

# Photon Unity Networking

## v1.18

Generated by Doxygen 1.8.2

Fri Apr 19 2013 11:08:57



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>General Documentation</b>	<b>3</b>
<b>3</b>	<b>Gui for Network Simulation</b>	<b>13</b>
<b>4</b>	<b>Gui for Network Statistics</b>	<b>15</b>
<b>5</b>	<b>What is the public PUN api</b>	<b>17</b>
<b>6</b>	<b>Module Index</b>	<b>19</b>
6.1	Modules . . . . .	19
<b>7</b>	<b>Namespace Index</b>	<b>21</b>
7.1	Packages . . . . .	21
<b>8</b>	<b>Hierarchical Index</b>	<b>23</b>
8.1	Class Hierarchy . . . . .	23
<b>9</b>	<b>Class Index</b>	<b>25</b>
9.1	Class List . . . . .	25
<b>10</b>	<b>File Index</b>	<b>27</b>
10.1	File List . . . . .	27
<b>11</b>	<b>Module Documentation</b>	<b>29</b>
11.1	Optional Gui Elements . . . . .	29
11.1.1	Detailed Description . . . . .	29
11.2	Public API . . . . .	30
11.2.1	Detailed Description . . . . .	31
11.2.2	Enumeration Type Documentation . . . . .	31
11.2.2.1	DisconnectCause . . . . .	31
11.2.2.2	PeerState . . . . .	31
11.2.2.3	PhotonLogLevel . . . . .	32
11.2.2.4	PhotonNetworkingMessage . . . . .	32

11.2.2.5 PhotonTargets . . . . .	34
<b>12 Namespace Documentation</b>	<b>35</b>
12.1 Package Photon . . . . .	35
<b>13 Class Documentation</b>	<b>37</b>
13.1 ActorProperties Class Reference . . . . .	37
13.1.1 Detailed Description . . . . .	37
13.1.2 Member Data Documentation . . . . .	37
13.1.2.1 playerName . . . . .	37
13.2 AuthenticationValues Class Reference . . . . .	37
13.2.1 Detailed Description . . . . .	38
13.2.2 Member Function Documentation . . . . .	38
13.2.2.1 SetAuthParameters . . . . .	38
13.2.3 Member Data Documentation . . . . .	38
13.2.3.1 AuthParameters . . . . .	38
13.2.3.2 AuthType . . . . .	38
13.2.3.3 Secret . . . . .	38
13.3 ErrorCode Class Reference . . . . .	39
13.3.1 Detailed Description . . . . .	39
13.3.2 Member Data Documentation . . . . .	40
13.3.2.1 AlreadyMatched . . . . .	40
13.3.2.2 CustomAuthenticationFailed . . . . .	40
13.3.2.3 GameClosed . . . . .	40
13.3.2.4 GameDoesNotExist . . . . .	40
13.3.2.5 GameFull . . . . .	40
13.3.2.6 GameldAlreadyExists . . . . .	40
13.3.2.7 InternalServerError . . . . .	40
13.3.2.8 InvalidAuthentication . . . . .	40
13.3.2.9 InvalidOperationCode . . . . .	40
13.3.2.10 InvalidRegion . . . . .	40
13.3.2.11 MaxCcuReached . . . . .	41
13.3.2.12 NoRandomMatchFound . . . . .	41
13.3.2.13 Ok . . . . .	41
13.3.2.14 OperationNotAllowedInCurrentState . . . . .	41
13.3.2.15 ServerFull . . . . .	41
13.3.2.16 UserBlocked . . . . .	41
13.4 EventCode Class Reference . . . . .	41
13.4.1 Detailed Description . . . . .	42
13.4.2 Member Data Documentation . . . . .	42
13.4.2.1 AppStats . . . . .	42

13.4.2.2	AzureNodeInfo	42
13.4.2.3	GameList	42
13.4.2.4	GameListUpdate	42
13.4.2.5	Join	42
13.4.2.6	Leave	43
13.4.2.7	Match	43
13.4.2.8	PropertiesChanged	43
13.4.2.9	QueueState	43
13.4.2.10	SetProperties	43
13.5	Extensions Class Reference	43
13.5.1	Detailed Description	44
13.5.2	Member Function Documentation	44
13.5.2.1	AlmostEquals	44
13.5.2.2	AlmostEquals	44
13.5.2.3	AlmostEquals	44
13.5.2.4	AlmostEquals	44
13.5.2.5	Contains	44
13.5.2.6	GetPhotonView	44
13.5.2.7	GetPhotonViewsInChildren	44
13.5.2.8	Merge	44
13.5.2.9	MergeStringKeys	45
13.5.2.10	StripKeysWithNullValues	45
13.5.2.11	StripToStringKeys	45
13.5.2.12	ToStringFull	45
13.6	FriendInfo Class Reference	46
13.6.1	Detailed Description	46
13.6.2	Member Function Documentation	46
13.6.2.1	ToString	46
13.6.3	Property Documentation	46
13.6.3.1	IsInRoom	46
13.6.3.2	IsOnline	46
13.6.3.3	Name	46
13.6.3.4	Room	46
13.7	GameProperties Class Reference	46
13.7.1	Detailed Description	47
13.7.2	Member Data Documentation	47
13.7.2.1	CleanupCacheOnLeave	47
13.7.2.2	IsOpen	47
13.7.2.3	IsVisible	47
13.7.2.4	MaxPlayers	47

13.7.2.5	PlayerCount	47
13.7.2.6	PropsListedInLobby	47
13.7.2.7	Removed	47
13.8	Photon.MonoBehaviour Class Reference	48
13.8.1	Detailed Description	48
13.8.2	Property Documentation	48
13.8.2.1	networkView	48
13.8.2.2	photonView	48
13.9	OperationCode Class Reference	48
13.9.1	Detailed Description	49
13.9.2	Member Data Documentation	49
13.9.2.1	Authenticate	49
13.9.2.2	ChangeGroups	49
13.9.2.3	CreateGame	49
13.9.2.4	FindFriends	49
13.9.2.5	GetProperties	49
13.9.2.6	JoinGame	49
13.9.2.7	JoinLobby	49
13.9.2.8	JoinRandomGame	50
13.9.2.9	Leave	50
13.9.2.10	LeaveLobby	50
13.9.2.11	RaiseEvent	50
13.9.2.12	SetProperties	50
13.10	ParameterCode Class Reference	50
13.10.1	Detailed Description	52
13.10.2	Member Data Documentation	52
13.10.2.1	ActorList	52
13.10.2.2	ActorNr	52
13.10.2.3	Add	52
13.10.2.4	Address	52
13.10.2.5	ApplicationId	52
13.10.2.6	AppVersion	52
13.10.2.7	Broadcast	52
13.10.2.8	Cache	52
13.10.2.9	CleanupCacheOnLeave	52
13.10.2.10	ClientAuthenticationParams	52
13.10.2.11	ClientAuthenticationType	53
13.10.2.12	Code	53
13.10.2.13	CustomEventContent	53
13.10.2.14	Data	53

13.10.2.15FindFriendsRequestList . . . . .	53
13.10.2.16FindFriendsResponseOnlineList . . . . .	53
13.10.2.17FindFriendsResponseRoomIdList . . . . .	53
13.10.2.18GameCount . . . . .	53
13.10.2.19GameList . . . . .	53
13.10.2.20GameProperties . . . . .	53
13.10.2.21Group . . . . .	53
13.10.2.22MasterPeerCount . . . . .	54
13.10.2.23MatchMakingType . . . . .	54
13.10.2.24PeerCount . . . . .	54
13.10.2.25PlayerProperties . . . . .	54
13.10.2.26Position . . . . .	54
13.10.2.27Properties . . . . .	54
13.10.2.28ReceiverGroup . . . . .	54
13.10.2.29Remove . . . . .	54
13.10.2.30RoomName . . . . .	54
13.10.2.31Secret . . . . .	54
13.10.2.32TargetActorNr . . . . .	54
13.10.2.33UserId . . . . .	55
13.11 PBitStream Class Reference . . . . .	55
13.11.1 Constructor & Destructor Documentation . . . . .	55
13.11.1.1 PBitStream . . . . .	55
13.11.1.2 PBitStream . . . . .	55
13.11.1.3 PBitStream . . . . .	55
13.11.2 Member Function Documentation . . . . .	55
13.11.2.1 Add . . . . .	55
13.11.2.2 BytesForBits . . . . .	55
13.11.2.3 Get . . . . .	55
13.11.2.4 GetNext . . . . .	55
13.11.2.5 Set . . . . .	55
13.11.2.6 ToBytes . . . . .	55
13.11.3 Property Documentation . . . . .	56
13.11.3.1 BitCount . . . . .	56
13.11.3.2 ByteCount . . . . .	56
13.11.3.3 Position . . . . .	56
13.12 PhotonLagSimulationGui Class Reference . . . . .	56
13.12.1 Detailed Description . . . . .	56
13.12.2 Member Function Documentation . . . . .	57
13.12.2.1 OnGUI . . . . .	57
13.12.2.2 Start . . . . .	57

13.12.3 Member Data Documentation . . . . .	57
13.12.3.1 Visible . . . . .	57
13.12.3.2 WindowId . . . . .	57
13.12.3.3 WindowRect . . . . .	57
13.12.4 Property Documentation . . . . .	57
13.12.4.1 Peer . . . . .	57
13.13 PhotonMessageInfo Class Reference . . . . .	57
13.13.1 Detailed Description . . . . .	58
13.13.2 Constructor & Destructor Documentation . . . . .	58
13.13.2.1 PhotonMessageInfo . . . . .	58
13.13.2.2 PhotonMessageInfo . . . . .	58
13.13.3 Member Function Documentation . . . . .	58
13.13.3.1 ToString . . . . .	58
13.13.4 Member Data Documentation . . . . .	58
13.13.4.1 photonView . . . . .	58
13.13.4.2 sender . . . . .	58
13.13.5 Property Documentation . . . . .	58
13.13.5.1 timestamp . . . . .	58
13.14 PhotonNetwork Class Reference . . . . .	58
13.14.1 Detailed Description . . . . .	63
13.14.2 Member Function Documentation . . . . .	63
13.14.2.1 AllocateViewID . . . . .	63
13.14.2.2 CloseConnection . . . . .	63
13.14.2.3 Connect . . . . .	63
13.14.2.4 Connect . . . . .	63
13.14.2.5 ConnectToBestCloudServer . . . . .	64
13.14.2.6 ConnectUsingSettings . . . . .	64
13.14.2.7 ConnectUsingSettings . . . . .	65
13.14.2.8 CreateRoom . . . . .	65
13.14.2.9 CreateRoom . . . . .	65
13.14.2.10 CreateRoom . . . . .	65
13.14.2.11 Destroy . . . . .	66
13.14.2.12 Destroy . . . . .	66
13.14.2.13 DestroyPlayerObjects . . . . .	66
13.14.2.14 Disconnect . . . . .	66
13.14.2.15 FetchServerTimestamp . . . . .	66
13.14.2.16 FindFriends . . . . .	67
13.14.2.17 GetPing . . . . .	67
13.14.2.18 GetRoomList . . . . .	67
13.14.2.19 InitializeSecurity . . . . .	67



13.14.2.20	Instantiate	67
13.14.2.21	Instantiate	68
13.14.2.22	InstantiateSceneObject	68
13.14.2.23	InternalCleanPhotonMonoFromScenelfStuck	68
13.14.2.24	JoinRandomRoom	69
13.14.2.25	JoinRandomRoom	69
13.14.2.26	JoinRandomRoom	69
13.14.2.27	JoinRoom	69
13.14.2.28	JoinRoom	69
13.14.2.29	LeaveRoom	70
13.14.2.30	LoadLevel	70
13.14.2.31	LoadLevel	70
13.14.2.32	NetworkStatisticsReset	70
13.14.2.33	NetworkStatisticsToString	70
13.14.2.34	OverrideBestCloudServer	70
13.14.2.35	RemoveAllBufferedMessages	70
13.14.2.36	RemoveAllBufferedMessages	71
13.14.2.37	RemoveAllInstantiatedObjects	71
13.14.2.38	RemoveAllInstantiatedObjects	71
13.14.2.39	RemoveRPCs	71
13.14.2.40	RemoveRPCs	71
13.14.2.41	RemoveRPCs	71
13.14.2.42	RemoveRPCsInGroup	71
13.14.2.43	SendOutgoingCommands	71
13.14.2.44	SetLevelPrefix	72
13.14.2.45	SetPlayerCustomProperties	72
13.14.2.46	SetReceivingEnabled	72
13.14.2.47	SetSendingEnabled	72
13.14.2.48	UnAllocateViewID	72
13.14.3	Member Data Documentation	73
13.14.3.1	logLevel	73
13.14.3.2	MAX_VIEW_IDS	73
13.14.3.3	PhotonServerSettings	73
13.14.3.4	precisionForFloatSynchronization	73
13.14.3.5	precisionForQuaternionSynchronization	73
13.14.3.6	precisionForVectorSynchronization	73
13.14.3.7	PrefabCache	73
13.14.3.8	serverSettingsAssetFile	73
13.14.3.9	serverSettingsAssetPath	73
13.14.3.10	UsePrefabCache	74

13.14.3.11versionPUN	74
13.14.4 Property Documentation	74
13.14.4.1 AuthValues	74
13.14.4.2 autoCleanUpPlayerObjects	74
13.14.4.3 autoJoinLobby	74
13.14.4.4 automaticallySyncScene	74
13.14.4.5 connected	75
13.14.4.6 connectionState	75
13.14.4.7 connectionStateDetailed	75
13.14.4.8 countOfPlayers	75
13.14.4.9 countOfPlayersInRooms	75
13.14.4.10countOfPlayersOnMaster	75
13.14.4.11countOfRooms	75
13.14.4.12Friends	75
13.14.4.13FriendsListAge	75
13.14.4.14insideLobby	75
13.14.4.15sMasterClient	76
13.14.4.16sMessageQueueRunning	76
13.14.4.17sNonMasterClientInRoom	76
13.14.4.18masterClient	76
13.14.4.19maxConnections	76
13.14.4.20NetworkStatisticsEnabled	76
13.14.4.21offlineMode	76
13.14.4.22otherPlayers	76
13.14.4.23player	76
13.14.4.24playerList	76
13.14.4.25playerName	77
13.14.4.26room	77
13.14.4.27sendRate	77
13.14.4.28sendRateOnSerialize	77
13.14.4.29ServerAddress	77
13.14.4.30time	77
13.14.4.31unreliableCommandsLimit	77
13.15PhotonPlayer Class Reference	77
13.15.1 Detailed Description	78
13.15.2 Constructor & Destructor Documentation	79
13.15.2.1 PhotonPlayer	79
13.15.2.2 PhotonPlayer	79
13.15.3 Member Function Documentation	79
13.15.3.1 Equals	79

13.15.3.2 Find	79
13.15.3.3 GetHashCode	79
13.15.3.4 SetCustomProperties	79
13.15.3.5 ToString	80
13.15.4 Member Data Documentation	80
13.15.4.1 isLocal	80
13.15.5 Property Documentation	80
13.15.5.1 allProperties	80
13.15.5.2 customProperties	80
13.15.5.3 ID	80
13.15.5.4 isMasterClient	80
13.15.5.5 name	80
13.16 PhotonStatsGui Class Reference	80
13.16.1 Detailed Description	81
13.16.2 Member Function Documentation	81
13.16.2.1 OnGUI	81
13.16.2.2 Start	81
13.16.2.3 TrafficStatsWindow	81
13.16.2.4 Update	81
13.16.3 Member Data Documentation	81
13.16.3.1 buttonsOn	81
13.16.3.2 healthStatsVisible	82
13.16.3.3 statsOn	82
13.16.3.4 statsRect	82
13.16.3.5 statsWindowOn	82
13.16.3.6 trafficStatsOn	82
13.16.3.7 WindowId	82
13.17 PhotonStream Class Reference	82
13.17.1 Detailed Description	83
13.17.2 Constructor & Destructor Documentation	83
13.17.2.1 PhotonStream	83
13.17.3 Member Function Documentation	83
13.17.3.1 ReceiveNext	83
13.17.3.2 SendNext	83
13.17.3.3 Serialize	83
13.17.3.4 Serialize	83
13.17.3.5 Serialize	83
13.17.3.6 Serialize	83
13.17.3.7 Serialize	83
13.17.3.8 Serialize	83

13.17.3.9 Serialize . . . . .	83
13.17.3.10Serialize . . . . .	83
13.17.3.11Serialize . . . . .	83
13.17.3.12Serialize . . . . .	83
13.17.3.13ToArray . . . . .	83
13.17.4 Property Documentation . . . . .	83
13.17.4.1 Count . . . . .	83
13.17.4.2 isReading . . . . .	83
13.17.4.3 isWriting . . . . .	83
13.18PhotonView Class Reference . . . . .	84
13.18.1 Detailed Description . . . . .	85
13.18.2 Member Function Documentation . . . . .	85
13.18.2.1 Awake . . . . .	85
13.18.2.2 Find . . . . .	85
13.18.2.3 Get . . . . .	85
13.18.2.4 Get . . . . .	85
13.18.2.5 OnApplicationQuit . . . . .	85
13.18.2.6 OnDestroy . . . . .	85
13.18.2.7 RPC . . . . .	85
13.18.2.8 RPC . . . . .	85
13.18.2.9 ToString . . . . .	85
13.18.3 Member Data Documentation . . . . .	85
13.18.3.1 group . . . . .	85
13.18.3.2 instantiationId . . . . .	85
13.18.3.3 observed . . . . .	85
13.18.3.4 onSerializeRigidBodyOption . . . . .	85
13.18.3.5 onSerializeTransformOption . . . . .	85
13.18.3.6 ownerId . . . . .	85
13.18.3.7 prefixBackup . . . . .	85
13.18.3.8 subId . . . . .	85
13.18.3.9 synchronization . . . . .	85
13.18.4 Property Documentation . . . . .	86
13.18.4.1 instantiationData . . . . .	86
13.18.4.2 isMine . . . . .	86
13.18.4.3 isSceneView . . . . .	86
13.18.4.4 owner . . . . .	86
13.18.4.5 OwnerActorNr . . . . .	86
13.18.4.6 prefix . . . . .	86
13.18.4.7 viewID . . . . .	86
13.19PingCloudRegions Class Reference . . . . .	86

13.19.1 Member Enumeration Documentation . . . . .	87
13.19.1.1 CloudRegions . . . . .	87
13.19.2 Member Function Documentation . . . . .	87
13.19.2.1 ConnectToBestRegion . . . . .	87
13.19.2.2 OverrideRegion . . . . .	87
13.19.2.3 PingAllRegions . . . . .	87
13.19.2.4 ResolveHost . . . . .	87
13.19.3 Member Data Documentation . . . . .	87
13.19.3.1 SP . . . . .	87
13.20Room Class Reference . . . . .	87
13.20.1 Detailed Description . . . . .	88
13.20.2 Member Function Documentation . . . . .	88
13.20.2.1 SetCustomProperties . . . . .	88
13.20.3 Property Documentation . . . . .	88
13.20.3.1 autoCleanUp . . . . .	89
13.20.3.2 maxPlayers . . . . .	89
13.20.3.3 name . . . . .	89
13.20.3.4 open . . . . .	89
13.20.3.5 playerCount . . . . .	89
13.20.3.6 propertiesListedInLobby . . . . .	89
13.20.3.7 visible . . . . .	89
13.21RoomInfo Class Reference . . . . .	89
13.21.1 Detailed Description . . . . .	90
13.21.2 Member Function Documentation . . . . .	91
13.21.2.1 Equals . . . . .	91
13.21.2.2 GetHashCode . . . . .	91
13.21.2.3 ToString . . . . .	91
13.21.3 Member Data Documentation . . . . .	91
13.21.3.1 autoCleanUpField . . . . .	91
13.21.3.2 maxPlayersField . . . . .	91
13.21.3.3 nameField . . . . .	91
13.21.3.4 openField . . . . .	91
13.21.3.5 visibleField . . . . .	91
13.21.4 Property Documentation . . . . .	91
13.21.4.1 customProperties . . . . .	91
13.21.4.2 isLocalClientInside . . . . .	92
13.21.4.3 maxPlayers . . . . .	92
13.21.4.4 name . . . . .	92
13.21.4.5 open . . . . .	92
13.21.4.6 playerCount . . . . .	92

13.21.4.7 removedFromList . . . . .	92
13.21.4.8 visible . . . . .	92
13.22RpcIndexComponent Class Reference . . . . .	92
13.22.1 Member Function Documentation . . . . .	93
13.22.1.1 Awake . . . . .	93
13.22.1.2 GetShortcut . . . . .	93
13.22.2 Member Data Documentation . . . . .	93
13.22.2.1 RpcIndex . . . . .	93
13.23ServerSettings Class Reference . . . . .	93
13.23.1 Detailed Description . . . . .	94
13.23.2 Member Enumeration Documentation . . . . .	94
13.23.2.1 HostingOption . . . . .	94
13.23.3 Member Function Documentation . . . . .	94
13.23.3.1 FindRegionForServerAddress . . . . .	94
13.23.3.2 FindServerAddressForRegion . . . . .	94
13.23.3.3 ToString . . . . .	94
13.23.3.4 UseCloud . . . . .	94
13.23.3.5 UseMyServer . . . . .	94
13.23.4 Member Data Documentation . . . . .	94
13.23.4.1 AppID . . . . .	95
13.23.4.2 CloudServerRegionNames . . . . .	95
13.23.4.3 CloudServerRegionPrefixes . . . . .	95
13.23.4.4 DefaultAppID . . . . .	95
13.23.4.5 DefaultCloudServerUrl . . . . .	95
13.23.4.6 DefaultMasterPort . . . . .	95
13.23.4.7 DefaultServerAddress . . . . .	95
13.23.4.8 DisableAutoOpenWizard . . . . .	95
13.23.4.9 HostType . . . . .	95
13.23.4.10RpcList . . . . .	95
13.23.4.11ServerAddress . . . . .	95
13.23.4.12ServerPort . . . . .	95
<b>14 File Documentation</b>	<b>97</b>
14.1 _Doc/general.md File Reference . . . . .	97
14.2 _Doc/main.md File Reference . . . . .	97
14.3 _Doc/optionalGui.md File Reference . . . . .	97
14.4 _Doc/photonStatsGui.md File Reference . . . . .	97
14.5 _Doc/publicApi.md File Reference . . . . .	97
14.6 Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs File Reference . . . . .	97
14.7 Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs File Reference . . . . .	97

14.7.1 Enumeration Type Documentation . . . . .	98
14.7.1.1 ConnectionState . . . . .	98
14.8 Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs File Reference . . . . .	98
14.8.1 Typedef Documentation . . . . .	98
14.8.1.1 SupportClass . . . . .	98
14.9 Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs File Reference . . . . .	98
14.10 Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs File Reference . . . . .	99
14.10.1 Enumeration Type Documentation . . . . .	99
14.10.1.1 CustomAuthenticationType . . . . .	99
14.10.1.2 MatchmakingMode . . . . .	100
14.11 Photon Unity Networking/Plugins/PhotonNetwork/NetworkingPeer.cs File Reference . . . . .	100
14.12 Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs File Reference . . . . .	100
14.13 Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs File Reference . . . . .	101
14.14 Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs File Reference . . . . .	101
14.15 Photon Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs File Reference . . . . .	101
14.15.1 Typedef Documentation . . . . .	101
14.15.1.1 Debug . . . . .	101
14.16 Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs File Reference . . . . .	101
14.17 Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs File Reference . . . . .	101
14.18 Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs File Reference . . . . .	102
14.18.1 Enumeration Type Documentation . . . . .	102
14.18.1.1 OnSerializeRigidBody . . . . .	102
14.18.1.2 OnSerializeTransform . . . . .	102
14.18.1.3 ViewSynchronization . . . . .	102
14.19 Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs File Reference . . . . .	102
14.20 Photon Unity Networking/Plugins/PhotonNetwork/Room.cs File Reference . . . . .	103
14.21 Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs File Reference . . . . .	103
14.22 Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs File Reference . . . . .	103
14.23 Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs File Reference . . . . .	103





# Chapter 1

## Main Page

### Introduction

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

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

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

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

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

Most notable features:

- Dead-easy API
- Server available as hosted service (free of charge for development)
- Partially automatic conversion from Unity Networking to PUN
- Offline mode: re-use your multiplayer code in singleplayer game modes
- Outstanding performance of the **Photon** Server
- Load balanced workflow scales across servers (with no extra effort)
- No direct P2P and no NAT punch-through needed

### First Steps

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

To read up on PUN, this documentation is split into a [General Documentation](#) and a [Public API reference](#) documentation.

