https://doc.photonengine.com/en-us/pun/v2/gameplay/interestgroups`

See: `https://doc.photonengine.com/en-us/pun/v2/demos-and-tutorials/package-demos/culling-`
Parameters

group	The interest group to affect.
enabled	Sets if receiving from group to enabled (or not).

8.60.2.50   static void Photon.Pun.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

8.60.2.51   static bool Photon.Pun.PhotonNetwork.SetMasterClient (Player *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).

8.60.2.52   static void Photon.Pun.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.

8.60.2.53   static void Photon.Pun.PhotonNetwork.SetSendingEnabled (byte *group,* bool *enabled*)   `[static]`

Enable/disable sending on given group (applied to PhotonViews)

This does not interact with the Photon server-side. It's just a client-side setting to suppress updates, should they be sent to one of the blocked groups.

This setting is not particularly useful, as it means that updates literally never reach the server or anyone else. Use with care.

Parameters

group	The interest group to affect.
enabled	Sets if sending to group is enabled (or not).

8.60.2.54   static void Photon.Pun.PhotonNetwork.SetSendingEnabled (byte[ ] *disableGroups,* byte[ ] *enableGroups*)   `[static]`

Enable/disable sending on given groups (applied to PhotonViews)

This does not interact with the Photon server-side. It's just a client-side setting to suppress updates, should they be sent to one of the blocked groups.

This setting is not particularly useful, as it means that updates literally never reach the server or anyone else. Use with care.

Parameters

enableGroups	The interest groups to enable sending on (or null).
disableGroups	The interest groups to disable sending on (or null).

8.60.2.55   static void Photon.Pun.PhotonNetwork.UnAllocateViewID (int *viewID*)   `[static]`

Unregister a viewID (of manually instantiated and destroyed networked objects).

Parameters

viewID	A viewID manually allocated by this player.

8.60.2.56   static bool Photon.Pun.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. https://doc.↵ photonengine.com/en-us/pun/v2/gameplay/web-extensions/webrpc The Parameters will be converted into JSon format, so make sure your parameters are compatible.

