

# Photon Unity Networking

## v1.25

Generated by Doxygen 1.8.4

Fri Mar 21 2014 13:40:22



# 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 Documentation</b>	<b>19</b>
6.1	Optional Gui Elements . . . . .	19
6.1.1	Detailed Description . . . . .	19
6.2	Public API . . . . .	20
6.2.1	Detailed Description . . . . .	21
6.2.2	Enumeration Type Documentation . . . . .	21
6.2.2.1	DisconnectCause . . . . .	21
6.2.2.2	PeerState . . . . .	21
6.2.2.3	PhotonLogLevel . . . . .	22
6.2.2.4	PhotonNetworkingMessage . . . . .	22
6.2.2.5	PhotonTargets . . . . .	24
<b>7</b>	<b>Namespace Documentation</b>	<b>25</b>
7.1	Package Photon . . . . .	25
<b>8</b>	<b>Class Documentation</b>	<b>27</b>
8.1	ActorProperties Class Reference . . . . .	27
8.1.1	Detailed Description . . . . .	27
8.1.2	Member Data Documentation . . . . .	27
8.1.2.1	PlayerName . . . . .	27
8.2	AuthenticationValues Class Reference . . . . .	27
8.2.1	Detailed Description . . . . .	28
8.2.2	Member Function Documentation . . . . .	28
8.2.2.1	SetAuthParameters . . . . .	28

8.2.2.2	<a href="#">SetAuthPostData</a>	28
8.2.2.3	<a href="#">SetAuthPostData</a>	28
8.2.2.4	<a href="#">ToString</a>	28
8.2.3	<a href="#">Member Data Documentation</a>	28
8.2.3.1	<a href="#">AuthParameters</a>	28
8.2.3.2	<a href="#">AuthType</a>	28
8.2.3.3	<a href="#">Secret</a>	29
8.2.4	<a href="#">Property Documentation</a>	29
8.2.4.1	<a href="#">AuthPostData</a>	29
8.3	<a href="#">ErrorCode Class Reference</a>	29
8.3.1	<a href="#">Detailed Description</a>	29
8.3.2	<a href="#">Member Data Documentation</a>	29
8.3.2.1	<a href="#">AlreadyMatched</a>	29
8.3.2.2	<a href="#">CustomAuthenticationFailed</a>	29
8.3.2.3	<a href="#">GameClosed</a>	29
8.3.2.4	<a href="#">GameDoesNotExist</a>	30
8.3.2.5	<a href="#">GameFull</a>	30
8.3.2.6	<a href="#">GameIdAlreadyExists</a>	30
8.3.2.7	<a href="#">InternalServerError</a>	30
8.3.2.8	<a href="#">InvalidAuthentication</a>	30
8.3.2.9	<a href="#">InvalidOperationCode</a>	30
8.3.2.10	<a href="#">InvalidRegion</a>	30
8.3.2.11	<a href="#">MaxCcuReached</a>	30
8.3.2.12	<a href="#">NoRandomMatchFound</a>	31
8.3.2.13	<a href="#">Ok</a>	31
8.3.2.14	<a href="#">OperationNotAllowedInCurrentState</a>	31
8.3.2.15	<a href="#">ServerFull</a>	31
8.3.2.16	<a href="#">UserBlocked</a>	31
8.4	<a href="#">EventCode Class Reference</a>	31
8.4.1	<a href="#">Detailed Description</a>	31
8.4.2	<a href="#">Member Data Documentation</a>	31
8.4.2.1	<a href="#">AppStats</a>	31
8.4.2.2	<a href="#">AzureNodeInfo</a>	32
8.4.2.3	<a href="#">GameList</a>	32
8.4.2.4	<a href="#">GameListUpdate</a>	32
8.4.2.5	<a href="#">Join</a>	32
8.4.2.6	<a href="#">Leave</a>	32
8.4.2.7	<a href="#">Match</a>	32
8.4.2.8	<a href="#">PropertiesChanged</a>	32
8.4.2.9	<a href="#">QueueState</a>	32

8.4.2.10	SetProperties	32
8.5	Extensions Class Reference	32
8.5.1	Detailed Description	33
8.5.2	Member Function Documentation	33
8.5.2.1	AlmostEquals	33
8.5.2.2	AlmostEquals	33
8.5.2.3	AlmostEquals	33
8.5.2.4	AlmostEquals	33
8.5.2.5	Contains	33
8.5.2.6	GetPhotonView	33
8.5.2.7	GetPhotonViewsInChildren	33
8.5.2.8	Merge	33
8.5.2.9	MergeStringKeys	34
8.5.2.10	StripKeysWithNullValues	34
8.5.2.11	StripToStringKeys	34
8.5.2.12	ToStringFull	34
8.6	FriendInfo Class Reference	34
8.6.1	Detailed Description	35
8.6.2	Member Function Documentation	35
8.6.2.1	ToString	35
8.6.3	Property Documentation	35
8.6.3.1	IsInRoom	35
8.6.3.2	IsOnline	35
8.6.3.3	Name	35
8.6.3.4	Room	35
8.7	GameProperties Class Reference	35
8.7.1	Detailed Description	35
8.7.2	Member Data Documentation	35
8.7.2.1	CleanupCacheOnLeave	35
8.7.2.2	IsOpen	35
8.7.2.3	IsVisible	36
8.7.2.4	MaxPlayers	36
8.7.2.5	PlayerCount	36
8.7.2.6	PropsListedInLobby	36
8.7.2.7	Removed	36
8.8	Photon.MonoBehaviour Class Reference	36
8.8.1	Detailed Description	36
8.8.2	Property Documentation	36
8.8.2.1	networkView	36
8.8.2.2	photonView	36

8.9	OperationCode Class Reference	36
8.9.1	Detailed Description	37
8.9.2	Member Data Documentation	37
8.9.2.1	Authenticate	37
8.9.2.2	ChangeGroups	37
8.9.2.3	CreateGame	37
8.9.2.4	FindFriends	37
8.9.2.5	GetProperties	37
8.9.2.6	GetRegions	37
8.9.2.7	JoinGame	37
8.9.2.8	JoinLobby	37
8.9.2.9	JoinRandomGame	38
8.9.2.10	Leave	38
8.9.2.11	LeaveLobby	38
8.9.2.12	RaiseEvent	38
8.9.2.13	SetProperties	38
8.10	ParameterCode Class Reference	38
8.10.1	Detailed Description	39
8.10.2	Member Data Documentation	39
8.10.2.1	ActorList	39
8.10.2.2	ActorNr	39
8.10.2.3	Add	39
8.10.2.4	Address	39
8.10.2.5	ApplicationId	39
8.10.2.6	AppVersion	39
8.10.2.7	Broadcast	39
8.10.2.8	Cache	40
8.10.2.9	CleanupCacheOnLeave	40
8.10.2.10	ClientAuthenticationData	40
8.10.2.11	ClientAuthenticationParams	40
8.10.2.12	ClientAuthenticationType	40
8.10.2.13	Code	40
8.10.2.14	CreateIfNotExists	40
8.10.2.15	CustomEventContent	40
8.10.2.16	Data	40
8.10.2.17	EventForward	40
8.10.2.18	FindFriendsRequestList	40
8.10.2.19	FindFriendsResponseOnlineList	41
8.10.2.20	FindFriendsResponseRoomIdList	41
8.10.2.21	GameCount	41

8.10.2.22 GameList . . . . .	41
8.10.2.23 GameProperties . . . . .	41
8.10.2.24 Group . . . . .	41
8.10.2.25 LobbyName . . . . .	41
8.10.2.26 LobbyType . . . . .	41
8.10.2.27 MasterPeerCount . . . . .	41
8.10.2.28 MatchMakingType . . . . .	41
8.10.2.29 PeerCount . . . . .	41
8.10.2.30 PlayerProperties . . . . .	42
8.10.2.31 Position . . . . .	42
8.10.2.32 Properties . . . . .	42
8.10.2.33 ReceiverGroup . . . . .	42
8.10.2.34 Region . . . . .	42
8.10.2.35 Remove . . . . .	42
8.10.2.36 RoomName . . . . .	42
8.10.2.37 Secret . . . . .	42
8.10.2.38 SuppressRoomEvents . . . . .	42
8.10.2.39 TargetActorNr . . . . .	42
8.10.2.40 UserId . . . . .	42
8.11 PBitStream Class Reference . . . . .	43
8.11.1 Constructor & Destructor Documentation . . . . .	43
8.11.1.1 PBitStream . . . . .	43
8.11.1.2 PBitStream . . . . .	43
8.11.1.3 PBitStream . . . . .	43
8.11.2 Member Function Documentation . . . . .	43
8.11.2.1 Add . . . . .	43
8.11.2.2 BytesForBits . . . . .	43
8.11.2.3 Get . . . . .	43
8.11.2.4 GetNext . . . . .	43
8.11.2.5 Set . . . . .	43
8.11.2.6 ToBytes . . . . .	43
8.11.3 Property Documentation . . . . .	43
8.11.3.1 BitCount . . . . .	43
8.11.3.2 ByteCount . . . . .	43
8.11.3.3 Position . . . . .	43
8.12 PhotonLagSimulationGui Class Reference . . . . .	44
8.12.1 Detailed Description . . . . .	44
8.12.2 Member Function Documentation . . . . .	44
8.12.2.1 OnGUI . . . . .	44
8.12.2.2 Start . . . . .	44