Aside from that, the source of Photon Unity Networking is available to you.



## Chapter 2

# General Documentation

### Photon

Unlike Unity's built-in networking, PUN always connects to a dedicated server which runs a specific game logic that provides rooms, matchmaking and in-room communication between players. Actually, [Photon](#) Unity Networking uses more than one server behind the scenes: Several "Game Servers" run the actual rooms (matches) while a Master will organize and match rooms.

You have two options for the server side.

### Exit Games Cloud

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

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

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

### Subscriptions bought in Asset Store

Follow these steps, if you bought a package with [Photon](#) Cloud Subscription in the Asset Store:

- Register a Photon Cloud Account: [cloud.exitgames.com](https://cloud.exitgames.com)
- Get your AppID from the Dashboard
- Send a Mail to: [developer@exitgames.com](mailto:developer@exitgames.com)
- With:
  - Your Name and Company (if applicable)
  - Invoice/Purchase ID from the Asset Store
  - Photon Cloud AppID

### Photon Server SDK

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

The Photon v3.2 SDK can be downloaded on: <http://www.exitgames.com/Download/Photon> Starting the Server: <http://doc.exitgames.com/photon-server/PhotonIn5Min>

## First steps

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

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

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

If you are using Unityscript, you'll need to move the **Photon** Unity Networking folder to the root of your project.

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

## Master Server And Lobby

**PhotonNetwork** always uses a master server and one or more game servers. The master server just manages the currently running games on those servers.

**To get a list of currently used rooms, clients have to join the Lobby** on the master server. The lobby automatically sends and updates the room list to clients. This is done in intervals to keep traffic low. Also, players won't notice each other in the Lobby and can't send data (to prevent issues when it's getting crowded).

PUN enters the Lobby by default on connect but, that is not a requirement: Creating and joining rooms is always possible while on the master server. Using JoinRandomRoom, you can distribute your players across matches quickly and hassle-free.

When a player joins or creates a room, this is sent to the master server. The master will forward the clients to a game server where the actual gameplay is happening.

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

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

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

## Versioning

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

## Creating and Joining Games

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

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

---

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

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

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

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

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

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

## Advanced Matchmaking & [Room](#) Properties

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

In [Photon](#) Cloud and Loadbalancing, you can set arbitrary room properties and filter for those in `JoinRandom`.

### [Room](#) Properties and the Lobby

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

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

```
string[] roomPropsInLobby = { "map", "ai" };
Hashtable customRoomProperties = new Hashtable() { { "map", 1 } };
CreateRoom(roomName, true, true, 4, customRoomProperties, roomPropsInLobby);
```

Note that "ai" has no value yet. It won't show up in the lobby until it's set in the game via [Room.SetCustomProperties\(\)](#). When you change the values for "map" or "ai", they will be updated in the lobby with a short delay, too.

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

### Filtering [Room](#) Properties in Join Random

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

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

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

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

## MonoBehaviour Callbacks

[PhotonNetwork](#) implements several callbacks to let your game know about state changes, like “connected” or “joined a game”. Each of the methods used as callback is part of the `PhotonNetworkingMessage` enum. Per enum item, the use is explained (check the tooltip when you type in e.g. `PhotonNetworkingMessage.OnConnectedToPhoton`). You can add these methods on any number of `MonoBehaviours`, they will be called in the respective situation. The complete list of callbacks is also in the [Plugin reference](#).

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

## Sending messages in game rooms

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

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

### Using Groups in PUN

Groups are not synchronized when they are changed on any [PhotonView](#). It's up to the developer to keep photonviews in the same groups on all clients, if that's needed. Using different group numbers for the same photonview on several clients will cause some inconsistent behaviour. Some network messages are checked for their receiver group at the receiver side only, namely: RPCS that are targetted to a single player (or MasterClient) RPCS that are buffered (`AllBuffered/OthersBuffered`). This includes [PhotonNetwork.Instantiate](#) (as it is buffered).

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

## PhotonView

[PhotonView](#) is a script component that is used to send messages (RPCs and `OnSerializePhotonView`). You need to attach the [PhotonView](#) to your games gameobjects. Note that the [PhotonView](#) is very similar to Unity's `NetworkView`.

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

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

### Observe Transform

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

### Observe MonoBehaviour

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

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

```

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

```

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

## Remote Procedure Calls

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

```

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

```

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

```

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

```

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

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

```

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

```

## Timing for RPCs and Loading Levels

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

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

The solution is to pause the message queue, while during scene setup:

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

Disabling the message queue will delay incoming and outgoing messages until the queue is unlocked. Obviously, it's very important to unlock the queue when you're ready to go on.

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

## Various topics

### Differences to Unity Networking

#### 1. Host model

- Unity networking is server-client based (NOT P2P!). Servers are run via a Unity client (so via one of the players)
- [Photon](#) is server-client based as well, but has a dedicated server; No more dropped connections due to hosts leaving.

#### 2. Connectivity

- Unity networking works with NAT punchthrough to try to improve connectivity: since players host the network servers, the connection often fails due to firewalls/routers etc. Connectivity can never be guaranteed, there is a low success rate.
- [Photon](#) has a dedicated server, there is no need for NAT punchthrough or other concepts. Connectivity is a guaranteed 100%. If, in the rare case, a connection fails it must be due to a very strict client side network (a business VPN for example).

#### 3. Performance

- [Photon](#) beats Unity networking performance wise. We do not have the figures to prove this yet but the library has been optimized for years now. Furthermore, since the Unity servers are player hosted latency/ping is usually worse; you rely on the connection of the player acting as server. These connections are never any better than the connection of your dedicated [Photon](#) server.

#### 4. Price

- Like the Unity Networking solution, the [Photon](#) Unity Networking plugin is free as well. You can subscribe to use [Photon](#) Cloud hosting service for your game. Alternatively, you can rent your own servers and run [Photon](#) on them. The free license enables up to 100 concurrent players. Other licenses cost a one-time fee (as you do the hosting) and lift the concurrent user limits.

#### 5. Features & maintenance

- Unity does not seem to give much priority to their Networking implementation. There are rarely feature improvements and bugfixes are as seldom. The [Photon](#) solution is actively maintained and parts of it are available with source code. Furthermore, [Photon](#) already offers more features than Unity, such as the built-in load balancing and offline mode.

#### 6. Master Server

- The Master Server for [Photon](#) is a bit different from the Master Server for plain Unity Networking: In our case, it's a [Photon](#) Server that lists room-names of currently played games in so called "lobbies". Like Unity's Master, it will forward clients to the Game Server(s), where the actual gameplay is done.



## Instantiating objects over the network

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

### PhotonNetwork.Instantiate

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

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

### Gain more control: Manually instantiate

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

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

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

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

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

public Transform playerPrefab; //set this in the inspector

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

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

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

## Offline mode

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

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

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

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

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

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

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

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

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

## Limitations

### Views and players

For performance reasons, the [PhotonNetwork](#) API supports up to 1000 PhotonViews per player and a maximum of 2,147,483 players (note that this is WAY higher than your hardware can support!). You can easily allow for more PhotonViews per player, at the cost of maximum players. This works as follows: PhotonViews send out a viewID for every network message. This viewID is an integer and it is composed of the player ID and the player's view ID. The maximum size of an int is 2,147,483,647, divided by our MAX\_VIEW\_IDS(1000) that allows for over 2 million players, each having 1000 view IDs. As you can see, you can easily increase the player count by reducing the MAX\_VIEW\_IDS. The other way around, you can give all players more VIEW\_IDS at the cost of less maximum players. It is important to note that most games will never need more than a few view ID's per player (one or two for the character..and that's usually it). If you need much more then you might be doing something wrong! It is extremely inefficient to assign a [PhotonView](#) and ID for every bullet that your weapon fires, instead keep track of your fire bullets via the player or weapon's [PhotonView](#).

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

## Groups and Scoping

The [PhotonNetwork](#) plugin does not support real network groups and no scoping yet. While Unity's "scope" feature is not implemented, the network groups are currently implemented purely client side: Any RPC that should be ignored due to grouping, will be discarded after it's received. This way, groups are working but won't save bandwidth.

## Feedback

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

## F.A.Q.

### Can I use multiple PhotonViews per GameObject? Why?

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

### Can I use it from Javascript?

To be able to use the [Photon](#) classes you'll need to move the Plugins folder to the root of your project.

## Converting your Unity networking project to Photon

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

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

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

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

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

This is where [Photon](#) differs most from Unity's solution:

There is only one [Photon](#) server and you connect using the room names. Therefore all references to IPs/ports can be removed from your code (usually GUI code). `PhotonNetwork.JoinRoom(string room)` only takes a room

argument, you'll need to remove your old IP/port/NAT arguments. If you have been using the "Ultimate Unity networking project" by M2H, you should remove the MultiplayerFunctions class.

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

## Chapter 3

# Gui for Network Simulation

As tool for developers, the Photon client library can simulate network conditions for lag (message delay) and loss.

The PUN package contains a small GUI component, to set the relevant values at runtime of a game.

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

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



## Chapter 4

# Gui for Network Statistics

The PhotonStatsGui is a simple GUI component to shows tracked network metrics easily at runtime.

### Usage

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

A few toggles let you configure the window:

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

### Message Statistics

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

### Traffic Statistics

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

### Health Statistics

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

### Button "Reset"

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

### Button "To Log"

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

**Button "Stats On" (Enabling Traffic Stats)**

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

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



## Chapter 5

# What is the public PUN api

The **public api of PUN** consists of any code that is considered useful for you as developer.

These classes are grouped into a "module" in this reference, to make it easier to learn about the important stuff of PUN.



## Chapter 6

# Module Index

### 6.1 Modules

Here is a list of all modules:

Optional Gui Elements . . . . .	29
Public API . . . . .	30



## Chapter 7

# Namespace Index

### 7.1 Packages

Here are the packages with brief descriptions (if available):

<a href="#">Photon</a>	.....	<a href="#">35</a>
------------------------	-------	--------------------



## Chapter 8

# Hierarchical Index

### 8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ActorProperties . . . . .	37
AuthenticationValues . . . . .	37
ErrorCode . . . . .	39
EventCode . . . . .	41
Extensions . . . . .	43
FriendInfo . . . . .	46
GameProperties . . . . .	46
MonoBehaviour	
Photon.MonoBehaviour . . . . .	48
PhotonView . . . . .	84
PhotonLagSimulationGui . . . . .	56
PingCloudRegions . . . . .	86
MonoBehaviour	
PhotonStatsGui . . . . .	80
RpcIndexComponent . . . . .	92
OperationCode . . . . .	48
ParameterCode . . . . .	50
PBitStream . . . . .	55
PhotonMessageInfo . . . . .	57
PhotonNetwork . . . . .	58
PhotonPlayer . . . . .	77
PhotonStream . . . . .	82
RoomInfo . . . . .	89
Room . . . . .	87
ScriptableObject	
ServerSettings . . . . .	93





## Chapter 9

# Class Index

### 9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">ActorProperties</a>	Class for constants. These (byte) values define "well known" properties for an Actor / Player. Pun uses these constants internally. . . . .	37
<a href="#">AuthenticationValues</a>	Container for "Custom Authentication" values in <a href="#">Photon</a> (default: user and token). Set Auth-Parameters before connecting - all else is handled. . . . .	37
<a href="#">ErrorCode</a>	Class for constants. These (int) values represent error codes, as defined and sent by the <a href="#">Photon</a> LoadBalancing logic. Pun uses these constants internally. . . . .	39
<a href="#">EventCode</a>	Class for constants. These values are for events defined by <a href="#">Photon</a> Loadbalancing. Pun uses these constants internally. . . . .	41
<a href="#">Extensions</a>	This static class defines some useful extension methods for several existing classes (e.g. Vector3, float and others). . . . .	43
<a href="#">FriendInfo</a>	Used to store info about a friend's online state and in which room he/she is. . . . .	46
<a href="#">GameProperties</a>	Class for constants. These (byte) values are for "well known" room/game properties used in <a href="#">Photon</a> Loadbalancing. Pun uses these constants internally. . . . .	46
<a href="#">Photon.MonoBehaviour</a>	This class adds the property photonView, while logging a warning when your game still uses the networkView. . . . .	48
<a href="#">OperationCode</a>	Class for constants. Contains operation codes. Pun uses these constants internally. . . . .	48
<a href="#">ParameterCode</a>	Class for constants. Codes for parameters of Operations and Events. Pun uses these constants internally. . . . .	50
<a href="#">PBitStream</a>	. . . . .	55
<a href="#">PhotonLagSimulationGui</a>	This MonoBehaviour is a basic GUI for the <a href="#">Photon</a> client's network-simulation feature. It can modify lag (fixed delay), jitter (random lag) and packet loss. . . . .	56
<a href="#">PhotonMessageInfo</a>	Container class for info about a particular message, RPC or update. . . . .	57
<a href="#">PhotonNetwork</a>	The main class to use the <a href="#">PhotonNetwork</a> plugin. This class is static. . . . .	58
<a href="#">PhotonPlayer</a>	Summarizes a "player" within a room, identified (in that room) by actorID. . . . .	77

<a href="#">PhotonStatsGui</a>	
Basic GUI to show traffic and health statistics of the connection to <a href="#">Photon</a> , toggled by shift+tab.	80
<a href="#">PhotonStream</a>	
This "container" class is used to carry your data as written by OnPhotonSerializeView. . . . .	82
<a href="#">PhotonView</a>	
PUN's NetworkView replacement class for networking. Use it like a NetworkView. . . . .	84
<a href="#">PingCloudRegions</a> . . . . .	86
<a href="#">Room</a>	
This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a <a href="#">RoomInfo</a> and you can close or hide "your" room. . . . .	87
<a href="#">RoomInfo</a>	
A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc). . . . .	89
<a href="#">RpcIndexComponent</a> . . . . .	92
<a href="#">ServerSettings</a>	
Collection of connection-relevant settings, used internally by <a href="#">PhotonNetwork.ConnectUsingSettings</a> . . . . .	93

## Chapter 10

# File Index

### 10.1 File List

Here is a list of all files with brief descriptions:

Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs	97
Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs	97
Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs	98
Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs	98
Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs	99
Photon Unity Networking/Plugins/PhotonNetwork/NetworkingPeer.cs	100
Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs	100
Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs	101
Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs	101
Photon Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs	101
Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs	101
Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs	101
Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs	102
Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs	102
Photon Unity Networking/Plugins/PhotonNetwork/Room.cs	103
Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs	103
Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs	103
Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs	103



# Chapter 11

## Module Documentation

### 11.1 Optional Gui Elements

#### Classes

- class [PhotonLagSimulationGui](#)

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

- class [PhotonStatsGui](#)

*Basic GUI to show traffic and health statistics of the connection to [Photon](#), toggled by shift+tab.*

#### 11.1.1 Detailed Description

While the PUN package does not provide in-game Gui components, there are some that try to make your life as developer easier.

## 11.2 Public API

### Classes

- class [PhotonMessageInfo](#)  
*Container class for info about a particular message, RPC or update.*
- class [PhotonStream](#)  
*This "container" class is used to carry your data as written by [OnPhotonSerializeView](#).*
- class [PhotonNetwork](#)  
*The main class to use the [PhotonNetwork](#) plugin. This class is static.*
- class [PhotonPlayer](#)  
*Summarizes a "player" within a room, identified (in that room) by actorID.*
- class [PhotonView](#)  
*PUN's [NetworkView](#) replacement class for networking. Use it like a [NetworkView](#).*
- class [Room](#)  
*This class resembles a room that PUN joins (or joined). The properties are settable as opposed to those of a [RoomInfo](#) and you can close or hide "your" room.*
- class [RoomInfo](#)  
*A simplified room with just the info required to list and join, used for the room listing in the lobby. The properties are not settable (open, maxPlayers, etc).*

### Enumerations

- enum [PeerState](#) {  
[Uninitialized](#), [PeerCreated](#), [Connecting](#), [Connected](#),  
[Queued](#), [Authenticated](#), [JoinedLobby](#), [DisconnectingFromMasterserver](#),  
[ConnectingToGameserver](#), [ConnectedToGameserver](#), [Joining](#), [Joined](#),  
[Leaving](#), [DisconnectingFromGameserver](#), [ConnectingToMasterserver](#), [ConnectedComingFromGameserver](#),  
[QueuedComingFromGameserver](#), [Disconnecting](#), [Disconnected](#), [ConnectedToMaster](#) }  
*Detailed connection / networking peer state. PUN implements a loadbalancing and authentication workflow "behind the scenes", so some states will automatically advance to some follow up state. Those states are commented with "(will-change)".*
- enum [PhotonNetworkingMessage](#) {  
[OnConnectedToPhoton](#), [OnLeftRoom](#), [OnMasterClientSwitched](#), [OnPhotonCreateRoomFailed](#),  
[OnPhotonJoinRoomFailed](#), [OnCreatedRoom](#), [OnJoinedLobby](#), [OnLeftLobby](#),  
[OnDisconnectedFromPhoton](#), [OnConnectionFail](#), [OnFailedToConnectToPhoton](#), [OnReceivedRoomListUpdate](#),  
[OnJoinedRoom](#), [OnPhotonPlayerConnected](#), [OnPhotonPlayerDisconnected](#), [OnPhotonRandomJoinFailed](#),  
[OnConnectedToMaster](#), [OnPhotonSerializeView](#), [OnPhotonInstantiate](#), [OnPhotonMaxCccuReached](#),  
[OnPhotonCustomRoomPropertiesChanged](#), [OnPhotonPlayerPropertiesChanged](#), [OnUpdatedFriendList](#),  
[OnCustomAuthenticationFailed](#) }  
*This enum makes up the set of [MonoMessages](#) sent by [Photon](#) Unity Networking. Implement any of these constant names as method and it will be called in the respective situation.*
- enum [DisconnectCause](#) {  
[ExceptionOnConnect](#) = [StatusCode.ExceptionOnConnect](#), [SecurityExceptionOnConnect](#) = [StatusCode.SecurityExceptionOnConnect](#), [TimeoutDisconnect](#) = [StatusCode.TimeoutDisconnect](#), [InternalReceiveException](#) = [StatusCode.InternalReceiveException](#),  
[DisconnectByServer](#) = [StatusCode.DisconnectByServer](#), [DisconnectByServerLogic](#) = [StatusCode.DisconnectByServerLogic](#), [DisconnectByServerUserLimit](#) = [StatusCode.DisconnectByServerUserLimit](#), [Exception](#) = [StatusCode.Exception](#),  
[InvalidRegion](#) = [ErrorCode.InvalidRegion](#), [MaxCcuReached](#) = [ErrorCode.MaxCcuReached](#) }  
*Summarizes the cause for a disconnect. Used in: [OnConnectionFail](#) and [OnFailedToConnectToPhoton](#).*
- enum [PhotonTargets](#) {  
[All](#), [Others](#), [MasterClient](#), [AllBuffered](#),  
[OthersBuffered](#) }

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

- enum [PhotonLogLevel](#) { [ErrorsOnly](#), [Informational](#), [Full](#) }

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

### 11.2.1 Detailed Description

**The public api of PUN consists of any code that is considered useful for you as developer.**

For documentation, we concentrate on the public api.

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

### 11.2.2 Enumeration Type Documentation

#### 11.2.2.1 enum DisconnectCause

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

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

See Also

[PhotonNetworkingMessage](#)

Enumerator:

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

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

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

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

**DisconnectByServer** Server actively disconnected this client.

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

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

**Exception** Some exception caused the connection to close.

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

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

#### 11.2.2.2 enum PeerState

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

Enumerator:

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

**PeerCreated** Created and available to connect.

**Connecting** Working to establish the initial connection to the master server (until this process is finished, no operations can be sent).(will-change)

**Connected** Connection is setup, now PUN will exchange keys for encryption or authenticate.(will-change)

**Queued** Not used at the moment.

**Authenticated** The application is authenticated. PUN usually joins the lobby now.(will-change) Unless AutoJoinLobby is false.

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

**DisconnectingFromMasterserver** Disconnecting.(will-change)

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

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

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

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

**Leaving** Leaving a room.(will-change)

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

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

**ConnectedComingFromGameserver** Same as Connected but coming from game server.(will-change)

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

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

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

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

### 11.2.2.3 enum PhotonLogLevel

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

Enumerator:

**ErrorsOnly**

**Informational**

**Full**

### 11.2.2.4 enum PhotonNetworkingMessage

This enum makes up the set of MonoMessages sent by [Photon](#) Unity Networking. Implement any of these constant names as method and it will be called in the respective situation.

Implement: public void OnLeftRoom() { //some work }

Enumerator:

**OnConnectedToPhoton** Called when the server is available and before client authenticates. Wait for the call to OnJoinedLobby (or OnConnectedToMaster) before the client does anything! Example: void OnConnectedToPhoton(){ ... } This is not called for transitions from the masterserver to game servers, which is hidden for PUN users.