See Photon.Realtime.IWebRpcCallback.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
}
```

8.60.3 Member Data Documentation

8.60.3.1 float Photon.Pun.PhotonNetwork.BackgroundTimeout = 60.0f `[static]`

Defines how many seconds PUN keeps the connection, after Unity's OnApplicationPause(true) call. Default: 60 seconds.

It's best practice to disconnect inactive apps/connections after a while but to also allow users to take calls, etc.. We think a reasonable backgroung timeout is 60 seconds.

To handle the timeout, implement: OnDisconnected(), as usual. Your application will "notice" the background disconnect when it becomes active again (running the Update() loop).

If you need to separate this case from others, you need to track if the app was in the background (there is no special callback by PUN).

A value below 0.1 seconds will disable this timeout (careful: connections can be kept indefinitely).

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.

8.60.3.2 ConnectMethod Photon.Pun.PhotonNetwork.ConnectMethod = ConnectMethod.NotCalled `[static]`

Tracks, which Connect method was called last.

ConnectToMaster sets this to ConnectToMaster. ConnectToRegion sets this to ConnectToRegion. ConnectTo↩BestCloudServer sets this to ConnectToBest. PhotonNetwork.ConnectUsingSettings will call either ConnectTo↩Master, ConnectToRegion or ConnectToBest, depending on the settings.

8.60.3.3 bool Photon.Pun.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.

8.60.3.4 PunLogLevel Photon.Pun.PhotonNetwork.LogLevel = PunLogLevel.ErrorsOnly `[static]`

Controls how verbose PUN is.

8.60.3.5 readonly int Photon.Pun.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.

8.60.3.6   int Photon.Pun.PhotonNetwork.ObjectsInOneUpdate = 10 `[static]`

Defines how many OnPhotonSerialize()-calls might get summarized in one message.

A low number increases overhead, a high number might mean fragmentation.

8.60.3.7   ServerSettings Photon.Pun.PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(Photon↩
Network.ServerSettingsFileName, typeof(ServerSettings)) `[static]`

Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.

8.60.3.8   float Photon.Pun.PhotonNetwork.precisionForFloatSynchronization = 0.01f `[static]`

The minimum difference between floats before we send it via a PhotonView's OnSerialize/ObservingComponent.

8.60.3.9   float Photon.Pun.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.

8.60.3.10   float Photon.Pun.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.

8.60.3.11   const string Photon.Pun.PhotonNetwork.PunVersion = "2.0"

Version number of PUN. Used in the AppVersion, which separates your playerbase in matchmaking.

8.60.3.12   bool Photon.Pun.PhotonNetwork.UseRpcMonoBehaviourCache `[static]`

While enabled, the MonoBehaviours on which we call RPCs are cached, avoiding costly GetComponents<Mono↩
Behaviour>() 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.).

8.60.4   Property Documentation

8.60.4.1   string Photon.Pun.PhotonNetwork.AppVersion `[static]`,`[get]`

Sent to Photon Server to specifiy the "Virtual AppId".

Sent with the operation Authenticate. When using PUN, you should set the GameVersion or use ConnectUsing↩
Settings().

8.60.4.2 AuthenticationValues Photon.Pun.PhotonNetwork.AuthValues `[static],[get],[set]`

A user's authentication values used during connect.

Set these before calling Connect if you want custom authentication. These values set the userId, if and how that userId gets verified (server-side), etc..

If authentication fails for any values, PUN will call your implementation of OnCustomAuthenticationFailed(string debugMessage). See Photon.Realtime.IConnectionCallbacks.OnCustomAuthenticationFailed.

8.60.4.3 bool Photon.Pun.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.

8.60.4.4 string Photon.Pun.PhotonNetwork.BestRegionSummaryInPreferences `[static],[get],[set]`

Used to store and access the "Best Region Summary" in the Player Preferences.

8.60.4.5 string Photon.Pun.PhotonNetwork.CloudRegion `[static],[get]`

Currently used Cloud Region (if any). As long as the client is not on a Master Server or Game Server, the region is not yet defined.

8.60.4.6 int Photon.Pun.PhotonNetwork.CountOfPlayers `[static],[get]`

The count of players currently using this application (available on MasterServer in 5sec intervals).

8.60.4.7 int Photon.Pun.PhotonNetwork.CountOfPlayersInRooms `[static],[get]`

Count of users currently playing your app in some room (sent every 5sec by Master Server). Use PhotonNetwork.↩
PlayerList.Length or PhotonNetwork.CurrentRoom.PlayerCount to get the count of players in the room you're in!

8.60.4.8 int Photon.Pun.PhotonNetwork.CountOfPlayersOnMaster `[static],[get]`

The count of players currently looking for a room (available on MasterServer in 5sec intervals).

8.60.4.9 int Photon.Pun.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).

8.60.4.10 bool Photon.Pun.PhotonNetwork.CrcCheckEnabled `[static],[get],[set]`

Crc checks can be useful to detect and avoid issues with broken datagrams. Can be enabled while not connected.

8.60.4.11 TypedLobby Photon.Pun.PhotonNetwork.CurrentLobby `[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. So when you connect or leave a room, PUN automatically gets you into a lobby again.

Check PhotonNetwork.InLobby if the client is in a lobby. (masterServerAndLobby)

8.60.4.12 Room Photon.Pun.PhotonNetwork.CurrentRoom `[static],[get]`

Get the room we're currently in (also when in OfflineMode). Null if we aren't in any room.

LoadBalancing Client is not aware of the Photon Offline Mode, so never use PhotonNetwork.NetworkingClient.↩
CurrentRoom will be null if you are using OffLine Mode, while PhotonNetwork.CurrentRoom will be set when
offlineMode is true

8.60.4.13 bool Photon.Pun.PhotonNetwork.EnableLobbyStatistics `[static],[get]`

If enabled, the client will get a list of available lobbies from the Master Server.

Set this value before the client connects to the Master Server. While connected to the Master Server, a change has no effect.

Implement OptionalInfoCallbacks.OnLobbyStatisticsUpdate, to get the list of used lobbies.

The lobby statistics can be useful if your title dynamically uses lobbies, depending (e.g.) on current player activity or such. In this case, getting a list of available lobbies, their room-count and player-count can be useful info.

ConnectUsingSettings sets this to the PhotonServerSettings value.

8.60.4.14 string Photon.Pun.PhotonNetwork.GameVersion `[static],[get],[set]`

Version number of your game. Setting this updates the AppVersion, which separates your playerbase in matchmaking.

In PUN, the GameVersion is only one component of the LoadBalancingClient.AppVersion. Setting the GameVersion will also set the LoadBalancingClient.AppVersion to: value+'_'+ PhotonNetwork.PunVersion.

The AppVersion is used to split your playerbase as needed. One AppId may have various AppVersions and each is a separate set of users for matchmaking.

The AppVersion gets sent in the "Authenticate" step. This means you can set the GameVersion right after calling ConnectUsingSettings (e.g.) and the new value will be used on the server. Once the client is connected, authentication is done and the value won't be sent to the server anymore.

8.60.4.15 bool Photon.Pun.PhotonNetwork.InLobby `[static],[get]`

True while this client is in a lobby.

Implement IPunCallbacks.OnRoomListUpdate() 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).

8.60.4.16 bool Photon.Pun.PhotonNetwork.InRoom `[static],[get]`

Is true while being in a room (NetworkClientState == ClientState.Joined).

Aside from polling this value, game logic should implement IMatchmakingCallbacks in some class and react when that gets called.

Many actions can only be executed in a room, like Instantiate or Leave, etc.
A client can join a room in offline mode. In that case, don't use LoadBalancingClient.InRoom, which does not cover offline mode.

8.60.4.17 bool Photon.Pun.PhotonNetwork.IsConnected `[static],[get]`

False until you connected to Photon initially. True in offline mode, while connected to any server and even while switching servers.

8.60.4.18 bool Photon.Pun.PhotonNetwork.IsConnectedAndReady `[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.

8.60.4.19 bool Photon.Pun.PhotonNetwork.IsMasterClient `[static],[get]`

Are we the master client?

8.60.4.20 bool Photon.Pun.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.

8.60.4.21 float Photon.Pun.PhotonNetwork.LevelLoadingProgress `[static],[get]`

Gets the networked level loading progress. Value will be be zero until the first loading, and remain at one in between loadings Use PhotonNetwork.LoadLevel() to initiate a networked level Loading

The level loading progress. Ranges from 0 to 1

8.60.4.22 Player Photon.Pun.PhotonNetwork.LocalPlayer `[static],[get]`

This client's Player instance is always available, unless the app shuts down.

Useful (e.g.) to set the Custom Player Properties or the NickName for this client anytime. When the client joins a room, the Custom Properties and other values are synced.

8.60.4.23 Player Photon.Pun.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.

8.60.4.24   int Photon.Pun.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.

8.60.4.25   ClientState Photon.Pun.PhotonNetwork.NetworkClientState   `[static],[get]`

Directly provides the network-level client state, unless in OfflineMode.

In context of PUN, you should usually use IsConnected or IsConnectedAndReady.

This is the lower level connection state. Keep in mind that PUN uses more than one server, so the client may become Disconnected, even though it's just switching servers.

While OfflineMode is true, this is ClientState.Joined (after create/join) or ConnectedToMasterserver in all other cases.

8.60.4.26   bool Photon.Pun.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

8.60.4.27   string Photon.Pun.PhotonNetwork.NickName   `[static],[get],[set]`

Set to synchronize the player's nickname with everyone in the room(s) you enter. This sets PhotonNetwork.player.↵
NickName.

The NickName 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: Player.NickName.
PhotonNetwork.PlayerListOthers is a list of other players - each contains the NickName the remote player set.

8.60.4.28   bool Photon.Pun.PhotonNetwork.OfflineMode   `[static],[get],[set]`

Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on Photon↵
Network will not create any connections and there is near to no overhead. Mostly usefull for reusing RPC's and
PhotonNetwork.Instantiate

8.60.4.29   int Photon.Pun.PhotonNetwork.PacketLossByCrcCheck   `[static],[get]`

If CrcCheckEnabled, this counts the incoming packages that don't have a valid CRC checksum and got rejected.

8.60.4.30   PhotonView [ ] Photon.Pun.PhotonNetwork.PhotonViews   `[static],[get]`

Gets the photon views.

This is an expensive operation as it returns a copy of the internal list.

The photon views.

8.60.4.31   IPunPrefabPool Photon.Pun.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.

8.60.4.32   int Photon.Pun.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.

8.60.4.33   int Photon.Pun.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.

8.60.4.34   int Photon.Pun.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 'SerializationRate'.

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.

8.60.4.35   int Photon.Pun.PhotonNetwork.SerializationRate   `[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.

8.60.4.36   ServerConnection Photon.Pun.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.

8.60.4.37   string Photon.Pun.PhotonNetwork.ServerAddress   `[static],[get]`

Currently used server address (no matter if master or game server).

8.60.4.38   int Photon.Pun.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.

8.60.4.39 double Photon.Pun.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!

8.60.4.40 bool Photon.Pun.PhotonNetwork.UseAlternativeUdpPorts `[static],[get],[set]`

Switch to alternative ports for a UDP connection to the Public Cloud.

This should be used when a customer has issues with connection stability. Some players reported better connectivity for Steam games. The effect might vary, which is why the alternative ports are not the new default.

The alternative (server) ports are 27000 up to 27003.

The values are appplied by replacing any incoming server-address string accordingly. You only need to set this to true though.

This value does not affect TCP or WebSocket connections.

8.61 Photon.Realtime.PhotonPing Class Reference

Inherits IDisposable.

Inherited by Photon.Realtime.PingMono.

Public Member Functions

- virtual bool **StartPing** (string ip)
- virtual bool **Done** ()
- virtual void **Dispose** ()

Public Attributes

- string **DebugString** = ""
- bool **Successful**

8.62 Photon.Pun.PhotonRigidbody2DView Class Reference

Inherits MonoBehaviour, and Photon.Pun.IPunObservable.

Public Member Functions

- void **Awake** ()
- void **FixedUpdate** ()
- 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.

Public Attributes

- bool **m_SynchronizeVelocity** = true
- bool **m_SynchronizeAngularVelocity** = false

8.62.1 Member Function Documentation

8.62.1.1 void Photon.Pun.PhotonRigidbody2DView.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.SerializationRate 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 Photon↩ View.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 Photon.Pun.IPunObservable.

8.63 Photon.Pun.PhotonRigidbodyView Class Reference

Inherits MonoBehaviour, and Photon.Pun.IPunObservable.

Public Member Functions

- void **Awake** ()
- void **FixedUpdate** ()
- 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.

Public Attributes

- bool **m_SynchronizeVelocity** = true
- bool **m_SynchronizeAngularVelocity** = false

8.63.1 Member Function Documentation

8.63.1.1 void Photon.Pun.PhotonRigidbodyView.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.SerializationRate 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 Photon↩
View.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 Photon.Pun.IPunObservable.

8.64 Photon.Pun.UtilityScripts.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).*

8.64.1 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).

8.64.2 Member Function Documentation

8.64.2.1 void Photon.Pun.UtilityScripts.PhotonStatsGui.Update ( )

Checks for shift+tab input combination (to toggle statsOn).

8.64.3 Member Data Documentation

8.64.3.1 bool Photon.Pun.UtilityScripts.PhotonStatsGui.buttonsOn

Show buttons to control stats and reset them.

8.64.3.2 bool Photon.Pun.UtilityScripts.PhotonStatsGui.healthStatsVisible

Shows additional "health" values of connection.

8.64.3.3 bool Photon.Pun.UtilityScripts.PhotonStatsGui.statsOn = true

Option to turn collecting stats on or off (used in Update()).

8.64.3.4 Rect Photon.Pun.UtilityScripts.PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)