8.12.3	Member Data Documentation . . . . .	44
8.12.3.1	Visible . . . . .	44
8.12.3.2	WindowId . . . . .	44
8.12.3.3	WindowRect . . . . .	44
8.12.4	Property Documentation . . . . .	44
8.12.4.1	Peer . . . . .	44
8.13	PhotonMessageInfo Class Reference . . . . .	45
8.13.1	Detailed Description . . . . .	45
8.13.2	Constructor & Destructor Documentation . . . . .	45
8.13.2.1	PhotonMessageInfo . . . . .	45
8.13.2.2	PhotonMessageInfo . . . . .	45
8.13.3	Member Function Documentation . . . . .	45
8.13.3.1	ToString . . . . .	45
8.13.4	Member Data Documentation . . . . .	45
8.13.4.1	photonView . . . . .	45
8.13.4.2	sender . . . . .	45
8.13.5	Property Documentation . . . . .	45
8.13.5.1	timestamp . . . . .	45
8.14	PhotonNetwork Class Reference . . . . .	45
8.14.1	Detailed Description . . . . .	48
8.14.2	Member Function Documentation . . . . .	48
8.14.2.1	AllocateViewID . . . . .	48
8.14.2.2	CloseConnection . . . . .	48
8.14.2.3	ConnectToBestCloudServer . . . . .	48
8.14.2.4	ConnectToMaster . . . . .	49
8.14.2.5	ConnectUsingSettings . . . . .	49
8.14.2.6	CreateRoom . . . . .	50
8.14.2.7	CreateRoom . . . . .	50
8.14.2.8	CreateRoom . . . . .	51
8.14.2.9	CreateRoom . . . . .	51
8.14.2.10	Destroy . . . . .	52
8.14.2.11	Destroy . . . . .	52
8.14.2.12	DestroyAll . . . . .	53
8.14.2.13	DestroyPlayerObjects . . . . .	53
8.14.2.14	DestroyPlayerObjects . . . . .	53
8.14.2.15	Disconnect . . . . .	54
8.14.2.16	EventCallback . . . . .	54
8.14.2.17	FetchServerTimestamp . . . . .	54
8.14.2.18	FindFriends . . . . .	54
8.14.2.19	GetPing . . . . .	55



8.14.2.20	GetRoomList	55
8.14.2.21	InitializeSecurity	55
8.14.2.22	Instantiate	55
8.14.2.23	Instantiate	55
8.14.2.24	InstantiateSceneObject	56
8.14.2.25	InternalCleanPhotonMonoFromScenelfStuck	56
8.14.2.26	JoinLobby	56
8.14.2.27	JoinLobby	57
8.14.2.28	JoinOrCreateRoom	57
8.14.2.29	JoinRandomRoom	57
8.14.2.30	JoinRandomRoom	58
8.14.2.31	JoinRandomRoom	58
8.14.2.32	JoinRoom	59
8.14.2.33	JoinRoom	59
8.14.2.34	LeaveLobby	59
8.14.2.35	LeaveRoom	59
8.14.2.36	LoadLevel	60
8.14.2.37	LoadLevel	60
8.14.2.38	NetworkStatisticsReset	60
8.14.2.39	NetworkStatisticsToString	60
8.14.2.40	OverrideBestCloudServer	60
8.14.2.41	RaiseEvent	61
8.14.2.42	RefreshCloudServerRating	61
8.14.2.43	RemoveRPCs	61
8.14.2.44	RemoveRPCs	62
8.14.2.45	RemoveRPCsInGroup	62
8.14.2.46	SendOutgoingCommands	62
8.14.2.47	SetLevelPrefix	62
8.14.2.48	SetMasterClient	63
8.14.2.49	SetPlayerCustomProperties	63
8.14.2.50	SetReceivingEnabled	63
8.14.2.51	SetSendingEnabled	64
8.14.2.52	UnAllocateViewID	64
8.14.3	Member Data Documentation	64
8.14.3.1	InstantiateInRoomOnly	64
8.14.3.2	logLevel	64
8.14.3.3	MAX_VIEW_IDS	64
8.14.3.4	OnEventCall	64
8.14.3.5	PhotonServerSettings	64
8.14.3.6	precisionForFloatSynchronization	64

8.14.3.7	<a href="#">precisionForQuaternionSynchronization</a>	65
8.14.3.8	<a href="#">precisionForVectorSynchronization</a>	65
8.14.3.9	<a href="#">PrefabCache</a>	65
8.14.3.10	<a href="#">SendMonoMessageTargets</a>	65
8.14.3.11	<a href="#">serverSettingsAssetFile</a>	65
8.14.3.12	<a href="#">serverSettingsAssetPath</a>	65
8.14.3.13	<a href="#">UseNameServer</a>	65
8.14.3.14	<a href="#">UsePrefabCache</a>	65
8.14.3.15	<a href="#">versionPUN</a>	66
8.14.4	<a href="#">Property Documentation</a>	66
8.14.4.1	<a href="#">AuthValues</a>	66
8.14.4.2	<a href="#">autoCleanUpPlayerObjects</a>	66
8.14.4.3	<a href="#">autoJoinLobby</a>	66
8.14.4.4	<a href="#">automaticallySyncScene</a>	66
8.14.4.5	<a href="#">connected</a>	66
8.14.4.6	<a href="#">connectedAndReady</a>	67
8.14.4.7	<a href="#">connecting</a>	67
8.14.4.8	<a href="#">connectionState</a>	67
8.14.4.9	<a href="#">connectionStateDetailed</a>	67
8.14.4.10	<a href="#">countOfPlayers</a>	67
8.14.4.11	<a href="#">countOfPlayersInRooms</a>	67
8.14.4.12	<a href="#">countOfPlayersOnMaster</a>	67
8.14.4.13	<a href="#">countOfRooms</a>	67
8.14.4.14	<a href="#">CrcCheckEnabled</a>	67
8.14.4.15	<a href="#">Friends</a>	67
8.14.4.16	<a href="#">FriendsListAge</a>	68
8.14.4.17	<a href="#">inRoom</a>	68
8.14.4.18	<a href="#">insideLobby</a>	68
8.14.4.19	<a href="#">isMasterClient</a>	68
8.14.4.20	<a href="#">isMessageQueueRunning</a>	68
8.14.4.21	<a href="#">isNonMasterClientInRoom</a>	68
8.14.4.22	<a href="#">lobby</a>	68
8.14.4.23	<a href="#">masterClient</a>	68
8.14.4.24	<a href="#">maxConnections</a>	68
8.14.4.25	<a href="#">MaxResendsBeforeDisconnect</a>	68
8.14.4.26	<a href="#">NetworkStatisticsEnabled</a>	69
8.14.4.27	<a href="#">offlineMode</a>	69
8.14.4.28	<a href="#">otherPlayers</a>	69
8.14.4.29	<a href="#">PacketLossByCrcCheck</a>	69
8.14.4.30	<a href="#">player</a>	69

8.14.4.31	playerList	69
8.14.4.32	playerName	69
8.14.4.33	ResentReliableCommands	69
8.14.4.34	room	69
8.14.4.35	sendRate	69
8.14.4.36	sendRateOnSerialize	70
8.14.4.37	Server	70
8.14.4.38	ServerAddress	70
8.14.4.39	time	70
8.14.4.40	unreliableCommandsLimit	70
8.15	PhotonPlayer Class Reference	70
8.15.1	Detailed Description	71
8.15.2	Constructor & Destructor Documentation	71
8.15.2.1	PhotonPlayer	71
8.15.2.2	PhotonPlayer	71
8.15.3	Member Function Documentation	71
8.15.3.1	Equals	71
8.15.3.2	Find	71
8.15.3.3	GetHashCode	72
8.15.3.4	SetCustomProperties	72
8.15.3.5	ToString	72
8.15.4	Member Data Documentation	72
8.15.4.1	isLocal	72
8.15.5	Property Documentation	72
8.15.5.1	allProperties	72
8.15.5.2	customProperties	72
8.15.5.3	ID	72
8.15.5.4	isMasterClient	72
8.15.5.5	name	73
8.16	PhotonStatsGui Class Reference	73
8.16.1	Detailed Description	73
8.16.2	Member Function Documentation	73
8.16.2.1	OnGUI	73
8.16.2.2	Start	73
8.16.2.3	TrafficStatsWindow	73
8.16.2.4	Update	73
8.16.3	Member Data Documentation	73
8.16.3.1	buttonsOn	73
8.16.3.2	healthStatsVisible	74
8.16.3.3	statsOn	74

8.16.3.4	statsRect	74
8.16.3.5	statsWindowOn	74
8.16.3.6	trafficStatsOn	74
8.16.3.7	WindowId	74
8.17	PhotonStream Class Reference	74
8.17.1	Detailed Description	75
8.17.2	Constructor & Destructor Documentation	75
8.17.2.1	PhotonStream	75
8.17.3	Member Function Documentation	75
8.17.3.1	ReceiveNext	75
8.17.3.2	SendNext	75
8.17.3.3	Serialize	75
8.17.3.4	Serialize	75
8.17.3.5	Serialize	75
8.17.3.6	Serialize	75
8.17.3.7	Serialize	75
8.17.3.8	Serialize	76
8.17.3.9	Serialize	76
8.17.3.10	Serialize	76
8.17.3.11	Serialize	76
8.17.3.12	Serialize	76
8.17.3.13	ToArray	76
8.17.4	Property Documentation	76
8.17.4.1	Count	76
8.17.4.2	isReading	76
8.17.4.3	isWriting	76
8.18	PhotonView Class Reference	76
8.18.1	Detailed Description	77
8.18.2	Member Function Documentation	77
8.18.2.1	Awake	77
8.18.2.2	ExecuteOnSerialize	77
8.18.2.3	Find	77
8.18.2.4	Get	77
8.18.2.5	Get	77
8.18.2.6	OnApplicationQuit	77
8.18.2.7	OnDestroy	78
8.18.2.8	RPC	78
8.18.2.9	RPC	78
8.18.2.10	ToString	78
8.18.3	Member Data Documentation	78