- OnLeftRoom** Called once the local user left a room. Example: `void OnLeftRoom(){ ... }`
- OnMasterClientSwitched** Called -after- switching to a new MasterClient because the previous MC left the room (not when getting into a room). The last MC will already be removed at this time. Example: `void OnMasterClientSwitched(PhotonPlayer newMasterClient){ ... }`
- OnPhotonCreateRoomFailed** Called if a `CreateRoom()` call failed. Most likely because the room name is already in use. Example: `void OnPhotonCreateRoomFailed(){ ... }`
- OnPhotonJoinRoomFailed** Called if a `JoinRoom()` call failed. Most likely because the room does not exist or the room is full. Example: `void OnPhotonJoinRoomFailed(){ ... }`
- OnCreatedRoom** Called when `CreateRoom` finishes creating the room. After this, `OnJoinedRoom` will be called, too (no matter if creating one or joining). Example: `void OnCreatedRoom(){ ... }` This implies the local client is the MasterClient.
- OnJoinedLobby** Called on entering the Master Server's lobby. Client can create/join rooms but room list is not available until `OnReceivedRoomListUpdate` is called! Example: `void OnJoinedLobby(){ ... }` Note: When [PhotonNetwork.autoJoinLobby](#) is false, `OnConnectedToMaster` will be called instead and the room list won't be available. While in the lobby, the roomlist is automatically updated in fixed intervals (which you can't modify).
- OnLeftLobby** Called after leaving the lobby. Example: `void OnLeftLobby(){ ... }`
- OnDisconnectedFromPhoton** Called after disconnecting from the [Photon](#) server. In some cases, other events are sent before `OnDisconnectedFromPhoton` is called. Examples: `OnConnectionFail` and `OnFailedToConnectToPhoton`. Example: `void OnDisconnectedFromPhoton(){ ... }`
- OnConnectionFail** Called when something causes the connection to fail (after it was established), followed by a call to `OnDisconnectedFromPhoton`. If the server could not be reached in the first place, `OnFailedToConnectToPhoton` is called instead. The reason for the error is provided as `StatusCode`. Example: `void OnConnectionFail(DisconnectCause cause){ ... }`
- OnFailedToConnectToPhoton** Called if a connect call to the [Photon](#) server failed before the connection was established, followed by a call to `OnDisconnectedFromPhoton`. If the connection was established but then fails, `OnConnectionFail` is called. Example: `void OnFailedToConnectToPhoton(DisconnectCause cause){ ... }`
- OnReceivedRoomListUpdate** Called for any update of the room listing (no matter if "new" list or "update for known" list). Only called in the Lobby state (on master server). Example: `void OnReceivedRoomListUpdate(){ ... }`
- OnJoinedRoom** Called when entering a room (by creating or joining it). Called on all clients (including the Master Client). Example: `void OnJoinedRoom(){ ... }`
- OnPhotonPlayerConnected** Called after a remote player connected to the room. This [PhotonPlayer](#) is already added to the playerlist at this time. Example: `void OnPhotonPlayerConnected(PhotonPlayer newPlayer){ ... }`
- OnPhotonPlayerDisconnected** Called after a remote player disconnected from the room. This [PhotonPlayer](#) is already removed from the playerlist at this time. Example: `void OnPhotonPlayerDisconnected(PhotonPlayer otherPlayer){ ... }`
- OnPhotonRandomJoinFailed** Called after a `JoinRandom()` call failed. Most likely all rooms are full or no rooms are available. Example: `void OnPhotonRandomJoinFailed(){ ... }`
- OnConnectedToMaster** Called after the connection to the master is established and authenticated but only when `PhotonNetwork.AutoJoinLobby` is false. If `AutoJoinLobby` is false, the list of available rooms won't become available but you could join (random or by name) and create rooms anyways. Example: `void OnConnectedToMaster(){ ... }`
- OnPhotonSerializeView** Called every network 'update' on MonoBehaviours that are being observed by a [PhotonView](#). Example: `void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info){ ... }`
- OnPhotonInstantiate** Called on all scripts on a `GameObject` (and it's children) that have been spawned using [PhotonNetwork.Instantiate](#) Example: `void OnPhotonInstantiate(PhotonMessageInfo info){ ... }`
- OnPhotonMaxCcuReached** Because the concurrent user limit was (temporarily) reached, this client is rejected by the server and disconnecting. When this happens, the user might try again later. You can't create or join rooms in `OnPhotonMaxCcuReached()`, cause the client will be disconnecting. You can raise

the CCU limits with a new license (when you host yourself) or extended subscription (when using the [Photon](#) Cloud). The [Photon](#) Cloud will mail you when the CCU limit was reached. This is also visible in the Dashboard (webpage). Example: `void OnPhotonMaxCcuReached(){ ... }`

***OnPhotonCustomRoomPropertiesChanged*** Called when inside a room when its custom properties have changed. This is ALSO called for room property changes by the local players. Example: `void OnPhotonCustomRoomPropertiesChanged(){ ... }`

***OnPhotonPlayerPropertiesChanged*** Called when inside a room when a players custom properties change. Example: `void OnPhotonPlayerPropertiesChanged(PhotonPlayer player){ ... }`

***OnUpdatedFriendList*** Called when the server sent the response to a FindFriends request and updated [PhotonNetwork.Friends](#). Example: `void OnUpdatedFriendList(){ ... }`

***OnCustomAuthenticationFailed*** Called when the custom authentication failed (due to user-input, bad tokens/secrets or wrong configuration). Followed by disconnect! Example: `void OnCustomAuthenticationFailed(string debugMessage){ ... }` Unless you setup a custom authentication service for your app (in the Dashboard), this won't be called! If authentication is successful, this method is also not called but `OnJoinedLobby`, `OnConnectedToMaster` will be called (just as usual).

#### 11.2.2.5 enum PhotonTargets

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

Enumerator:

***All***

***Others***

***MasterClient***

***AllBuffered***

***OthersBuffered***

## Chapter 12

# Namespace Documentation

### 12.1 Package Photon

#### Classes

- class [MonoBehaviour](#)

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



## Chapter 13

# Class Documentation

### 13.1 ActorProperties Class Reference

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

#### Public Attributes

- const byte [PlayerName](#) = 255  
*(255) Name of a player/actor.*

#### 13.1.1 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.

#### 13.1.2 Member Data Documentation

##### 13.1.2.1 const byte ActorProperties.PlayerName = 255

(255) Name of a player/actor.

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

### 13.2 AuthenticationValues Class Reference

Container for "Custom Authentication" values in [Photon](#) (default: user and token). Set AuthParameters before connecting - all else is handled.

#### Public Member Functions

- virtual void [SetAuthParameters](#) (string user, string token)  
*Creates the default parameter string from a user and token value, escaping both. Alternatively set AuthParameters yourself.*

## Public Attributes

- **CustomAuthenticationType AuthType** = CustomAuthenticationType.Custom  
The type of custom authentication provider that should be used. Currently only "Custom" or "None" (turns this off).
- string **AuthParameters**  
This string must contain any (http get) parameters expected by the used authentication service. By default, username and token.
- string **Secret**  
After initial authentication, **Photon** provides a secret for this client / user, which is subsequently used as (cached) validation.

### 13.2.1 Detailed Description

Container for "Custom Authentication" values in **Photon** (default: user and token). Set AuthParameters before connecting - all else is handled.

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://cloud.exitgames.com/dashboard>

### 13.2.2 Member Function Documentation

#### 13.2.2.1 virtual void AuthenticationValues.SetAuthParameters ( string user, string token ) [virtual]

Creates the default parameter string from a user and token value, escaping both. Alternatively set AuthParameters yourself.

The default parameter string is: "username={user}&token={token}"

#### Parameters

<i>user</i>	Name or other end-user ID used in custom authentication service.
<i>token</i>	Token provided by authentication service to be used on initial "login" to <b>Photon</b> .

### 13.2.3 Member Data Documentation

#### 13.2.3.1 string AuthenticationValues.AuthParameters

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

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

#### 13.2.3.2 CustomAuthenticationType AuthenticationValues.AuthType = CustomAuthenticationType.Custom

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

#### 13.2.3.3 string AuthenticationValues.Secret

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

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.3 ErrorCode Class Reference

Class for constants. These (int) values represent error codes, as defined and sent by the [Photon](#) LoadBalancing logic. Pun uses these constants internally.

### 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: Appld is unknown to [Photon](#) (in cloud service).
- const int [GameldAlreadyExists](#) = 0x7FFF - 1  
(32766) Gameld (name) already in use (can't create another). Change name.
- const int [GameFull](#) = 0x7FFF - 2  
(32765) Game is full. This can when players took over while you joined the game.
- 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 because 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.

### 13.3.1 Detailed Description

Class for constants. These (int) values represent error codes, as defined and sent by the [Photon](#) LoadBalancing logic. Pun uses these constants internally.

<note>Codes from the [Photon](#) Core are negative. Default-app error codes go down from short.max.</note>

### 13.3.2 Member Data Documentation

13.3.2.1 `const int ErrorCode.AlreadyMatched = 0x7FFF - 4`

13.3.2.2 `const int ErrorCode.CustomAuthenticationFailed = 0x7FFF - 12`

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

13.3.2.3 `const int ErrorCode.GameClosed = 0x7FFF - 3`

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

13.3.2.4 `const int ErrorCode.GameDoesNotExist = 0x7FFF - 9`

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

13.3.2.5 `const int ErrorCode.GameFull = 0x7FFF - 2`

(32765) Game is full. This can when players took over while you joined the game.

13.3.2.6 `const int ErrorCode.GameIdAlreadyExists = 0x7FFF - 1`

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

13.3.2.7 `const int ErrorCode.InternalServerError = -1`

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

13.3.2.8 `const int ErrorCode.InvalidAuthentication = 0x7FFF`

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

13.3.2.9 `const int ErrorCode.InvalidOperationCode = -2`

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

13.3.2.10 `const int ErrorCode.InvalidRegion = 0x7FFF - 11`

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

Some subscription plans for the [Photon](#) Cloud are region-bound. Servers of other regions can't be used then. Check your master server address and compare it with your [Photon](#) Cloud Dashboard's info. <https://cloud.-exitgames.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.



**13.3.2.11** `const int ErrorCode.MaxCcuReached = 0x7FFF - 10`

(32757) Authorization on the [Photon](#) Cloud failed because 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 and 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.

**13.3.2.12** `const int ErrorCode.NoRandomMatchFound = 0x7FFF - 7`

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

**13.3.2.13** `const int ErrorCode.Ok = 0`

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

**13.3.2.14** `const int ErrorCode.OperationNotAllowedInCurrentState = -3`

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

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

**13.3.2.15** `const int ErrorCode.ServerFull = 0x7FFF - 5`

(32762) Not in use currently.

**13.3.2.16** `const int ErrorCode.UserBlocked = 0x7FFF - 6`

(32761) Not in use currently.

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.4 EventCode Class Reference

Class for constants. These values are for events defined by [Photon](#) Loadbalancing. Pun uses these constants internally.

### Public Attributes

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

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

  - const byte `QueueState` = 228

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

- const byte `Match` = 227
- (227) Currently not used. Event for matchmaking

  - const byte `AppStats` = 226

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

- const byte `AzureNodeInfo` = 210
- (210) Internally used in case of hosting by Azure

  - const byte `Join` = (byte)LiteEventCode.Join

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

  - const byte `PropertiesChanged` = (byte)LiteEventCode.PropertiesChanged

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

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

### 13.4.1 Detailed Description

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

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

### 13.4.2 Member Data Documentation

#### 13.4.2.1 const byte EventCode.AppStats = 226

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

#### 13.4.2.2 const byte EventCode.AzureNodeInfo = 210

(210) Internally used in case of hosting by Azure

#### 13.4.2.3 const byte EventCode.GameList = 230

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

#### 13.4.2.4 const byte EventCode.GameListUpdate = 229

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

#### 13.4.2.5 const byte EventCode.Join = (byte)LiteEventCode.Join

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

#### 13.4.2.6 `const byte EventCode.Leave = (byte)LiteEventCode.Leave`

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

#### 13.4.2.7 `const byte EventCode.Match = 227`

(227) Currently not used. Event for matchmaking

#### 13.4.2.8 `const byte EventCode.PropertiesChanged = (byte)LiteEventCode.PropertiesChanged`

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

#### 13.4.2.9 `const byte EventCode.QueueState = 228`

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

#### 13.4.2.10 `const byte EventCode.SetProperties = (byte)LiteEventCode.PropertiesChanged`

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

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.5 Extensions Class Reference

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

### Static Public Member Functions

- static [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 square magnitude of target - second to given float value*
- static bool [AlmostEquals](#) (this `Vector2` target, `Vector2` second, float `sqrMagnitudePrecision`)  
*compares the square 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 if their difference is less than floatDiff*
- static void [Merge](#) (this `IDictionary` target, `IDictionary` addHash)  
*Merges all keys from addHash into the target. Adds new keys and updates the values of existing keys in target.*
- static void [MergeStringKeys](#) (this `IDictionary` target, `IDictionary` addHash)  
*Merges keys of type string to target Hashtable.*
- static string [ToStringFull](#) (this `IDictionary` origin)  
*Returns a string-representation of the IDictionary's content, including type-information. Note: This might turn out a "heavy-duty" call if used frequently but it's useful to debug Dictionary or Hashtable content.*
- static `Hashtable` [StripToStringKeys](#) (this `IDictionary` original)

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

- static void [StripKeysWithNullValues](#) (this IDictionary original)

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

- static bool [Contains](#) (this int[] target, int nr)

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

### 13.5.1 Detailed Description

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

### 13.5.2 Member Function Documentation

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

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

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

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

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

compares the angle between target and second to given float value

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

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

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

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

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

#### Parameters

<i>target</i>	The array of ints to check.
<i>nr</i>	The number to lookup in target.

#### Returns

True if nr was found in target.

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

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

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

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

## Parameters

<i>target</i>	The IDictionary to update.
<i>addHash</i>	The IDictionary containing data to merge into target.

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

Merges keys of type string to target Hashtable.

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

## Parameters

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

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

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

## Parameters

<i>original</i>	The IDictionary to strip of keys with null-values.
-----------------	--

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

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

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

## Parameters

<i>original</i>	The original IDictionary to get string-typed keys from.
-----------------	---

## Returns

New Hashtable containing parts of the original.

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

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

## Parameters

<i>origin</i>	Some Dictionary or Hashtable.
---------------	-------------------------------

## Returns

String of the content of the IDictionary.

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[Extensions.cs](#)

## 13.6 FriendInfo Class Reference

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

### Public Member Functions

- override string [ToString](#) ()

### Properties

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

#### 13.6.1 Detailed Description

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

#### 13.6.2 Member Function Documentation

13.6.2.1 override string [FriendInfo.ToString](#) ( )

#### 13.6.3 Property Documentation

13.6.3.1 bool [FriendInfo.IsInRoom](#) [get]

13.6.3.2 bool [FriendInfo.IsOnline](#) [get], [set]

13.6.3.3 string [FriendInfo.Name](#) [get], [set]

13.6.3.4 string [FriendInfo.Room](#) [get], [set]

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[FriendInfo.cs](#)

## 13.7 GameProperties Class Reference

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

### Public Attributes

- const byte [MaxPlayers](#) = 255  
(255) Max number of players that "fit" into this room. 0 is for "unlimited".
- const byte [IsVisible](#) = 254  
(254) Makes this room listed or not in the lobby on master.
- const byte [IsOpen](#) = 253  
(253) Allows more players to join a room (or not).

- const byte [PlayerCount](#) = 252  
(252) Current count of players in the room. Used only in the lobby on master.
- const byte [Removed](#) = 251  
(251) True if the room is to be removed from room listing (used in update to room list in lobby on master)
- const byte [PropsListedInLobby](#) = 250  
(250) A list of the room properties to pass to the [RoomInfo](#) list in a lobby. This is used in *CreateRoom*, which defines this list once per room.
- const byte [CleanupCacheOnLeave](#) = 249  
Equivalent of Operation Join parameter *CleanupCacheOnLeave*.

### 13.7.1 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.

### 13.7.2 Member Data Documentation

#### 13.7.2.1 const byte GameProperties.CleanupCacheOnLeave = 249

Equivalent of Operation Join parameter *CleanupCacheOnLeave*.

#### 13.7.2.2 const byte GameProperties.IsOpen = 253

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

#### 13.7.2.3 const byte GameProperties.IsVisible = 254

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

#### 13.7.2.4 const byte GameProperties.MaxPlayers = 255

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

#### 13.7.2.5 const byte GameProperties.PlayerCount = 252

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

#### 13.7.2.6 const byte GameProperties.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.

#### 13.7.2.7 const byte GameProperties.Removed = 251

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

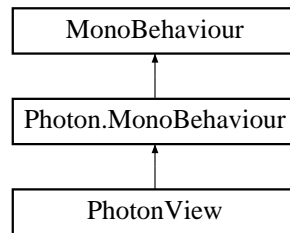
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.8 Photon.MonoBehaviour Class Reference

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

Inheritance diagram for `Photon.MonoBehaviour`:



### Properties

- `PhotonView photonView` [get]
- `new PhotonView networkView` [get]

#### 13.8.1 Detailed Description

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

#### 13.8.2 Property Documentation

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

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

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonClasses.cs](#)

## 13.9 OperationCode Class Reference

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

### Public Attributes

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

  - const byte `RaiseEvent` = (byte)LiteOpCode.RaiseEvent
- (253) Raise event (in a room, for other actors/players)

  - const byte `SetProperties` = (byte)LiteOpCode.SetProperties
- (252) Set Properties (of room or actor/player)

  - const byte `GetProperties` = (byte)LiteOpCode.GetProperties
- (251) Get Properties

  - const byte `ChangeGroups` = (byte)LiteOpCode.ChangeGroups
- (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).

### 13.9.1 Detailed Description

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

### 13.9.2 Member Data Documentation

#### 13.9.2.1 const byte OperationCode.Authenticate = 230

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

#### 13.9.2.2 const byte OperationCode.ChangeGroups = (byte)LiteOpCode.ChangeGroups

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

#### 13.9.2.3 const byte OperationCode.CreateGame = 227

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

#### 13.9.2.4 const byte OperationCode.FindFriends = 222

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

#### 13.9.2.5 const byte OperationCode.GetProperties = (byte)LiteOpCode.GetProperties

(251) Get Properties

#### 13.9.2.6 const byte OperationCode.JoinGame = 226

(226) Join game (by name)

#### 13.9.2.7 const byte OperationCode.JoinLobby = 229

(229) Joins lobby (on master)

#### 13.9.2.8 `const byte OperationCode.JoinRandomGame = 225`

(225) Joins random game (on master)

#### 13.9.2.9 `const byte OperationCode.Leave = (byte)LiteOpCode.Leave`

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

#### 13.9.2.10 `const byte OperationCode.LeaveLobby = 228`

(228) Leaves lobby (on master)

#### 13.9.2.11 `const byte OperationCode.RaiseEvent = (byte)LiteOpCode.RaiseEvent`

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

#### 13.9.2.12 `const byte OperationCode.SetProperties = (byte)LiteOpCode.SetProperties`

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

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.10 ParameterCode Class Reference

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

### Public Attributes

- `const byte Address = 230`  
(230) Address of a (game) server to use.
- `const byte PeerCount = 229`  
(229) Count of players in rooms (connected to game servers for this application, 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 (connected to master server for this application, looking for games, used in stats event)
- `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 (as "Position" currently). 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 [RoomName](#) = (byte)LiteOpKey.GameId  
(255) Code for the gameId/roomName (a unique name per room). Used in OpJoin and similar.
- const byte [Broadcast](#) = (byte)LiteOpKey.Broadcast  
(250) Code for broadcast parameter of OpSetProperties method.
- const byte [ActorList](#) = (byte)LiteOpKey.ActorList  
(252) Code for list of players in a room. Currently not used.
- const byte [ActorNr](#) = (byte)LiteOpKey.ActorNr  
(254) Code of the Actor of an operation. Used for property get and set.
- const byte [PlayerProperties](#) = (byte)LiteOpKey.ActorProperties  
(249) Code for property set (Hashtable).
- const byte [CustomEventContent](#) = (byte)LiteOpKey.Data  
(245) Code of data/custom content of an event. Used in OpRaiseEvent.
- const byte [Data](#) = (byte)LiteOpKey.Data  
(245) Code of data of an event. Used in OpRaiseEvent.
- const byte [Code](#) = (byte)LiteOpKey.Code  
(244) Code used when sending some code-related parameter, like OpRaiseEvent's event-code.
- const byte [GameProperties](#) = (byte)LiteOpKey.GameProperties  
(248) Code for property set (Hashtable).
- const byte [Properties](#) = (byte)LiteOpKey.Properties  
(251) Code for property-set (Hashtable). This key is used when sending only one set of properties. If either [ActorProperties](#) or [GameProperties](#) are used (or both), check those keys.
- const byte [TargetActorNr](#) = (byte)LiteOpKey.TargetActorNr  
(253) Code of the target Actor of an operation. Used for property set. Is 0 for game
- const byte [ReceiverGroup](#) = (byte)LiteOpKey.ReceiverGroup  
(246) Code to select the receivers of events (used in Lite, Operation RaiseEvent).
- const byte [Cache](#) = (byte)LiteOpKey.Cache  
(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](#) = LiteOpKey.Group  
(240) Code for "group" operation-parameter (as used in Op RaiseEvent).
- const byte [Remove](#) = LiteOpKey.Remove  
(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 [Add](#) = LiteOpKey.Add  
(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 [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 [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).

### 13.10.1 Detailed Description

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

### 13.10.2 Member Data Documentation

13.10.2.1 `const byte ParameterCode.ActorList = (byte)LiteOpKey.ActorList`

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

13.10.2.2 `const byte ParameterCode.ActorNr = (byte)LiteOpKey.ActorNr`

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

13.10.2.3 `const byte ParameterCode.Add = LiteOpKey.Add`

(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.

13.10.2.4 `const byte ParameterCode.Address = 230`

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

13.10.2.5 `const byte ParameterCode.ApplicationId = 224`

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

13.10.2.6 `const byte ParameterCode.AppVersion = 220`

(220) Version of your application

13.10.2.7 `const byte ParameterCode.Broadcast = (byte)LiteOpKey.Broadcast`

(250) Code for broadcast parameter of OpSetProperties method.

13.10.2.8 `const byte ParameterCode.Cache = (byte)LiteOpKey.Cache`

(247) Code for caching events while raising them.

13.10.2.9 `const byte ParameterCode.CleanupCacheOnLeave = (byte)241`

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

13.10.2.10 `const byte ParameterCode.ClientAuthenticationParams = 216`

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

**13.10.2.11 const byte ParameterCode.ClientAuthenticationType = 217**

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

**13.10.2.12 const byte ParameterCode.Code = (byte)LiteOpKey.Code**

(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](#).

**13.10.2.13 const byte ParameterCode.CustomEventContent = (byte)LiteOpKey.Data**

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

**13.10.2.14 const byte ParameterCode.Data = (byte)LiteOpKey.Data**

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

**13.10.2.15 const byte ParameterCode.FindFriendsRequestList = (byte)1**

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

**13.10.2.16 const byte ParameterCode.FindFriendsResponseOnlineList = (byte)1**

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

**13.10.2.17 const byte ParameterCode.FindFriendsResponseRoomIdList = (byte)2**

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

**13.10.2.18 const byte ParameterCode.GameCount = 228**

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

**13.10.2.19 const byte ParameterCode.GameList = 222**

(222) List of RoomInfos about open / listed rooms

**13.10.2.20 const byte ParameterCode.GameProperties = (byte)LiteOpKey.GameProperties**

(248) Code for property set (Hashtable).

**13.10.2.21 const byte ParameterCode.Group = LiteOpKey.Group**

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

**13.10.2.22** `const byte ParameterCode.MasterPeerCount = 227`

(227) Count of players on the master server (connected to master server for this application, looking for games, used in stats event)

**13.10.2.23** `const byte ParameterCode.MatchMakingType = 223`

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

**13.10.2.24** `const byte ParameterCode.PeerCount = 229`

(229) Count of players in rooms (connected to game servers for this application, used in stats event)

**13.10.2.25** `const byte ParameterCode.PlayerProperties = (byte)LiteOpKey.ActorProperties`

(249) Code for property set (Hashtable).

**13.10.2.26** `const byte ParameterCode.Position = 223`

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

**13.10.2.27** `const byte ParameterCode.Properties = (byte)LiteOpKey.Properties`

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

**13.10.2.28** `const byte ParameterCode.ReceiverGroup = (byte)LiteOpKey.ReceiverGroup`

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

**13.10.2.29** `const byte ParameterCode.Remove = LiteOpKey.Remove`

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

**13.10.2.30** `const byte ParameterCode.RoomName = (byte)LiteOpKey.Gameld`

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

**13.10.2.31** `const byte ParameterCode.Secret = 221`

(221) Internally used to establish encryption

**13.10.2.32** `const byte ParameterCode.TargetActorNr = (byte)LiteOpKey.TargetActorNr`

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

13.10.2.33 `const byte ParameterCode.UserId = 225`

(225) User's ID

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[LoadbalancingPeer.cs](#)

## 13.11 PBitStream Class Reference

### Public Member Functions

- [PBitStream](#) ()
- [PBitStream](#) (int bitCount)
- [PBitStream](#) (IEnumerable< byte > bytes, int bitCount)
- void [Add](#) (bool val)
- byte[] [ToBytes](#) ()
- bool [GetNext](#) ()
- bool [Get](#) (int bitIndex)
- void [Set](#) (int bitIndex, bool value)

### Static Public Member Functions

- static int [BytesForBits](#) (int bitCount)

### Properties

- int [ByteCount](#) [get]
- int [BitCount](#) [get, set]
- int [Position](#) [get, set]

### 13.11.1 Constructor & Destructor Documentation

13.11.1.1 `PBitStream.PBitStream ( )`

13.11.1.2 `PBitStream.PBitStream ( int bitCount )`

13.11.1.3 `PBitStream.PBitStream ( IEnumerable< byte > bytes, int bitCount )`

### 13.11.2 Member Function Documentation

13.11.2.1 `void PBitStream.Add ( bool val )`

13.11.2.2 `static int PBitStream.BytesForBits ( int bitCount )` [static]

13.11.2.3 `bool PBitStream.Get ( int bitIndex )`

13.11.2.4 `bool PBitStream.GetNext ( )`

13.11.2.5 `void PBitStream.Set ( int bitIndex, bool value )`

13.11.2.6 `byte [] PBitStream.ToBytes ( )`

### 13.11.3 Property Documentation

13.11.3.1 `int PBitStream.BitCount` `[get]`, `[set]`

13.11.3.2 `int PBitStream.ByteCount` `[get]`

13.11.3.3 `int PBitStream.Position` `[get]`, `[set]`

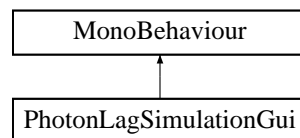
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonClasses.cs](#)

## 13.12 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.

Inheritance diagram for PhotonLagSimulationGui:



### 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).*

### 13.12.1 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.



### 13.12.2 Member Function Documentation

13.12.2.1 void PhotonLagSimulationGui.OnGUI ( )

13.12.2.2 void PhotonLagSimulationGui.Start ( )

### 13.12.3 Member Data Documentation

13.12.3.1 bool PhotonLagSimulationGui.Visible = true

Shows or hides GUI (does not affect settings).