Positioning rect for window.

8.64.3.5 bool Photon.Pun.UtilityScripts.PhotonStatsGui.statsWindowOn = true

Shows or hides GUI (does not affect if stats are collected).

8.64.3.6 bool Photon.Pun.UtilityScripts.PhotonStatsGui.trafficStatsOn

Shows additional "lower level" traffic stats.

Unity GUI Window ID (must be unique or will cause issues).

8.65   Photon.Pun.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.
- void **SetReadStream** (object[ ] incomingData, byte pos=0)
- 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 Player 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.

8.65.1 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.

8.65.2 Constructor & Destructor Documentation

8.65.2.1 Photon.Pun.PhotonStream.PhotonStream (bool *write,* object[ ] *incomingData*)

Creates a stream and initializes it. Used by PUN internally.

8.65.3 Member Function Documentation

8.65.3.1 object Photon.Pun.PhotonStream.PeekNext ( )

Read next piece of data from the stream without advancing the "current" item.

8.65.3.2 object Photon.Pun.PhotonStream.ReceiveNext ( )

Read next piece of data from the stream when IsReading is true.

8.65.3.3 void Photon.Pun.PhotonStream.SendNext (object *obj*)

Add another piece of data to send it when IsWriting is true.

8.65.3.4 void Photon.Pun.PhotonStream.Serialize (ref bool *myBool*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.5 void Photon.Pun.PhotonStream.Serialize (ref int *myInt*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.6 void Photon.Pun.PhotonStream.Serialize (ref string *value*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.7 void Photon.Pun.PhotonStream.Serialize (ref char *value*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.8 void Photon.Pun.PhotonStream.Serialize (ref short *value*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.9   void Photon.Pun.PhotonStream.Serialize (ref float *obj*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.10   void Photon.Pun.PhotonStream.Serialize (ref Player *obj*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.11   void Photon.Pun.PhotonStream.Serialize (ref Vector3 *obj*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.12   void Photon.Pun.PhotonStream.Serialize (ref Vector2 *obj*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.13   void Photon.Pun.PhotonStream.Serialize (ref Quaternion *obj*)

Will read or write the value, depending on the stream's IsWriting value.

8.65.3.14   object [ ] Photon.Pun.PhotonStream.ToArray (   )

Turns the stream into a new object[].

8.65.4   Property Documentation

8.65.4.1   int Photon.Pun.PhotonStream.Count `[get]`

Count of items in the stream.

8.65.4.2   bool Photon.Pun.PhotonStream.IsReading `[get]`

If true, this client should read data send by another client.

8.65.4.3   bool Photon.Pun.PhotonStream.IsWriting `[get]`

If true, this client should add data to the stream to send it.

8.66   Photon.Pun.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.

8.66.1 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.

8.66.2 Constructor & Destructor Documentation

8.66.2.1 Photon.Pun.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

8.66.3 Member Function Documentation

8.66.3.1 void Photon.Pun.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

8.66.3.2 bool Photon.Pun.PhotonStreamQueue.HasQueuedObjects ( )

Determines whether the queue has stored any objects

8.66.3.3 object Photon.Pun.PhotonStreamQueue.ReceiveNext ( )

Receives the next object from the queue. This works just like PhotonStream.ReceiveNext

Returns

8.66.3.4   void Photon.Pun.PhotonStreamQueue.Reset (   )

Resets the PhotonStreamQueue. You need to do this whenever the amount of objects you are observing changes

8.66.3.5   void Photon.Pun.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

8.66.3.6   void Photon.Pun.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

8.67   Photon.Pun.PhotonTransformView Class Reference

Inherits MonoBehaviour, and Photon.Pun.IPunObservable.

Public Member Functions

- void **Awake** ()
- void **Update** ()
- 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.

Public Attributes

- bool **m_SynchronizePosition** = true
- bool **m_SynchronizeRotation** = true
- bool **m_SynchronizeScale** = false

8.67.1   Member Function Documentation

8.67.1.1   void Photon.Pun.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 Photon←
View.

This method will be called in scripts that are assigned as Observed component of a PhotonView.
PhotonNetwork.SerializationRate 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 Photon↩ View.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 [Photon.Pun.IPunObservable](#).

8.68 Photon.Pun.PhotonView Class Reference

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

Inherits MonoBehaviour.

Public Member Functions

- void [RequestOwnership](#) ()

  Depending on the [PhotonView](#)'s OwnershipTransfer setting, any client can request to become owner of the [Photon↩ View](#).

- void [TransferOwnership](#) ([Player](#) 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.UseRpcMono↩ BehaviourCache](#) is true.

- void [RPC](#) (string methodName, [RpcTarget](#) 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, [RpcTarget](#) 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, [Player](#) 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, [Player](#) 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

- byte **Group** = 0
- bool OwnershipWasTransfered

    Flag to check if ownership of this photonView was set during the lifecycle. Used for checking when joining late if event with mismatched owner and sender needs addressing.

- int **prefixField** = -1
- ViewSynchronization **Synchronization**
- 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.

- Player 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, set]`
- Player **Controller** `[get]`
- int **ControllerActorNr** `[get]`
- bool **IsOwnerActive** `[get]`
- int **CreatorActorNr** `[get]`
- bool IsMine `[get]`

    True if the PhotonView is "mine" and can be controlled by this client.

8.68.1 Detailed Description

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

8.68.2 Member Function Documentation

8.68.2.1 void Photon.Pun.PhotonView.RefreshRpcMonoBehaviourCache ( )

Can be used to refesh the list of MonoBehaviours on this GameObject while PhotonNetwork.UseRpcMono↵BehaviourCache 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.

8.68.2.2   void Photon.Pun.PhotonView.RequestOwnership (   )

Depending on the PhotonView's OwnershipTransfer setting, any client can request to become owner of the Photon↩
View.

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.

8.68.2.3   void Photon.Pun.PhotonView.RPC (string *methodName,* RpcTarget *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!).

8.68.2.4   void Photon.Pun.PhotonView.RPC (string *methodName,* Player *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!).

8.68.2.5   void Photon.Pun.PhotonView.RpcSecure (string *methodName,* RpcTarget *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!).

8.68.2.6   void Photon.Pun.PhotonView.RpcSecure (string *methodName,* Player *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!).

8.68.2.7   void Photon.Pun.PhotonView.TransferOwnership (Player *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.

8.68.2.8   void Photon.Pun.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.

8.68.3   Member Data Documentation

8.68.3.1   OwnershipOption Photon.Pun.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.

8.68.3.2   bool Photon.Pun.PhotonView.OwnershipWasTransfered

Flag to check if ownership of this photonView was set during the lifecycle. Used for checking when joining late if event with mismatched owner and sender needs addressing.

`true` if owner ship was transfered; otherwise, `false`.

8.68.4 Property Documentation

8.68.4.1 object [ ] Photon.Pun.PhotonView.InstantiationData `[get],[set]`

This is the InstantiationData that was passed when calling PhotonNetwork.Instantiate∗ (if that was used to spawn this prefab)

8.68.4.2 bool Photon.Pun.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 Player. True if this is a scene photonview on the Master client.

8.68.4.3 bool Photon.Pun.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).

8.68.4.4 Player Photon.Pun.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 IPun↩ Callbacks.OnOwnershipRequest to react to the ownership request.

8.68.4.5 int Photon.Pun.PhotonView.ViewID `[get],[set]`

The ID of the PhotonView. Identifies it in a networked game (per room).

See: Network Instantiation

8.69 Photon.Realtime.PingMono Class Reference

Uses C# Socket class from System.Net.Sockets (as Unity usually does).

Inherits Photon.Realtime.PhotonPing.

Public Member Functions

- override bool StartPing (string ip)

    Sends a "Photon Ping" to a server.
- override bool **Done** ()
- override void **Dispose** ()

Additional Inherited Members

8.69.1 Detailed Description

Uses C# Socket class from System.Net.Sockets (as Unity usually does).

Incompatible with Windows 8 Store/Phone API.

8.69.2 Member Function Documentation

8.69.2.1 override bool Photon.Realtime.PingMono.StartPing (string *ip*) `[virtual]`

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.

Reimplemented from Photon.Realtime.PhotonPing.

8.70 Photon.Realtime.Player Class Reference

Summarizes a "player" within a room, identified (in that room) by ID (or "actorNumber").

Public Member Functions

- Player Get (int id)

    Get a Player by ActorNumber (Player.ID).
- Player GetNext ()

    Gets this Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.
- Player GetNextFor (Player currentPlayer)

    Gets a Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.
- Player GetNextFor (int currentPlayerId)

    Gets a Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.
- virtual void InternalCacheProperties (Hashtable properties)

    Caches properties for new Players or when updates of remote players are received. Use SetCustomProperties() for a synced update.
- override string ToString ()

    Brief summary string of the Player. Includes name or player.ID and if it's the Master Client.
- string ToStringFull ()

    String summary of the Player: player.ID, name and all custom properties of this user.
- override bool Equals (object p)

    If players are equal (by GetHasCode, which returns this.ID).
- override int GetHashCode ()

    Accompanies Equals, using the ID (actorNumber) as HashCode to return.
- void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedValues=null, WebFlags web↩Flags=null)

    Updates and synchronizes this Player's Custom Properties. Optionally, expectedProperties can be provided as condition.

Public Attributes

- readonly bool IsLocal

    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".

Properties

- int ActorNumber `[get]`

    Identifier of this player in current room. Also known as: actorNumber or actorNumber. It's -1 outside of rooms.
- string NickName `[get, set]`

    Non-unique nickname of this player. Synced automatically in a room.
- 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]`

    If this player is active in the room (and getting events which are currently being sent).
- Hashtable CustomProperties `[get, set]`

    Read-only cache for custom properties of player. Set via Player.SetCustomProperties.

8.70.1 Detailed Description

Summarizes a "player" within a room, identified (in that room) by ID (or "actorNumber").

Each player has a actorNumber, valid for that room. It's -1 until assigned by server (and client logic).

8.70.2 Member Function Documentation

8.70.2.1 override bool Photon.Realtime.Player.Equals (object *p*)

If players are equal (by GetHasCode, which returns this.ID).

8.70.2.2 Player Photon.Realtime.Player.Get (int *id*)

Get a Player by ActorNumber (Player.ID).

Parameters

id	ActorNumber of the a player in this room.

Returns

Player or null.

8.70.2.3 override int Photon.Realtime.Player.GetHashCode ( )

Accompanies Equals, using the ID (actorNumber) as HashCode to return.

8.70.2.4   Player Photon.Realtime.Player.GetNext (   )

Gets this Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.

Returns

> Player or null.

8.70.2.5   Player Photon.Realtime.Player.GetNextFor (Player *currentPlayer*)

Gets a Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.

Useful when you pass something to the next player. For example: passing the turn to the next player.

Parameters

currentPlayer	The Player for which the next is being needed.

Returns

> Player or null.

8.70.2.6   Player Photon.Realtime.Player.GetNextFor (int *currentPlayerId*)

Gets a Player's next Player, as sorted by ActorNumber (Player.ID). Wraps around.

Useful when you pass something to the next player. For example: passing the turn to the next player.

Parameters

currentPlayerId	The ActorNumber (Player.ID) for which the next is being needed.

Returns

> Player or null.

8.70.2.7   virtual void Photon.Realtime.Player.InternalCacheProperties (Hashtable *properties*) `[virtual]`

Caches properties for new Players or when updates of remote players are received. Use SetCustomProperties() for a synced update.

This only updates the CustomProperties and doesn't send them to the server. Mostly used when creating new remote players, where the server sends their properties.

8.70.2.8   void Photon.Realtime.Player.SetCustomProperties (Hashtable *propertiesToSet,* Hashtable *expectedValues =* `null,` WebFlags *webFlags =* `null`)

Updates and synchronizes this Player's Custom Properties. Optionally, expectedProperties can be provided as condition.

Custom Properties are a set of string keys and arbitrary values which is synchronized for the players in a Room. They are available when the client enters the room, as they are in the response of OpJoin and OpCreate.

Custom Properties either relate to the (current) Room or a Player (in that Room).

Both classes locally cache the current key/values and make them available as property: CustomProperties. This is provided only to read them. You must use the method SetCustomProperties to set/modify them.

Any client can set any Custom Properties anytime (when in a room). It's up to the game logic to organize how they are best used.

You should call SetCustomProperties only with key/values that are new or changed. This reduces traffic and performance.

Unless you define some expectedProperties, setting key/values is always permitted. In this case, the property-setting client will not receive the new values from the server but instead update its local cache in SetCustom↩ Properties.

If you define expectedProperties, the server will skip updates if the server property-cache does not contain all expectedProperties with the same values. In this case, the property-setting client will get an update from the server and update it's cached key/values at about the same time as everyone else.