8.18.3.1	group	78
8.18.3.2	instantiationId	78
8.18.3.3	observed	78
8.18.3.4	onSerializeRigidBodyOption	78
8.18.3.5	onSerializeTransformOption	78
8.18.3.6	ownerId	78
8.18.3.7	prefixBackup	78
8.18.3.8	subId	78
8.18.3.9	synchronization	78
8.18.4	Property Documentation	78
8.18.4.1	instantiationData	78
8.18.4.2	isMine	78
8.18.4.3	isSceneView	78
8.18.4.4	owner	78
8.18.4.5	OwnerActorNr	78
8.18.4.6	prefix	78
8.18.4.7	viewID	79
8.19	PingCloudRegions Class Reference	79
8.19.1	Detailed Description	79
8.19.2	Member Function Documentation	79
8.19.2.1	Awake	79
8.19.2.2	ConnectToBestRegion	79
8.19.2.3	DeleteRegionPref	79
8.19.2.4	OverrideRegion	79
8.19.2.5	RefreshCloudServerRating	79
8.19.2.6	ResolveHost	79
8.19.3	Member Data Documentation	80
8.19.3.1	ClosestRegion	80
8.19.3.2	Instance	80
8.19.4	Property Documentation	80
8.19.4.1	ClosestRegionAvailable	80
8.20	RaiseEventOptions Class Reference	80
8.20.1	Detailed Description	80
8.20.2	Member Data Documentation	80
8.20.2.1	CacheSliceIndex	80
8.20.2.2	CachingOption	80
8.20.2.3	Default	81
8.20.2.4	ForwardToWebhook	81
8.20.2.5	InterestGroup	81
8.20.2.6	Receivers	81

8.20.2.7	SequenceChannel	81
8.20.2.8	TargetActors	81
8.21	Room Class Reference	81
8.21.1	Detailed Description	82
8.21.2	Member Function Documentation	82
8.21.2.1	SetCustomProperties	82
8.21.2.2	SetPropertiesListedInLobby	82
8.21.3	Property Documentation	82
8.21.3.1	autoCleanUp	82
8.21.3.2	maxPlayers	82
8.21.3.3	name	82
8.21.3.4	open	83
8.21.3.5	playerCount	83
8.21.3.6	propertiesListedInLobby	83
8.21.3.7	visible	83
8.22	RoomInfo Class Reference	83
8.22.1	Detailed Description	84
8.22.2	Member Function Documentation	84
8.22.2.1	Equals	84
8.22.2.2	GetHashCode	84
8.22.2.3	ToString	84
8.22.3	Member Data Documentation	84
8.22.3.1	autoCleanUpField	84
8.22.3.2	maxPlayersField	84
8.22.3.3	nameField	84
8.22.3.4	openField	84
8.22.3.5	visibleField	84
8.22.4	Property Documentation	85
8.22.4.1	customProperties	85
8.22.4.2	isLocalClientInside	85
8.22.4.3	maxPlayers	85
8.22.4.4	name	85
8.22.4.5	open	85
8.22.4.6	playerCount	85
8.22.4.7	removedFromList	85
8.22.4.8	visible	85
8.23	RoomOptions Class Reference	85
8.23.1	Detailed Description	86
8.23.2	Member Data Documentation	86
8.23.2.1	cleanupCacheOnLeave	86

8.23.2.2	<a href="#">customRoomProperties</a>	86
8.23.2.3	<a href="#">customRoomPropertiesForLobby</a>	86
8.23.2.4	<a href="#">isOpen</a>	86
8.23.2.5	<a href="#">isVisible</a>	86
8.23.2.6	<a href="#">maxPlayers</a>	86
8.24	<a href="#">ServerSettings Class Reference</a>	86
8.24.1	<a href="#">Detailed Description</a>	87
8.24.2	<a href="#">Member Enumeration Documentation</a>	87
8.24.2.1	<a href="#">HostingOption</a>	87
8.24.3	<a href="#">Member Function Documentation</a>	87
8.24.3.1	<a href="#">ExtractRegionFromAddress</a>	87
8.24.3.2	<a href="#">FindRegionForServerAddress</a>	87
8.24.3.3	<a href="#">FindServerAddressForRegion</a>	87
8.24.3.4	<a href="#">FindServerAddressForRegion</a>	87
8.24.3.5	<a href="#">ToString</a>	87
8.24.3.6	<a href="#">TryParseCloudServerRegion</a>	88
8.24.3.7	<a href="#">UseCloud</a>	88
8.24.3.8	<a href="#">UseMyServer</a>	88
8.24.4	<a href="#">Member Data Documentation</a>	88
8.24.4.1	<a href="#">AppID</a>	88
8.24.4.2	<a href="#">CloudServerRegionPrefixes</a>	88
8.24.4.3	<a href="#">DefaultAppID</a>	88
8.24.4.4	<a href="#">DefaultCloudServerUrl</a>	88
8.24.4.5	<a href="#">DefaultMasterPort</a>	88
8.24.4.6	<a href="#">DefaultNameServerPort</a>	88
8.24.4.7	<a href="#">DefaultServerAddress</a>	88
8.24.4.8	<a href="#">DisableAutoOpenWizard</a>	88
8.24.4.9	<a href="#">HostType</a>	88
8.24.4.10	<a href="#">PingCloudServersOnAwake</a>	88
8.24.4.11	<a href="#">RpcList</a>	88
8.24.4.12	<a href="#">ServerAddress</a>	88
8.24.4.13	<a href="#">ServerPort</a>	88
8.24.5	<a href="#">Property Documentation</a>	88
8.24.5.1	<a href="#">Region</a>	88
8.25	<a href="#">TypedLobby Class Reference</a>	88
8.25.1	<a href="#">Detailed Description</a>	89
8.25.2	<a href="#">Constructor &amp; Destructor Documentation</a>	89
8.25.2.1	<a href="#">TypedLobby</a>	89
8.25.2.2	<a href="#">TypedLobby</a>	89
8.25.3	<a href="#">Member Function Documentation</a>	89

---

8.25.3.1 ToString . . . . .	89
8.25.4 Member Data Documentation . . . . .	89
8.25.4.1 Default . . . . .	89
8.25.4.2 Name . . . . .	89
8.25.4.3 Type . . . . .	89
8.25.5 Property Documentation . . . . .	89
8.25.5.1 IsDefault . . . . .	89
 Index	 90



# 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: [exitgames.com/en/Account/SignUp](https://exitgames.com/en/Account/SignUp)
- Create an App and 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 Server SDK can be downloaded on: [www.exitgames.com/en/OnPremise/Download](http://www.exitgames.com/en/OnPremise/Download)

Starting the Server: [doc.exitgames.com/en/onpremise/current/getting-started/photon-server-in-5m](http://doc.exitgames.com/en/onpremise/current/getting-started/photon-server-in-5m)

## 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: PhotonServerSettings.

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 \Plugins 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 Hashtable with string keys. Preferably short keys.

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

```
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”. All you have to do is implement the fitting method in any MonoBehaviour and it gets called when the event happens.

The names of all callback methods are defined in the **PhotonNetworkingMessage enum**. The description of each item also describes the parameters and the situation when it’s called.

The complete list of callbacks is also in the reference part of this documentation.

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

## Sending messages in rooms

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

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

### Using Groups in PUN

Groups are not synchronized when they are changed on any [PhotonView](#). It’s up to the developer to keep photonviews in the same groups on all clients, if that’s needed. Using different group numbers for the same photonview on several clients will cause some inconsistent behaviour. Some network messages are checked for their receiver group at the receiver side only, namely: RPCS that are 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 [Photon.MonoBehaviour](#), it has a `photonView` field. Any regular *MonoBehaviour* or *GameObject* can use: `PhotonView.Get(this)` to get access to its [PhotonView](#) component and then call RPCs on it.

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

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

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

```
[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, during scene loading. This code shows how you can do it:

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

Alternatively you can use [PhotonNetwork.LoadLevel](#). It temporarily disables the message queue as well.

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

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

## Various topics

### Differences to Unity Networking

#### 1. Host model

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

#### 2. Connectivity

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

#### 3. Performance

- [Photon](#) beats Unity networking performance wise. We do not have the figures to prove this yet but the library has been optimized for years now. Furthermore, since the Unity servers are player hosted latency/ping is usually worse; you rely on the connection of the player acting as server. These connections are never any better 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 (value range: 32,768 to 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 UnityScript / Javascript?

To be able to use the [Photon](#) classes in Unity Script, you'll need to move the Plugins folder in "Photon Unity Networking/Plugins" folder to the root of your project to make sure it's compiled before your code.