13.12.3.2 int PhotonLagSimulationGui.WindowId = 101

Unity GUI Window ID (must be unique or will cause issues).

13.12.3.3 Rect PhotonLagSimulationGui.WindowRect = new Rect(0, 100, 120, 100)

Positioning rect for window.

### 13.12.4 Property Documentation

13.12.4.1 PhotonPeer PhotonLagSimulationGui.Peer [get], [set]

The peer currently in use (to set the network simulation).

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonLagSimulationGui.cs](#)

## 13.13 PhotonMessageInfo Class Reference

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

### Public Member Functions

- [PhotonMessageInfo](#) ()  
*Initializes a new instance of the [PhotonMessageInfo](#) class. To create an empty messageinfo only!*
- [PhotonMessageInfo](#) ([PhotonPlayer](#) player, int timestamp, [PhotonView](#) view)
- override string [ToString](#) ()

### Public Attributes

- [PhotonPlayer](#) sender
- [PhotonView](#) photonView

### Properties

- double [timestamp](#) [get]

### 13.13.1 Detailed Description

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

### 13.13.2 Constructor & Destructor Documentation

#### 13.13.2.1 PhotonMessageInfo.PhotonMessageInfo ( )

Initializes a new instance of the [PhotonMessageInfo](#) class. To create an empty messageinfo only!

#### 13.13.2.2 PhotonMessageInfo.PhotonMessageInfo ( PhotonPlayer player, int timestamp, PhotonView view )

### 13.13.3 Member Function Documentation

#### 13.13.3.1 override string PhotonMessageInfo.ToString ( )

### 13.13.4 Member Data Documentation

#### 13.13.4.1 PhotonView PhotonMessageInfo.photonView

#### 13.13.4.2 PhotonPlayer PhotonMessageInfo.sender

### 13.13.5 Property Documentation

#### 13.13.5.1 double PhotonMessageInfo.timestamp [get]

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonClasses.cs](#)

## 13.14 PhotonNetwork Class Reference

The main class to use the [PhotonNetwork](#) plugin. This class is static.

### Static Public Member Functions

- static void [NetworkStatisticsReset](#) ()  
*Resets the traffic stats and re-enables them.*
- static string [NetworkStatisticsToString](#) ()  
*Only available when NetworkStatisticsEnabled was used to gather some stats.*
- static void [InternalCleanPhotonMonoFromSceneIfStuck](#) ()  
*Internally used by Editor scripts, called on Hierarchy change (includes scene save) to remove surplus hidden Photon-Handlers.*
- static void [ConnectUsingSettings](#) (string gameVersion)  
*Connect to the configured Photon server: Reads PhotonNetwork.serverSettingsAssetPath and connects to cloud or your own server.*
- static void [ConnectUsingSettings](#) ()
- static void [Connect](#) (string serverAddress, int port, string uniqueGameID)
- static void [ConnectToBestCloudServer](#) (string gameVersion)  
*Connect to the PUN cloud server with the lowest 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(..)*

- static void [OverrideBestCloudServer](#) ([PingCloudRegions.CloudRegions](#) region)
 

*Overwrites the region that is used for [ConnectToBestCloudServer\(string gameVersion\)](#) This will overwrite the result that was gotten by pinging all cloud servers.*
- static void [Connect](#) (string serverAddress, int port, string appId, string gameVersion)
 

*Connect to the photon server by address, port, appId and game(client) version. This method is used by [Connect-UsingSettings](#) which applies values from the settings file.*
- static void [Disconnect](#) ()
 

*Makes this client disconnect from the photon server, a process that leaves any room and calls [OnDisconnectedFromPhoton](#) on completion.*
- static void [InitializeSecurity](#) ()
 

*Used for compatibility with Unity networking only. Encryption is automatically initialized while connecting.*
- static bool [FindFriends](#) (string[] friendsToFind)
 

*Request the rooms and online status for a list of friends. All client must set a unique username via [PlayerName](#) property. The result is available in [this.Friends](#).*
- static void [CreateRoom](#) (string roomName)
 

*Creates a room with given name but fails if this room is existing already.*
- static void [CreateRoom](#) (string roomName, bool isVisible, bool isOpen, int maxPlayers)
 

*Creates a room with given name but fails if this room is existing already.*
- static void [CreateRoom](#) (string roomName, bool isVisible, bool isOpen, int maxPlayers, Hashtable customRoomProperties, string[] propsToListInLobby)
 

*Creates a room with given name but fails if this room is existing already.*
- static void [JoinRoom](#) ([RoomInfo](#) listedRoom)
 

*Join room by room.Name. This fails if the room is either full or no longer available (might close at the same time).*
- static void [JoinRoom](#) (string roomName)
 

*Join room with given title. This fails if the room is either full or no longer available (might close at the same time).*
- static void [JoinRandomRoom](#) ()
 

*Joins any available room but will fail if none is currently available.*
- static void [JoinRandomRoom](#) (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)
 

*Attempts to join an open room with fitting, custom properties but fails if none is currently available.*
- static void [JoinRandomRoom](#) (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, [MatchmakingMode](#) matchingType)
 

*Attempts to join an open room with fitting, custom properties but fails if none is currently available.*
- static void [LeaveRoom](#) ()
 

*Leave the current room*
- static [RoomInfo](#)[] [GetRoomList](#) ()
 

*Gets an array of (currently) known rooms as [RoomInfo](#). This list is automatically updated every few seconds while this client is in the lobby (on the Master Server). Not available while being in a room.*
- static void [SetPlayerCustomProperties](#) (Hashtable customProperties)
 

*Sets this (local) player's properties. This caches the properties in [PhotonNetwork.player.customProperties](#). [CreateRoom](#), [JoinRoom](#) and [JoinRandomRoom](#) will all apply your player's custom properties when you enter the room. While in a room, your properties are synced with the other players. 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.*
- static int [AllocateViewID](#) ()
 

*Allocates a viewID that's valid for the current/local player.*
- static void [UnAllocateViewID](#) (int viewID)
 

*Unregister a viewID (of manually instantiated and destroyed networked objects).*
- static GameObject [Instantiate](#) (string prefabName, Vector3 position, Quaternion rotation, int group)
 

*Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.*
- static GameObject [Instantiate](#) (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)
 

*Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.*

- static GameObject [InstantiateSceneObject](#) (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)  
*Instantiate a scene-owned prefab over the network. The PhotonViews will be controllable by the MasterClient. This prefab needs to be located in the root of a "Resources" folder.*
- static int [GetPing](#) ()  
*The current roundtrip time to the photon server*
- static void [FetchServerTimestamp](#) ()  
*Refreshes the server timestamp (async operation, takes a roundtrip).*
- static void [SendOutgoingCommands](#) ()  
*Can be used to immediately send the RPCs and Instantiates just made, so they are on their way to the other players.*
- static void [CloseConnection](#) (PhotonPlayer kickPlayer)  
*Request a client to disconnect (KICK). Only the master client can do this.*
- static void [Destroy](#) (PhotonView view)  
*Destroy supplied PhotonView. This will remove all Buffered RPCs and destroy the GameObject this view is attached to (plus all childs, if any) This has the same effect as calling Destroy by passing a GameObject*
- static void [Destroy](#) (GameObject go)  
*Destroys given GameObject. This GameObject must've been instantiated using PhotonNetwork.Instantiate and must have a PhotonView at it's root. This has the same effect as calling Destroy by passing an attached PhotonView from this GameObject*
- static void [DestroyPlayerObjects](#) (PhotonPlayer destroyPlayer)  
*Destroy all GameObjects/PhotonViews of this player. can only be called on the local player. The only exception is the master client which call call this for all players.*
- static void [RemoveAllInstantiatedObjects](#) ()  
*MasterClient method only: Destroy ALL instantiated GameObjects*
- static void [RemoveAllInstantiatedObjects](#) (PhotonPlayer targetPlayer)  
*Destroy ALL PhotonNetwork.Instantiated GameObjects by given player. Can only be called on the local player or MasterClient. The MasterClient can call this for all players.*
- static void [RemoveRPCs](#) ()  
*Remove ALL buffered RPCs of the local player*
- static void [RemoveRPCs](#) (PhotonPlayer targetPlayer)  
*Remove ALL buffered RPCs of a player*
- static void [RemoveAllBufferedMessages](#) ()  
*Remove ALL buffered messages of the local player (RPC's and Instantiation calls) Note that this only removed the buffered messages on the server, you will still need to remove the Instantiated GameObjects yourself.*
- static void [RemoveAllBufferedMessages](#) (PhotonPlayer targetPlayer)  
*Remove ALL buffered messages of a player (RPC's and Instantiation calls) Note that this only removed the buffered messages on the server, you will still need to remove the Instantiated GameObjects yourself.*
- static void [RemoveRPCs](#) (PhotonView view)  
*Remove all buffered RPCs on given PhotonView (if they are owned by this player).*
- static void [RemoveRPCsInGroup](#) (int group)  
*Remove all buffered RPCs with given group*
- static void [SetReceivingEnabled](#) (int group, bool enabled)  
*Enable/disable receiving on given group (applied to PhotonViews)*
- static void [SetSendingEnabled](#) (int group, bool enabled)  
*Enable/disable sending on given group (applied to PhotonViews)*
- static void [SetLevelPrefix](#) (short prefix)  
*Sets level prefix for PhotonViews instantiated later on. Don't set it if you need only one!*
- static void [LoadLevel](#) (int levelNumber)  
*Loads the level and automatically pauses the network queue. Call this in OnJoinedRoom to make sure no cached RPCs are fired in the wrong scene.*
- static void [LoadLevel](#) (string levelTitle)  
*Loads the level and automatically pauses the network queue. Call this in OnJoinedRoom to make sure no cached RPCs are fired in the wrong scene.*

## Public Attributes

- const string [versionPUN](#) = "1.20"  
*Version number of PUN. Also used in GameVersion to separate client version from each other.*
- const string [serverSettingsAssetFile](#) = "PhotonServerSettings"  
*Name of the PhotonServerSettings file (used to load and by PhotonEditor to save new files).*
- const string [serverSettingsAssetPath](#) = "Assets/Photon Unity Networking/Resources/" + PhotonNetwork.serverSettingsAssetFile + ".asset"  
*Path to the PhotonServerSettings file (used by PhotonEditor).*

## Static Public Attributes

- static readonly int [MAX\\_VIEW\\_IDS](#) = 1000  
*The maximum amount of assigned PhotonViews PER player (or scene). See the documentation on how to raise this limitation*
- static [ServerSettings](#) [PhotonServerSettings](#) = ([ServerSettings](#))Resources.Load([PhotonNetwork.serverSettingsAssetFile](#), typeof([ServerSettings](#)))  
*Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.*
- static 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 Note that this is the sqrMagnitude. E.g. to send only after a 0.01 change on the Y-axis, we use  $0.01f * 0.01f = 0.0001f$ . As a remedy against float inaccuracy we use 0.000099f instead of 0.0001f.*
- static float [precisionForQuaternionSynchronization](#) = 1.0f  
*The minimum angle that a rotation needs to change before we send it via a [PhotonView](#)'s OnSerialize/ObservingComponent*
- static float [precisionForFloatSynchronization](#) = 0.01f  
*The minimum difference between floats before we send it via a [PhotonView](#)'s OnSerialize/ObservingComponent*
- static [PhotonLogLevel](#) [logLevel](#) = [PhotonLogLevel.ErrorsOnly](#)  
*Network log level. Controls how verbose PUN is.*
- static bool [UsePrefabCache](#) = true  
*While enabled (true), Instantiate uses [PhotonNetwork.PrefabCache](#) to keep game objects in memory (improving instantiation of the same prefab).*
- static Dictionary< string, GameObject > [PrefabCache](#) = new Dictionary<string, GameObject>()  
*Keeps references to GameObjects for frequent instantiation (out of memory instead of loading the Resources).*

## Properties

- static string [ServerAddress](#) [get]  
*Currently used server address (no matter if master or game server).*
- static bool [connected](#) [get]  
*Are we connected to the photon server (can be IN or OUTSIDE a room)*
- static [ConnectionState](#) [connectionState](#) [get]  
*Simplified connection state*
- static [PeerState](#) [connectionStateDetailed](#) [get]  
*Detailed connection state (ignorant of PUN, so it can be "disconnected" while switching servers).*
- static [AuthenticationValues](#) [AuthValues](#) [get, set]  
*A user's authentication values used during connect for Custom Authentication with [Photon](#) (and a custom service/-community). Set these before calling Connect if you want custom authentication.*
- static [Room](#) [room](#) [get]  
*Get the room we're currently in. Null if we aren't in any room.*

- static [PhotonPlayer player](#) [get]
 

The local [PhotonPlayer](#). Always available and represents this player. CustomProperties can be set before entering a room and will be synced as well.
- static [PhotonPlayer masterClient](#) [get]
 

The [PhotonPlayer](#) of the master client. The master client is the 'virtual owner' of the room. You can use it if you need authoritative decision made by one of the players.
- static string [playerName](#) [get, set]
 

This local player's name.
- static [PhotonPlayer\[\] playerList](#) [get]
 

The full [PhotonPlayer](#) list, including the local player.
- static [PhotonPlayer\[\] otherPlayers](#) [get]
 

The other PhotonPlayers, not including our local player.
- static List< [FriendInfo](#) > [Friends](#) [get, set]
 

Read-only list of friends, their online status and the room they are in. Null until initialized by a FindFriends call.
- static int [FriendsListAge](#) [get]
 

Age of friend list info (in milliseconds). It's 0 until a friend list is fetched.
- static bool [offlineMode](#) [get, set]
 

Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on [PhotonNetwork](#) will not create any connections and there is near to no overhead. Mostly usefull for reusing RPC's and [PhotonNetwork.Instantiate](#)
- static int [maxConnections](#) [get, set]
 

The maximum number of players for a room. Better: Set it in CreateRoom. If no room is opened, this will return 0.
- static bool [automaticallySyncScene](#) [get, set]
 

If true, PUN will make sure that all users are in the same scene at all times. If the MasterClient switches, all clients will load the new scene. This also takes care of smooth loading of the game scene after joining a game from your main menu.
- static bool [autoCleanUpPlayerObjects](#) [get, set]
 

This setting defines if players in a room should destroy a leaving player's instantiated GameObjects and PhotonViews.
- static bool [autoJoinLobby](#) [get, set]
 

Defines if the [PhotonNetwork](#) should join the "lobby" when connected to the Master server. If this is false, OnConnectedToMaster() will be called when connection to the Master is available. OnJoinedLobby() will NOT be called if this is false.
- static bool [insideLobby](#) [get]
 

Returns true when we are connected to [Photon](#) and in the lobby state
- static int [sendRate](#) [get, set]
 

Defines how many times per second [PhotonNetwork](#) should send a package. If you change this, do not forget to also change 'sendRateOnSerialize'.
- static int [sendRateOnSerialize](#) [get, set]
 

Defines how many times per second OnPhotonSerialize should be called on PhotonViews.
- static bool [isMessageQueueRunning](#) [get, set]
 

Can be used to pause dispatch of incoming evtents (RPCs, Instantiates and anything else incoming). 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.
- static int [unreliableCommandsLimit](#) [get, set]
 

Used once per dispatch to limit unreliable commands per channel (so after a pause, many channels can still cause a lot of unreliable commands)
- static double [time](#) [get]
 

[Photon](#) network time, synched with the server
- static bool [isMasterClient](#) [get]
 

Are we the master client?
- static bool [isNonMasterClientInRoom](#) [get]
 

True if we are in a room (client) and NOT the room's masterclient

- static int [countOfPlayersOnMaster](#) [get]  
*The count of players currently looking for a room. This is updated on the MasterServer (only) in 5sec intervals (if any count changed).*
- static int [countOfPlayersInRooms](#) [get]  
*The count of players currently inside a room. This is updated on the MasterServer (only) in 5sec intervals (if any count changed).*
- static int [countOfPlayers](#) [get]  
*The count of players currently using this application. This is updated on the MasterServer (only) in 5sec intervals (if any count changed).*
- static int [countOfRooms](#) [get]  
*The count of rooms currently in use. When inside the lobby this is based on [PhotonNetwork.GetRoomList\(\).Length](#). When not inside the lobby, this value updated on the MasterServer (only) in 5sec intervals (if any count changed).*
- static bool [NetworkStatisticsEnabled](#) [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.*

### 13.14.1 Detailed Description

The main class to use the [PhotonNetwork](#) plugin. This class is static.

### 13.14.2 Member Function Documentation

#### 13.14.2.1 static int PhotonNetwork.AllocateViewID ( ) [static]

Allocates a viewID that's valid for the current/local player.

##### Returns

A viewID that can be used for a new [PhotonView](#).

#### 13.14.2.2 static void PhotonNetwork.CloseConnection ( PhotonPlayer kickPlayer ) [static]

Request a client to disconnect (KICK). Only the master client can do this.

##### Parameters

<i>kickPlayer</i>	The <a href="#">PhotonPlayer</a> to kick.
-------------------	---

#### 13.14.2.3 static void PhotonNetwork.Connect ( string serverAddress, int port, string uniqueGameID ) [static]

#### 13.14.2.4 static void PhotonNetwork.Connect ( string serverAddress, int port, string appId, string gameVersion ) [static]

Connect to the photon server by address, port, appId and game(client) version. This method is used by Connect-UsingSettings which applies values from the settings file.

To connect to the [Photon](#) Cloud, a valid AppId must be in the settings file (shown in the [Photon](#) Cloud Dashboard). <https://cloud.exitgames.com/dashboard>

Connecting to the [Photon](#) Cloud might fail due to:

- Network issues (calls: OnFailedToConnectToPhoton())
- Invalid region (calls: OnConnectionFail() with DisconnectCause.InvalidRegion)



- Subscription CCU limit reached (calls: `OnConnectionFail()` with `DisconnectCause.MaxCcuReached`. also calls: `OnPhotonMaxCcuReached()`)

More about the connection limitations: <http://doc.exitgames.com/photon-cloud/>

#### Parameters

<i>serverAddress</i>	The master server's address (either your own or <a href="#">Photon</a> Cloud address).
<i>port</i>	The master server's port to connect to.
<i>appId</i>	Your application ID ( <a href="#">Photon</a> Cloud provides you with a GUID for your game).
<i>gameVersion</i>	This client's version number. Users are separated from each other by gameversion (which allows you to make breaking changes).

#### 13.14.2.5 static void PhotonNetwork.ConnectToBestCloudServer ( string gameVersion ) [static]

Connect to the PUN cloud server with the lowest 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. This is used for [Connect\(string serverAddress, int port, string appId, string gameVersion\)](#).

To connect to the [Photon](#) Cloud, a valid AppId must be in the settings file (shown in the [Photon](#) Cloud Dashboard). <https://cloud.exitgames.com/dashboard>

Connecting to the [Photon](#) Cloud might fail due to:

- Network issues (calls: `OnFailedToConnectToPhoton()`)
- Invalid region (calls: `OnConnectionFail()` with `DisconnectCause.InvalidRegion`)
- Subscription CCU limit reached (calls: `OnConnectionFail()` with `DisconnectCause.MaxCcuReached`. also calls: `OnPhotonMaxCcuReached()`)

More about the connection limitations: <http://doc.exitgames.com/photon-cloud>

#### Parameters

<i>gameVersion</i>	This client's version number. Users are separated from each other by gameversion (which allows you to make breaking changes).
--------------------	---

#### 13.14.2.6 static void PhotonNetwork.ConnectUsingSettings ( string gameVersion ) [static]

Connect to the configured [Photon](#) server: Reads [PhotonNetwork.serverSettingsAssetPath](#) and connects to cloud or your own server.

The PUN Setup Wizard stores your appId in a settings file and applies a server address/port. This is used for [Connect\(string serverAddress, int port, string appId, string gameVersion\)](#).

To connect to the [Photon](#) Cloud, a valid AppId must be in the settings file (shown in the [Photon](#) Cloud Dashboard). <https://cloud.exitgames.com/dashboard>

Connecting to the [Photon](#) Cloud might fail due to:

- Network issues (calls: `OnFailedToConnectToPhoton()`)
- Invalid region (calls: `OnConnectionFail()` with `DisconnectCause.InvalidRegion`)



- Subscription CCU limit reached (calls: OnConnectionFail() with DisconnectCause.MaxCcuReached. also calls: OnPhotonMaxCcuReached())

More about the connection limitations: <http://doc.exitgames.com/photon-cloud>

#### Parameters

<i>gameVersion</i>	This client's version number. Users are separated from each other by gameversion (which allows you to make breaking changes).
--------------------	---

13.14.2.7 `static void PhotonNetwork.ConnectUsingSettings ( ) [static]`

13.14.2.8 `static void PhotonNetwork.CreateRoom ( string roomName ) [static]`

Creates a room with given name but fails if this room is existing already.

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). Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName.

[PhotonNetwork.autoCleanUpPlayerObjects](#) will become this room's AutoCleanUp property and that's used by all clients that join this room.

#### Parameters

<i>roomName</i>	Unique name of the room to create.
-----------------	------------------------------------

13.14.2.9 `static void PhotonNetwork.CreateRoom ( string roomName, bool isVisible, bool isOpen, int maxPlayers ) [static]`

Creates a room with given name but fails if this room is existing already.

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). Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName

#### Parameters

<i>roomName</i>	Unique name of the room to create. Pass null or "" to make the server generate a name.
<i>isVisible</i>	Shows (or hides) this room from the lobby's listing of rooms.
<i>isOpen</i>	Allows (or disallows) others to join this room.
<i>maxPlayers</i>	Max number of players that can join the room.

13.14.2.10 `static void PhotonNetwork.CreateRoom ( string roomName, bool isVisible, bool isOpen, int maxPlayers, Hashtable customRoomProperties, string[] propsToListInLobby ) [static]`

Creates a room with given name but fails if this room is existing already.

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). Call this only on the master server. Internally, the master will respond with a server-address (and roomName, if needed). Both are used internally to switch to the assigned game server and roomName.

[PhotonNetwork.autoCleanUpPlayerObjects](#) will become this room's AutoCleanUp property and that's used by all clients that join this room.

## Parameters

<i>roomName</i>	Unique name of the room to create. Pass null or "" to make the server generate a name.
<i>isVisible</i>	Shows (or hides) this room from the lobby's listing of rooms.
<i>isOpen</i>	Allows (or disallows) others to join this room.
<i>maxPlayers</i>	Max number of players that can join the room.
<i>customRoom-Properties</i>	Custom properties of the new room (set on create, so they are immediately available).
<i>propsToListIn-Lobby</i>	Array of custom-property-names that should be forwarded to the lobby (include only the useful ones).