The benefit of using expectedProperties can be only one client successfully sets a key from one known value to another. As example: Store who owns an item in a Custom Property "ownedBy". It's 0 initally. When multiple players reach the item, they all attempt to change "ownedBy" from 0 to their actorNumber. If you use expectedProperties {"ownedBy", 0} as condition, the first player to take the item will have it (and the others fail to set the ownership).

Properties get saved with the game state for Turnbased games (which use IsPersistent = true).

Parameters

propertiesToSet	Hashtable of Custom Properties to be set.
expectedValues	If non-null, these are the property-values the server will check as condition for this update.
webFlags	Defines if this SetCustomProperties-operation gets forwarded to your WebHooks. Client must be in room.

8.70.2.9   override string Photon.Realtime.Player.ToString ( )

Brief summary string of the Player. Includes name or player.ID and if it's the Master Client.

8.70.2.10   string Photon.Realtime.Player.ToStringFull ( )

String summary of the Player: 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.

8.70.3   Member Data Documentation

8.70.3.1   readonly bool Photon.Realtime.Player.IsLocal

Only one player is controlled by each client. Others are not local.

8.70.3.2   object Photon.Realtime.Player.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.

8.70.4   Property Documentation

8.70.4.1   int Photon.Realtime.Player.ActorNumber   `[get]`

Identifier of this player in current room. Also known as: actorNumber or actorNumber. It's -1 outside of rooms.

The ID is assigned per room and only valid in that context. It will change even on leave and re-join. IDs are never re-used per room.

8.70.4.2 Hashtable Photon.Realtime.Player.CustomProperties `[get],[set]`

Read-only cache for custom properties of player. Set via Player.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

8.70.4.3 bool Photon.Realtime.Player.IsInactive `[get],[set]`

If this player is active in the room (and getting events which are currently being sent).

Inactive players keep their spot in a room but otherwise behave as if offline (no matter what their actual connection status is). The room needs a PlayerTTL != 0. If a player is inactive for longer than PlayerTTL, the server will remove this player from the room. For a client "rejoining" a room, is the same as joining it: It gets properties, cached events and then the live events.

8.70.4.4 bool Photon.Realtime.Player.IsMasterClient `[get]`

True if this player is the Master Client of the current room.

See also: PhotonNetwork.MasterClient.

8.70.4.5 string Photon.Realtime.Player.NickName `[get],[set]`

Non-unique nickname of this player. Synced automatically in a room.

A player might change his own playername in a room (it's only a property). Setting this value updates the server and other players (using an operation).

8.70.4.6 string Photon.Realtime.Player.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).

8.71 Photon.Pun.UtilityScripts.PlayerNumbering Class Reference

Implements consistent numbering in a room/game with help of room properties. Access them by Player.GetPlayer↩
Number() extension.

Inherits Photon.Pun.MonoBehaviourPunCallbacks.

Public Member Functions

- delegate void PlayerNumberingChanged ()

    OnPlayerNumberingChanged delegate. Use
- void **Awake** ()
- override void OnJoinedRoom ()

    Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.
- override void OnLeftRoom ()

    Called when the local user/client left a room, so the game's logic can clean up it's internal state.
- override void OnPlayerEnteredRoom (Player newPlayer)

Called when a remote player entered the room. This Player is already added to the playerlist.

- override void OnPlayerLeftRoom (Player otherPlayer)

  Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

- override void OnPlayerPropertiesUpdate (Player targetPlayer, Hashtable changedProps)

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

- void RefreshData ()

  Internal call Refresh the cached data and call the OnPlayerNumberingChanged delegate.

Public Attributes

- const string RoomPlayerIndexedProp = "pNr"

  Defines the room custom property name to use for room player indexing tracking.

- bool dontDestroyOnLoad = false

  dont destroy on load flag for this Component's GameObject to survive Level Loading.

Static Public Attributes

- static PlayerNumbering instance

  The instance. EntryPoint to query about Room Indexing.

- static Player[ ] **SortedPlayers**

Events

- static PlayerNumberingChanged OnPlayerNumberingChanged

  Called everytime the room Indexing was updated. Use this for discrete updates. Always better than brute force calls every frame.

Additional Inherited Members

8.71.1 Detailed Description

Implements consistent numbering in a room/game with help of room properties. Access them by Player.GetPlayer↩
Number() extension.

indexing ranges from 0 to the maximum number of Players. indexing remains for the player while in room. If a Player
is numbered 2 and player numbered 1 leaves, numbered 1 become vacant and will assigned to the future player
joining (the first available vacant number is assigned when joining)

8.71.2 Member Function Documentation

8.71.2.1 override void Photon.Pun.UtilityScripts.PlayerNumbering.OnJoinedRoom ( ) `[virtual]`

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↩
CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom
Property).

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.71.2.2 override void Photon.Pun.UtilityScripts.PlayerNumbering.OnLeftRoom ( ) `[virtual]`

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.71.2.3 override void Photon.Pun.UtilityScripts.PlayerNumbering.OnPlayerEnteredRoom (Player *newPlayer*) `[virtual]`

Called when a remote player entered the room. This Player is already added to the playerlist.

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.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.71.2.4 override void Photon.Pun.UtilityScripts.PlayerNumbering.OnPlayerLeftRoom (Player *otherPlayer*) `[virtual]`

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

If another player leaves the room or if the server detects a lost connection, this callback will be used to notify your game logic.

Depending on the room's setup, players may become inactive, which means they may return and retake their spot in the room. In such cases, the Player stays in the Room.Players dictionary.

If the player is not just inactive, it gets removed from the Room.Players dictionary, before the callback is called.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.71.2.5 override void Photon.Pun.UtilityScripts.PlayerNumbering.OnPlayerPropertiesUpdate (Player *target,* Hashtable *changedProps*) `[virtual]`

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

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

Parameters

targetPlayer	Contains Player that changed.
changedProps	Contains the properties that changed.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.71.2.6 delegate void Photon.Pun.UtilityScripts.PlayerNumbering.PlayerNumberingChanged ( )

OnPlayerNumberingChanged delegate. Use

8.71.2.7 void Photon.Pun.UtilityScripts.PlayerNumbering.RefreshData ( )

Internal call Refresh the cached data and call the OnPlayerNumberingChanged delegate.

8.71.3 Member Data Documentation

8.71.3.1 bool Photon.Pun.UtilityScripts.PlayerNumbering.dontDestroyOnLoad = false

dont destroy on load flag for this Component's GameObject to survive Level Loading.

8.71.3.2 PlayerNumbering Photon.Pun.UtilityScripts.PlayerNumbering.instance `[static]`

The instance. EntryPoint to query about Room Indexing.

8.71.3.3 const string Photon.Pun.UtilityScripts.PlayerNumbering.RoomPlayerIndexedProp = "pNr"

Defines the room custom property name to use for room player indexing tracking.

8.71.4 Event Documentation

8.71.4.1 PlayerNumberingChanged Photon.Pun.UtilityScripts.PlayerNumbering.OnPlayerNumberingChanged
`[static]`

Called everytime the room Indexing was updated. Use this for discrete updates. Always better than brute force calls every frame.

8.72 Photon.Pun.UtilityScripts.PlayerNumberingExtensions Class Reference

Extension used for PlayerRoomIndexing and Player class.

Static Public Member Functions

- static int GetPlayerNumber (this Player player)

  Extension for Player class to wrap up access to the player's custom property. Make sure you use the delegate 'On↩ PlayerNumberingChanged' to knoiw when you can query the PlayerNumber. Numbering can changes over time or not be yet assigned during the initial phase (when player creates a room for example)
- static void SetPlayerNumber (this Player player, int playerNumber)

  Sets the player number. It's not recommanded to manually interfere with the playerNumbering, but possible.

8.72.1 Detailed Description

Extension used for PlayerRoomIndexing and Player class.

8.72.2 Member Function Documentation

8.72.2.1 static int Photon.Pun.UtilityScripts.PlayerNumberingExtensions.GetPlayerNumber (this Player *player*)
`[static]`

Extension for Player class to wrap up access to the player's custom property. Make sure you use the delegate 'OnPlayerNumberingChanged' to knoiw when you can query the PlayerNumber. Numbering can changes over time or not be yet assigned during the initial phase (when player creates a room for example)

Returns

persistent index in room. -1 for no indexing

8.72.2.2 **static void Photon.Pun.UtilityScripts.PlayerNumberingExtensions.SetPlayerNumber (this Player** *player,* **int** *playerNumber* **)** `[static]`

Sets the player number. It's not recommanded to manually interfere with the playerNumbering, but possible.

Parameters

player	Player.
playerNumber	Player number.

8.73 Photon.Pun.UtilityScripts.PointedAtGameObjectInfo Class Reference

Display ViewId, OwnerActorNr, IsCeneView and IsMine when clicked.

Inherits MonoBehaviour.

Public Member Functions

- void **SetFocus** ([PhotonView](#) pv)
- void **RemoveFocus** ([PhotonView](#) pv)

Public Attributes

- Text **text**

Static Public Attributes

- static [PointedAtGameObjectInfo](#) **Instance**

8.73.1 Detailed Description

Display ViewId, OwnerActorNr, IsCeneView and IsMine when clicked.

8.74 Photon.Pun.PunExtensions Class Reference

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

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 Public Attributes

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

8.74.1   Detailed Description

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

8.74.2   Member Function Documentation

8.74.2.1   static bool Photon.Pun.PunExtensions.AlmostEquals (this Vector3 *target,* Vector3 *second,* float *sqrMagnitudePrecision*) `[static]`

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

8.74.2.2   static bool Photon.Pun.PunExtensions.AlmostEquals (this Vector2 *target,* Vector2 *second,* float *sqrMagnitudePrecision*) `[static]`

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

8.74.2.3   static bool Photon.Pun.PunExtensions.AlmostEquals (this Quaternion *target,* Quaternion *second,* float *maxAngle*) `[static]`

compares the angle between target and second to given float value

8.74.2.4   static bool Photon.Pun.PunExtensions.AlmostEquals (this float *target,* float *second,* float *floatDiff*) `[static]`

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

8.75   Photon.Pun.UtilityScripts.PunPlayerScores Class Reference

Scoring system for PhotonPlayer

Inherits MonoBehaviour.

Public Attributes

- const string **PlayerScoreProp** = "score"

8.75.1   Detailed Description

Scoring system for PhotonPlayer

8.76   Photon.Pun.PunRPC Class Reference

Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

Inherits Attribute.

8.76.1   Detailed Description

Replacement for RPC attribute with different name. Used to flag methods as remote-callable.

8.77 Photon.Pun.UtilityScripts.PunTeams Class Reference

Implements teams in a room/game with help of player properties. Access them by Player.GetTeam extension.

Inherits Photon.Pun.MonoBehaviourPunCallbacks.

Public Types

- enum Team : byte

    Enum defining the teams available. First team should be neutral (it's the default value any field of this enum gets).

Public Member Functions

- void **Start** ()
- override void **OnDisable** ()
- override void OnJoinedRoom ()

    Needed to update the team lists when joining a room.

- override void OnLeftRoom ()

    Called when the local user/client left a room, so the game's logic can clean up it's internal state.

- override void OnPlayerPropertiesUpdate (Player targetPlayer, Hashtable changedProps)

    Refreshes the team lists. It could be a non-team related property change, too.

- override void OnPlayerLeftRoom (Player otherPlayer)

    Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

- override void OnPlayerEnteredRoom (Player newPlayer)

    Called when a remote player entered the room. This Player is already added to the playerlist.

- void **UpdateTeams** ()

Public Attributes

- const string TeamPlayerProp = "team"

    Defines the player custom property name to use for team affinity of "this" player.

Static Public Attributes

- static Dictionary< Team, List< Player > > PlayersPerTeam

    The main list of teams with their player-lists. Automatically kept up to date.

Additional Inherited Members

8.77.1 Detailed Description

Implements teams in a room/game with help of player properties. Access them by Player.GetTeam extension.

Teams are defined by enum Team. Change this to get more / different teams. There are no rules when / if you can join a team. You could add this in JoinTeam or something.

8.77.2 Member Enumeration Documentation

8.77.2.1 enum Photon.Pun.UtilityScripts.PunTeams.Team : byte [strong]

Enum defining the teams available. First team should be neutral (it's the default value any field of this enum gets).

8.77.3 Member Function Documentation

8.77.3.1 override void Photon.Pun.UtilityScripts.PunTeams.OnJoinedRoom ( ) `[virtual]`

Needed to update the team lists when joining a room.

Called by PUN. See enum MonoBehaviourPunCallbacks for an explanation.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.77.3.2 override void Photon.Pun.UtilityScripts.PunTeams.OnLeftRoom ( ) `[virtual]`

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.77.3.3 override void Photon.Pun.UtilityScripts.PunTeams.OnPlayerEnteredRoom (Player *newPlayer*) `[virtual]`

Called when a remote player entered the room. This Player is already added to the playerlist.

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.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.77.3.4 override void Photon.Pun.UtilityScripts.PunTeams.OnPlayerLeftRoom (Player *otherPlayer*) `[virtual]`

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