## Converting your Unity networking project to Photon

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

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

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

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

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

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

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

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



## Chapter 3

# 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 Documentation

### 6.1 Optional Gui Elements

#### Classes

- class [PhotonLagSimulationGui](#)
- class [PhotonStatsGui](#)

#### 6.1.1 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.

## 6.2 Public API

### Classes

- class [PhotonMessageInfo](#)
- class [PhotonStream](#)
- class [PhotonNetwork](#)
- class [PhotonPlayer](#)
- class [PhotonView](#)
- class [Room](#)
- class [RoomInfo](#)

### Enumerations

- enum [PeerState](#) {  
[PeerState.Uninitialized](#), [PeerState.PeerCreated](#), [PeerState.Queued](#), [PeerState.Authenticated](#),  
[PeerState.JoinedLobby](#), [PeerState.DisconnectingFromMasterserver](#), [PeerState.ConnectingToGameserver](#),  
[PeerState.ConnectedToGameserver](#),  
[PeerState.Joining](#), [PeerState.Joined](#), [PeerState.Leaving](#), [PeerState.DisconnectingFromGameserver](#),  
[PeerState.ConnectingToMasterserver](#), [PeerState.QueuedComingFromGameserver](#), [PeerState.Disconnecting](#),  
[PeerState.Disconnected](#),  
[PeerState.ConnectedToMaster](#), [PeerState.ConnectingToNameServer](#), [PeerState.ConnectedToNameServer](#),  
[PeerState.DisconnectingFromNameServer](#),  
[PeerState.Authenticating](#) }
- enum [PhotonNetworkingMessage](#) {  
[PhotonNetworkingMessage.OnConnectedToPhoton](#), [PhotonNetworkingMessage.OnLeftRoom](#), [PhotonNetworkingMessage.OnMasterClientSwitched](#), [PhotonNetworkingMessage.OnPhotonCreateRoomFailed](#),  
[PhotonNetworkingMessage.OnPhotonJoinRoomFailed](#), [PhotonNetworkingMessage.OnCreatedRoom](#),  
[PhotonNetworkingMessage.OnJoinedLobby](#), [PhotonNetworkingMessage.OnLeftLobby](#),  
[PhotonNetworkingMessage.OnDisconnectedFromPhoton](#), [PhotonNetworkingMessage.OnConnectionFail](#),  
[PhotonNetworkingMessage.OnFailedToConnectToPhoton](#), [PhotonNetworkingMessage.OnReceivedRoomListUpdate](#),  
[PhotonNetworkingMessage.OnJoinedRoom](#), [PhotonNetworkingMessage.OnPhotonPlayerConnected](#),  
[PhotonNetworkingMessage.OnPhotonPlayerDisconnected](#), [PhotonNetworkingMessage.OnPhotonRandomJoinFailed](#),  
[PhotonNetworkingMessage.OnConnectedToMaster](#), [PhotonNetworkingMessage.OnPhotonSerializeView](#),  
[PhotonNetworkingMessage.OnPhotonInstantiate](#), [PhotonNetworkingMessage.OnPhotonMaxCcuReached](#),  
[PhotonNetworkingMessage.OnPhotonCustomRoomPropertiesChanged](#), [PhotonNetworkingMessage.OnPhotonPlayerPropertiesChanged](#), [PhotonNetworkingMessage.OnUpdatedFriendList](#), [PhotonNetworkingMessage.OnCustomAuthenticationFailed](#) }
- enum [DisconnectCause](#) {  
[DisconnectCause.ExceptionOnConnect](#) = [StatusCode.ExceptionOnConnect](#), [DisconnectCause.SecurityExceptionOnConnect](#) = [StatusCode.SecurityExceptionOnConnect](#), [DisconnectCause.TimeoutDisconnect](#) = [StatusCode.TimeoutDisconnect](#), [DisconnectCause.DisconnectByClientTimeout](#) = [StatusCode.TimeoutDisconnect](#),  
[DisconnectCause.InternalReceiveException](#) = [StatusCode.ExceptionOnReceive](#), [DisconnectCause.DisconnectByServer](#) = [StatusCode.DisconnectByServer](#), [DisconnectCause.DisconnectByServerTimeout](#) = [StatusCode.DisconnectByServer](#), [DisconnectCause.DisconnectByServerLogic](#) = [StatusCode.DisconnectByServerLogic](#),  
[DisconnectCause.DisconnectByServerUserLimit](#) = [StatusCode.DisconnectByServerUserLimit](#), [DisconnectCause.Exception](#) = [StatusCode.Exception](#), [DisconnectCause.InvalidRegion](#) = [ErrorCode.InvalidRegion](#),  
[DisconnectCause.MaxCcuReached](#) = [ErrorCode.MaxCcuReached](#),  
[DisconnectCause.InvalidAuthentication](#) = [ErrorCode.InvalidAuthentication](#) }
- enum [PhotonTargets](#) {  
[PhotonTargets.All](#), [PhotonTargets.Others](#), [PhotonTargets.MasterClient](#), [PhotonTargets.AllBuffered](#),  
[PhotonTargets.OthersBuffered](#), [PhotonTargets.AllViaServer](#), [PhotonTargets.AllBufferedViaServer](#) }
- enum [PhotonLogLevel](#) { [PhotonLogLevel.ErrorsOnly](#), [PhotonLogLevel.Informational](#), [PhotonLogLevel.Full](#) }

### 6.2.1 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.

### 6.2.2 Enumeration Type Documentation

#### 6.2.2.1 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).

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

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

**DisconnectByServer** Server actively disconnected this client.

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

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

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

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

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

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

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

#### 6.2.2.2 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.
- 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 Join-Random to get into a room to play.
- DisconnectingFromMasterserver** Disconnecting.(will-change)
- ConnectingToGameserver** Connecting to game server (to join/create a room and play).(will-change)
- ConnectedToGameserver** Similar to Connected state but on game server. Still in process to join/create room.(will-change)
- Joining** In process to join/create room (on game server).(will-change)
- Joined** Final state of a room join/create sequence. This client can now exchange events / call RPCs with other clients.
- Leaving** Leaving a room.(will-change)
- DisconnectingFromGameserver** Workflow is leaving the game server and will re-connect to the master server.(will-change)
- ConnectingToMasterserver** Workflow is connected to master server and will establish encryption and authenticate your app.(will-change)
- QueuedComingFromGameserver** Same Queued but coming from game server.(will-change)
- Disconnecting** PUN is disconnecting. This leads to Disconnected.(will-change)
- Disconnected** No connection is setup, ready to connect. Similar to PeerCreated.
- ConnectedToMaster** Final state for connecting to master without joining the lobby (AutoJoinLobby is false).
- ConnectingToNameServer** Client connects to the NameServer. This process includes low level connecting and setting up encryption. When done, state becomes ConnectedToNameServer.
- ConnectedToNameServer** Client is connected to the NameServer and established encryption already. You should call OpGetRegions or ConnectToRegionMaster.
- DisconnectingFromNameServer** When disconnecting from a [Photon](#) NameServer.(will-change)
- Authenticating** When connecting to a [Photon](#) Server, this state is intermediate before you can call any operations.(will-change)

## 6.2.2.3 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**

## 6.2.2.4 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 master server 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\(\)](#){ ... } If you set PhotonNetwork.AutoJoinLobby to true, OnJoinedLobby will be called instead of this.

**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 a room's custom properties changed. The propertiesThatChanged contains all that was set via [Room.SetCustomProperties](#). Since v1.25 this method has one parameter Hashtable propertiesThatChanged. Changing properties must be done by [Room.SetCustomProperties](#), which causes this callback locally, too. Example: void OnPhotonCustomRoomPropertiesChanged(Hashtable propertiesThatChanged){ ... }

**OnPhotonPlayerPropertiesChanged** Called when custom player-properties are changed. Player and the changed properties are passed as object[]. Since v1.25 this method has one parameter object[], which contains two entries: [0] is the affected [PhotonPlayer](#). [1] is the Hashtable of properties that changed. We are using a object[] due to limitations of Unity's GameObject.SendMessage (which has only one optional parameter)

Changing properties must be done by [PhotonPlayer.SetCustomProperties](#), which causes this callback locally, too. Example: void OnPhotonPlayerPropertiesChanged(object[] playerAndUpdatedProps) { [PhotonPlayer](#) player = playerAndUpdatedProps[0] as [PhotonPlayer](#); Hashtable props = playerAndUpdatedProps[1] as Hashtable; //... }

**OnUpdatedFriendList** Called when the server sent the response to a FindFriends request and updated [PhotonNetwork.Friends](#). 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).

#### 6.2.2.5 enum PhotonTargets

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

##### Enumerator

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

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

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

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

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

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

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



## Chapter 7

# Namespace Documentation

### 7.1 Package Photon

#### Classes

- class [MonoBehaviour](#)



## Chapter 8

# Class Documentation

### 8.1 ActorProperties Class Reference

#### Public Attributes

- const byte [PlayerName](#) = 255

#### 8.1.1 Detailed Description

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

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

#### 8.1.2 Member Data Documentation

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

(255) Name of a player/actor.