## 13.14.2.11 static void PhotonNetwork.Destroy ( PhotonView view ) [static]

Destroy supplied [PhotonView](#). This will remove all Buffered RPCs and destroy the GameObject this view is attached to (plus all childs, if any) This has the same effect as calling Destroy by passing a GameObject

## Parameters

<i>view</i>	
-------------	--

## 13.14.2.12 static void PhotonNetwork.Destroy ( GameObject go ) [static]

Destroys given GameObject. This GameObject must've been instantiated using [PhotonNetwork.Instantiate](#) and must have a [PhotonView](#) at it's root. This has the same effect as calling Destroy by passing an attached [PhotonView](#) from this GameObject

## Parameters

<i>go</i>	
-----------	--

## 13.14.2.13 static void PhotonNetwork.DestroyPlayerObjects ( PhotonPlayer destroyPlayer ) [static]

Destroy all GameObjects/PhotonViews of this player. can only be called on the local player. The only exception is the master client which call call this for all players.

## Parameters

<i>player</i>	
---------------	--

## 13.14.2.14 static void PhotonNetwork.Disconnect ( ) [static]

Makes this client disconnect from the photon server, a process that leaves any room and calls OnDisconnectedFromPhoton on completion.

When the client is connected, the server is being informed that this client disconnects. This speeds up leave/disconnect messages for players in the same room as you (otherwise the server would timeout this client's connection). When used in offlineMode, the state-change and event-call OnDisconnectedFromPhoton are immediate. Offline mode is set to false as well. Once disconnected, the client can connect again. Use ConnectUsingSettings.

## 13.14.2.15 static void PhotonNetwork.FetchServerTimestamp ( ) [static]

Refreshes the server timestamp (async operation, takes a roundtrip).

Can be useful if a bad connection made the timestamp unusable or imprecise.

13.14.2.16 `static bool PhotonNetwork.FindFriends ( string[] friendsToFind ) [static]`

Request the rooms and online status for a list of friends. All client must set a unique username via `PlayerName` property. The result is available in `this.Friends`.

Used on Master Server to find the rooms played by a selected list of users. The result will be mapped to `LoadBalancingClient.Friends` when available. The list is initialized by `OpFindFriends` on first use (before that, it is null).

Users identify themselves by setting a `PlayerName` in the `LoadBalancingClient` instance. This in turn will send the name in `OpAuthenticate` after each 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): `ParameterCode.FindFriendsResponseOnlineList` = `bool[]` of online states `ParameterCode.FindFriendsResponseRoomIdList` = `string[]` of room names (empty string if not in a room)

#### Parameters

<i>friendsToFind</i>	Array of friend's names (make sure they are unique).
----------------------	--

#### Returns

If the operation could be sent (requires connection, only one request is allowed at any time). Always false in offline mode.

13.14.2.17 `static int PhotonNetwork.GetPing ( ) [static]`

The current roundtrip time to the photon server

#### Returns

Roundtrip time (to server and back).

13.14.2.18 `static RoomInfo [] PhotonNetwork.GetRoomList ( ) [static]`

Gets an array of (currently) known rooms as [RoomInfo](#). This list is automatically updated every few seconds while this client is in the lobby (on the Master Server). Not available while being in a room.

Creates a new instance of the list each time called. Copied from `networkingPeer.mGameList`.

#### Returns

[RoomInfo\[\]](#) of current rooms in lobby.

13.14.2.19 `static void PhotonNetwork.InitializeSecurity ( ) [static]`

Used for compatibility with Unity networking only. Encryption is automatically initialized while connecting.

13.14.2.20 `static GameObject PhotonNetwork.Instantiate ( string prefabName, Vector3 position, Quaternion rotation, int group ) [static]`

Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.

Instead of using prefabs in the Resources folder, you can manually Instantiate and assign `PhotonViews`. See doc.

## Parameters

<i>prefabName</i>	Name of the prefab to instantiate.
<i>position</i>	Position Vector3 to apply on instantiation.
<i>rotation</i>	Rotation Quaternion to apply on instantiation.
<i>group</i>	The group for this <a href="#">PhotonView</a> .

## Returns

The new instance of a GameObject with initialized [PhotonView](#).

**13.14.2.21** `static GameObject PhotonNetwork.Instantiate ( string prefabName, Vector3 position, Quaternion rotation, int group, object[] data ) [static]`

Instantiate a prefab over the network. This prefab needs to be located in the root of a "Resources" folder.

Instead of using prefabs in the Resources folder, you can manually Instantiate and assign PhotonViews. See doc.

## Parameters

<i>prefabName</i>	Name of the prefab to instantiate.
<i>position</i>	Position Vector3 to apply on instantiation.
<i>rotation</i>	Rotation Quaternion to apply on instantiation.
<i>group</i>	The group for this <a href="#">PhotonView</a> .
<i>data</i>	Optional instantiation data. This will be saved to it's <a href="#">PhotonView.instantiationData</a> .

## Returns

The new instance of a GameObject with initialized [PhotonView](#).

**13.14.2.22** `static GameObject PhotonNetwork.InstantiateSceneObject ( string prefabName, Vector3 position, Quaternion rotation, int group, object[] data ) [static]`

Instantiate a scene-owned prefab over the network. The PhotonViews will be controllable by the MasterClient. This prefab needs to be located in the root of a "Resources" folder.

Only the master client can Instantiate scene objects. Instead of using prefabs in the Resources folder, you can manually Instantiate and assign PhotonViews. See doc.

## Parameters

<i>prefabName</i>	Name of the prefab to instantiate.
<i>position</i>	Position Vector3 to apply on instantiation.
<i>rotation</i>	Rotation Quaternion to apply on instantiation.
<i>group</i>	The group for this <a href="#">PhotonView</a> .
<i>data</i>	Optional instantiation data. This will be saved to it's <a href="#">PhotonView.instantiationData</a> .

## Returns

The new instance of a GameObject with initialized [PhotonView](#).

**13.14.2.23** `static void PhotonNetwork.InternalCleanPhotonMonoFromScenelfStuck ( ) [static]`

Internally used by Editor scripts, called on Hierarchy change (includes scene save) to remove surplus hidden PhotonHandlers.

13.14.2.24 `static void PhotonNetwork.JoinRandomRoom ( ) [static]`

Joins any available room but will fail if none is currently available.

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

13.14.2.25 `static void PhotonNetwork.JoinRandomRoom ( Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers ) [static]`

Attempts to join an open room with fitting, custom properties but fails if none is currently available.

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

#### Parameters

<i>expected-CustomRoom-Properties</i>	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
<i>expectedMax-Players</i>	Filters for a particular maxplayer setting. Use 0 to accept any maxPlayer value.

13.14.2.26 `static void PhotonNetwork.JoinRandomRoom ( Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType ) [static]`

Attempts to join an open room with fitting, custom properties but fails if none is currently available.

If this fails, you can still create a room (and make this available for the next who uses JoinRandomRoom). Alternatively, try again in a moment.

#### Parameters

<i>expected-CustomRoom-Properties</i>	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
<i>expectedMax-Players</i>	Filters for a particular maxplayer setting. Use 0 to accept any maxPlayer value.
<i>matchingType</i>	Selects one of the available matchmaking algorithms. See MatchmakingMode enum for options.

13.14.2.27 `static void PhotonNetwork.JoinRoom ( RoomInfo listedRoom ) [static]`

Join room by room.Name. This fails if the room is either full or no longer available (might close at the same time).

#### Parameters

<i>roomName</i>	The room instance to join (only listedRoom.Name is used).
-----------------	---

13.14.2.28 `static void PhotonNetwork.JoinRoom ( string roomName ) [static]`

Join room with given title. This fails if the room is either full or no longer available (might close at the same time).

## Parameters

<i>roomName</i>	Unique name of the room to create.
-----------------	------------------------------------

**13.14.2.29** `static void PhotonNetwork.LeaveRoom ( ) [static]`

Leave the current room

**13.14.2.30** `static void PhotonNetwork.LoadLevel ( int levelNumber ) [static]`

Loads the level and automatically pauses the network queue. Call this in OnJoinedRoom to make sure no cached RPCs are fired in the wrong scene.

## Parameters

<i>levelNumber</i>	Number of the level to load (make sure it's in the build preferences).
--------------------	--

**13.14.2.31** `static void PhotonNetwork.LoadLevel ( string levelTitle ) [static]`

Loads the level and automatically pauses the network queue. Call this in OnJoinedRoom to make sure no cached RPCs are fired in the wrong scene.

## Parameters

<i>levelTitle</i>	Name of the level to load.
-------------------	----------------------------

**13.14.2.32** `static void PhotonNetwork.NetworkStatisticsReset ( ) [static]`

Resets the traffic stats and re-enables them.

**13.14.2.33** `static string PhotonNetwork.NetworkStatisticsToString ( ) [static]`

Only available when NetworkStatisticsEnabled was used to gather some stats.

## Returns

A string with vital networking statistics.

**13.14.2.34** `static void PhotonNetwork.OverrideBestCloudServer ( PingCloudRegions.CloudRegions region ) [static]`

Overwrites the region that is used for [ConnectToBestCloudServer\(string gameVersion\)](#) This will overwrite the result that was gotten by pinging all cloud servers.

Use this to allow your users to specify their region manually.

**13.14.2.35** `static void PhotonNetwork.RemoveAllBufferedMessages ( ) [static]`

Remove ALL buffered messages of the local player (RPC's and Instantiation calls) Note that this only removed the buffered messages on the server, you will still need to remove the Instantiated GameObjects yourself.

13.14.2.36 `static void PhotonNetwork.RemoveAllBufferedMessages ( PhotonPlayer targetPlayer ) [static]`

Remove ALL buffered messages of a player (RPC's and Instantiation calls) Note that this only removed the buffered messages on the server, you will still need to remove the Instantiated GameObjects yourself.

13.14.2.37 `static void PhotonNetwork.RemoveAllInstantiatedObjects ( ) [static]`

MasterClient method only: Destroy ALL instantiated GameObjects

13.14.2.38 `static void PhotonNetwork.RemoveAllInstantiatedObjects ( PhotonPlayer targetPlayer ) [static]`

Destroy ALL PhotonNetwork.Instantiated GameObjects by given player. Can only be called on the local player or MasterClient. The MasterClient can call this for all players.

#### Parameters

<i>player</i>	
---------------	--

13.14.2.39 `static void PhotonNetwork.RemoveRPCs ( ) [static]`

Remove ALL buffered RPCs of the local player

13.14.2.40 `static void PhotonNetwork.RemoveRPCs ( PhotonPlayer targetPlayer ) [static]`

Remove ALL buffered RPCs of a player

13.14.2.41 `static void PhotonNetwork.RemoveRPCs ( PhotonView view ) [static]`

Remove all buffered RPCs on given [PhotonView](#) (if they are owned by this player).

#### Parameters

<i>view</i>	
-------------	--

13.14.2.42 `static void PhotonNetwork.RemoveRPCsInGroup ( int group ) [static]`

Remove all buffered RPCs with given group

#### Parameters

<i>group</i>	
--------------	--

13.14.2.43 `static void PhotonNetwork.SendOutgoingCommands ( ) [static]`

Can be used to immediately send the RPCs and Instantiates just made, 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.

#### 13.14.2.44 static void PhotonNetwork.SetLevelPrefix ( short *prefix* ) [static]

Sets level prefix for PhotonViews instantiated later on. Don't set it if you need only one!

Important: If you don't use multiple level prefixes, simply don't set this value. The default value is optimized out of the traffic.

This won't affect existing PhotonViews (they can't be changed yet for existing PhotonViews).

Messages sent with a different level prefix will be received but not executed. This affects RPCs, Instantiates and synchronization.

Be aware that PUN never resets this value, you'll have to do so yourself.

##### Parameters

<i>prefix</i>	Max value is short.MaxValue = 32767
---------------	-------------------------------------

#### 13.14.2.45 static void PhotonNetwork.SetPlayerCustomProperties ( Hashtable *customProperties* ) [static]

Sets this (local) player's properties. This caches the properties in PhotonNetwork.player.customProperties. CreateRoom, JoinRoom and JoinRandomRoom will all apply your player's custom properties when you enter the room. While in a room, your properties are synced with the other players. 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

<i>custom-Properties</i>	Only string-typed keys will be used from this hashtable. If null, custom properties are all deleted.
--------------------------	--

#### 13.14.2.46 static void PhotonNetwork.SetReceivingEnabled ( int *group*, bool *enabled* ) [static]

Enable/disable receiving on given group (applied to PhotonViews)

##### Parameters

<i>group</i>	
<i>enabled</i>	

#### 13.14.2.47 static void PhotonNetwork.SetSendingEnabled ( int *group*, bool *enabled* ) [static]

Enable/disable sending on given group (applied to PhotonViews)

##### Parameters

<i>group</i>	
<i>enabled</i>	

#### 13.14.2.48 static void PhotonNetwork.UnAllocateViewID ( int *viewID* ) [static]

Unregister a viewID (of manually instantiated and destroyed networked objects).



## Parameters

<i>viewID</i>	A viewID manually allocated by this player.
---------------	---

## 13.14.3 Member Data Documentation

13.14.3.1 **PhotonLogLevel** PhotonNetwork.logLevel = PhotonLogLevel.ErrorsOnly [static]

Network log level. Controls how verbose PUN is.

13.14.3.2 **readonly int** PhotonNetwork.MAX\_VIEW\_IDS = 1000 [static]

The maximum amount of assigned PhotonViews PER player (or scene). See the documentation on how to raise this limitation

13.14.3.3 **ServerSettings** PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(PHOTON\_SERVER\_SETTINGS\_ASSET\_FILE, typeof(ServerSettings)) [static]

Serialized server settings, written by the Setup Wizard for use in ConnectUsingSettings.

13.14.3.4 **float** PhotonNetwork.precisionForFloatSynchronization = 0.01f [static]

The minimum difference between floats before we send it via a PhotonView's OnSerialize/ObservingComponent

13.14.3.5 **float** PhotonNetwork.precisionForQuaternionSynchronization = 1.0f [static]

The minimum angle that a rotation needs to change before we send it via a PhotonView's OnSerialize/ObservingComponent

13.14.3.6 **float** PhotonNetwork.precisionForVectorSynchronization = 0.000099f [static]

The minimum difference that a Vector2 or Vector3(e.g. a transforms rotation) needs to change before we send it via a PhotonView's OnSerialize/ObservingComponent Note that this is the sqrMagnitude. E.g. to send only after a 0.01 change on the Y-axis, we use 0.01f\*0.01f=0.0001f. As a remedy against float inaccuracy we use 0.000099f instead of 0.0001f.

13.14.3.7 **Dictionary<string, GameObject>** PhotonNetwork.PrefabCache = new Dictionary<string, GameObject>() [static]

Keeps references to GameObjects for frequent instantiation (out of memory instead of loading the Resources).

You should be able to modify the cache anytime you like, except while Instantiate is used. Best do it only in the main-Thread.

13.14.3.8 **const string** PhotonNetwork.serverSettingsAssetFile = "PhotonServerSettings"

Name of the PhotonServerSettings file (used to load and by PhotonEditor to save new files).

13.14.3.9 **const string** PhotonNetwork.serverSettingsAssetPath = "Assets/Photon Unity Networking/Resources/" + PhotonNetwork.serverSettingsAssetFile + ".asset"

Path to the PhotonServerSettings file (used by PhotonEditor).

#### 13.14.3.10 `bool PhotonNetwork.UsePrefabCache = true` [static]

While enabled (true), Instantiate uses `PhotonNetwork.PrefabCache` to keep game objects in memory (improving instantiation of the same prefab).

Setting UsePrefabCache to false during runtime will not clear PrefabCache but will ignore it right away. You could clean and modify the cache yourself. Read its comments.

#### 13.14.3.11 `const string PhotonNetwork.versionPUN = "1.20"`

Version number of PUN. Also used in GameVersion to separate client version from each other.

### 13.14.4 Property Documentation

#### 13.14.4.1 `AuthenticationValues PhotonNetwork.AuthValues` [static], [get], [set]

A user's authentication values used during connect for Custom Authentication with `Photon` (and a custom service/-community). Set these before calling Connect if you want custom authentication.

If authentication fails for any values, PUN will call your implementation of `OnCustomAuthenticationFailed(string debugMsg)`. See: `PhotonNetworkingMessage.OnCustomAuthenticationFailed`

#### 13.14.4.2 `bool PhotonNetwork.autoCleanUpPlayerObjects` [static], [get], [set]

This setting defines if players in a room should destroy a leaving player's instantiated GameObjects and PhotonViews.

When "this client" creates a room/game, autoCleanUpPlayerObjects is copied to that room's properties and used by all PUN clients in that room (no matter what their autoCleanUpPlayerObjects value is).

If room.AutoCleanUp is enabled in a room, the PUN clients will destroy a player's objects on leave.

When enabled, the server will clean RPCs, instantiated GameObjects and PhotonViews of the leaving player and joining players won't get those anymore.

Once a room is created, this setting can't be changed anymore.

Enabled by default.

#### 13.14.4.3 `bool PhotonNetwork.autoJoinLobby` [static], [get], [set]

Defines if the `PhotonNetwork` should join the "lobby" when connected to the Master server. If this is false, `OnConnectedToMaster()` will be called when connection to the Master is available. `OnJoinedLobby()` will NOT be called if this is false.

Enabled by default.

The room listing will not become available. Rooms can be created and joined (randomly) without joining the lobby (and getting sent the room list).

#### 13.14.4.4 `bool PhotonNetwork.automaticallySyncScene` [static], [get], [set]

If true, PUN will make sure that all users are in the same scene at all times. If the MasterClient switches, all clients will load the new scene. This also takes care of smooth loading of the game scene after joining a game from your main menu.

true if automatically sync scene; otherwise, false.

13.14.4.5 **bool PhotonNetwork.connected** [static], [get]

Are we connected to the photon server (can be IN or OUTSIDE a room)

13.14.4.6 **ConnectionState PhotonNetwork.connectionState** [static], [get]

Simplified connection state

13.14.4.7 **PeerState PhotonNetwork.connectionStateDetailed** [static], [get]

Detailed connection state (ignorant of PUN, so it can be "disconnected" while switching servers).

13.14.4.8 **int PhotonNetwork.countOfPlayers** [static], [get]

The count of players currently using this application. This is updated on the MasterServer (only) in 5sec intervals (if any count changed).

13.14.4.9 **int PhotonNetwork.countOfPlayersInRooms** [static], [get]

The count of players currently inside a room This is updated on the MasterServer (only) in 5sec intervals (if any count changed).

13.14.4.10 **int PhotonNetwork.countOfPlayersOnMaster** [static], [get]

The count of players currently looking for a room. This is updated on the MasterServer (only) in 5sec intervals (if any count changed).

13.14.4.11 **int PhotonNetwork.countOfRooms** [static], [get]

The count of rooms currently in use. When inside the lobby this is based on [PhotonNetwork.GetRoomList\(\)](#).Length. When not inside the lobby, this value updated on the MasterServer (only) in 5sec intervals (if any count changed).

13.14.4.12 **List<FriendInfo> PhotonNetwork.Friends** [static], [get], [set]

Read-only list of friends, their online status and the room they are in. Null until initialized by a FindFriends call.

Do not modify this list! It's internally handles by FindFriends and only useful to read the values. The value of FriendsListAge tells you how old the data is in milliseconds.

Don't get this list more often than useful (> 10 seconds). In best case, keep the list you fetch really short. You could (e.g.) get the full list only once, then request a few updates only for friends who are online. After a while (e.g. 1 minute), you can get the full list again (to update online states).

13.14.4.13 **int PhotonNetwork.FriendsListAge** [static], [get]

Age of friend list info (in milliseconds). It's 0 until a friend list is fetched.

13.14.4.14 **bool PhotonNetwork.insideLobby** [static], [get]

Returns true when we are connected to [Photon](#) and in the lobby state

**13.14.4.15** `bool PhotonNetwork.isMasterClient` `[static], [get]`

Are we the master client?

**13.14.4.16** `bool PhotonNetwork.isMessageQueueRunning` `[static], [get], [set]`

Can be used to pause dispatch of incoming events (RPCs, Instantiates and anything else incoming). 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.

**13.14.4.17** `bool PhotonNetwork.isNonMasterClientInRoom` `[static], [get]`

True if we are in a room (client) and NOT the room's masterclient

**13.14.4.18** `PhotonPlayer PhotonNetwork.masterClient` `[static], [get]`

The [PhotonPlayer](#) of the master client. The master client is the 'virtual owner' of the room. You can use it if you need authoritative decision made by one of the players.

The masterClient is null until a room is joined and becomes null again when the room is left.

**13.14.4.19** `int PhotonNetwork.maxConnections` `[static], [get], [set]`

The maximum number of players for a room. Better: Set it in CreateRoom. If no room is opened, this will return 0.

**13.14.4.20** `bool PhotonNetwork.NetworkStatisticsEnabled` `[static], [get], [set]`

Enables or disables the collection of statistics about this client's traffic. If you encounter issues with clients, the traffic stats are a good starting point to find solutions.

Only with enabled stats, you can use GetVitalStats

**13.14.4.21** `bool PhotonNetwork.offlineMode` `[static], [get], [set]`

Offline mode can be set to re-use your multiplayer code in singleplayer game modes. When this is on [PhotonNetwork](#) will not create any connections and there is near to no overhead. Mostly useful for reusing RPC's and [PhotonNetwork.Instantiate](#)

**13.14.4.22** `PhotonPlayer [] PhotonNetwork.otherPlayers` `[static], [get]`

The other PhotonPlayers, not including our local player.

**13.14.4.23** `PhotonPlayer PhotonNetwork.player` `[static], [get]`

The local [PhotonPlayer](#). Always available and represents this player. CustomProperties can be set before entering a room and will be synced as well.

**13.14.4.24** `PhotonPlayer [] PhotonNetwork.playerList` `[static], [get]`

The full [PhotonPlayer](#) list, including the local player.

**13.14.4.25** `string PhotonNetwork.playerName` `[static], [get], [set]`

This local player's name.

Setting the name will automatically send it, if connected. Setting null, won't change the name.

**13.14.4.26** `Room PhotonNetwork.room` `[static], [get]`

Get the room we're currently in. Null if we aren't in any room.

**13.14.4.27** `int PhotonNetwork.sendRate` `[static], [get], [set]`

Defines how many times per second [PhotonNetwork](#) should send a package. If you change this, do not forget to also change 'sendRateOnSerialize'.

Less packages are less overhead but more delay. Setting the sendRate to 50 will create up to 50 packages per second (which is a lot!). Keep your target platform in mind: mobile networks are slower and less reliable.

**13.14.4.28** `int PhotonNetwork.sendRateOnSerialize` `[static], [get], [set]`

Defines how many times per second OnPhotonSerialize should be called on PhotonViews.

Choose this value in relation to 'sendRate'. OnPhotonSerialize will create the commands to be put into packages. A lower rate takes up less performance but will cause more lag.

**13.14.4.29** `string PhotonNetwork.ServerAddress` `[static], [get]`

Currently used server address (no matter if master or game server).

**13.14.4.30** `double PhotonNetwork.time` `[static], [get]`

[Photon](#) network time, synched with the server

v1.3: This time reflects milliseconds since start of the server, cut down to 4 bytes. It will overflow every 49 days from a high value to 0. We do not (yet) compensate this overflow. Master- and Game-Server will have different time values. v1.10: Fixed issues with precision for high server-time values. This should update with 15ms precision by default.