If another player leaves the room or if the server detects a lost connection, this callback will be used to notify your game logic.

Depending on the room's setup, players may become inactive, which means they may return and retake their spot in the room. In such cases, the Player stays in the Room.Players dictionary.

If the player is not just inactive, it gets removed from the Room.Players dictionary, before the callback is called.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.77.3.5 override void Photon.Pun.UtilityScripts.PunTeams.OnPlayerPropertiesUpdate (Player *targetPlayer,* Hashtable *changedProps*) `[virtual]`

Refreshes the team lists. It could be a non-team related property change, too.

Called by PUN. See enum MonoBehaviourPunCallbacks for an explanation.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.77.4 Member Data Documentation

8.77.4.1 Dictionary<**Team,** List<**Player**> > Photon.Pun.UtilityScripts.PunTeams.PlayersPerTeam `[static]`

The main list of teams with their player-lists. Automatically kept up to date.

Note that this is static. Can be accessed by PunTeam.PlayersPerTeam. You should not modify this.

8.77.4.2 const string Photon.Pun.UtilityScripts.PunTeams.TeamPlayerProp = "team"

Defines the player custom property name to use for team affinity of "this" player.

8.78 Photon.Pun.UtilityScripts.PunTurnManager Class Reference

Pun turnBased Game manager. Provides an Interface (IPunTurnManagerCallbacks) for the typical turn flow and logic, between players Provides Extensions for Player, Room and RoomInfo to feature dedicated api for TurnBased Needs

Inherits Photon.Pun.MonoBehaviourPunCallbacks.

Public Member Functions

- void BeginTurn ()

    Tells the TurnManager to begins a new turn.
- void SendMove (object move, bool finished)

    Call to send an action. Optionally finish the turn, too. The move object can be anything. Try to optimize though and only send the strict minimum set of information to define the turn move.
- bool GetPlayerFinishedTurn (Player player)

    Gets if the player finished the current turn.
- void OnEvent (byte eventCode, object content, int senderId)

    Called by PhotonNetwork.OnEventCall registration
- override void OnRoomPropertiesUpdate (Hashtable propertiesThatChanged)

    Called by PhotonNetwork

Public Attributes

- float TurnDuration = 20f

    The duration of the turn in seconds.
- IPunTurnManagerCallbacks TurnManagerListener

    The turn manager listener. Set this to your own script instance to catch Callbacks
- const byte TurnManagerEventOffset = 0

    The turn manager event offset event message byte. Used internaly for defining data in Room Custom Properties
- const byte EvMove = 1 + TurnManagerEventOffset

    The Move event message byte. Used internaly for saving data in Room Custom Properties
- const byte EvFinalMove = 2 + TurnManagerEventOffset

    The Final Move event message byte. Used internaly for saving data in Room Custom Properties

Properties

- int Turn [get]

    Wraps accessing the "turn" custom properties of a room.
- float ElapsedTimeInTurn [get]

    Gets the elapsed time in the current turn in seconds
- float RemainingSecondsInTurn [get]

    Gets the remaining seconds for the current turn. Ranges from 0 to TurnDuration
- bool IsCompletedByAll [get]

    Gets a value indicating whether the turn is completed by all.
- bool IsFinishedByMe [get]

Gets a value indicating whether the current turn is finished by me.

- bool IsOver `[get]`

    Gets a value indicating whether the current turn is over. That is the ElapsedTimeinTurn is greater or equal to the TurnDuration

8.78.1 Detailed Description

Pun turnBased Game manager. Provides an Interface (IPunTurnManagerCallbacks) for the typical turn flow and logic, between players Provides Extensions for Player, Room and RoomInfo to feature dedicated api for TurnBased Needs

8.78.2 Member Function Documentation

8.78.2.1 void Photon.Pun.UtilityScripts.PunTurnManager.BeginTurn ( )

Tells the TurnManager to begins a new turn.

8.78.2.2 bool Photon.Pun.UtilityScripts.PunTurnManager.GetPlayerFinishedTurn (Player *player*)

Gets if the player finished the current turn.

Returns

`true`, if player finished the current turn, `false` otherwise.

Parameters

player	The Player to check for

8.78.2.3 void Photon.Pun.UtilityScripts.PunTurnManager.OnEvent (byte *eventCode,* object *content,* int *senderId*)

Called by PhotonNetwork.OnEventCall registration

Parameters

eventCode	Event code.
content	Content.
senderId	Sender identifier.

8.78.2.4 override void Photon.Pun.UtilityScripts.PunTurnManager.OnRoomPropertiesUpdate (Hashtable *propertiesThatChanged*) `[virtual]`

Called by PhotonNetwork

Parameters

propertiesThat↩ Changed	Properties that changed.

Reimplemented from Photon.Pun.MonoBehaviourPunCallbacks.

8.78.2.5 void Photon.Pun.UtilityScripts.PunTurnManager.SendMove (object *move,* bool *finished*)

Call to send an action. Optionally finish the turn, too. The move object can be anything. Try to optimize though and only send the strict minimum set of information to define the turn move.

Parameters

move	
finished	

8.78.3 Member Data Documentation

8.78.3.1 const byte Photon.Pun.UtilityScripts.PunTurnManager.EvFinalMove = 2 + **TurnManagerEventOffset**

The Final Move event message byte. Used internaly for saving data in Room Custom Properties

8.78.3.2 const byte Photon.Pun.UtilityScripts.PunTurnManager.EvMove = 1 + **TurnManagerEventOffset**

The Move event message byte. Used internaly for saving data in Room Custom Properties

8.78.3.3 float Photon.Pun.UtilityScripts.PunTurnManager.TurnDuration = 20f

The duration of the turn in seconds.

8.78.3.4 const byte Photon.Pun.UtilityScripts.PunTurnManager.TurnManagerEventOffset = 0

The turn manager event offset event message byte. Used internaly for defining data in Room Custom Properties

8.78.3.5 IPunTurnManagerCallbacks Photon.Pun.UtilityScripts.PunTurnManager.TurnManagerListener

The turn manager listener. Set this to your own script instance to catch Callbacks

8.78.4 Property Documentation

8.78.4.1 float Photon.Pun.UtilityScripts.PunTurnManager.ElapsedTimeInTurn `[get]`

Gets the elapsed time in the current turn in seconds

The elapsed time in the turn.

8.78.4.2 bool Photon.Pun.UtilityScripts.PunTurnManager.IsCompletedByAll `[get]`

Gets a value indicating whether the turn is completed by all.

`true` if this turn is completed by all; otherwise, `false`.

8.78.4.3 bool Photon.Pun.UtilityScripts.PunTurnManager.IsFinishedByMe `[get]`

Gets a value indicating whether the current turn is finished by me.

`true` if the current turn is finished by me; otherwise, `false`.

8.78.4.4 bool Photon.Pun.UtilityScripts.PunTurnManager.IsOver `[get]`

Gets a value indicating whether the current turn is over. That is the ElapsedTimeinTurn is greater or equal to the TurnDuration

`true` if the current turn is over; otherwise, `false`.

8.78.4.5 float Photon.Pun.UtilityScripts.PunTurnManager.RemainingSecondsInTurn `[get]`

Gets the remaining seconds for the current turn. Ranges from 0 to TurnDuration

The remaining seconds fo the current turn

8.78.4.6 int Photon.Pun.UtilityScripts.PunTurnManager.Turn `[get]`

Wraps accessing the "turn" custom properties of a room.

The turn index

8.79 Photon.Realtime.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.
- WebFlags Flags = WebFlags.Default

    Optional flags to be used in Photon client SDKs with Op RaiseEvent and Op SetProperties.

Static Public Attributes

- static readonly RaiseEventOptions Default = new RaiseEventOptions()

    *Default options: CachingOption: DoNotCache, InterestGroup: 0, targetActors: null, receivers: Others, sequence↩
    Channel: 0.*

8.79.1 Detailed Description

Aggregates several less-often used options for operation RaiseEvent. See field descriptions for usage details.

8.79.2 Member Data Documentation

8.79.2.1 EventCaching Photon.Realtime.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.

8.79.2.2   readonly RaiseEventOptions Photon.Realtime.RaiseEventOptions.Default = new RaiseEventOptions()
`[static]`

Default options: CachingOption: DoNotCache, InterestGroup: 0, targetActors: null, receivers: Others, sequence↩
Channel: 0.

8.79.2.3   WebFlags Photon.Realtime.RaiseEventOptions.Flags = WebFlags.Default

Optional flags to be used in Photon client SDKs with Op RaiseEvent and Op SetProperties.

Introduced mainly for webhooks 1.2 to control behavior of forwarded HTTP requests.

8.79.2.4   byte Photon.Realtime.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.

8.79.2.5   ReceiverGroup Photon.Realtime.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.

8.79.2.6   byte Photon.Realtime.RaiseEventOptions.SequenceChannel

Events are ordered per "channel". If you have events that are independent of others, they can go into another
sequence or channel.

8.79.2.7   int [ ] Photon.Realtime.RaiseEventOptions.TargetActors

A list of PhotonPlayer.IDs to send this event to. You can implement events that just go to specific users this way.

8.80   Photon.Realtime.Region Class Reference

Public Member Functions

- **Region** (string code, string address)
- override string **ToString** ()

Properties

- string **Code** `[get]`
- string Cluster `[get]`
    Unlike the CloudRegionCode, this may contain cluster information.
- string **HostAndPort** `[get, set]`
- int **Ping** `[get, set]`
- bool **WasPinged** `[get]`

8.80.1   Property Documentation

8.80.1.1   string Photon.Realtime.Region.Cluster `[get]`

Unlike the CloudRegionCode, this may contain cluster information.

8.81 Photon.Realtime.RegionHandler Class Reference

Provides methods to work with Photon's regions (Photon Cloud) and can be use to find the one with best ping.

Public Member Functions

- void **SetRegions** (OperationResponse opGetRegions)
- bool **PingMinimumOfRegions** (Action< RegionHandler > onCompleteCallback, string previousSummary)

Properties

- List< Region > EnabledRegions `[get, set]`

    A list of region names for the Photon Cloud. Set by the result of OpGetRegions().
- Region BestRegion `[get]`

    When PingMinimumOfRegions was called and completed, the BestRegion is identified by best ping.
- string SummaryToCache `[get]`

    This value summarizes the results of pinging the currently available EnabledRegions (after PingMinimumOfRegions finished).
- bool **IsPinging** `[get]`

8.81.1 Detailed Description

Provides methods to work with Photon's regions (Photon Cloud) and can be use to find the one with best ping.

When a client uses a Name Server to fetch the list of available regions, the LoadBalancingClient will create a RegionHandler and provide it via the OnRegionListReceived callback.

Your logic can decide to either connect to one of those regional servers, or it may use PingMinimumOfRegions to test which region provides the best ping.

It makes sense to make clients "sticky" to a region when one gets selected. This can be achieved by storing the SummaryToCache value, once pinging was done. When the client connects again, the previous SummaryToCache helps limiting the number of regions to ping. In best case, only the previously selected region gets re-pinged and if the current ping is not much worse, this one region is used again.

8.81.2 Property Documentation

8.81.2.1 Region Photon.Realtime.RegionHandler.BestRegion `[get]`

When PingMinimumOfRegions was called and completed, the BestRegion is identified by best ping.

8.81.2.2 List<**Region**> Photon.Realtime.RegionHandler.EnabledRegions `[get],[set]`

A list of region names for the Photon Cloud. Set by the result of OpGetRegions().

Implement ILoadBalancingCallbacks and register for the callbacks to get OnRegionListReceived(RegionHandler regionHandler). You can also put a "case OperationCode.GetRegions:" into your OnOperationResponse method to notice when the result is available.

8.81.2.3 string Photon.Realtime.RegionHandler.SummaryToCache `[get]`

This value summarizes the results of pinging the currently available EnabledRegions (after PingMinimumOfRegions finished).

8.82 Photon.Realtime.RegionPinger Class Reference

Public Member Functions

- **RegionPinger** ([Region](#) region, Action< [Region](#) > onDoneCallback)
- bool **Start** ()

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

- int **CurrentAttempt** = 0

Static Public Attributes

- static int **Attempts** = 5
- static bool **IgnoreInitialAttempt** = true
- static int **MaxMilliseconsPerPing** = 800
- static int **PingWhenFailed** = Attempts ∗ MaxMilliseconsPerPing

Properties

- bool **Done** `[get]`

8.82.1 Member Function Documentation

8.82.1.1 static string Photon.Realtime.RegionPinger.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.ToString().Contains("6")) // ipv6...

Parameters

hostName	Hostname to resolve.

Returns

IP string or empty string if resolution fails

8.83 Photon.Realtime.Room Class Reference

This class represents a room a client joins/joined.

Inherits [Photon.Realtime.RoomInfo](#).

Public Member Functions

- virtual void SetCustomProperties (Hashtable propertiesToSet, Hashtable expectedProperties=null, WebFlags webFlags=null)

    Updates and synchronizes this Room's Custom Properties. Optionally, expectedProperties can be provided as condition.

- void SetPropertiesListedInLobby (string[ ] propertiesListedInLobby)

    Enables you to define the properties available in the lobby if not all properties are needed to pick a room.

- bool SetMasterClient (Player masterClientPlayer)

    Asks the server to assign another player as Master Client of your current room.

- virtual bool AddPlayer (Player player)

    Checks if the player is in the room's list already and calls StorePlayer() if not.

- virtual Player StorePlayer (Player player)

    Updates a player reference in the Players dictionary (no matter if it existed before or not).

- virtual Player GetPlayer (int id)

    Tries to find the player with given actorNumber (a.k.a. ID). Only useful when in a Room, as IDs are only valid per 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.

- bool **IsOffline** [get]
- new bool IsOpen [get, set]

    Defines if the room can be joined.

- new bool IsVisible [get, set]

    Defines if the room is listed in its lobby.

- new byte MaxPlayers [get, set]

    Sets a limit of players to this room. This property is synced and shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

- new byte PlayerCount [get]

    The count of players in this Room (using this.Players.Count).

- Dictionary< int, Player > Players [get]

    While inside a Room, this is the list of players who are also in that 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.

- int PlayerTtl [get, set]

    Player Time To Live. How long any player can be inactive (due to disconnect or leave) before the user gets removed from the playerlist (freeing a slot).

- int EmptyRoomTtl [get, set]

    Room Time To Live. How long a room stays available (and in server-memory), after the last player becomes inactive. After this time, the room gets persisted or destroyed.