### 8.2 AuthenticationValues Class Reference

#### Public Member Functions

- virtual void [SetAuthPostData](#) (string stringData)
- virtual void [SetAuthPostData](#) (byte[] byteData)
- virtual void [SetAuthParameters](#) (string user, string token)
- override string [ToString](#) ()

#### Public Attributes

- CustomAuthenticationType [AuthType](#) = CustomAuthenticationType.Custom
- string [AuthParameters](#)
- string [Secret](#)

#### Properties

- object [AuthPostData](#) [get, set]

### 8.2.1 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>

### 8.2.2 Member Function Documentation

#### 8.2.2.1 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 <a href="#">Photon</a> .

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

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

Parameters

<i>byteData</i>	Binary token / auth-data to pass on. Empty string will set AuthPostData to null.
-----------------	--

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

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

Parameters

<i>byteData</i>	Binary token / auth-data to pass on.
-----------------	--------------------------------------

#### 8.2.2.4 override string AuthenticationValues.ToString ( )

### 8.2.3 Member Data Documentation

#### 8.2.3.1 string AuthenticationValues.AuthParameters

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

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

#### 8.2.3.2 CustomAuthenticationType AuthenticationValues.AuthType = CustomAuthenticationType.Custom

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

### 8.2.3.3 string AuthenticationValues.Secret

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

## 8.2.4 Property Documentation

### 8.2.4.1 object AuthenticationValues.AuthPostData [get], [set]

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

## 8.3 ErrorCode Class Reference

### Public Attributes

- const int [Ok](#) = 0
- const int [OperationNotAllowedInCurrentState](#) = -3
- const int [InvalidOperationCode](#) = -2
- const int [InternalServerError](#) = -1
- const int [InvalidAuthentication](#) = 0x7FFF
- const int [GameIdAlreadyExists](#) = 0x7FFF - 1
- const int [GameFull](#) = 0x7FFF - 2
- const int [GameClosed](#) = 0x7FFF - 3
- const int [AlreadyMatched](#) = 0x7FFF - 4
- const int [ServerFull](#) = 0x7FFF - 5
- const int [UserBlocked](#) = 0x7FFF - 6
- const int [NoRandomMatchFound](#) = 0x7FFF - 7
- const int [GameDoesNotExist](#) = 0x7FFF - 9
- const int [MaxCcuReached](#) = 0x7FFF - 10
- const int [InvalidRegion](#) = 0x7FFF - 11
- const int [CustomAuthenticationFailed](#) = 0x7FFF - 12

### 8.3.1 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.

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

### 8.3.2 Member Data Documentation

#### 8.3.2.1 const int ErrorCode.AlreadyMatched = 0x7FFF - 4

#### 8.3.2.2 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.

#### 8.3.2.3 const int ErrorCode.GameClosed = 0x7FFF - 3

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

#### 8.3.2.4 `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.

#### 8.3.2.5 `const int ErrorCode.GameFull = 0x7FFF - 2`

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

#### 8.3.2.6 `const int ErrorCode.GameIdAlreadyExists = 0x7FFF - 1`

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

#### 8.3.2.7 `const int ErrorCode.InternalServerError = -1`

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

#### 8.3.2.8 `const int ErrorCode.InvalidAuthentication = 0x7FFF`

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

#### 8.3.2.9 `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.

#### 8.3.2.10 `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.

#### 8.3.2.11 `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 an play again.

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

**8.3.2.12** `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.

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

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

**8.3.2.14** `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`)

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

(32762) Not in use currently.

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

(32761) Not in use currently.

## 8.4 ErrorCode Class Reference

### Public Attributes

- `const byte GameList = 230`
- `const byte GameListUpdate = 229`
- `const byte QueueState = 228`
- `const byte Match = 227`
- `const byte AppStats = 226`
- `const byte AzureNodeInfo = 210`
- `const byte Join = (byte)LiteEventCode.Join`
- `const byte Leave = (byte)LiteEventCode.Leave`
- `const byte PropertiesChanged = (byte)LiteEventCode.PropertiesChanged`
- `const byte SetProperties = (byte)LiteEventCode.PropertiesChanged`

### 8.4.1 Detailed Description

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

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

### 8.4.2 Member Data Documentation

**8.4.2.1** `const byte EventCode.AppStats = 226`

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

#### 8.4.2.2 `const byte EventCode.AzureNodeInfo = 210`

(210) Internally used in case of hosting by Azure

#### 8.4.2.3 `const byte EventCode.GameList = 230`

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

#### 8.4.2.4 `const byte EventCode.GameListUpdate = 229`

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

#### 8.4.2.5 `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).

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

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

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

(227) Currently not used. Event for matchmaking

#### 8.4.2.8 `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.

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

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

#### 8.4.2.10 `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.

## 8.5 Extensions Class Reference

### 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)
- static bool [AlmostEquals](#) (this Vector2 target, Vector2 second, float sqrMagnitudePrecision)
- static bool [AlmostEquals](#) (this Quaternion target, Quaternion second, float maxAngle)
- static bool [AlmostEquals](#) (this float target, float second, float floatDiff)
- static void [Merge](#) (this IDictionary target, IDictionary addHash)



- static void [MergeStringKeys](#) (this IDictionary target, IDictionary addHash)
- static string [ToStringFull](#) (this IDictionary origin)
- static Hashtable [StripToStringKeys](#) (this IDictionary original)
- static void [StripKeysWithNullValues](#) (this IDictionary original)
- static bool [Contains](#) (this int[] target, int nr)

### 8.5.1 Detailed Description

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

### 8.5.2 Member Function Documentation

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

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

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

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

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

compares the angle between target and second to given float value

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

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

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

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

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

Parameters

<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.

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

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

**8.5.2.8** 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.

8.5.2.9 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.

8.5.2.10 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.
-----------------	--

8.5.2.11 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 only string-typed keys of the original.

8.5.2.12 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.

## 8.6 FriendInfo Class Reference

### Public Member Functions

- override string [ToString](#) ()

## Properties

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

### 8.6.1 Detailed Description

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

### 8.6.2 Member Function Documentation

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

### 8.6.3 Property Documentation

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

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

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

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

## 8.7 GameProperties Class Reference

### Public Attributes

- const byte [MaxPlayers](#) = 255
- const byte [IsVisible](#) = 254
- const byte [IsOpen](#) = 253
- const byte [PlayerCount](#) = 252
- const byte [Removed](#) = 251
- const byte [PropsListedInLobby](#) = 250
- const byte [CleanupCacheOnLeave](#) = 249

### 8.7.1 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.

### 8.7.2 Member Data Documentation

8.7.2.1 const byte [GameProperties.CleanupCacheOnLeave](#) = 249

Equivalent of Operation Join parameter CleanupCacheOnLeave.

8.7.2.2 const byte [GameProperties.IsOpen](#) = 253

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

#### 8.7.2.3 `const byte GameProperties.IsVisible = 254`

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

#### 8.7.2.4 `const byte GameProperties.MaxPlayers = 255`

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

#### 8.7.2.5 `const byte GameProperties.PlayerCount = 252`

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

#### 8.7.2.6 `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.

#### 8.7.2.7 `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)

## 8.8 Photon.MonoBehaviour Class Reference

Inherits `MonoBehaviour`.

Inherited by `PhotonHandler`, and [PhotonView](#).

### Properties

- [PhotonView](#) `photonView` [get]
- new [PhotonView](#) `networkView` [get]

#### 8.8.1 Detailed Description

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

#### 8.8.2 Property Documentation

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

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

## 8.9 OperationCode Class Reference

### Public Attributes

- const byte [Authenticate](#) = 230
- const byte [JoinLobby](#) = 229
- const byte [LeaveLobby](#) = 228
- const byte [CreateGame](#) = 227

- const byte `JoinGame` = 226
- const byte `JoinRandomGame` = 225
- const byte `Leave` = (byte)LiteOpCode.Leave
- const byte `RaiseEvent` = (byte)LiteOpCode.RaiseEvent
- const byte `SetProperties` = (byte)LiteOpCode.SetProperties
- const byte `GetProperties` = (byte)LiteOpCode.GetProperties
- const byte `ChangeGroups` = (byte)LiteOpCode.ChangeGroups
- const byte `FindFriends` = 222
- const byte `GetRegions` = 220

### 8.9.1 Detailed Description

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

### 8.9.2 Member Data Documentation

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

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

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

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

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

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

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

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

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

(251) Get Properties

#### 8.9.2.6 const byte OperationCode.GetRegions = 220

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

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

(226) Join game (by name)

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

(229) Joins lobby (on master)

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

(225) Joins random game (on master)

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

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

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

(228) Leaves lobby (on master)

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

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

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

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

## 8.10 ParameterCode Class Reference

### Public Attributes

- `const byte SuppressRoomEvents = 237`
- `const byte EventForward = 234`
- `const byte Address = 230`
- `const byte PeerCount = 229`
- `const byte GameCount = 228`
- `const byte MasterPeerCount = 227`
- `const byte UserId = 225`
- `const byte ApplicationId = 224`
- `const byte Position = 223`
- `const byte MatchMakingType = 223`
- `const byte GameList = 222`
- `const byte Secret = 221`
- `const byte AppVersion = 220`
- `const byte RoomName = (byte)LiteOpKey.GameId`
- `const byte Broadcast = (byte)LiteOpKey.Broadcast`
- `const byte ActorList = (byte)LiteOpKey.ActorList`
- `const byte ActorNr = (byte)LiteOpKey.ActorNr`
- `const byte PlayerProperties = (byte)LiteOpKey.ActorProperties`
- `const byte CustomEventContent = (byte)LiteOpKey.Data`
- `const byte Data = (byte)LiteOpKey.Data`
- `const byte Code = (byte)LiteOpKey.Code`
- `const byte GameProperties = (byte)LiteOpKey.GameProperties`
- `const byte Properties = (byte)LiteOpKey.Properties`
- `const byte TargetActorNr = (byte)LiteOpKey.TargetActorNr`
- `const byte ReceiverGroup = (byte)LiteOpKey.ReceiverGroup`
- `const byte Cache = (byte)LiteOpKey.Cache`
- `const byte CleanupCacheOnLeave = (byte)241`

- const byte [Group](#) = LiteOpKey.Group
- const byte [Remove](#) = LiteOpKey.Remove
- const byte [Add](#) = LiteOpKey.Add
- const byte [ClientAuthenticationType](#) = 217
- const byte [ClientAuthenticationParams](#) = 216
- const byte [CreateIfNotExists](#) = 215
- const byte [ClientAuthenticationData](#) = 214
- const byte [LobbyName](#) = (byte)213
- const byte [LobbyType](#) = (byte)212
- const byte [FindFriendsRequestList](#) = (byte)1
- const byte [FindFriendsResponseOnlineList](#) = (byte)1
- const byte [FindFriendsResponseRoomIdList](#) = (byte)2
- const byte [Region](#) = (byte)210

### 8.10.1 Detailed Description

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

### 8.10.2 Member Data Documentation

#### 8.10.2.1 const byte ParameterCode.ActorList = (byte)LiteOpKey.ActorList

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

#### 8.10.2.2 const byte ParameterCode.ActorNr = (byte)LiteOpKey.ActorNr

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

#### 8.10.2.3 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.

#### 8.10.2.4 const byte ParameterCode.Address = 230

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

#### 8.10.2.5 const byte ParameterCode.ApplicationId = 224

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

#### 8.10.2.6 const byte ParameterCode.AppVersion = 220

(220) Version of your application

#### 8.10.2.7 const byte ParameterCode.Broadcast = (byte)LiteOpKey.Broadcast

(250) Code for broadcast parameter of OpSetProperties method.