**13.14.4.31** `int PhotonNetwork.unreliableCommandsLimit` `[static], [get], [set]`

Used once per dispatch to limit unreliable commands per channel (so after a pause, many channels can still cause a lot of unreliable commands)

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonNetwork.cs](#)

## 13.15 PhotonPlayer Class Reference

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

## Public Member Functions

- [PhotonPlayer](#) (bool [isLocal](#), int actorID, string [name](#))  
*Creates a [PhotonPlayer](#) instance.*
- override string [ToString](#) ()  
*Gives the name.*
- override bool [Equals](#) (object p)  
*Makes [PhotonPlayer](#) comparable*
- override int [GetHashCode](#) ()
- void [SetCustomProperties](#) (Hashtable propertiesToSet)  
*Updates and synchronizes the named properties of this Player with the values of propertiesToSet.*

## Static Public Member Functions

- static [PhotonPlayer Find](#) (int [ID](#))  
*Try to get a specific player by id.*

## Public Attributes

- readonly bool [isLocal](#) = false  
*Only one player is controlled by each client. Others are not local.*

## Protected Member Functions

- [PhotonPlayer](#) (bool [isLocal](#), int actorID, Hashtable properties)  
*Internally used to create players from event Join*

## Properties

- int [ID](#) [get]  
*This player's actorID*
- string [name](#) [get, set]  
*Nickname of this player.*
- bool [isMasterClient](#) [get]  
*The player with the lowest actorID is the master and could be used for special tasks.*
- Hashtable [customProperties](#) [get, set]  
*Read-only cache for custom properties of player. Set via [Player.SetCustomProperties](#).*
- Hashtable [allProperties](#) [get]  
*Creates a Hashtable with all properties (custom and "well known" ones).*

### 13.15.1 Detailed Description

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

Each player has an actorId (or ID), valid for that room. It's -1 until it's assigned by server. Each client can set it's player's custom properties with [SetCustomProperties](#), even before being in a room. They are synced when joining a room.

## 13.15.2 Constructor & Destructor Documentation

### 13.15.2.1 PhotonPlayer.PhotonPlayer ( bool *isLocal*, int *actorID*, string *name* )

Creates a [PhotonPlayer](#) instance.

#### Parameters

<i>isLocal</i>	If this is the local peer's player (or a remote one).
<i>actorID</i>	ID or ActorNumber of this player in the current room (a shortcut to identify each player in room)
<i>name</i>	Name of the player (a "well known property").

### 13.15.2.2 PhotonPlayer.PhotonPlayer ( bool *isLocal*, int *actorID*, Hashtable *properties* ) [protected]

Internally used to create players from event Join

## 13.15.3 Member Function Documentation

### 13.15.3.1 override bool PhotonPlayer.Equals ( object *p* )

Makes [PhotonPlayer](#) comparable

### 13.15.3.2 static PhotonPlayer PhotonPlayer.Find ( int *ID* ) [static]

Try to get a specific player by id.

#### Parameters

<i>ID</i>	ActorID
-----------	---------

#### Returns

The player with matching actorID or null, if the actorID is not in use.

### 13.15.3.3 override int PhotonPlayer.GetHashCode ( )

### 13.15.3.4 void PhotonPlayer.SetCustomProperties ( Hashtable *propertiesToSet* )

Updates and synchronizes the named properties of this Player with the values of propertiesToSet.

Any player's properties are available in a [Room](#) only and only until the player disconnect or leaves. Access any player's properties by: Player.CustomProperties (read-only!) but don't modify that hashtable.

New properties are added, existing values are updated. Other values will not be changed, so only provide values that changed or are new. To delete a named (custom) property of this player, use null as value. Only string-typed keys are applied (everything else is ignored).

Local cache is updated immediately, other players are updated through [Photon](#) with a fitting operation. To reduce network traffic, set only values that actually changed.

#### Parameters

<i>propertiesToSet</i>	Hashtable of props to update, set and sync. See description.
------------------------	--

### 13.15.3.5 override string PhotonPlayer.ToString ( )

Gives the name.

## 13.15.4 Member Data Documentation

### 13.15.4.1 readonly bool PhotonPlayer.isLocal = false

Only one player is controlled by each client. Others are not local.

## 13.15.5 Property Documentation

### 13.15.5.1 Hashtable PhotonPlayer.allProperties [get]

Creates a Hashtable with all properties (custom and "well known" ones).

If used more often, this should be cached.

### 13.15.5.2 Hashtable PhotonPlayer.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.

### 13.15.5.3 int PhotonPlayer.ID [get]

This player's actorID

### 13.15.5.4 bool PhotonPlayer.isMasterClient [get]

The player with the lowest actorID is the master and could be used for special tasks.

### 13.15.5.5 string PhotonPlayer.name [get], [set]

Nickname of this player.

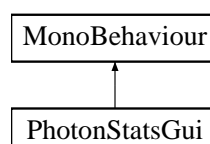
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonPlayer.cs](#)

## 13.16 PhotonStatsGui Class Reference

Basic GUI to show traffic and health statistics of the connection to [Photon](#), toggled by shift+tab.

Inheritance diagram for PhotonStatsGui:





## 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).*

### 13.16.1 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).

### 13.16.2 Member Function Documentation

13.16.2.1 void `PhotonStatsGui.OnGUI` ( )

13.16.2.2 void `PhotonStatsGui.Start` ( )

13.16.2.3 void `PhotonStatsGui.TrafficStatsWindow` ( int *windowID* )

13.16.2.4 void `PhotonStatsGui.Update` ( )

Checks for shift+tab input combination (to toggle statsOn).

### 13.16.3 Member Data Documentation

13.16.3.1 bool `PhotonStatsGui.buttonsOn`

Show buttons to control stats and reset them.

### 13.16.3.2 bool PhotonStatsGui.healthStatsVisible

Shows additional "health" values of connection.

### 13.16.3.3 bool PhotonStatsGui.statsOn = true

Option to turn collecting stats on or off (used in [Update\(\)](#)).

### 13.16.3.4 Rect PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)

Positioning rect for window.

### 13.16.3.5 bool PhotonStatsGui.statsWindowOn = true

Shows or hides GUI (does not affect if stats are collected).

### 13.16.3.6 bool PhotonStatsGui.trafficStatsOn

Shows additional "lower level" traffic stats.

### 13.16.3.7 int PhotonStatsGui.WindowId = 100

Unity GUI Window ID (must be unique or will cause issues).

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonStatsGui.cs](#)

## 13.17 PhotonStream Class Reference

This "container" class is used to carry your data as written by OnPhotonSerializeView.

### Public Member Functions

- [PhotonStream](#) (bool write, object[] incomingData)
- object [ReceiveNext](#) ()
- void [SendNext](#) (object obj)
- object[] [ToArray](#) ()
- void [Serialize](#) (ref bool myBool)
- void [Serialize](#) (ref int myInt)
- void [Serialize](#) (ref string value)
- void [Serialize](#) (ref char value)
- void [Serialize](#) (ref short value)
- void [Serialize](#) (ref float obj)
- void [Serialize](#) (ref [PhotonPlayer](#) obj)
- void [Serialize](#) (ref Vector3 obj)
- void [Serialize](#) (ref Vector2 obj)
- void [Serialize](#) (ref Quaternion obj)

## Properties

- bool [isWriting](#) [get]
- bool [isReading](#) [get]
- int [Count](#) [get]

### 13.17.1 Detailed Description

This "container" class is used to carry your data as written by OnPhotonSerializeView.

#### See Also

[PhotonNetworkingMessage](#)

### 13.17.2 Constructor & Destructor Documentation

13.17.2.1 `PhotonStream.PhotonStream ( bool write, object[] incomingData )`

### 13.17.3 Member Function Documentation

13.17.3.1 `object PhotonStream.ReceiveNext ( )`

13.17.3.2 `void PhotonStream.SendNext ( object obj )`

13.17.3.3 `void PhotonStream.Serialize ( ref bool myBool )`

13.17.3.4 `void PhotonStream.Serialize ( ref int myInt )`

13.17.3.5 `void PhotonStream.Serialize ( ref string value )`

13.17.3.6 `void PhotonStream.Serialize ( ref char value )`

13.17.3.7 `void PhotonStream.Serialize ( ref short value )`

13.17.3.8 `void PhotonStream.Serialize ( ref float obj )`

13.17.3.9 `void PhotonStream.Serialize ( ref PhotonPlayer obj )`

13.17.3.10 `void PhotonStream.Serialize ( ref Vector3 obj )`

13.17.3.11 `void PhotonStream.Serialize ( ref Vector2 obj )`

13.17.3.12 `void PhotonStream.Serialize ( ref Quaternion obj )`

13.17.3.13 `object [] PhotonStream.ToArray ( )`

### 13.17.4 Property Documentation

13.17.4.1 `int PhotonStream.Count` [get]

13.17.4.2 `bool PhotonStream.isReading` [get]

13.17.4.3 `bool PhotonStream.isWriting` [get]

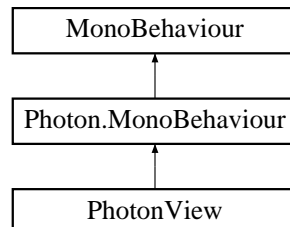
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonClasses.cs](#)

## 13.18 PhotonView Class Reference

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

Inheritance diagram for PhotonView:



### Public Member Functions

- void [Awake](#) ()  
*Called by Unity on start of the application and does a setup the [PhotonView](#).*
- void [OnApplicationQuit](#) ()
- void [OnDestroy](#) ()
- void [RPC](#) (string methodName, [PhotonTargets](#) target, params object[] parameters)
- void [RPC](#) (string methodName, [PhotonPlayer](#) targetPlayer, params object[] parameters)
- override string [ToString](#) ()

### Static Public Member Functions

- static [PhotonView Get](#) (Component component)
- static [PhotonView Get](#) (GameObject gameObj)
- static [PhotonView Find](#) (int viewID)

### Public Attributes

- int [subId](#)
- int [ownerId](#)
- int [group](#) = 0
- int [prefixBackup](#) = -1
- Component [observed](#)
- [ViewSynchronization](#) [synchronization](#)
- [OnSerializeTransform](#) [onSerializeTransformOption](#) = OnSerializeTransform.PositionAndRotation
- [OnSerializeRigidBody](#) [onSerializeRigidBodyOption](#) = OnSerializeRigidBody.All
- 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]

- bool [isSceneView](#) [get]
- [PhotonPlayer](#) owner [get]
- int [OwnerActorNr](#) [get]
- bool [isMine](#) [get]

*Is this photonView mine? True in case the owner matches the local [PhotonPlayer](#) ALSO true if this is a scene photonview on the Master client*

### 13.18.1 Detailed Description

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

### 13.18.2 Member Function Documentation

#### 13.18.2.1 void PhotonView.Awake ( )

Called by Unity on start of the application and does a setup the [PhotonView](#).

#### 13.18.2.2 static PhotonView PhotonView.Find ( int viewID ) [static]

#### 13.18.2.3 static PhotonView PhotonView.Get ( Component component ) [static]

#### 13.18.2.4 static PhotonView PhotonView.Get ( GameObject gameObj ) [static]

#### 13.18.2.5 void PhotonView.OnApplicationQuit ( )

#### 13.18.2.6 void PhotonView.OnDestroy ( )

#### 13.18.2.7 void PhotonView.RPC ( string methodName, PhotonTargets target, params object[] parameters )

#### 13.18.2.8 void PhotonView.RPC ( string methodName, PhotonPlayer targetPlayer, params object[] parameters )

#### 13.18.2.9 override string PhotonView.ToString ( )

### 13.18.3 Member Data Documentation

#### 13.18.3.1 int PhotonView.group = 0

#### 13.18.3.2 int PhotonView.instantiationId

#### 13.18.3.3 Component PhotonView.observed

#### 13.18.3.4 OnSerializeRigidBody PhotonView.onSerializeRigidBodyOption = OnSerializeRigidBody.All

#### 13.18.3.5 OnSerializeTransform PhotonView.onSerializeTransformOption = OnSerializeTransform.PositionAndRotation

#### 13.18.3.6 int PhotonView.ownerId

#### 13.18.3.7 int PhotonView.prefixBackup = -1

#### 13.18.3.8 int PhotonView.subId

#### 13.18.3.9 ViewSynchronization PhotonView.synchronization

### 13.18.4 Property Documentation

13.18.4.1 `object [] PhotonView.instantiationData` `[get]`, `[set]`

This is the instantiationData that was passed when calling [PhotonNetwork.Instantiate\\*](#) (if that was used to spawn this prefab)

13.18.4.2 `bool PhotonView.isMine` `[get]`

Is this photonView mine? True in case the owner matches the local [PhotonPlayer](#) ALSO true if this is a scene photonview on the Master client

13.18.4.3 `bool PhotonView.isSceneView` `[get]`

13.18.4.4 `PhotonPlayer PhotonView.owner` `[get]`

13.18.4.5 `int PhotonView.OwnerActorNr` `[get]`

13.18.4.6 `int PhotonView.prefix` `[get]`, `[set]`

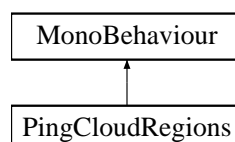
13.18.4.7 `int PhotonView.viewID` `[get]`, `[set]`

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PhotonView.cs](#)

## 13.19 PingCloudRegions Class Reference

Inheritance diagram for PingCloudRegions:



### Public Types

- enum [CloudRegions](#) { [us](#), [eu](#), [asia](#) }

### Public Member Functions

- IEnumerator [PingAllRegions](#) ()

### Static Public Member Functions

- static void [OverrideRegion](#) ([CloudRegions](#) region)
- static void [ConnectToBestRegion](#) (string gameVersion)
- static string [ResolveHost](#) (string hostString)

## Static Public Attributes

- static [PingCloudRegions.SP](#)

## 13.19.1 Member Enumeration Documentation

### 13.19.1.1 enum PingCloudRegions.CloudRegions

Enumerator:

***us***  
***eu***  
***asia***

## 13.19.2 Member Function Documentation

13.19.2.1 static void PingCloudRegions.ConnectToBestRegion ( string *gameVersion* ) [static]

13.19.2.2 static void PingCloudRegions.OverrideRegion ( CloudRegions *region* ) [static]

13.19.2.3 IEnumerator PingCloudRegions.PingAllRegions ( )

13.19.2.4 static string PingCloudRegions.ResolveHost ( string *hostString* ) [static]

## 13.19.3 Member Data Documentation

13.19.3.1 PingCloudRegions PingCloudRegions.SP [static]

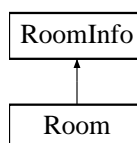
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[PingCloudRegions.cs](#)

## 13.20 Room Class Reference

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

Inheritance diagram for Room:



## Public Member Functions

- void [SetCustomProperties](#) (Hashtable *propertiesToSet*)

*Updates and synchronizes the named properties of this [Room](#) with the values of *propertiesToSet*.*

## Properties

- new int [playerCount](#) [get]  
*Count of players in this room.*
- new string [name](#) [get, set]  
*The name of a room. Unique identifier (per Loadbalancing group) for a room/match.*
- new int [maxPlayers](#) [get, set]  
*Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.*
- new bool [open](#) [get, set]  
*Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "visible" to not list the room.*
- new bool [visible](#) [get, set]  
*Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.*
- string[] [propertiesListedInLobby](#) [get, set]  
*A list of custom properties that should be forwarded to the lobby and listed there.*
- bool [autoCleanUp](#) [get]  
*Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.*

## Additional Inherited Members

### 13.20.1 Detailed Description

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

### 13.20.2 Member Function Documentation

#### 13.20.2.1 void Room.SetCustomProperties ( Hashtable *propertiesToSet* )

Updates and synchronizes the named properties of this [Room](#) with the values of *propertiesToSet*.

Any player can set a [Room](#)'s properties. [Room](#) properties are available until changed, deleted or until the last player leaves the room. Access them by: `Room.CustomProperties` (read-only!).

New properties are added, existing values are updated. Other values will not be changed, so only provide values that changed or are new. To delete a named (custom) property of this room, use null as value. Only string-typed keys are applied (everything else is ignored).

Local cache is updated immediately, other clients are updated through [Photon](#) with a fitting operation. To reduce network traffic, set only values that actually changed.

#### Parameters

<i>propertiesToSet</i>	Hashtable of props to update, set and sync. See description.
------------------------	--

### 13.20.3 Property Documentation



**13.20.3.1** `bool Room.autoCleanUp` `[get]`

Gets if this room uses autoCleanUp to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.

**13.20.3.2** `new int Room.maxPlayers` `[get], [set]`

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

**13.20.3.3** `new string Room.name` `[get], [set]`

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

**13.20.3.4** `new bool Room.open` `[get], [set]`

Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "visible" to not list the room.

**13.20.3.5** `new int Room.playerCount` `[get]`

Count of players in this room.

**13.20.3.6** `string [] Room.propertiesListedInLobby` `[get], [set]`

A list of custom properties that should be forwarded to the lobby and listed there.

**13.20.3.7** `new bool Room.visible` `[get], [set]`

Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

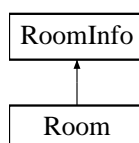
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[Room.cs](#)

## 13.21 RoomInfo Class Reference

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

Inheritance diagram for RoomInfo:



## Public Member Functions

- override bool [Equals](#) (object p)  
*Makes [RoomInfo](#) comparable (by name).*
- override int [GetHashCode](#) ()  
*Accompanies Equals, using the name's [HashCode](#) as return.*
- override string [ToString](#) ()  
*Simple printing in method.*

## Protected Attributes

- byte [maxPlayersField](#) = 0  
*Backing field for property.*
- bool [openField](#) = true  
*Backing field for property.*
- bool [visibleField](#) = true  
*Backing field for property.*
- bool [autoCleanUpField](#) = false  
*Backing field for property. False unless the [GameProperty](#) is set to true (else it's not sent).*
- string [nameField](#)  
*Backing field for property.*

## Properties

- bool [removedFromList](#) [get, set]  
*Used internally in lobby, to mark rooms that are no longer listed.*
- Hashtable [customProperties](#) [get]  
*Read-only "cache" of custom properties of a room. Set via [Room.SetCustomProperties](#) (not available for [RoomInfo](#) class!).*
- string [name](#) [get]  
*The name of a room. Unique identifier (per Loadbalancing group) for a room/match.*
- int [playerCount](#) [get, set]  
*Only used internally in lobby, to display number of players in room (while you're not in).*
- bool [isLocalClientInside](#) [get, set]  
*State if the local client is already in the game or still going to join it on gameserver (in lobby always false).*
- byte [maxPlayers](#) [get]  
*Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.*
- bool [open](#) [get]  
*Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.*
- bool [visible](#) [get]  
*Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.*

### 13.21.1 Detailed Description

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

This class resembles info about available rooms, as sent by the Master server's lobby. Consider all values as readonly. None are synced (only updated by events by server).

### 13.21.2 Member Function Documentation

#### 13.21.2.1 override bool RoomInfo.Equals ( object *p* )

Makes [RoomInfo](#) comparable (by name).

#### 13.21.2.2 override int RoomInfo.GetHashCode ( )

Accompanies Equals, using the name's HashCode as return.

Returns

#### 13.21.2.3 override string RoomInfo.ToString ( )

Simple printingin method.

Returns

String showing the [RoomInfo](#).

### 13.21.3 Member Data Documentation

#### 13.21.3.1 bool RoomInfo.autoCleanUpField = false [protected]

Backing field for property. False unless the GameProperty is set to true (else it's not sent).

#### 13.21.3.2 byte RoomInfo.maxPlayersField = 0 [protected]

Backing field for property.

#### 13.21.3.3 string RoomInfo.nameField [protected]

Backing field for property.

#### 13.21.3.4 bool RoomInfo.openField = true [protected]

Backing field for property.

#### 13.21.3.5 bool RoomInfo.visibleField = true [protected]

Backing field for property.

### 13.21.4 Property Documentation

#### 13.21.4.1 Hashtable RoomInfo.customProperties [get]

Read-only "cache" of custom properties of a room. Set via [Room.SetCustomProperties](#) (not available for [RoomInfo](#) class!).

All keys are string-typed and the values depend on the game/application.

#### 13.21.4.2 `bool RoomInfo.isLocalClientInside` `[get]`, `[set]`

State if the local client is already in the game or still going to join it on gameserver (in lobby always false).

#### 13.21.4.3 `byte RoomInfo.maxPlayers` `[get]`

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

#### 13.21.4.4 `string RoomInfo.name` `[get]`

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

#### 13.21.4.5 `bool RoomInfo.open` `[get]`

Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.

As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

#### 13.21.4.6 `int RoomInfo.playerCount` `[get]`, `[set]`

Only used internally in lobby, to display number of players in room (while you're not in).

#### 13.21.4.7 `bool RoomInfo.removedFromList` `[get]`, `[set]`

Used internally in lobby, to mark rooms that are no longer listed.

#### 13.21.4.8 `bool RoomInfo.visible` `[get]`

Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

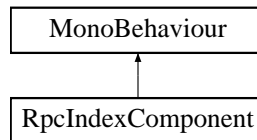
As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[RoomInfo.cs](#)

## 13.22 RpcIndexComponent Class Reference

Inheritance diagram for RpcIndexComponent:



### Public Member Functions

- void [Awake](#) ()
- byte [GetShortcut](#) (string method)

### Public Attributes

- string[] [RpcIndex](#)

#### 13.22.1 Member Function Documentation

13.22.1.1 void [RpcIndexComponent.Awake](#) ( )

13.22.1.2 byte [RpcIndexComponent.GetShortcut](#) ( string *method* )

#### 13.22.2 Member Data Documentation

13.22.2.1 string [] [RpcIndexComponent.RpcIndex](#)

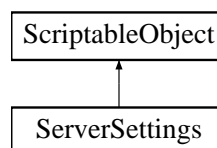
The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[RpcIndexComponent.cs](#)

## 13.23 ServerSettings Class Reference

Collection of connection-relevant settings, used internally by [PhotonNetwork.ConnectUsingSettings](#).

Inheritance diagram for ServerSettings:



### Public Types

- enum [HostingOption](#) { [NotSet](#), [PhotonCloud](#), [SelfHosted](#), [OfflineMode](#) }

### Public Member Functions

- void [UseCloud](#) (string cloudAppid, int regionIndex)
- void [UseMyServer](#) (string serverAddress, int serverPort, string application)
- override string [ToString](#) ()

## Static Public Member Functions

- static int [FindRegionForServerAddress](#) (string server)
- static string [FindServerAddressForRegion](#) (int regionIndex)

## Public Attributes

- [HostingOption HostType](#) = HostingOption.NotSet
- string [ServerAddress](#) = [DefaultServerAddress](#)
- int [ServerPort](#) = 5055
- string [AppID](#) = ""
- List< string > [RpcList](#)
- bool [DisableAutoOpenWizard](#)

## Static Public Attributes

- static string [DefaultCloudServerUrl](#) = "app-eu.exitgamescloud.com"
- static string[] [CloudServerRegionNames](#) = new string[] { "EU", "US", "Asia", "Japan" }
- static string[] [CloudServerRegionPrefixes](#) = new string[] { "app-eu", "app-us", "app-asia", "app-jp" }
- static string [DefaultServerAddress](#) = "127.0.0.1"
- static int [DefaultMasterPort](#) = 5055
- static string [DefaultAppID](#) = "Master"

### 13.23.1 Detailed Description

Collection of connection-relevant settings, used internally by [PhotonNetwork.ConnectUsingSettings](#).