- int MasterClientId [get]

    The ID (actorNumber, actorNumber) of the player who's the master of this Room. Note: This changes when the current master leaves the room.

- string[] PropertiesListedInLobby [get]

    Gets a list of custom properties that are in the RoomInfo of the Lobby. This list is defined when creating the room and can't be changed afterwards. Compare: LoadBalancingClient.OpCreateRoom()

- bool AutoCleanUp [get]

    Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.

Additional Inherited Members

8.83.1 Detailed Description

This class represents a room a client joins/joined.

Contains a list of current players, their properties and those of this room, too. A room instance has a number of "well known" properties like IsOpen, MaxPlayers which can be changed. Your own, custom properties can be set via SetCustomProperties() while being in the room.

Typically, this class should be extended by a game-specific implementation with logic and extra features.

8.83.2 Member Function Documentation

8.83.2.1 virtual bool Photon.Realtime.Room.AddPlayer (Player *player*) [virtual]

Checks if the player is in the room's list already and calls StorePlayer() if not.

Parameters

player	The new player - identified by ID.

Returns

    False if the player could not be added (cause it was in the list already).

8.83.2.2 void Photon.Realtime.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.

8.83.2.3 virtual Player Photon.Realtime.Room.GetPlayer (int *id*) [virtual]

Tries to find the player with given actorNumber (a.k.a. ID). Only useful when in a Room, as IDs are only valid per Room.

Parameters

id	ID to look for.

Returns

    The player with the ID or null.

8.83.2.4   virtual void Photon.Realtime.Room.SetCustomProperties (Hashtable *propertiesToSet,* Hashtable *expectedProperties =* `null`, WebFlags *webFlags =* `null`) `[virtual]`

Updates and synchronizes this Room's Custom Properties. Optionally, expectedProperties can be provided as condition.

Custom Properties are a set of string keys and arbitrary values which is synchronized for the players in a Room. They are available when the client enters the room, as they are in the response of OpJoin and OpCreate.

Custom Properties either relate to the (current) Room or a Player (in that Room).

Both classes locally cache the current key/values and make them available as property: CustomProperties. This is provided only to read them. You must use the method SetCustomProperties to set/modify them.

Any client can set any Custom Properties anytime (when in a room). It's up to the game logic to organize how they are best used.

You should call SetCustomProperties only with key/values that are new or changed. This reduces traffic and performance.

Unless you define some expectedProperties, setting key/values is always permitted. In this case, the property-setting client will not receive the new values from the server but instead update its local cache in SetCustom↩ Properties.

If you define expectedProperties, the server will skip updates if the server property-cache does not contain all expectedProperties with the same values. In this case, the property-setting client will get an update from the server and update it's cached key/values at about the same time as everyone else.

The benefit of using expectedProperties can be only one client successfully sets a key from one known value to another. As example: Store who owns an item in a Custom Property "ownedBy". It's 0 initally. When multiple players reach the item, they all attempt to change "ownedBy" from 0 to their actorNumber. If you use expectedProperties {"ownedBy", 0} as condition, the first player to take the item will have it (and the others fail to set the ownership).

Properties get saved with the game state for Turnbased games (which use IsPersistent = true).

Parameters

propertiesToSet	Hashtable of Custom Properties that changes.
expected↩ Properties	Provide some keys/values to use as condition for setting the new values. Client must be in room.
webFlags	Defines if this SetCustomProperties-operation gets forwarded to your WebHooks. Client must be in room.

8.83.2.5   bool Photon.Realtime.Room.SetMasterClient (Player *masterClientPlayer*)

Asks the server to assign another player as Master Client of your current room.

RaiseEvent has 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: MasterClientId

Parameters

masterClient↩ Player	The player to become the next Master Client.

Returns

      False when this operation couldn't be done currently. Requires a v4 Photon Server.

8.83.2.6   void Photon.Realtime.Room.SetPropertiesListedInLobby (string[ ] *propertiesListedInLobby*)

Enables you to define the properties available in the lobby if not all properties are needed to pick a room.

Limit the amount of properties sent to users in the lobby to improve speed and stability.

Parameters

properties↩ ListedInLobby	An array of custom room property names to forward to the lobby.

8.83.2.7   virtual Player Photon.Realtime.Room.StorePlayer (Player *player*)   `[virtual]`

Updates a player reference in the Players dictionary (no matter if it existed before or not).

Parameters

player	The Player instance to insert into the room.

8.83.2.8   override string Photon.Realtime.Room.ToString (   )

Returns a summary of this Room instance as string.

Returns

      Summary of this Room instance.

8.83.2.9   new string Photon.Realtime.Room.ToStringFull (   )

Returns a summary of this Room instance as longer string, including Custom Properties.

Returns

      Summary of this Room instance.

8.83.3   Property Documentation

8.83.3.1   bool Photon.Realtime.Room.AutoCleanUp   `[get]`

Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.

8.83.3.2   int Photon.Realtime.Room.EmptyRoomTtl   `[get],[set]`

Room Time To Live. How long a room stays available (and in server-memory), after the last player becomes inactive. After this time, the room gets persisted or destroyed.

8.83.3.3 string [ ] Photon.Realtime.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.

8.83.3.4 new bool Photon.Realtime.Room.IsOpen `[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 while users are trying to join. 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.

8.83.3.5 new bool Photon.Realtime.Room.IsVisible `[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.

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.

8.83.3.6 int Photon.Realtime.Room.MasterClientId `[get]`

The ID (actorNumber, actorNumber) of the player who's the master of this Room. Note: This changes when the current master leaves the room.

8.83.3.7 new byte Photon.Realtime.Room.MaxPlayers `[get],[set]`

Sets a limit of players to this room. This property is synced and 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.

8.83.3.8 new string Photon.Realtime.Room.Name `[get],[set]`

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

The name can't be changed once it's set by the server.

8.83.3.9 new byte Photon.Realtime.Room.PlayerCount `[get]`

The count of players in this Room (using this.Players.Count).

8.83.3.10 Dictionary<int, Player> Photon.Realtime.Room.Players `[get]`

While inside a Room, this is the list of players who are also in that room.

8.83.3.11 int Photon.Realtime.Room.PlayerTtl `[get],[set]`

Player Time To Live. How long any player can be inactive (due to disconnect or leave) before the user gets removed from the playerlist (freeing a slot).

8.83.3.12 string [ ] Photon.Realtime.Room.PropertiesListedInLobby `[get]`

Gets a list of custom properties that are in the RoomInfo of the Lobby. This list is defined when creating the room and can't be changed afterwards. Compare: LoadBalancingClient.OpCreateRoom()

You could name properties that are not set from the beginning. Those will be synced with the lobby when added later on.

8.84 Photon.Realtime.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 (IsOpen, MaxPlayers, etc).

Inherited by Photon.Realtime.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 ()

    Returns most interesting room values as string.
- string ToStringFull ()

    Returns most interesting room values as string, including custom properties.

Public Attributes

- bool removedFromList

    Used in lobby, to mark rooms that are no longer listed (for being full, closed or hidden).

Protected Attributes

- byte maxPlayers = 0

    Backing field for property.
- int emptyRoomTtl = 0

    Backing field for property.
- int playerTtl = 0

    Backing field for property.
- string[ ] expectedUsers

    Backing field for property.
- bool isOpen = true

    Backing field for property.
- bool isVisible = true

    Backing field for property.
- bool autoCleanUp = true

Backing field for property. False unless the GameProperty is set to true (else it's not sent).

- string name

    Backing field for property.

- string[ ] propertiesListedInLobby

    Backing field for property.

Properties

- 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 for a room/match (per AppId + game-Version).

- int PlayerCount `[get]`

    Count of players currently in room. This property is overwritten by the Room class (used when you're in a Room).

- byte MaxPlayers `[get]`

    The limit of players for this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

- bool IsOpen `[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 even while you join them. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.

- bool IsVisible `[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.

8.84.1   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 (IsOpen, 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).

8.84.2   Member Function Documentation

8.84.2.1   override bool Photon.Realtime.RoomInfo.Equals (object *other*)

Makes RoomInfo comparable (by name).

8.84.2.2   override int Photon.Realtime.RoomInfo.GetHashCode ( )

Accompanies Equals, using the name's HashCode as return.

Returns

8.84.2.3 override string Photon.Realtime.RoomInfo.ToString ( )

Returns most interesting room values as string.

Returns

Summary of this [RoomInfo](#) instance.

8.84.2.4 string Photon.Realtime.RoomInfo.ToStringFull ( )

Returns most interesting room values as string, including custom properties.

Returns

Summary of this [RoomInfo](#) instance.

8.84.3 Member Data Documentation

8.84.3.1 bool Photon.Realtime.RoomInfo.autoCleanUp = true `[protected]`

Backing field for property. False unless the GameProperty is set to true (else it's not sent).

8.84.3.2 int Photon.Realtime.RoomInfo.emptyRoomTtl = 0 `[protected]`

Backing field for property.

8.84.3.3 string [ ] Photon.Realtime.RoomInfo.expectedUsers `[protected]`

Backing field for property.

8.84.3.4 bool Photon.Realtime.RoomInfo.isOpen = true `[protected]`

Backing field for property.

8.84.3.5 bool Photon.Realtime.RoomInfo.isVisible = true `[protected]`

Backing field for property.

8.84.3.6 byte Photon.Realtime.RoomInfo.maxPlayers = 0 `[protected]`

Backing field for property.

8.84.3.7 string Photon.Realtime.RoomInfo.name `[protected]`

Backing field for property.

8.84.3.8 int Photon.Realtime.RoomInfo.playerTtl = 0 `[protected]`

Backing field for property.

8.84.3.9   string [ ] Photon.Realtime.RoomInfo.propertiesListedInLobby `[protected]`

Backing field for property.

8.84.3.10   bool Photon.Realtime.RoomInfo.removedFromList

Used in lobby, to mark rooms that are no longer listed (for being full, closed or hidden).

8.84.4   Property Documentation

8.84.4.1   Hashtable Photon.Realtime.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

8.84.4.2   bool Photon.Realtime.RoomInfo.IsOpen `[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 even while you join them. 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.

8.84.4.3   bool Photon.Realtime.RoomInfo.IsVisible `[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.

8.84.4.4   byte Photon.Realtime.RoomInfo.MaxPlayers `[get]`

The limit of players for 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.

8.84.4.5   string Photon.Realtime.RoomInfo.Name `[get]`

The name of a room. Unique identifier for a room/match (per AppId + game-Version).

8.84.4.6   int Photon.Realtime.RoomInfo.PlayerCount `[get]`

Count of players currently in room. This property is overwritten by the Room class (used when you're in a Room).

8.85 Photon.Realtime.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.
- int EmptyRoomTtl

  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.
- 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]`

  Removes a user's events and properties from the room when a user leaves.
- bool SuppressRoomEvents `[get, set]`

  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.
- bool DeleteNullProperties `[get, set]`

  Optionally, properties get deleted, when null gets assigned as value. Defaults to off / false.
- bool BroadcastPropsChangeToAll `[get, set]`

  By default, property changes are sent back to the client that's setting them to avoid de-sync when properties are set concurrently.

8.85.1 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.

8.85.2 Member Data Documentation

8.85.2.1 Hashtable Photon.Realtime.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").

8.85.2.2 string [ ] Photon.Realtime.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.

8.85.2.3 int Photon.Realtime.RoomOptions.EmptyRoomTtl

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.

8.85.2.4 byte Photon.Realtime.RoomOptions.MaxPlayers

Max number of players that can be in the room at any time. 0 means "no limit".

8.85.2.5 int Photon.Realtime.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.

8.85.2.6 string [ ] Photon.Realtime.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.

8.85.3 Property Documentation

8.85.3.1 bool Photon.Realtime.RoomOptions.BroadcastPropsChangeToAll `[get],[set]`

By default, property changes are sent back to the client that's setting them to avoid de-sync when properties are set concurrently.

This option is enables by default to fix this scenario:

1) On server, room property ABC is set to value FOO, which triggers notifications to all the clients telling them that the property changed. 2) While that notification is in flight, a client sets the ABC property to value BAR. 3) Client receives notification from the server and changes its local copy of ABC to FOO. 4) Server receives the set operation and changes the official value of ABC to BAR, but never notifies the client that sent the set operation that the value is now BAR.

Without this option, the client that set the value to BAR never hears from the server that the official copy has been updated to BAR, and thus gets stuck with a value of FOO.

8.85.3.2 bool Photon.Realtime.RoomOptions.CleanupCacheOnLeave `[get],[set]`

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.

8.85.3.3 **bool Photon.Realtime.RoomOptions.DeleteNullProperties** `[get],[set]`