**8.10.2.8** `const byte ParameterCode.Cache = (byte)LiteOpKey.Cache`

(247) Code for caching events while raising them.

**8.10.2.9** `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).

**8.10.2.10** `const byte ParameterCode.ClientAuthenticationData = 214`

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

**8.10.2.11** `const byte ParameterCode.ClientAuthenticationParams = 216`

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

**8.10.2.12** `const byte ParameterCode.ClientAuthenticationType = 217`

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

**8.10.2.13** `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.

**8.10.2.14** `const byte ParameterCode.CreateIfNotExists = 215`

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

**8.10.2.15** `const byte ParameterCode.CustomEventContent = (byte)LiteOpKey.Data`

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

**8.10.2.16** `const byte ParameterCode.Data = (byte)LiteOpKey.Data`

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

**8.10.2.17** `const byte ParameterCode.EventForward = 234`

(234) Optional parameter of OpRaiseEvent to forward the event to some web-service.

**8.10.2.18** `const byte ParameterCode.FindFriendsRequestList = (byte)1`

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



8.10.2.19 `const byte ParameterCode.FindFriendsResponseOnlineList = (byte)1`

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

8.10.2.20 `const byte ParameterCode.FindFriendsResponseRoomIdList = (byte)2`

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

8.10.2.21 `const byte ParameterCode.GameCount = 228`

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

8.10.2.22 `const byte ParameterCode.GameList = 222`

(222) List of RoomInfos about open / listed rooms

8.10.2.23 `const byte ParameterCode.GameProperties = (byte)LiteOpKey.GameProperties`

(248) Code for property set (Hashtable).

8.10.2.24 `const byte ParameterCode.Group = LiteOpKey.Group`

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

8.10.2.25 `const byte ParameterCode.LobbyName = (byte)213`

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

8.10.2.26 `const byte ParameterCode.LobbyType = (byte)212`

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

8.10.2.27 `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)

8.10.2.28 `const byte ParameterCode.MatchMakingType = 223`

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

8.10.2.29 `const byte ParameterCode.PeerCount = 229`

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

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

(249) Code for property set (Hashtable).

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

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

**8.10.2.32** `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 [Actor-Properties](#) or [GameProperties](#) are used (or both), check those keys.

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

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

**8.10.2.34** `const byte ParameterCode.Region = (byte)210`

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

**8.10.2.35** `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.

**8.10.2.36** `const byte ParameterCode.RoomName = (byte)LiteOpKey.GameId`

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

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

(221) Internally used to establish encryption

**8.10.2.38** `const byte ParameterCode.SuppressRoomEvents = 237`

(237) Optional parameter to suppress events Join and Leave for a room (which might be used as lobby/chat room then).

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

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

**8.10.2.40** `const byte ParameterCode.UserId = 225`

(225) User's ID

## 8.11 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]

### 8.11.1 Constructor & Destructor Documentation

8.11.1.1 [PBitStream.PBitStream](#) ( )

8.11.1.2 [PBitStream.PBitStream](#) ( int *bitCount* )

8.11.1.3 [PBitStream.PBitStream](#) ( IEnumerable< byte > *bytes*, int *bitCount* )

### 8.11.2 Member Function Documentation

8.11.2.1 void [PBitStream.Add](#) ( bool *val* )

8.11.2.2 static int [PBitStream.BytesForBits](#) ( int *bitCount* ) [static]

8.11.2.3 bool [PBitStream.Get](#) ( int *bitIndex* )

8.11.2.4 bool [PBitStream.GetNext](#) ( )

8.11.2.5 void [PBitStream.Set](#) ( int *bitIndex*, bool *value* )

8.11.2.6 byte [] [PBitStream.ToBytes](#) ( )

### 8.11.3 Property Documentation

8.11.3.1 int [PBitStream.BitCount](#) [get], [set]

8.11.3.2 int [PBitStream.ByteCount](#) [get]

8.11.3.3 int [PBitStream.Position](#) [get], [set]

## 8.12 PhotonLagSimulationGui Class Reference

Inherits MonoBehaviour.

### Public Member Functions

- void [Start](#) ()
- void [OnGUI](#) ()

### Public Attributes

- Rect [WindowRect](#) = new Rect(0, 100, 120, 100)
- int [WindowId](#) = 101
- bool [Visible](#) = true

### Properties

- PhotonPeer [Peer](#) [get, set]

#### 8.12.1 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.

#### 8.12.2 Member Function Documentation

8.12.2.1 void PhotonLagSimulationGui.OnGUI ( )

8.12.2.2 void PhotonLagSimulationGui.Start ( )

#### 8.12.3 Member Data Documentation

8.12.3.1 bool PhotonLagSimulationGui.Visible = true

Shows or hides GUI (does not affect settings).

8.12.3.2 int PhotonLagSimulationGui.WindowId = 101

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

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

Positioning rect for window.

#### 8.12.4 Property Documentation

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

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

## 8.13 PhotonMessageInfo Class Reference

### Public Member Functions

- [PhotonMessageInfo](#) ()
- [PhotonMessageInfo](#) ([PhotonPlayer](#) player, int [timestamp](#), [PhotonView](#) view)
- override string [ToString](#) ()

### Public Attributes

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

### Properties

- double [timestamp](#) [get]

#### 8.13.1 Detailed Description

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

#### 8.13.2 Constructor & Destructor Documentation

##### 8.13.2.1 PhotonMessageInfo.PhotonMessageInfo ( )

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

##### 8.13.2.2 PhotonMessageInfo.PhotonMessageInfo ( [PhotonPlayer](#) player, int [timestamp](#), [PhotonView](#) view )

#### 8.13.3 Member Function Documentation

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

#### 8.13.4 Member Data Documentation

##### 8.13.4.1 PhotonView PhotonMessageInfo.photonView

##### 8.13.4.2 PhotonPlayer PhotonMessageInfo.sender

#### 8.13.5 Property Documentation

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

## 8.14 PhotonNetwork Class Reference

### Public Member Functions

- delegate void [EventCallback](#) (byte eventCode, object content, int senderId)

## Static Public Member Functions

- static bool [SetMasterClient](#) ([PhotonPlayer](#) masterClientPlayer)
- static void [NetworkStatisticsReset](#) ()
- static string [NetworkStatisticsToString](#) ()
- static void [InternalCleanPhotonMonoFromScenelfStuck](#) ()
- static bool [ConnectUsingSettings](#) (string gameVersion)
- static bool [ConnectToMaster](#) (string masterServerAddress, int port, string applID, string gameVersion)
- static bool [ConnectToBestCloudServer](#) (string gameVersion)
- static void [OverrideBestCloudServer](#) (CloudServerRegion region)
- static void [RefreshCloudServerRating](#) ()
- static void [Disconnect](#) ()
- static void [InitializeSecurity](#) ()
- static bool [FindFriends](#) (string[] friendsToFind)
- static bool [CreateRoom](#) (string roomName, bool isVisible, bool isOpen, int maxPlayers)
- static bool [CreateRoom](#) (string roomName, bool isVisible, bool isOpen, int maxPlayers, Hashtable customRoomProperties, string[] propsToListInLobby)
- static bool [CreateRoom](#) (string roomName)
- static bool [CreateRoom](#) (string roomName, [RoomOptions](#) roomOptions, [TypedLobby](#) typedLobby)
- static bool [JoinRoom](#) (string roomName, bool createlfNotExists)
- static bool [JoinRoom](#) (string roomName)
- static bool [JoinOrCreateRoom](#) (string roomName, [RoomOptions](#) roomOptions, [TypedLobby](#) typedLobby)
- static bool [JoinRandomRoom](#) ()
- static bool [JoinRandomRoom](#) (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)
- static bool [JoinRandomRoom](#) (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType, [TypedLobby](#) typedLobby, string sqlLobbyFilter)
- static bool [JoinLobby](#) ()
- static bool [JoinLobby](#) ([TypedLobby](#) typedLobby)
- static bool [LeaveLobby](#) ()
- static bool [LeaveRoom](#) ()
- static [RoomInfo](#)[] [GetRoomList](#) ()
- static void [SetPlayerCustomProperties](#) (Hashtable customProperties)
- static bool [RaiseEvent](#) (byte eventCode, object eventContent, bool sendReliable, [RaiseEventOptions](#) options)
- static int [AllocateViewID](#) ()
- static void [UnAllocateViewID](#) (int viewID)
- static GameObject [Instantiate](#) (string prefabName, Vector3 position, Quaternion rotation, int group)
- static GameObject [Instantiate](#) (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)
- static GameObject [InstantiateSceneObject](#) (string prefabName, Vector3 position, Quaternion rotation, int group, object[] data)
- static int [GetPing](#) ()
- static void [FetchServerTimestamp](#) ()
- static void [SendOutgoingCommands](#) ()
- static bool [CloseConnection](#) ([PhotonPlayer](#) kickPlayer)
- static void [Destroy](#) ([PhotonView](#) targetView)
- static void [Destroy](#) (GameObject targetGo)
- static void [DestroyPlayerObjects](#) ([PhotonPlayer](#) targetPlayer)
- static void [DestroyPlayerObjects](#) (int targetPlayerId)
- static void [DestroyAll](#) ()
- static void [RemoveRPCs](#) ([PhotonPlayer](#) targetPlayer)
- static void [RemoveRPCs](#) ([PhotonView](#) targetPhotonView)
- static void [RemoveRPCsInGroup](#) (int targetGroup)
- static void [SetReceivingEnabled](#) (int group, bool enabled)
- static void [SetSendingEnabled](#) (int group, bool enabled)
- static void [SetLevelPrefix](#) (short prefix)
- static void [LoadLevel](#) (int levelNumber)
- static void [LoadLevel](#) (string levelName)