### 13.23.2 Member Enumeration Documentation

#### 13.23.2.1 enum [ServerSettings.HostingOption](#)

Enumerator:

***NotSet***  
***PhotonCloud***  
***SelfHosted***  
***OfflineMode***

### 13.23.3 Member Function Documentation

- 13.23.3.1 static int [ServerSettings.FindRegionForServerAddress](#) ( string *server* ) [static]
- 13.23.3.2 static string [ServerSettings.FindServerAddressForRegion](#) ( int *regionIndex* ) [static]
- 13.23.3.3 override string [ServerSettings.ToString](#) ( )
- 13.23.3.4 void [ServerSettings.UseCloud](#) ( string *cloudAppid*, int *regionIndex* )
- 13.23.3.5 void [ServerSettings.UseMyServer](#) ( string *serverAddress*, int *serverPort*, string *application* )

### 13.23.4 Member Data Documentation

- 13.23.4.1 `string ServerSettings.AppID = ""`
- 13.23.4.2 `string [] ServerSettings.CloudServerRegionNames = new string[] { "EU", "US", "Asia", "Japan" } [static]`
- 13.23.4.3 `string [] ServerSettings.CloudServerRegionPrefixes = new string[] { "app-eu", "app-us", "app-asia", "app-jp" } [static]`
- 13.23.4.4 `string ServerSettings.DefaultAppID = "Master" [static]`
- 13.23.4.5 `string ServerSettings.DefaultCloudServerUrl = "app-eu.exitgamescloud.com" [static]`
- 13.23.4.6 `int ServerSettings.DefaultMasterPort = 5055 [static]`
- 13.23.4.7 `string ServerSettings.DefaultServerAddress = "127.0.0.1" [static]`
- 13.23.4.8 `bool ServerSettings.DisableAutoOpenWizard`
- 13.23.4.9 `HostingOption ServerSettings.HostType = HostingOption.NotSet`
- 13.23.4.10 `List<string> ServerSettings.RpcList`
- 13.23.4.11 `string ServerSettings.ServerAddress = DefaultServerAddress`
- 13.23.4.12 `int ServerSettings.ServerPort = 5055`

The documentation for this class was generated from the following file:

- Photon Unity Networking/Plugins/PhotonNetwork/[ServerSettings.cs](#)





## Chapter 14

# File Documentation

### 14.1 `_Doc/general.md` File Reference

### 14.2 `_Doc/main.md` File Reference

### 14.3 `_Doc/optionalGui.md` File Reference

### 14.4 `_Doc/photonStatsGui.md` File Reference

### 14.5 `_Doc/publicApi.md` File Reference

### 14.6 Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs File Reference

#### 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.*

### 14.7 Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs File Reference

#### Enumerations

- enum [ConnectionState](#) {  
[Disconnected](#), [Connecting](#), [Connected](#), [Disconnecting](#),  
[InitializingApplication](#) }  
*High level connection state of the client. Better use the more detailed [PeerState](#).*
- enum [PeerState](#) {  
[Uninitialized](#), [PeerCreated](#), [Connecting](#), [Connected](#),  
[Queued](#), [Authenticated](#), [JoinedLobby](#), [DisconnectingFromMasterserver](#),  
[ConnectingToGameserver](#), [ConnectedToGameserver](#), [Joining](#), [Joined](#),  
[Leaving](#), [DisconnectingFromGameserver](#), [ConnectingToMasterserver](#), [ConnectedComingFromGameserver](#),  
[QueuedComingFromGameserver](#), [Disconnecting](#), [Disconnected](#), [ConnectedToMaster](#) }  
*Detailed connection / networking peer state. PUN implements a loadbalancing and authentication workflow "behind the scenes", so some states will automatically advance to some follow up state. Those states are commented with "(will-change)".*

- enum [PhotonNetworkingMessage](#) {  
[OnConnectedToPhoton](#), [OnLeftRoom](#), [OnMasterClientSwitched](#), [OnPhotonCreateRoomFailed](#),  
[OnPhotonJoinRoomFailed](#), [OnCreatedRoom](#), [OnJoinedLobby](#), [OnLeftLobby](#),  
[OnDisconnectedFromPhoton](#), [OnConnectionFail](#), [OnFailedToConnectToPhoton](#), [OnReceivedRoomList-Update](#),  
[OnJoinedRoom](#), [OnPhotonPlayerConnected](#), [OnPhotonPlayerDisconnected](#), [OnPhotonRandomJoinFailed](#),  
[OnConnectedToMaster](#), [OnPhotonSerializeView](#), [OnPhotonInstantiate](#), [OnPhotonMaxCcuReached](#),  
[OnPhotonCustomRoomPropertiesChanged](#), [OnPhotonPlayerPropertiesChanged](#), [OnUpdatedFriendList](#),  
[OnCustomAuthenticationFailed](#) }

*This enum makes up the set of MonoMessages sent by [Photon](#) Unity Networking. Implement any of these constant names as method and it will be called in the respective situation.*

- enum [DisconnectCause](#) {  
[ExceptionOnConnect](#) = StatusCode.ExceptionOnConnect, [SecurityExceptionOnConnect](#) = StatusCode.-SecurityExceptionOnConnect, [TimeoutDisconnect](#) = StatusCode.TimeoutDisconnect, [InternalReceive-Exception](#) = StatusCode.InternalReceiveException,  
[DisconnectByServer](#) = StatusCode.DisconnectByServer, [DisconnectByServerLogic](#) = StatusCode.-DisconnectByServerLogic, [DisconnectByServerUserLimit](#) = StatusCode.DisconnectByServerUserLimit, [Ex-ception](#) = StatusCode.Exception,  
[InvalidRegion](#) = ErrorCode.InvalidRegion, [MaxCcuReached](#) = ErrorCode.MaxCcuReached }

*Summarizes the cause for a disconnect. Used in: [OnConnectionFail](#) and [OnFailedToConnectToPhoton](#).*

## 14.7.1 Enumeration Type Documentation

### 14.7.1.1 enum [ConnectionState](#)

High level connection state of the client. Better use the more detailed [PeerState](#).

Enumerator:

***[Disconnected](#)***  
***[Connecting](#)***  
***[Connected](#)***  
***[Disconnecting](#)***  
***[InitializingApplication](#)***

## 14.8 Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs File Reference

### Classes

- class [Extensions](#)  
*This static class defines some useful extension methods for several existing classes (e.g. [Vector3](#), [float](#) and others).*

### Typedefs

- using [SupportClass](#) = ExitGames.Client.Photon.SupportClass

### 14.8.1 Typedef Documentation

#### 14.8.1.1 using [SupportClass](#) = ExitGames.Client.Photon.SupportClass

## 14.9 Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs File Reference

## Classes

- class [FriendInfo](#)

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

## 14.10 Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs File Reference

### Classes

- class **LoadbalancingPeer**

*Internally used by PUN, a LoadbalancingPeer provides the operations and enum definitions needed to use the [Photon](#) Loadbalancing server (or the [Photon](#) Cloud).*

- class [ErrorCode](#)

*Class for constants. These (int) values represent error codes, as defined and sent by the [Photon](#) LoadBalancing logic. Pun uses these constants internally.*

- class [ActorProperties](#)

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

- class [GameProperties](#)

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

- class [EventCode](#)

*Class for constants. These values are for events defined by [Photon](#) Loadbalancing. Pun uses these constants internally.*

- class [ParameterCode](#)

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

- class [OperationCode](#)

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

- class [AuthenticationValues](#)

*Container for "Custom Authentication" values in [Photon](#) (default: user and token). Set AuthParameters before connecting - all else is handled.*

### Enumerations

- enum [MatchmakingMode](#) : byte { [FillRoom](#) = 0, [SerialMatching](#) = 1, [RandomMatching](#) = 2 }

*Options for matchmaking rules for OpJoinRandom.*

- enum [CustomAuthenticationType](#) : byte { [Custom](#) = 0, [None](#) = byte.MaxValue }

*Options for optional "Custom Authentication" services used with [Photon](#). Used by OpAuthenticate after connecting to [Photon](#).*

### 14.10.1 Enumeration Type Documentation

#### 14.10.1.1 enum CustomAuthenticationType : byte

Options for optional "Custom Authentication" services used with [Photon](#). Used by OpAuthenticate after connecting to [Photon](#).

Enumerator:

**Custom** Use a custom authentication service. Currently the only implemented option.

**None** Disables custom authentication. Same as not providing any [AuthenticationValues](#) for connect (more precisely for: OpAuthenticate).

#### 14.10.1.2 enum MatchmakingMode : byte

Options for matchmaking rules for OpJoinRandom.

Enumerator:

**FillRoom** Fills up rooms (oldest first) to get players together as fast as possible. Default. Makes most sense with MaxPlayers > 0 and games that can only start with more players.

**SerialMatching** Distributes players across available rooms sequentially but takes filter into account. Without filter, rooms get players evenly distributed.

**RandomMatching** Joins a (fully) random room. Expected properties must match but aside from this, any available room might be selected.

## 14.11 Photon Unity Networking/Plugins/PhotonNetwork/NetworkingPeer.cs File Reference

### Classes

- class **NetworkingPeer**

Implements [Photon](#) LoadBalancing used in PUN. This class is used internally by [PhotonNetwork](#) and not intended as public API.

## 14.12 Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs File Reference

### Classes

- class **PhotonNetworkMessages**

Class for constants. Defines photon-event-codes for PUN usage.

- class [Photon.MonoBehaviour](#)

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

- class [PhotonMessageInfo](#)

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

- class [PBitStream](#)

- class [PhotonStream](#)

This "container" class is used to carry your data as written by OnPhotonSerializeView.

### Namespaces

- package [Photon](#)

### Enumerations

- enum [PhotonTargets](#) {  
All, Others, MasterClient, AllBuffered,  
OthersBuffered }

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

- enum [PhotonLogLevel](#) { ErrorsOnly, Informational, Full }

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

## 14.13 Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs File Reference

### Classes

- class **PhotonHandler**  
*Internal MonoBehaviour that allows [Photon](#) to run an Update loop.*

## 14.14 Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs File Reference

### Classes

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

## 14.15 Photon Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs File Reference

### Classes

- class [PhotonNetwork](#)  
*The main class to use the [PhotonNetwork](#) plugin. This class is static.*

### Typedefs

- using [Debug](#) = UnityEngine.Debug

#### 14.15.1 Typedef Documentation

##### 14.15.1.1 using Debug = UnityEngine.Debug

## 14.16 Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs File Reference

### Classes

- class [PhotonPlayer](#)  
*Summarizes a "player" within a room, identified (in that room) by actorID.*

## 14.17 Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs File Reference

### Classes

- class [PhotonStatsGui](#)  
*Basic GUI to show traffic and health statistics of the connection to [Photon](#), toggled by shift+tab.*

## 14.18 Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs File Reference

### Classes

- class [PhotonView](#)

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

### Enumerations

- enum [ViewSynchronization](#) { [Off](#), [ReliableDeltaCompressed](#), [Unreliable](#) }
- enum [OnSerializeTransform](#) { [OnlyPosition](#), [OnlyRotation](#), [OnlyScale](#), [PositionAndRotation](#), [All](#) }
- enum [OnSerializeRigidBody](#) { [OnlyVelocity](#), [OnlyAngularVelocity](#), [All](#) }

### 14.18.1 Enumeration Type Documentation

#### 14.18.1.1 enum OnSerializeRigidBody

Enumerator:

***OnlyVelocity***  
***OnlyAngularVelocity***  
***All***

#### 14.18.1.2 enum OnSerializeTransform

Enumerator:

***OnlyPosition***  
***OnlyRotation***  
***OnlyScale***  
***PositionAndRotation***  
***All***

#### 14.18.1.3 enum ViewSynchronization

Enumerator:

***Off***  
***ReliableDeltaCompressed***  
***Unreliable***

## 14.19 Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs File Reference

### Classes

- class [PingCloudRegions](#)

## 14.20 Photon Unity Networking/Plugins/PhotonNetwork/Room.cs File Reference

### Classes

- class [Room](#)

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

## 14.21 Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs File Reference

### Classes

- class [RoomInfo](#)

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

## 14.22 Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs File Reference

### Classes

- class [RpcIndexComponent](#)

## 14.23 Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs File Reference

### Classes

- class [ServerSettings](#)

*Collection of connection-relevant settings, used internally by [PhotonNetwork.ConnectUsingSettings](#).*

# Index

[\\_Doc/general.md](#), 97  
[\\_Doc/main.md](#), 97  
[\\_Doc/optionalGui.md](#), 97  
[\\_Doc/photonStatsGui.md](#), 97  
[\\_Doc/publicApi.md](#), 97

ActorList  
    ParameterCode, 52  
ActorNr  
    ParameterCode, 52  
ActorProperties, 37  
    PlayerName, 37  
Add  
    ParameterCode, 52  
    PBitStream, 55  
Address  
    ParameterCode, 52  
All  
    PhotonView.cs, 102  
    Public API, 34  
AllBuffered  
    Public API, 34  
allProperties  
    PhotonPlayer, 80  
AllocateViewID  
    PhotonNetwork, 63  
AlmostEquals  
    Extensions, 44  
AlreadyMatched  
    ErrorCode, 40  
AppID  
    ServerSettings, 94  
AppStats  
    EventCode, 42  
AppVersion  
    ParameterCode, 52  
ApplicationId  
    ParameterCode, 52  
asia  
    PingCloudRegions, 87  
AuthParameters  
    AuthenticationValues, 38  
AuthType  
    AuthenticationValues, 38  
AuthValues  
    PhotonNetwork, 74  
Authenticate  
    OperationCode, 49  
Authenticated  
    Public API, 32

AuthenticationValues, 37  
    AuthParameters, 38  
    AuthType, 38  
    Secret, 38  
    SetAuthParameters, 38  
autoCleanUp  
    Room, 88  
autoCleanUpField  
    RoomInfo, 91  
autoCleanUpPlayerObjects  
    PhotonNetwork, 74  
autoJoinLobby  
    PhotonNetwork, 74  
automaticallySyncScene  
    PhotonNetwork, 74  
Awake  
    PhotonView, 85  
    RpcIndexComponent, 93  
AzureNodeInfo  
    EventCode, 42  
  
BitCount  
    PBitStream, 56  
Broadcast  
    ParameterCode, 52  
buttonsOn  
    PhotonStatsGui, 81  
ByteCount  
    PBitStream, 56  
BytesForBits  
    PBitStream, 55  
  
Cache  
    ParameterCode, 52  
ChangeGroups  
    OperationCode, 49  
CleanupCacheOnLeave  
    GameProperties, 47  
    ParameterCode, 52  
ClientAuthenticationParams  
    ParameterCode, 52  
ClientAuthenticationType  
    ParameterCode, 52  
CloseConnection  
    PhotonNetwork, 63  
CloudRegions  
    PingCloudRegions, 87  
CloudServerRegionNames  
    ServerSettings, 95  
CloudServerRegionPrefixes



- ServerSettings, 95
- Code
  - ParameterCode, 53
- Connect
  - PhotonNetwork, 63
- ConnectToBestCloudServer
  - PhotonNetwork, 64
- ConnectToBestRegion
  - PingCloudRegions, 87
- ConnectUsingSettings
  - PhotonNetwork, 64, 65
- Connected
  - Enums.cs, 98
  - Public API, 32
- connected
  - PhotonNetwork, 74
- ConnectedComingFromGameserver
  - Public API, 32
- ConnectedToGameserver
  - Public API, 32
- ConnectedToMaster
  - Public API, 32
- Connecting
  - Enums.cs, 98
  - Public API, 32
- ConnectingToGameserver
  - Public API, 32
- ConnectingToMasterserver
  - Public API, 32
- ConnectionState
  - Enums.cs, 98
- connectionState
  - PhotonNetwork, 75
- connectionStateDetailed
  - PhotonNetwork, 75
- Contains
  - Extensions, 44
- Count
  - PhotonStream, 83
- countOfPlayers
  - PhotonNetwork, 75
- countOfPlayersInRooms
  - PhotonNetwork, 75
- countOfPlayersOnMaster
  - PhotonNetwork, 75
- countOfRooms
  - PhotonNetwork, 75
- CreateGame
  - OperationCode, 49
- CreateRoom
  - PhotonNetwork, 65
- Custom
  - LoadbalancingPeer.cs, 99
- CustomAuthenticationFailed
  - ErrorCode, 40
- CustomAuthenticationType
  - LoadbalancingPeer.cs, 99
- CustomEventContent
  - ParameterCode, 53
- customProperties
  - PhotonPlayer, 80
  - RoomInfo, 91
- Data
  - ParameterCode, 53
- Debug
  - PhotonNetwork.cs, 101
- DefaultAppID
  - ServerSettings, 95
- DefaultCloudServerUrl
  - ServerSettings, 95
- DefaultMasterPort
  - ServerSettings, 95
- DefaultServerAddress
  - ServerSettings, 95
- Destroy
  - PhotonNetwork, 66
- DestroyPlayerObjects
  - PhotonNetwork, 66
- DisableAutoOpenWizard
  - ServerSettings, 95
- Disconnect
  - PhotonNetwork, 66
- DisconnectByServer
  - Public API, 31
- DisconnectByServerLogic
  - Public API, 31
- DisconnectByServerUserLimit
  - Public API, 31
- DisconnectCause
  - Public API, 31
- Disconnected
  - Enums.cs, 98
  - Public API, 32
- Disconnecting
  - Enums.cs, 98
  - Public API, 32
- DisconnectingFromGameserver
  - Public API, 32
- DisconnectingFromMasterserver
  - Public API, 32
- Enums.cs
  - Connected, 98
  - Connecting, 98
  - Disconnected, 98
  - Disconnecting, 98
  - InitializingApplication, 98
- Enums.cs
  - ConnectionState, 98
- Equals
  - PhotonPlayer, 79
  - RoomInfo, 91
- ErrorCode, 39
  - AlreadyMatched, 40
  - CustomAuthenticationFailed, 40
  - GameClosed, 40

- GameDoesNotExist, 40
- GameFull, 40
- GameIdAlreadyExists, 40
- InternalServerError, 40
- InvalidAuthentication, 40
- InvalidOperationCode, 40
- InvalidRegion, 40
- MaxCcuReached, 40
- NoRandomMatchFound, 41
- Ok, 41
- OperationNotAllowedInCurrentState, 41
- ServerFull, 41
- UserBlocked, 41
- ErrorsOnly
  - Public API, 32
- eu
  - PingCloudRegions, 87
- EventCode, 41
  - AppStats, 42
  - AzureNodeInfo, 42
  - GameList, 42
  - GameListUpdate, 42
  - Join, 42
  - Leave, 42
  - Match, 43
  - PropertiesChanged, 43
  - QueueState, 43
  - SetProperties, 43
- Exception
  - Public API, 31
- ExceptionOnConnect
  - Public API, 31
- Extensions, 43
  - AlmostEquals, 44
  - Contains, 44
  - GetPhotonView, 44
  - GetPhotonViewsInChildren, 44
  - Merge, 44
  - MergeStringKeys, 45
  - StripKeysWithNullValues, 45
  - StripToStringKeys, 45
  - ToStringFull, 45
- Extensions.cs
  - SupportClass, 98
- FetchServerTimestamp
  - PhotonNetwork, 66
- FillRoom
  - LoadbalancingPeer.cs, 100
- Find
  - PhotonPlayer, 79
  - PhotonView, 85
- FindFriends
  - OperationCode, 49
  - PhotonNetwork, 66
- FindFriendsRequestList
  - ParameterCode, 53
- FindFriendsResponseOnlineList
  - ParameterCode, 53
- FindFriendsResponseRoomIdList
  - ParameterCode, 53
- FindRegionForServerAddress
  - ServerSettings, 94
- FindServerAddressForRegion
  - ServerSettings, 94
- FriendInfo, 46
  - IsInRoom, 46
  - IsOnline, 46
  - Name, 46
  - Room, 46
  - ToString, 46
- Friends
  - PhotonNetwork, 75
- FriendsListAge
  - PhotonNetwork, 75
- Full
  - Public API, 32
- GameClosed
  - ErrorCode, 40
- GameCount
  - ParameterCode, 53
- GameDoesNotExist
  - ErrorCode, 40
- GameFull
  - ErrorCode, 40
- GameIdAlreadyExists
  - ErrorCode, 40
- GameList
  - EventCode, 42
  - ParameterCode, 53
- GameListUpdate
  - EventCode, 42
- GameProperties, 46
  - CleanupCacheOnLeave, 47
  - IsOpen, 47
  - IsVisible, 47
  - MaxPlayers, 47
  - ParameterCode, 53
  - PlayerCount, 47
  - PropsListedInLobby, 47
  - Removed, 47
- Get
  - PBitStream, 55
  - PhotonView, 85
- GetHashCode
  - PhotonPlayer, 79
  - RoomInfo, 91
- GetNext
  - PBitStream, 55
- GetPhotonView
  - Extensions, 44
- GetPhotonViewsInChildren
  - Extensions, 44
- GetPing
  - PhotonNetwork, 67
- GetProperties
  - OperationCode, 49

- GetRoomList
  - PhotonNetwork, [67](#)
- GetShortcut
  - RpclIndexComponent, [93](#)
- Group
  - ParameterCode, [53](#)
- group
  - PhotonView, [85](#)
- healthStatsVisible
  - PhotonStatsGui, [81](#)
- HostType
  - ServerSettings, [95](#)
- HostingOption
  - ServerSettings, [94](#)
- ID
  - PhotonPlayer, [80](#)
- Informational
  - Public API, [32](#)
- InitializeSecurity
  - PhotonNetwork, [67](#)
- InitializingApplication
  - Enums.cs, [98](#)
- insideLobby
  - PhotonNetwork, [75](#)
- Instantiate
  - PhotonNetwork, [67](#), [68](#)
- InstantiateSceneObject
  - PhotonNetwork, [68](#)
- instantiationData
  - PhotonView, [86](#)
- instantiationId
  - PhotonView, [85](#)
- InternalReceiveException
  - Public API, [31](#)
- InternalCleanPhotonMonoFromScenelfStuck
  - PhotonNetwork, [68](#)
- InternalServerError
  - ErrorCode, [40](#)
- InvalidRegion
  - Public API, [31](#)
- InvalidAuthentication
  - ErrorCode, [40](#)
- InvalidOperationCode
  - ErrorCode, [40](#)
- InvalidRegion
  - ErrorCode, [40](#)
- IsInRoom
  - FriendInfo, [46](#)
- isLocal
  - PhotonPlayer, [80](#)
- isLocalClientInside
  - RoomInfo, [91](#)
- isMasterClient
  - PhotonNetwork, [75](#)
  - PhotonPlayer, [80](#)
- isMessageQueueRunning
  - PhotonNetwork, [76](#)
- isMine
  - PhotonView, [86](#)
- isNonMasterClientInRoom
  - PhotonNetwork, [76](#)
- IsOnline
  - FriendInfo, [46](#)
- IsOpen
  - GameProperties, [47](#)
- isReading
  - PhotonStream, [83](#)
- isSceneView
  - PhotonView, [86](#)
- IsVisible
  - GameProperties, [47](#)
- isWriting
  - PhotonStream, [83](#)
- Join
  - EventCode, [42](#)
- JoinGame
  - OperationCode, [49](#)
- JoinLobby
  - OperationCode, [49](#)
- JoinRandomGame
  - OperationCode, [49](#)
- JoinRandomRoom
  - PhotonNetwork, [68](#), [69](#)
- JoinRoom
  - PhotonNetwork, [69](#)
- Joined
  - Public API, [32](#)
- JoinedLobby
  - Public API, [32](#)
- Joining
  - Public API, [32](#)
- Leave
  - EventCode, [42](#)
  - OperationCode, [50](#)
- LeaveLobby
  - OperationCode, [50](#)
- LeaveRoom
  - PhotonNetwork, [70](#)
- Leaving
  - Public API, [32](#)
- LoadLevel
  - PhotonNetwork, [70](#)
- LoadbalancingPeer.cs
  - Custom, [99](#)
  - FillRoom, [100](#)
  - None, [99](#)
  - RandomMatching, [100](#)
  - SerialMatching, [100](#)
- LoadbalancingPeer.cs
  - CustomAuthenticationType, [99](#)
  - MatchmakingMode, [99](#)
- logLevel
  - PhotonNetwork, [73](#)