Optionally, properties get deleted, when null gets assigned as value. Defaults to off / false.

When Op SetProperties is setting a key's value to null, the server and clients should remove the key/value from the Custom Properties. By default, the server keeps the keys (and null values) and sends them to joining players.

Important: Only when SetProperties does a "broadcast", the change (key, value = null) is sent to clients to update accordingly. This applies to Custom Properties for rooms and actors/players.

8.85.3.4 **bool Photon.Realtime.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).

8.85.3.5 **bool Photon.Realtime.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.

8.85.3.6 **bool Photon.Realtime.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.).

8.85.3.7 **bool Photon.Realtime.RoomOptions.SuppressRoomEvents** `[get],[set]`

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.

8.86 Photon.Pun.SceneManagerHelper Class Reference

Properties

- static string **ActiveSceneName** `[get]`
- static int **ActiveSceneBuildIndex** `[get]`

8.87 Photon.Pun.UtilityScripts.ScoreExtensions Class Reference

Static Public Member Functions

- static void **SetScore** (this Player player, int newScore)

- static void **AddScore** (this Player player, int scoreToAddToCurrent)
- static int **GetScore** (this Player player)

8.88 Photon.Pun.ServerSettings Class Reference

Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

Inherits ScriptableObject.

Public Member Functions

- void **UseCloud** (string cloudAppid, string code="")
- override string **ToString** ()

Static Public Member Functions

- static bool IsAppId (string val)

    Checks if a string is a Guid by attempting to create one.
- static void **ResetBestRegionCodeInPreferences** ()

Public Attributes

- AppSettings **AppSettings**
- bool **StartInOfflineMode**
- PunLogLevel **PunLogging** = PunLogLevel.ErrorsOnly
- bool **EnableSupportLogger**
- bool **RunInBackground** = true
- List< string > **RpcList** = new List<string>()
- bool **DisableAutoOpenWizard**

Properties

- static string BestRegionSummaryInPreferences `[get]`

    Gets the best region code in preferences. This composes the PhotonHandler, since its Internal and can not be accessed by the custom inspector

8.88.1 Detailed Description

Collection of connection-relevant settings, used internally by PhotonNetwork.ConnectUsingSettings.

8.88.2 Member Function Documentation

8.88.2.1 static bool Photon.Pun.ServerSettings.IsAppId (string *val*) `[static]`

Checks if a string is a Guid by attempting to create one.

Parameters

	val	The potential guid to check.

Returns

True if new Guid(val) did not fail.

8.88.3 Property Documentation

8.88.3.1 string Photon.Pun.ServerSettings.BestRegionSummaryInPreferences `[static],[get]`

Gets the best region code in preferences. This composes the PhotonHandler, since its Internal and can not be accessed by the custom inspector

The best region code in preferences.

8.89 Photon.Pun.UtilityScripts.SmoothSyncMovement Class Reference

Smoothed out movement for network gameobjects

Inherits Photon.Pun.MonoBehaviourPun, and Photon.Pun.IPunObservable.

Public Member Functions

- void **Awake** ()
- 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.
- void **Update** ()

Public Attributes

- float **SmoothingDelay** = 5

Additional Inherited Members

8.89.1 Detailed Description

Smoothed out movement for network gameobjects

8.89.2 Member Function Documentation

8.89.2.1 void Photon.Pun.UtilityScripts.SmoothSyncMovement.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.SerializationRate 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 Photon↩
View.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 Photon.Pun.IPunObservable.

8.90 Photon.Pun.UtilityScripts.StatesGui Class Reference

Output detailed information about Pun Current states, using the old Unity UI framework.

Inherits MonoBehaviour.

Public Attributes

- Rect **GuiOffset** = new Rect(250, 0, 300, 300)
- bool **DontDestroy** = true
- bool **ServerTimestamp**
- bool **DetailedConnection**
- bool **Server**
- bool **AppVersion**
- bool **UserId**
- bool **Room**
- bool **RoomProps**
- bool **LocalPlayer**
- bool **PlayerProps**
- bool **Others**
- bool **Buttons**
- bool **ExpectedUsers**

8.90.1 Detailed Description

Output detailed information about Pun Current states, using the old Unity UI framework.

8.91 Photon.Realtime.SupportLogger Class Reference

Helper class to debug log basic information about Photon client and vital traffic statistics.

Inherits Photon.Realtime.IConnectionCallbacks, Photon.Realtime.IInRoomCallbacks, Photon.Realtime.I↩
MatchmakingCallbacks, and Photon.Realtime.ILobbyCallbacks.

Public Member Functions

- void LogStats ()

  Debug logs vital traffic statistics about the attached Photon Client.
- void OnConnected ()

Called to signal that the "low level connection" got established but before the client can call operation on the server.

• void OnConnectedToMaster ()

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

• void OnFriendListUpdate (List< FriendInfo > friendList)

Called when the server sent the response to a FindFriends request.

• void OnJoinedLobby ()

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

• void OnLeftLobby ()

Called after leaving a lobby.

• void OnCreateRoomFailed (short returnCode, string message)

Called when the server couldn't create a room (OpCreateRoom failed).

• void OnJoinedRoom ()

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

• void OnJoinRoomFailed (short returnCode, string message)

Called when a previous OpJoinRoom call failed on the server.

• void OnJoinRandomFailed (short returnCode, string message)

Called when a previous OpJoinRandom call failed on the server.

• void OnCreatedRoom ()

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

• void OnLeftRoom ()

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

• void OnDisconnected (DisconnectCause cause)

Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call

• void OnRegionListReceived (RegionHandler regionHandler)

Called when the Name Server provided a list of regions for your title.

• void OnRoomListUpdate (List< RoomInfo > roomList)

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

• void OnPlayerEnteredRoom (Player newPlayer)

Called when a remote player entered the room. This Player is already added to the playerlist.

• void OnPlayerLeftRoom (Player otherPlayer)

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

• void OnRoomPropertiesUpdate (Hashtable propertiesThatChanged)

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

• void OnPlayerPropertiesUpdate (Player targetPlayer, Hashtable changedProps)

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

• void OnMasterClientSwitched (Player newMasterClient)

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

• void OnCustomAuthenticationResponse (Dictionary< string, object > data)

Called when your Custom Authentication service responds with additional data.

• void OnCustomAuthenticationFailed (string debugMessage)

Called when the custom authentication failed. Followed by disconnect!

• void OnLobbyStatisticsUpdate (List< TypedLobbyInfo > lobbyStatistics)

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

Public Attributes

• bool LogTrafficStats

Toggle to enable or disable traffic statistics logging.

Properties

- **LoadBalancingClient Client** `[get, set]`

    Photon client to log information and statistics from.

8.91.1 Detailed Description

Helper class to debug log basic information about Photon client and vital traffic statistics.

Set SupportLogger.Client for this to work.

8.91.2 Member Function Documentation

8.91.2.1 void Photon.Realtime.SupportLogger.LogStats (   )

Debug logs vital traffic statistics about the attached Photon Client.

8.91.2.2 void Photon.Realtime.SupportLogger.OnConnected (   )

Called to signal that the "low level connection" got established but before the client can call operation on the server.

After the (low level transport) connection is established, the client will automatically send the Authentication operation, which needs to get a response before the client can call other operations.

Your logic should wait for either: OnRegionListReceived or OnConnectedToMaster.

This callback is useful to detect if the server can be reached at all (technically). Most often, it's enough to implement OnDisconnected(DisconnectCause cause) and check for the cause.

This is not called for transitions from the masterserver to game servers.

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.3 void Photon.Realtime.SupportLogger.OnConnectedToMaster (   )

Called when the client is connected to the Master Server and ready for matchmaking and other tasks.

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

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.4 void Photon.Realtime.SupportLogger.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 OpCreateRoom).

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", implement OnMasterClientSwitched() and make each new MasterClient check the room's state.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.5 void Photon.Realtime.SupportLogger.OnCreateRoomFailed (short *returnCode,* string *message*)

Called when the server couldn't create a room (OpCreateRoom failed).

Creating a room may fail for various reasons. Most often, the room already exists (roomname in use) or the Room↩ Options clash and it's impossible to create the room.

When creating a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.6   void Photon.Realtime.SupportLogger.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.

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.7   void Photon.Realtime.SupportLogger.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-us/realtime/current/reference/custom-authentication

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.8   void Photon.Realtime.SupportLogger.OnDisconnected (DisconnectCause *cause*)

Called after disconnecting from the Photon server. It could be a failure or an explicit disconnect call

The reason for this disconnect is provided as DisconnectCause.

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.9   void Photon.Realtime.SupportLogger.OnFriendListUpdate (List< FriendInfo > *friendList*)

Called when the server sent the response to a FindFriends request.

After calling OpFindFriends, the Master Server will cache the friend list and send updates to the friend list. The friends includes the name, userId, online state and the room (if any) for each requested user/friend.

Use the friendList to update your UI and store it, if the UI should highlight changes.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.10 void Photon.Realtime.SupportLogger.OnJoinedLobby ( )

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

While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify in the public cloud). The room list gets available via OnRoomListUpdate.

Implements Photon.Realtime.ILobbyCallbacks.

8.91.2.11 void Photon.Realtime.SupportLogger.OnJoinedRoom ( )

Called when the LoadBalancingClient entered a room, no matter if this client created it or simply joined.

When this is called, you can access the existing players in Room.Players, their custom properties and Room.↩
CustomProperties.

In this callback, you could create player objects. For example in Unity, instantiate a prefab for the player.

If you want a match to be started "actively", enable the user to signal "ready" (using OpRaiseEvent or a Custom Property).

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.12 void Photon.Realtime.SupportLogger.OnJoinRandomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRandom call failed on the server.

The most common causes are that a room is full or does not exist (due to someone else being faster or closing the room).

This operation is only ever sent to the Master Server. Once a room is found by the Master Server, the client will head off to the designated Game Server and use the operation Join on the Game Server.

When using multiple lobbies (via OpJoinLobby or a TypedLobby parameter), another lobby might have more/fitting rooms.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.13 void Photon.Realtime.SupportLogger.OnJoinRoomFailed (short *returnCode,* string *message*)

Called when a previous OpJoinRoom call failed on the server.

Joining a room may fail for various reasons. Most often, the room is full or does not exist anymore (due to someone else being faster or closing the room).

When joining a room fails on a Game Server: The client will cache the failure internally and returns to the Master Server before it calls the fail-callback. This way, the client is ready to find/create a room at the moment of the callback. In this case, the client skips calling OnConnectedToMaster but returning to the Master Server will still call OnConnected. Treat callbacks of OnConnected as pure information that the client could connect.

Parameters

returnCode	Operation ReturnCode from the server.
message	Debug message for the error.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.14   void Photon.Realtime.SupportLogger.OnLeftLobby (   )

Called after leaving a lobby.

When you leave a lobby, OpCreateRoom and OpJoinRandomRoom automatically refer to the default lobby.

Implements Photon.Realtime.ILobbyCallbacks.

8.91.2.15   void Photon.Realtime.SupportLogger.OnLeftRoom (   )

Called when the local user/client left a room, so the game's logic can clean up it's internal state.

When leaving a room, the LoadBalancingClient will disconnect the Game Server and connect to the Master Server. This wraps up multiple internal actions.

Wait for the callback OnConnectedToMaster, before you use lobbies and join or create rooms.

Implements Photon.Realtime.IMatchmakingCallbacks.

8.91.2.16   void Photon.Realtime.SupportLogger.OnLobbyStatisticsUpdate (List< TypedLobbyInfo > *lobbyStatistics*)

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 Photon.Realtime.ILobbyCallbacks.

8.91.2.17   void Photon.Realtime.SupportLogger.OnMasterClientSwitched (Player *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.

Implements Photon.Realtime.IInRoomCallbacks.

8.91.2.18   void Photon.Realtime.SupportLogger.OnPlayerEnteredRoom (Player *newPlayer*)

Called when a remote player entered the room. This Player is already added to the playerlist.

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 Photon.Realtime.IInRoomCallbacks.

8.91.2.19   void Photon.Realtime.SupportLogger.OnPlayerLeftRoom (Player *otherPlayer*)

Called when a remote player left the room or became inactive. Check otherPlayer.IsInactive.

If another player leaves the room or if the server detects a lost connection, this callback will be used to notify your game logic.