## Public Attributes

- const string [versionPUN](#) = "1.25"
- const string [serverSettingsAssetFile](#) = "PhotonServerSettings"
- const string [serverSettingsAssetPath](#) = "Assets/Photon Unity Networking/Resources/" + PhotonNetwork.serverSettingsAssetFile + ".asset"

## Static Public Attributes

- static readonly int [MAX\\_VIEW\\_IDS](#) = 1000
- static [ServerSettings](#) [PhotonServerSettings](#) = ([ServerSettings](#))Resources.Load([PhotonNetwork.serverSettingsAssetFile](#), typeof([ServerSettings](#)))
- static float [precisionForVectorSynchronization](#) = 0.000099f
- static float [precisionForQuaternionSynchronization](#) = 1.0f
- static float [precisionForFloatSynchronization](#) = 0.01f
- static bool [InstantiateInRoomOnly](#) = true
- static [PhotonLogLevel](#) [logLevel](#) = [PhotonLogLevel.ErrorsOnly](#)
- static bool [UsePrefabCache](#) = true
- static Dictionary< string, [GameObject](#) > [PrefabCache](#) = new Dictionary<string, [GameObject](#)>()
- static bool [UseNameServer](#) = false
- static HashSet< [GameObject](#) > [SendMonoMessageTargets](#)
- static [EventCallback](#) [OnEventCall](#)

## Properties

- static string [ServerAddress](#) [get]
- static bool [connected](#) [get]
- static bool [connecting](#) [get]
- static bool [connectedAndReady](#) [get]
- static [ConnectionState](#) [connectionState](#) [get]
- static [PeerState](#) [connectionStateDetailed](#) [get]
- static [AuthenticationValues](#) [AuthValues](#) [get, set]
- static [Room](#) [room](#) [get]
- static [PhotonPlayer](#) [player](#) [get]
- static [PhotonPlayer](#) [masterClient](#) [get]
- static string [playerName](#) [get, set]
- static [PhotonPlayer\[\]](#) [playerList](#) [get]
- static [PhotonPlayer\[\]](#) [otherPlayers](#) [get]
- static List< [FriendInfo](#) > [Friends](#) [get, set]
- static int [FriendsListAge](#) [get]
- static bool [offlineMode](#) [get, set]
- static int [maxConnections](#) [get, set]
- static bool [automaticallySyncScene](#) [get, set]
- static bool [autoCleanUpPlayerObjects](#) [get, set]
- static bool [autoJoinLobby](#) [get, set]
- static bool [insideLobby](#) [get]
- static [TypedLobby](#) [lobby](#) [get, set]
- static int [sendRate](#) [get, set]
- static int [sendRateOnSerialize](#) [get, set]
- static bool [isMessageQueueRunning](#) [get, set]
- static int [unreliableCommandsLimit](#) [get, set]
- static double [time](#) [get]
- static bool [isMasterClient](#) [get]

- static bool [inRoom](#) [get]
- static bool [isNonMasterClientInRoom](#) [get]
- static int [countOfPlayersOnMaster](#) [get]
- static int [countOfPlayersInRooms](#) [get]
- static int [countOfPlayers](#) [get]
- static int [countOfRooms](#) [get]
- static bool [NetworkStatisticsEnabled](#) [get, set]
- static int [ResentReliableCommands](#) [get]
- static bool [CrcCheckEnabled](#) [get, set]
- static int [PacketLossByCrcCheck](#) [get]
- static int [MaxResendsBeforeDisconnect](#) [get, set]
- static ServerConnection [Server](#) [get]

### 8.14.1 Detailed Description

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

### 8.14.2 Member Function Documentation

#### 8.14.2.1 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](#).

#### 8.14.2.2 static bool PhotonNetwork.CloseConnection ( [PhotonPlayer](#) *kickPlayer* ) [static]

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

Only the target player gets this event. That player will disconnect automatically, which is what the others will notice, too.

##### Parameters

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

#### 8.14.2.3 static bool PhotonNetwork.ConnectToBestCloudServer ( string *gameVersion* ) [static]

Connect to the [Photon](#) Cloud region with the lowest ping (on platforms that support Unity's Ping).

Will save the result of pinging all cloud servers in PlayerPrefs. Calling this the first time can take +-2 seconds. The ping result can be overridden via [PhotonNetwork.OverrideBestCloudServer\(..\)](#) This call can take up to 2 seconds if it is the first time you are using this, all cloud servers will be pinged to check for the best region.

The PUN Setup Wizard stores your appId in a settings file and applies a server address/port. To connect to the [Photon](#) Cloud, a valid AppId must be in the settings file (shown in the [Photon](#) Cloud Dashboard). <https://cloud.exitgames.com/dashboard>

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

- Invalid AppId (calls: [OnFailedToConnectToPhoton\(\)](#). check exact AppId value)
- Network issues (calls: [OnFailedToConnectToPhoton\(\)](#))
- Invalid region (calls: [OnConnectionFail\(\)](#) with DisconnectCause.InvalidRegion)



- Subscription CCU limit reached (calls: [OnConnectionFail\(\)](#) with DisconnectCause.MaxCcuReached. also calls: [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).
--------------------	---

#### Returns

If this client is going to connect to cloud server based on ping. Even if true, this does not guarantee a connection but the attempt is being made.

**8.14.2.4** `static bool PhotonNetwork.ConnectToMaster ( string masterServerAddress, int port, string appId, string gameVersion ) [static]`

Connect to a [Photon](#) Master Server by address, port, appId and game(client) version.

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

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

- Invalid AppId (calls: [OnFailedToConnectToPhoton\(\)](#). check exact AppId value)
- Network issues (calls: [OnFailedToConnectToPhoton\(\)](#))
- Invalid region (calls: [OnConnectionFail\(\)](#) with DisconnectCause.InvalidRegion)
- Subscription CCU limit reached (calls: [OnConnectionFail\(\)](#) with DisconnectCause.MaxCcuReached. also calls: [OnPhotonMaxCcuReached\(\)](#))

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

#### Parameters

<i>masterServer-Address</i>	The server's address (either your own or <a href="#">Photon</a> Cloud address).
<i>port</i>	The 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 by gameversion (which allows you to make breaking changes).

**8.14.2.5** `static bool PhotonNetwork.ConnectUsingSettings ( string gameVersion ) [static]`

Connect to [Photon](#) as configured in the editor (saved in PhotonServerSettings file).

This method will disable offlineMode (which won't destroy any instantiated GOs) and it will set isMessageQueue-Running to true.

Your server configuration is created by the PUN Wizard and contains the AppId and region for [Photon](#) Cloud games and the server address if you host [Photon](#) yourself. These settings usually don't change often.

To ignore the config file and connect anywhere call: [PhotonNetwork.ConnectToMaster](#).

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

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

- Invalid AppId (calls: [OnFailedToConnectToPhoton\(\)](#). check exact AppId value)
- Network issues (calls: [OnFailedToConnectToPhoton\(\)](#))
- Invalid region (calls: [OnConnectionFail\(\)](#) with DisconnectCause.InvalidRegion)
- Subscription CCU limit reached (calls: [OnConnectionFail\(\)](#) with DisconnectCause.MaxCcuReached. also calls: [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).
--------------------	---

#### 8.14.2.6 `static bool 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).

The created room is automatically placed in the currently used lobby or the default-lobby if you didn't explicitly join one.

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

#### 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.

#### 8.14.2.7 `static bool 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).

The created room is automatically placed in the currently used lobby or the default-lobby if you didn't explicitly join one.

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

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

#### Parameters

<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).

#### 8.14.2.8 static bool PhotonNetwork.CreateRoom ( string *roomName* ) [static]

Creates a room with given name but fails if this room(name) is existing already. Creates random name for room-Name null.

If you don't want to create a unique room-name, pass null or "" as name and the server will assign a roomName (a GUID as string).

The created room is automatically placed in the currently used lobby (if any) or the default-lobby if you didn't explicitly join one.

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

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

##### Parameters

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

#### 8.14.2.9 static bool PhotonNetwork.CreateRoom ( string *roomName*, RoomOptions *roomOptions*, TypedLobby *typedLobby* ) [static]

Creates a room but fails if this room is existing already. Can only be called on Master Server.

When successful, this calls the callbacks OnCreatedRoom and OnJoinedRoom (the latter, cause you join as first player). If the room can't be created (because it exists already), OnPhotonCreateRoomFailed gets called.

If you don't want to create a unique room-name, pass null or "" as name and the server will assign a roomName (a GUID as string).

Rooms can be created in any number of lobbies. Those don't have to exist before you create a room in them (they get auto-created on demand). Lobbies can be useful to split room lists on the server-side already. That can help keep the room lists short and manageable. If you set a typedLobby parameter, the room will be created in that lobby (no matter if you are active in any). If you don't set a typedLobby, the room is automatically placed in the currently active lobby (if any) or the default-lobby.