- MAX\_VIEW\_IDS
  - PhotonNetwork, [73](#)
- MasterClient
  - Public API, [34](#)
- masterClient
  - PhotonNetwork, [76](#)
- MasterPeerCount
  - ParameterCode, [53](#)
- Match
  - EventCode, [43](#)
- MatchMakingType
  - ParameterCode, [54](#)
- MatchmakingMode
  - LoadbalancingPeer.cs, [99](#)
- MaxCcuReached
  - Public API, [31](#)
- MaxCcuReached
  - ErrorCode, [40](#)
- maxConnections
  - PhotonNetwork, [76](#)
- MaxPlayers
  - GameProperties, [47](#)
- maxPlayers
  - Room, [89](#)
  - RoomInfo, [92](#)
- maxPlayersField
  - RoomInfo, [91](#)
- Merge
  - Extensions, [44](#)
- MergeStringKeys
  - Extensions, [45](#)
- Name
  - FriendInfo, [46](#)
- name
  - PhotonPlayer, [80](#)
  - Room, [89](#)
  - RoomInfo, [92](#)
- nameField
  - RoomInfo, [91](#)
- NetworkStatisticsEnabled
  - PhotonNetwork, [76](#)
- NetworkStatisticsReset
  - PhotonNetwork, [70](#)
- NetworkStatisticsToString
  - PhotonNetwork, [70](#)
- networkView
  - Photon::MonoBehaviour, [48](#)
- NoRandomMatchFound
  - ErrorCode, [41](#)
- None
  - LoadbalancingPeer.cs, [99](#)
- NotSet
  - ServerSettings, [94](#)
- observed
  - PhotonView, [85](#)
- Off
  - PhotonView.cs, [102](#)
- OfflineMode
  - ServerSettings, [94](#)
- offlineMode
  - PhotonNetwork, [76](#)
- Ok
  - ErrorCode, [41](#)
- OnConnectedToMaster
  - Public API, [33](#)
- OnConnectedToPhoton
  - Public API, [32](#)
- OnConnectionFail
  - Public API, [33](#)
- OnCreatedRoom
  - Public API, [33](#)
- OnCustomAuthenticationFailed
  - Public API, [34](#)
- OnDisconnectedFromPhoton
  - Public API, [33](#)
- OnFailedToConnectToPhoton
  - Public API, [33](#)
- OnJoinedLobby
  - Public API, [33](#)
- OnJoinedRoom
  - Public API, [33](#)
- OnLeftLobby
  - Public API, [33](#)
- OnLeftRoom
  - Public API, [32](#)
- OnMasterClientSwitched
  - Public API, [33](#)
- OnPhotonCreateRoomFailed
  - Public API, [33](#)
- OnPhotonCustomRoomPropertiesChanged
  - Public API, [34](#)
- OnPhotonInstantiate
  - Public API, [33](#)
- OnPhotonJoinRoomFailed
  - Public API, [33](#)
- OnPhotonMaxCccuReached
  - Public API, [33](#)
- OnPhotonPlayerConnected
  - Public API, [33](#)
- OnPhotonPlayerDisconnected
  - Public API, [33](#)
- OnPhotonPlayerPropertiesChanged
  - Public API, [34](#)
- OnPhotonRandomJoinFailed
  - Public API, [33](#)
- OnPhotonSerializeView
  - Public API, [33](#)
- OnReceivedRoomListUpdate
  - Public API, [33](#)
- OnUpdatedFriendList
  - Public API, [34](#)
- OnApplicationQuit
  - PhotonView, [85](#)
- OnDestroy
  - PhotonView, [85](#)

- OnGUI
  - PhotonLagSimulationGui, [57](#)
  - PhotonStatsGui, [81](#)
- OnSerializeRigidBody
  - PhotonView.cs, [102](#)
- onSerializeRigidBodyOption
  - PhotonView, [85](#)
- OnSerializeTransform
  - PhotonView.cs, [102](#)
- onSerializeTransformOption
  - PhotonView, [85](#)
- OnlyAngularVelocity
  - PhotonView.cs, [102](#)
- OnlyPosition
  - PhotonView.cs, [102](#)
- OnlyRotation
  - PhotonView.cs, [102](#)
- OnlyScale
  - PhotonView.cs, [102](#)
- OnlyVelocity
  - PhotonView.cs, [102](#)
- open
  - Room, [89](#)
  - RoomInfo, [92](#)
- openField
  - RoomInfo, [91](#)
- OperationCode, [48](#)
  - Authenticate, [49](#)
  - ChangeGroups, [49](#)
  - CreateGame, [49](#)
  - FindFriends, [49](#)
  - GetProperties, [49](#)
  - JoinGame, [49](#)
  - JoinLobby, [49](#)
  - JoinRandomGame, [49](#)
  - Leave, [50](#)
  - LeaveLobby, [50](#)
  - RaiseEvent, [50](#)
  - SetProperties, [50](#)
- OperationNotAllowedInCurrentState
  - ErrorCode, [41](#)
- Optional Gui Elements, [29](#)
- otherPlayers
  - PhotonNetwork, [76](#)
- Others
  - Public API, [34](#)
- OthersBuffered
  - Public API, [34](#)
- OverrideBestCloudServer
  - PhotonNetwork, [70](#)
- OverrideRegion
  - PingCloudRegions, [87](#)
- owner
  - PhotonView, [86](#)
- OwnerActorNr
  - PhotonView, [86](#)
- ownerId
  - PhotonView, [85](#)
- PBitStream, [55](#)
  - Add, [55](#)
  - BitCount, [56](#)
  - ByteCount, [56](#)
  - BytesForBits, [55](#)
  - Get, [55](#)
  - GetNext, [55](#)
  - PBitStream, [55](#)
  - PBitStream, [55](#)
  - Position, [56](#)
  - Set, [55](#)
  - ToBytes, [55](#)
- ParameterCode, [50](#)
  - ActorList, [52](#)
  - ActorNr, [52](#)
  - Add, [52](#)
  - Address, [52](#)
  - AppVersion, [52](#)
  - ApplicationId, [52](#)
  - Broadcast, [52](#)
  - Cache, [52](#)
  - CleanupCacheOnLeave, [52](#)
  - ClientAuthenticationParams, [52](#)
  - ClientAuthenticationType, [52](#)
  - Code, [53](#)
  - CustomEventContent, [53](#)
  - Data, [53](#)
  - FindFriendsRequestList, [53](#)
  - FindFriendsResponseOnlineList, [53](#)
  - FindFriendsResponseRoomIdList, [53](#)
  - GameCount, [53](#)
  - GameList, [53](#)
  - GameProperties, [53](#)
  - Group, [53](#)
  - MasterPeerCount, [53](#)
  - MatchMakingType, [54](#)
  - PeerCount, [54](#)
  - PlayerProperties, [54](#)
  - Position, [54](#)
  - Properties, [54](#)
  - ReceiverGroup, [54](#)
  - Remove, [54](#)
  - RoomName, [54](#)
  - Secret, [54](#)
  - TargetActorNr, [54](#)
  - UserId, [54](#)
- Peer
  - PhotonLagSimulationGui, [57](#)
- PeerCreated
  - Public API, [31](#)
- PeerCount
  - ParameterCode, [54](#)
- PeerState
  - Public API, [31](#)
- Photon, [35](#)
  - Photon Unity Networking/Plugins/PhotonNetwork/-CustomTypes.cs, [97](#)

- Photon Unity Networking/Plugins/PhotonNetwork/-Enums.cs, [97](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-Extensions.cs, [98](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-FriendInfo.cs, [98](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-LoadbalancingPeer.cs, [99](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-NetworkingPeer.cs, [100](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonClasses.cs, [100](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonHandler.cs, [101](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonLagSimulationGui.cs, [101](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonNetwork.cs, [101](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonPlayer.cs, [101](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonStatsGui.cs, [101](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-PhotonView.cs, [102](#)
- Photon Unity Networking/Plugins/PhotonNetwork/Ping-CloudRegions.cs, [102](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-Room.cs, [103](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-RoomInfo.cs, [103](#)
- Photon Unity Networking/Plugins/PhotonNetwork/Rpc-IndexComponent.cs, [103](#)
- Photon Unity Networking/Plugins/PhotonNetwork/-ServerSettings.cs, [103](#)
- Photon.MonoBehaviour, [48](#)
- PhotonCloud
  - ServerSettings, [94](#)
- PhotonView.cs
  - All, [102](#)
  - Off, [102](#)
  - OnlyAngularVelocity, [102](#)
  - OnlyPosition, [102](#)
  - OnlyRotation, [102](#)
  - OnlyScale, [102](#)
  - OnlyVelocity, [102](#)
  - PositionAndRotation, [102](#)
  - ReliableDeltaCompressed, [102](#)
  - Unreliable, [102](#)
- Photon::MonoBehaviour
  - networkView, [48](#)
  - photonView, [48](#)
- PhotonLagSimulationGui, [56](#)
  - OnGUI, [57](#)
  - Peer, [57](#)
  - Start, [57](#)
  - Visible, [57](#)
  - WindowId, [57](#)
  - WindowRect, [57](#)
- PhotonLogLevel
  - Public API, [32](#)
- PhotonMessageInfo, [57](#)
  - PhotonMessageInfo, [58](#)
  - photonView, [58](#)
  - PhotonMessageInfo, [58](#)
  - sender, [58](#)
  - timestamp, [58](#)
  - ToString, [58](#)
- PhotonNetwork, [58](#)
  - AllocateViewID, [63](#)
  - AuthValues, [74](#)
  - autoCleanUpPlayerObjects, [74](#)
  - autoJoinLobby, [74](#)
  - automaticallySyncScene, [74](#)
  - CloseConnection, [63](#)
  - Connect, [63](#)
  - ConnectToBestCloudServer, [64](#)
  - ConnectUsingSettings, [64](#), [65](#)
  - connected, [74](#)
  - connectionState, [75](#)
  - connectionStateDetailed, [75](#)
  - countOfPlayers, [75](#)
  - countOfPlayersInRooms, [75](#)
  - countOfPlayersOnMaster, [75](#)
  - countOfRooms, [75](#)
  - CreateRoom, [65](#)
  - Destroy, [66](#)
  - DestroyPlayerObjects, [66](#)
  - Disconnect, [66](#)
  - FetchServerTimestamp, [66](#)
  - FindFriends, [66](#)
  - Friends, [75](#)
  - FriendsListAge, [75](#)
  - GetPing, [67](#)
  - GetRoomList, [67](#)
  - InitializeSecurity, [67](#)
  - insideLobby, [75](#)
  - Instantiate, [67](#), [68](#)
  - InstantiateSceneObject, [68](#)
  - InternalCleanPhotonMonoFromScenelfStuck, [68](#)
  - isMasterClient, [75](#)
  - isMessageQueueRunning, [76](#)
  - isNonMasterClientInRoom, [76](#)
  - JoinRandomRoom, [68](#), [69](#)
  - JoinRoom, [69](#)
  - LeaveRoom, [70](#)
  - LoadLevel, [70](#)
  - logLevel, [73](#)
  - MAX\_VIEW\_IDS, [73](#)
  - masterClient, [76](#)
  - maxConnections, [76](#)
  - NetworkStatisticsEnabled, [76](#)
  - NetworkStatisticsReset, [70](#)
  - NetworkStatisticsToString, [70](#)
  - offlineMode, [76](#)
  - otherPlayers, [76](#)
  - OverrideBestCloudServer, [70](#)

- PhotonServerSettings, 73
- player, 76
- playerList, 76
- playerName, 76
- precisionForFloatSynchronization, 73
- precisionForQuaternionSynchronization, 73
- precisionForVectorSynchronization, 73
- PrefabCache, 73
- RemoveAllBufferedMessages, 70
- RemoveAllInstantiatedObjects, 71
- RemoveRPCs, 71
- RemoveRPCsInGroup, 71
- room, 77
- SendOutgoingCommands, 71
- sendRate, 77
- sendRateOnSerialize, 77
- ServerAddress, 77
- serverSettingsAssetFile, 73
- serverSettingsAssetPath, 73
- SetLevelPrefix, 71
- SetPlayerCustomProperties, 72
- SetReceivingEnabled, 72
- SetSendingEnabled, 72
- time, 77
- UnAllocateViewID, 72
- unreliableCommandsLimit, 77
- UsePrefabCache, 73
- versionPUN, 74
- PhotonNetwork.cs
  - Debug, 101
- PhotonNetworkingMessage
  - Public API, 32
- PhotonPlayer, 77
  - allProperties, 80
  - customProperties, 80
  - Equals, 79
  - Find, 79
  - GetHashCode, 79
  - ID, 80
  - isLocal, 80
  - isMasterClient, 80
  - name, 80
  - PhotonPlayer, 79
  - PhotonPlayer, 79
  - SetCustomProperties, 79
  - ToString, 79
- PhotonServerSettings
  - PhotonNetwork, 73
- PhotonStatsGui, 80
  - buttonsOn, 81
  - healthStatsVisible, 81
  - OnGUI, 81
  - Start, 81
  - statsOn, 82
  - statsRect, 82
  - statsWindowOn, 82
  - trafficStatsOn, 82
  - TrafficStatsWindow, 81
- Update, 81
- WindowId, 82
- PhotonStream, 82
  - Count, 83
  - isReading, 83
  - isWriting, 83
  - PhotonStream, 83
  - PhotonStream, 83
  - ReceiveNext, 83
  - SendNext, 83
  - Serialize, 83
  - ToArray, 83
- PhotonTargets
  - Public API, 34
- PhotonView, 84
  - Awake, 85
  - Find, 85
  - Get, 85
  - group, 85
  - instantiationData, 86
  - instantiationId, 85
  - isMine, 86
  - isSceneView, 86
  - observed, 85
  - OnApplicationQuit, 85
  - OnDestroy, 85
  - onSerializeRigidBodyOption, 85
  - onSerializeTransformOption, 85
  - owner, 86
  - OwnerActorNr, 86
  - ownerId, 85
  - prefix, 86
  - prefixBackup, 85
  - RPC, 85
  - subId, 85
  - synchronization, 85
  - ToString, 85
  - viewID, 86
- photonView
  - Photon::MonoBehaviour, 48
  - PhotonMessageInfo, 58
- PhotonView.cs
  - OnSerializeRigidBody, 102
  - OnSerializeTransform, 102
  - ViewSynchronization, 102
- PingCloudRegions
  - asia, 87
  - eu, 87
  - us, 87
- PingAllRegions
  - PingCloudRegions, 87
- PingCloudRegions, 86
  - CloudRegions, 87
  - ConnectToBestRegion, 87
  - OverrideRegion, 87
  - PingAllRegions, 87
  - ResolveHost, 87
  - SP, 87



- player
  - PhotonNetwork, 76
- PlayerCount
  - GameProperties, 47
- playerCount
  - Room, 89
  - RoomInfo, 92
- playerList
  - PhotonNetwork, 76
- PlayerName
  - ActorProperties, 37
- playerName
  - PhotonNetwork, 76
- PlayerProperties
  - ParameterCode, 54
- Position
  - ParameterCode, 54
  - PBitStream, 56
- PositionAndRotation
  - PhotonView.cs, 102
- precisionForFloatSynchronization
  - PhotonNetwork, 73
- precisionForQuaternionSynchronization
  - PhotonNetwork, 73
- precisionForVectorSynchronization
  - PhotonNetwork, 73
- PrefabCache
  - PhotonNetwork, 73
- prefix
  - PhotonView, 86
- prefixBackup
  - PhotonView, 85
- Properties
  - ParameterCode, 54
- PropertiesChanged
  - EventCode, 43
- propertiesListedInLobby
  - Room, 89
- PropsListedInLobby
  - GameProperties, 47
- Public API
  - All, 34
  - AllBuffered, 34
  - Authenticated, 32
  - Connected, 32
  - ConnectedComingFromGameserver, 32
  - ConnectedToGameserver, 32
  - ConnectedToMaster, 32
  - Connecting, 32
  - ConnectingToGameserver, 32
  - ConnectingToMasterserver, 32
  - DisconnectByServer, 31
  - DisconnectByServerLogic, 31
  - DisconnectByServerUserLimit, 31
  - Disconnected, 32
  - Disconnecting, 32
  - DisconnectingFromGameserver, 32
  - DisconnectingFromMasterserver, 32
  - ErrorsOnly, 32
  - Exception, 31
  - ExceptionOnConnect, 31
  - Full, 32
  - Informational, 32
  - InternalReceiveException, 31
  - InvalidRegion, 31
  - Joined, 32
  - JoinedLobby, 32
  - Joining, 32
  - Leaving, 32
  - MasterClient, 34
  - MaxCcuReached, 31
  - OnConnectedToMaster, 33
  - OnConnectedToPhoton, 32
  - OnConnectionFail, 33
  - OnCreatedRoom, 33
  - OnCustomAuthenticationFailed, 34
  - OnDisconnectedFromPhoton, 33
  - OnFailedToConnectToPhoton, 33
  - OnJoinedLobby, 33
  - OnJoinedRoom, 33
  - OnLeftLobby, 33
  - OnLeftRoom, 32
  - OnMasterClientSwitched, 33
  - OnPhotonCreateRoomFailed, 33
  - OnPhotonCustomRoomPropertiesChanged, 34
  - OnPhotonInstantiate, 33
  - OnPhotonJoinRoomFailed, 33
  - OnPhotonMaxCcuReached, 33
  - OnPhotonPlayerConnected, 33
  - OnPhotonPlayerDisconnected, 33
  - OnPhotonPlayerPropertiesChanged, 34
  - OnPhotonRandomJoinFailed, 33
  - OnPhotonSerializeView, 33
  - OnReceivedRoomListUpdate, 33
  - OnUpdatedFriendList, 34
  - Others, 34
  - OthersBuffered, 34
  - PeerCreated, 31
  - Queued, 32
  - QueuedComingFromGameserver, 32
  - SecurityExceptionOnConnect, 31
  - TimeoutDisconnect, 31
  - Uninitialized, 31
- Public API, 30
  - DisconnectCause, 31
  - PeerState, 31
  - PhotonLogLevel, 32
  - PhotonNetworkingMessage, 32
  - PhotonTargets, 34
- QueueState
  - EventCode, 43
- Queued
  - Public API, 32
- QueuedComingFromGameserver
  - Public API, 32



- RPC
  - PhotonView, [85](#)
- RaiseEvent
  - OperationCode, [50](#)
- RandomMatching
  - LoadbalancingPeer.cs, [100](#)
- ReceiveNext
  - PhotonStream, [83](#)
- ReceiverGroup
  - ParameterCode, [54](#)
- ReliableDeltaCompressed
  - PhotonView.cs, [102](#)
- Remove
  - ParameterCode, [54](#)
- RemoveAllBufferedMessages
  - PhotonNetwork, [70](#)
- RemoveAllInstantiatedObjects
  - PhotonNetwork, [71](#)
- RemoveRPCs
  - PhotonNetwork, [71](#)
- RemoveRPCsInGroup
  - PhotonNetwork, [71](#)
- Removed
  - GameProperties, [47](#)
- removedFromList
  - RoomInfo, [92](#)
- ResolveHost
  - PingCloudRegions, [87](#)
- Room, [87](#)
  - autoCleanUp, [88](#)
  - FriendInfo, [46](#)
  - maxPlayers, [89](#)
  - name, [89](#)
  - open, [89](#)
  - playerCount, [89](#)
  - propertiesListedInLobby, [89](#)
  - SetCustomProperties, [88](#)
  - visible, [89](#)
- room
  - PhotonNetwork, [77](#)
- RoomInfo, [89](#)
  - autoCleanUpField, [91](#)
  - customProperties, [91](#)
  - Equals, [91](#)
  - GetHashCode, [91](#)
  - isLocalClientInside, [91](#)
  - maxPlayers, [92](#)
  - maxPlayersField, [91](#)
  - name, [92](#)
  - nameField, [91](#)
  - open, [92](#)
  - openField, [91](#)
  - playerCount, [92](#)
  - removedFromList, [92](#)
  - ToString, [91](#)
  - visible, [92](#)
  - visibleField, [91](#)
- RoomName
  - ParameterCode, [54](#)
- RpcIndex
  - RpcIndexComponent, [93](#)
- RpcIndexComponent, [92](#)
  - Awake, [93](#)
  - GetShortcut, [93](#)
  - RpcIndex, [93](#)
- RpcList
  - ServerSettings, [95](#)
- SP
  - PingCloudRegions, [87](#)
- Secret
  - AuthenticationValues, [38](#)
  - ParameterCode, [54](#)
- SecurityExceptionOnConnect
  - Public API, [31](#)
- SelfHosted
  - ServerSettings, [94](#)
- SendNext
  - PhotonStream, [83](#)
- SendOutgoingCommands
  - PhotonNetwork, [71](#)
- sendRate
  - PhotonNetwork, [77](#)
- sendRateOnSerialize
  - PhotonNetwork, [77](#)
- sender
  - PhotonMessageInfo, [58](#)
- SerialMatching
  - LoadbalancingPeer.cs, [100](#)
- Serialize
  - PhotonStream, [83](#)
- ServerSettings
  - NotSet, [94](#)
  - OfflineMode, [94](#)
  - PhotonCloud, [94](#)
  - SelfHosted, [94](#)
- ServerAddress
  - PhotonNetwork, [77](#)
  - ServerSettings, [95](#)
- ServerFull
  - ErrorCode, [41](#)
- ServerPort
  - ServerSettings, [95](#)
- ServerSettings, [93](#)
  - AppID, [94](#)
  - CloudServerRegionNames, [95](#)
  - CloudServerRegionPrefixes, [95](#)
  - DefaultAppID, [95](#)
  - DefaultCloudServerUrl, [95](#)
  - DefaultMasterPort, [95](#)
  - DefaultServerAddress, [95](#)
  - DisableAutoOpenWizard, [95](#)
  - FindRegionForServerAddress, [94](#)
  - FindServerAddressForRegion, [94](#)
  - HostType, [95](#)
  - HostingOption, [94](#)
  - RpcList, [95](#)

- ServerAddress, 95
- ServerPort, 95
- ToString, 94
- UseCloud, 94
- UseMyServer, 94
- serverSettingsAssetFile
  - PhotonNetwork, 73
- serverSettingsAssetPath
  - PhotonNetwork, 73
- Set
  - PBitStream, 55
- SetAuthParameters
  - AuthenticationValues, 38
- SetCustomProperties
  - PhotonPlayer, 79
  - Room, 88
- SetLevelPrefix
  - PhotonNetwork, 71
- SetPlayerCustomProperties
  - PhotonNetwork, 72
- SetProperties
  - EventCode, 43
  - OperationCode, 50
- SetReceivingEnabled
  - PhotonNetwork, 72
- SetSendingEnabled
  - PhotonNetwork, 72
- Start
  - PhotonLagSimulationGui, 57
  - PhotonStatsGui, 81
- statsOn
  - PhotonStatsGui, 82
- statsRect
  - PhotonStatsGui, 82
- statsWindowOn
  - PhotonStatsGui, 82
- StripKeysWithNullValues
  - Extensions, 45
- StripToStringKeys
  - Extensions, 45
- subId
  - PhotonView, 85
- SupportClass
  - Extensions.cs, 98
- synchronization
  - PhotonView, 85
- TargetActorNr
  - ParameterCode, 54
- time
  - PhotonNetwork, 77
- TimeoutDisconnect
  - Public API, 31
- timestamp
  - PhotonMessageInfo, 58
- ToArray
  - PhotonStream, 83
- ToBytes
  - PBitStream, 55
- ToString
  - FriendInfo, 46
  - PhotonMessageInfo, 58
  - PhotonPlayer, 79
  - PhotonView, 85
  - RoomInfo, 91
  - ServerSettings, 94
- ToStringFull
  - Extensions, 45
- trafficStatsOn
  - PhotonStatsGui, 82
- TrafficStatsWindow
  - PhotonStatsGui, 81
- UnAllocateViewID
  - PhotonNetwork, 72
- Uninitialized
  - Public API, 31
- Unreliable
  - PhotonView.cs, 102
- unreliableCommandsLimit
  - PhotonNetwork, 77
- Update
  - PhotonStatsGui, 81
- us
  - PingCloudRegions, 87
- UseCloud
  - ServerSettings, 94
- UseMyServer
  - ServerSettings, 94
- UsePrefabCache
  - PhotonNetwork, 73
- UserBlocked
  - ErrorCode, 41
- UserId
  - ParameterCode, 54
- versionPUN
  - PhotonNetwork, 74
- viewID
  - PhotonView, 86
- ViewSynchronization
  - PhotonView.cs, 102
- Visible
  - PhotonLagSimulationGui, 57
- visible
  - Room, 89
  - RoomInfo, 92
- visibleField
  - RoomInfo, 91
- WindowId
  - PhotonLagSimulationGui, 57
  - PhotonStatsGui, 82
- WindowRect
  - PhotonLagSimulationGui, 57