Depending on the room's setup, players may become inactive, which means they may return and retake their spot in the room. In such cases, the Player stays in the Room.Players dictionary.

If the player is not just inactive, it gets removed from the Room.Players dictionary, before the callback is called.

Implements Photon.Realtime.IInRoomCallbacks.

8.91.2.20   void Photon.Realtime.SupportLogger.OnPlayerPropertiesUpdate (Player *targetPlayer,* Hashtable *changedProps*)

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

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

Parameters

targetPlayer	Contains Player that changed.
changedProps	Contains the properties that changed.

Implements Photon.Realtime.IInRoomCallbacks.

8.91.2.21   void Photon.Realtime.SupportLogger.OnRegionListReceived (RegionHandler *regionHandler*)

Called when the Name Server provided a list of regions for your title.

Check the RegionHandler class description, to make use of the provided values.

Parameters

regionHandler	The currently used RegionHandler.

Implements Photon.Realtime.IConnectionCallbacks.

8.91.2.22   void Photon.Realtime.SupportLogger.OnRoomListUpdate (List< RoomInfo > *roomList*)

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

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 Photon.Realtime.ILobbyCallbacks.

8.91.2.23   void Photon.Realtime.SupportLogger.OnRoomPropertiesUpdate (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	

Implements Photon.Realtime.IInRoomCallbacks.

8.91.3   Member Data Documentation

8.91.3.1   bool Photon.Realtime.SupportLogger.LogTrafficStats

Toggle to enable or disable traffic statistics logging.

8.91.4 Property Documentation

8.91.4.1 LoadBalancingClient Photon.Realtime.SupportLogger.Client `[get],[set]`

Photon client to log information and statistics from.

8.92 Photon.Pun.PhotonAnimatorView.SynchronizedLayer Class Reference

Public Attributes

- SynchronizeType **SynchronizeType**
- int **LayerIndex**

8.93 Photon.Pun.PhotonAnimatorView.SynchronizedParameter Class Reference

Public Attributes

- ParameterType **Type**
- SynchronizeType **SynchronizeType**
- string **Name**

8.94 Photon.Pun.UtilityScripts.TabViewManager.Tab Class Reference

Public Attributes

- string **ID** = ""
- Toggle **Toggle**
- RectTransform **View**

8.95 Photon.Pun.UtilityScripts.TabViewManager.TabChangeEvent Class Reference

Tab change event.

Inherits UnityEvent< string >.

8.95.1 Detailed Description

Tab change event.

8.96 Photon.Pun.UtilityScripts.TabViewManager Class Reference

Tab view manager. Handles Tab views activation and deactivation, and provides a Unity Event Callback when a tab was selected.

Inherits MonoBehaviour.

Classes

- class Tab
- class TabChangeEvent

  Tab change event.

Public Member Functions

- void SelectTab (string id)

  Selects a given tab.

Public Attributes

- ToggleGroup ToggleGroup

  The toggle group component target.
- Tab[ ] Tabs

  all the tabs for this group
- TabChangeEvent OnTabChanged

  The on tab changed Event.

Protected Attributes

- Tab **CurrentTab**

8.96.1   Detailed Description

Tab view manager. Handles Tab views activation and deactivation, and provides a Unity Event Callback when a tab was selected.

8.96.2   Member Function Documentation

8.96.2.1   void Photon.Pun.UtilityScripts.TabViewManager.SelectTab (string *id*)

Selects a given tab.

Parameters

id	Tab Id

8.96.3   Member Data Documentation

8.96.3.1   TabChangeEvent Photon.Pun.UtilityScripts.TabViewManager.OnTabChanged

The on tab changed Event.

8.96.3.2   Tab [ ] Photon.Pun.UtilityScripts.TabViewManager.Tabs

all the tabs for this group

8.96.3.3   ToggleGroup Photon.Pun.UtilityScripts.TabViewManager.ToggleGroup

The toggle group component target.

8.97   Photon.Pun.UtilityScripts.TeamExtensions Class Reference

Extension used for PunTeams and Player class. Wraps access to the player's custom property.

Static Public Member Functions

- static PunTeams.Team GetTeam (this Player player)

   Extension for Player class to wrap up access to the player's custom property.
- static void SetTeam (this Player player, PunTeams.Team team)

   Switch that player's team to the one you assign.

8.97.1   Detailed Description

Extension used for PunTeams and Player class. Wraps access to the player's custom property.

8.97.2   Member Function Documentation

8.97.2.1   static PunTeams.Team Photon.Pun.UtilityScripts.TeamExtensions.GetTeam (this Player *player*) `[static]`

Extension for Player class to wrap up access to the player's custom property.

Returns

   PunTeam.Team.none if no team was found (yet).

8.97.2.2   static void Photon.Pun.UtilityScripts.TeamExtensions.SetTeam (this Player *player,* PunTeams.Team *team*) `[static]`

Switch that player's team to the one you assign.

Internally checks if this player is in that team already or not. Only team switches are actually sent.

Parameters

player	
team	

8.98   Photon.Pun.UtilityScripts.TextButtonTransition Class Reference

Use this on Button texts to have some color transition on the text as well without corrupting button's behaviour.

Inherits MonoBehaviour, IPointerEnterHandler, and IPointerExitHandler.

Public Member Functions

- void **Awake** ()
- void **OnEnable** ()

- void **OnDisable** ()
- void **OnPointerEnter** (PointerEventData eventData)
- void **OnPointerExit** (PointerEventData eventData)

Public Attributes

- Selectable [Selectable](#)

     The selectable Component.

- Color [NormalColor](#) = Color.white

     The color of the normal of the transition state.

- Color [HoverColor](#) = Color.black

     The color of the hover of the transition state.

8.98.1   Detailed Description

Use this on Button texts to have some color transition on the text as well without corrupting button's behaviour.

8.98.2   Member Data Documentation

8.98.2.1   Color Photon.Pun.UtilityScripts.TextButtonTransition.HoverColor = Color.black

The color of the hover of the transition state.

8.98.2.2   Color Photon.Pun.UtilityScripts.TextButtonTransition.NormalColor = Color.white

The color of the normal of the transition state.

8.98.2.3   Selectable Photon.Pun.UtilityScripts.TextButtonTransition.Selectable

The selectable Component.

8.99   Photon.Pun.UtilityScripts.TextToggleIsOnTransition Class Reference

Use this on toggles texts to have some color transition on the text depending on the isOn State.

Inherits MonoBehaviour, IPointerEnterHandler, and IPointerExitHandler.

Public Member Functions

- void **OnEnable** ()
- void **OnDisable** ()
- void **OnValueChanged** (bool isOn)
- void **OnPointerEnter** (PointerEventData eventData)
- void **OnPointerExit** (PointerEventData eventData)

Public Attributes

- Toggle toggle

  The toggle Component.
- Color NormalOnColor = Color.white

  The color of the normal on transition state.
- Color NormalOffColor = Color.black

  The color of the normal off transition state.
- Color HoverOnColor = Color.black

  The color of the hover on transition state.
- Color HoverOffColor = Color.black

  The color of the hover off transition state.

8.99.1 Detailed Description

Use this on toggles texts to have some color transition on the text depending on the isOn State.

8.99.2 Member Data Documentation

8.99.2.1 Color Photon.Pun.UtilityScripts.TextToggleIsOnTransition.HoverOffColor = Color.black

The color of the hover off transition state.

8.99.2.2 Color Photon.Pun.UtilityScripts.TextToggleIsOnTransition.HoverOnColor = Color.black

The color of the hover on transition state.

8.99.2.3 Color Photon.Pun.UtilityScripts.TextToggleIsOnTransition.NormalOffColor = Color.black

The color of the normal off transition state.

8.99.2.4 Color Photon.Pun.UtilityScripts.TextToggleIsOnTransition.NormalOnColor = Color.white

The color of the normal on transition state.

8.99.2.5 Toggle Photon.Pun.UtilityScripts.TextToggleIsOnTransition.toggle

The toggle Component.

8.100 Photon.Pun.UtilityScripts.TurnExtensions Class Reference

Static Public Member Functions

- static void SetTurn (this Room room, int turn, bool setStartTime=false)

  Sets the turn.
- static int GetTurn (this RoomInfo room)

  Gets the current turn from a RoomInfo
- static int GetTurnStart (this RoomInfo room)

  Returns the start time when the turn began. This can be used to calculate how long it's going on.

- static int GetFinishedTurn (this Player player)

  gets the player's finished turn (from the ROOM properties)
- static void SetFinishedTurn (this Player player, int turn)

  Sets the player's finished turn (in the ROOM properties)

Static Public Attributes

- static readonly string TurnPropKey = "Turn"

  currently ongoing turn number
- static readonly string TurnStartPropKey = "TStart"

  start (server) time for currently ongoing turn (used to calculate end)
- static readonly string FinishedTurnPropKey = "FToA"

  Finished Turn of Actor (followed by number)

8.100.1 Member Function Documentation

8.100.1.1 static int Photon.Pun.UtilityScripts.TurnExtensions.GetFinishedTurn (this Player *player* **)** `[static]`

gets the player's finished turn (from the ROOM properties)

Returns

The finished turn index

Parameters

player	Player reference

8.100.1.2 static int Photon.Pun.UtilityScripts.TurnExtensions.GetTurn (this RoomInfo *room* **)** `[static]`

Gets the current turn from a RoomInfo

Returns

The turn index

Parameters

room	RoomInfo reference

8.100.1.3 static int Photon.Pun.UtilityScripts.TurnExtensions.GetTurnStart (this RoomInfo *room* **)** `[static]`

Returns the start time when the turn began. This can be used to calculate how long it's going on.

Returns

The turn start.

Parameters

room	Room.

8.100.1.4 static void Photon.Pun.UtilityScripts.TurnExtensions.SetFinishedTurn (this Player *player,* int *turn*) `[static]`

Sets the player's finished turn (in the ROOM properties)

Parameters

player	Player Reference
turn	Turn Index

8.100.1.5 static void Photon.Pun.UtilityScripts.TurnExtensions.SetTurn (this Room *room,* int *turn,* bool *setStartTime =* `false` **)** `[static]`

Sets the turn.

Parameters

room	Room reference
turn	Turn index
setStartTime	If set to `true` set start time.

8.100.2 Member Data Documentation

8.100.2.1 readonly string Photon.Pun.UtilityScripts.TurnExtensions.FinishedTurnPropKey = "FToA" `[static]`

Finished Turn of Actor (followed by number)

8.100.2.2 readonly string Photon.Pun.UtilityScripts.TurnExtensions.TurnPropKey = "Turn" `[static]`

currently ongoing turn number

8.100.2.3 readonly string Photon.Pun.UtilityScripts.TurnExtensions.TurnStartPropKey = "TStart" `[static]`

start (server) time for currently ongoing turn (used to calculate end)

8.101 Photon.Realtime.TypedLobby Class Reference

Refers to a specific lobby (and type) on the server.

Inherited by Photon.Realtime.TypedLobbyInfo.

Public Member Functions

- **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]`

8.101.1 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.

8.101.2 Member Data Documentation

8.101.2.1 string Photon.Realtime.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.

8.101.2.2 LobbyType Photon.Realtime.TypedLobby.Type

Type of the (named)lobby this game gets added to

8.102 Photon.Realtime.TypedLobbyInfo Class Reference

Inherits Photon.Realtime.TypedLobby.

Public Member Functions

- override string **ToString** ()

Public Attributes

- int **PlayerCount**
- int **RoomCount**

Additional Inherited Members

8.103 Photon.Realtime.WebFlags Class Reference

Optional flags to be used in Photon client SDKs with Op RaiseEvent and Op SetProperties. Introduced mainly for webhooks 1.2 to control behavior of forwarded HTTP requests.

Public Member Functions

- **WebFlags** (byte webhookFlags)

Public Attributes

- byte **WebhookFlags**
- const byte **HttpForwardConst** = 0x01
- const byte **SendAuthCookieConst** = 0x02
- const byte **SendSyncConst** = 0x04
- const byte **SendStateConst** = 0x08

Static Public Attributes

- static readonly WebFlags **Default** = new WebFlags(0)

Properties

- bool HttpForward `[get, set]`

  Indicates whether to forward HTTP request to web service or not.
- bool SendAuthCookie `[get, set]`

  Indicates whether to send AuthCookie of actor in the HTTP request to web service or not.
- bool SendSync `[get, set]`

  Indicates whether to send HTTP request synchronously or asynchronously to web service.
- bool SendState `[get, set]`

  Indicates whether to send serialized game state in HTTP request to web service or not.

8.103.1 Detailed Description

Optional flags to be used in Photon client SDKs with Op RaiseEvent and Op SetProperties. Introduced mainly for webhooks 1.2 to control behavior of forwarded HTTP requests.

8.103.2 Property Documentation

8.103.2.1 bool Photon.Realtime.WebFlags.HttpForward `[get],[set]`

Indicates whether to forward HTTP request to web service or not.

8.103.2.2 bool Photon.Realtime.WebFlags.SendAuthCookie `[get],[set]`

Indicates whether to send AuthCookie of actor in the HTTP request to web service or not.

Indicates whether to send serialized game state in HTTP request to web service or not.

8.103.2.4 **bool Photon.Realtime.WebFlags.SendSync** `[get],[set]`

Indicates whether to send HTTP request synchronously or asynchronously to web service.

8.104 Photon.Realtime.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]`

    Name of the WebRpc that was called.
- int ReturnCode `[get]`

    ReturnCode of the WebService that answered the WebRpc.
- string DebugMessage `[get]`

    Might be empty or null.
- Dictionary< string, object > Parameters `[get]`

    Other key/values returned by the webservice that answered the WebRpc.

8.104.1 Detailed Description

Reads an operation response of a WebRpc and provides convenient access to most common values.

See LoadBalancingClient.OpWebRpc.
Create a WebRpcResponse to access common result values.
The operationResponse.OperationCode should be: OperationCode.WebRpc.

8.104.2 Constructor & Destructor Documentation

8.104.2.1 **Photon.Realtime.WebRpcResponse.WebRpcResponse (OperationResponse** *response* **)**

An OperationResponse for a WebRpc is needed to read it's values.

8.104.3 Member Function Documentation

8.104.3.1 **string Photon.Realtime.WebRpcResponse.ToStringFull ( )**

Turns the response into an easier to read string.

Returns

String resembling the result.

8.104.4 Property Documentation

8.104.4.1 string Photon.Realtime.WebRpcResponse.DebugMessage `[get]`

Might be empty or null.

8.104.4.2 string Photon.Realtime.WebRpcResponse.Name `[get]`

Name of the WebRpc that was called.

8.104.4.3 Dictionary<string, object> Photon.Realtime.WebRpcResponse.Parameters `[get]`

Other key/values returned by the webservice that answered the WebRpc.

8.104.4.4 int Photon.Realtime.WebRpcResponse.ReturnCode `[get]`

ReturnCode of the WebService that answered the WebRpc.

1 is: "OK" for WebRPCs.
-1 is: No ReturnCode by WebRpc service (check OperationResponse.ReturnCode).
Other ReturnCodes are defined by the individual WebRpc and service.

Index