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

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

##### Parameters

<i>roomName</i>	Unique name of the room to create. Pass null or "" to make the server generate a name.
<i>roomOptions</i>	Common options for the room like maxPlayers, initial custom room properties and similar. See <a href="#">RoomOptions</a> type..

<i>typedLobby</i>	If null, the room is automatically created in the currently used lobby (which is "default" when you didn't join one explicitly).
-------------------	--

#### 8.14.2.10 static void PhotonNetwork.Destroy ( PhotonView targetView ) [static]

Network-Destroy the GameObject associated with the [PhotonView](#), unless the [PhotonView](#) is static or not under this client's control.

Destroying a networked GameObject while in a [Room](#) includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the [PhotonNetwork.Instantiate](#) call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Usually, when you leave a room, the GOs get destroyed automatically. If you have to destroy a GO while not in a room, the Destroy is only done locally.

Destroying networked objects works only if they got created with [PhotonNetwork.Instantiate\(\)](#). Objects loaded with a scene are ignored, no matter if they have [PhotonView](#) components.

The GameObject must be under this client's control:

- Instantiated and owned by this client.
- Instantiated objects of players who left the room are controlled by the Master Client.
- Scene-owned game objects are controlled by the Master Client.
- GameObject can be destroyed while client is not in a room.

#### Returns

Nothing. Check error debug log for any issues.

#### 8.14.2.11 static void PhotonNetwork.Destroy ( GameObject targetGo ) [static]

Network-Destroy the GameObject, unless it is static or not under this client's control.

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the [PhotonNetwork.Instantiate](#) call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Usually, when you leave a room, the GOs get destroyed automatically. If you have to destroy a GO while not in a room, the Destroy is only done locally.

Destroying networked objects works only if they got created with [PhotonNetwork.Instantiate\(\)](#). Objects loaded with a scene are ignored, no matter if they have [PhotonView](#) components.

The GameObject must be under this client's control:

- Instantiated and owned by this client.
- Instantiated objects of players who left the room are controlled by the Master Client.
- Scene-owned game objects are controlled by the Master Client.
- GameObject can be destroyed while client is not in a room.

**Returns**

Nothing. Check error debug log for any issues.

**8.14.2.12 static void PhotonNetwork.DestroyAll ( ) [static]**

Network-Destroy all GameObjects, PhotonViews and their RPCs in the room. Removes anything buffered from the server. Can only be called by Master Client (for anyone).

Can only be called by Master Client (for anyone). Unlike the Destroy methods, this will remove anything from the server's room buffer. If your game buffers anything beyond Instantiate and RPC calls, that will be cleaned as well from server.

Destroying all includes:

- Remove anything from the server's room buffer (Instantiate, RPCs, anything buffered).
- Sending a message to other clients to destroy everything locally, too (affected by network lag).

Destroying networked objects works only if they got created with [PhotonNetwork.Instantiate\(\)](#). Objects loaded with a scene are ignored, no matter if they have [PhotonView](#) components.

**Returns**

Nothing. Check error debug log for any issues.

**8.14.2.13 static void PhotonNetwork.DestroyPlayerObjects ( PhotonPlayer targetPlayer ) [static]**

Network-Destroy all GameObjects, PhotonViews and their RPCs of targetPlayer. Can only be called on local player (for "self") or Master Client (for anyone).

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the [PhotonNetwork.Instantiate](#) call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Destroying networked objects works only if they got created with [PhotonNetwork.Instantiate\(\)](#). Objects loaded with a scene are ignored, no matter if they have [PhotonView](#) components.

**Returns**

Nothing. Check error debug log for any issues.

**8.14.2.14 static void PhotonNetwork.DestroyPlayerObjects ( int targetPlayerId ) [static]**

Network-Destroy all GameObjects, PhotonViews and their RPCs of this player (by ID). Can only be called on local player (for "self") or Master Client (for anyone).

Destroying a networked GameObject includes:

- Removal of the Instantiate call from the server's room buffer.
- Removing RPCs buffered for PhotonViews that got created indirectly with the [PhotonNetwork.Instantiate](#) call.
- Sending a message to other clients to remove the GameObject also (affected by network lag).

Destroying networked objects works only if they got created with [PhotonNetwork.Instantiate\(\)](#). Objects loaded with a scene are ignored, no matter if they have [PhotonView](#) components.

**Returns**

Nothing. Check error debug log for any issues.

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

**8.14.2.16 delegate void PhotonNetwork.EventCallback ( byte *eventCode*, object *content*, int *senderId* )**

Defines the delegate usable in OnEventCall.

**Parameters**

<i>eventCode</i>	The code assignend to the incoming event.
<i>content</i>	The content the sender put into the event.
<i>senderId</i>	The ID of the player who sent the event. It might be 0, if the "room" sent the event.

**8.14.2.17 static void PhotonNetwork.FetchServerTimestamp ( ) [static]**

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

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

**8.14.2.18 static bool PhotonNetwork.FindFriends ( string[] *friendsToFind* ) [static]**

Requests the rooms and online status for a list of friends (with *PlayerName*) and saves the result in [PhotonNetwork.Friends](#).

Works only on Master Server to find the rooms played by a selected list of users. All client must set a unique username via *PlayerName* property or *CustomAuthValues*.

The result will be mapped to *LoadBalancingClient.Friends* when available. The list is initialized by *OpFindFriends* on first use (before that, it is null). To refresh the list, call *FindFriends* again (but not too frequently).

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.

#### 8.14.2.19 static int PhotonNetwork.GetPing ( ) [static]

The current roundtrip time to the photon server.

##### Returns

Roundtrip time (to server and back).

#### 8.14.2.20 static RoomInfo [] PhotonNetwork.GetRoomList ( ) [static]

Gets currently known rooms as [RoomInfo](#) array. This is available and updated while in a lobby (check `insideLobby`).

This list is a cached copy of the internal rooms list so it can be accessed each frame if needed. Per [RoomInfo](#) you can check if the room is full by comparing `playerCount` and `maxPlayers` before you allow a join.

The name of a room must be used to join it (via `JoinRoom`).

Closed rooms are also listed by lobbies but they can't be joined. While in a room, any player can set [Room.visible](#) and [Room.open](#) to hide rooms from matchmaking and close them.

##### Returns

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

#### 8.14.2.21 static void PhotonNetwork.InitializeSecurity ( ) [static]

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

#### 8.14.2.22 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](#).

#### 8.14.2.23 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](#).

**8.14.2.24** `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](#).

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

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

**8.14.2.26** `static bool PhotonNetwork.JoinLobby ( ) [static]`

On MasterServer this joins the default lobby which list rooms currently in use.

The room list is sent and refreshed by the server. You can access this cached list by [PhotonNetwork.GetRoomList\(\)](#).

Per room you should check if it's full or not before joining. [Photon](#) also lists rooms that are full, unless you close and hide them (room.open = false and room.visible = false).

In best case, you make your clients join random games, as described here: <http://doc.exitgames.com/en/realtime/current/reference/matchmaking-and-lobby>

You can show your current players and room count without joining a lobby (but you must be on the master server). Use: `countOfPlayers`, `countOfPlayersOnMaster`, `countOfPlayersInRooms` and `countOfRooms`.

You can use more than one lobby to keep the room lists shorter. See [JoinLobby\(TypedLobby lobby\)](#). When creating new rooms, they will be "attached" to the currently used lobby or the default lobby.

You can use `JoinRandomRoom` without being in a lobby! Set `autoJoinLobby = false` before you connect, to not join a lobby. In that case, the connect-workflow will call `OnConnectedToMaster` (if you implement it) when it's done.



**8.14.2.27 static bool PhotonNetwork.JoinLobby ( TypedLobby typedLobby ) [static]**

On a Master Server you can join a lobby to get lists of available rooms.

The room list is sent and refreshed by the server. You can access this cached list by [PhotonNetwork.GetRoomList\(\)](#).

Any client can "make up" any lobby on the fly. Splitting rooms into multiple lobbies will keep each list shorter. However, having too many lists might ruin the matchmaking experience.

In best case, you create a limited number of lobbies. For example, create a lobby per game-mode: "koth" for king of the hill and "ffa" for free for all, etc.

There is no listing of lobbies at the moment.

Sql-typed lobbies offer a different filtering model for random matchmaking. This might be more suited for skillbased-games. However, you will also need to follow the conventions for naming filterable properties in sql-lobbies! Both is explained in the matchmaking doc linked below.

In best case, you make your clients join random games, as described here: <http://confluence.-exitgames.com/display/PTN/Op+JoinRandomGame>

Per room you should check if it's full or not before joining. [Photon](#) does list rooms that are full, unless you close and hide them (room.open = false and room.visible = false).

You can show your games current players and room count without joining a lobby (but you must be on the master server). Use: countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms and countOfRooms.

When creating new rooms, they will be "attached" to the currently used lobby or the default lobby.

You can use JoinRandomRoom without being in a lobby! Set autoJoinLobby = false before you connect, to not join a lobby. In that case, the connect-workflow will call OnConnectedToMaster (if you implement it) when it's done.

**Parameters**

<i>typedLobby</i>	A typed lobby to join (must have name and type).
-------------------	--

**8.14.2.28 static bool PhotonNetwork.JoinOrCreateRoom ( string roomName, RoomOptions roomOptions, TypedLobby typedLobby ) [static]**

Lets you either join a named room or create it on the fly - you don't have to know if someone created the room already.

This makes it easier for groups of players to get into the same room. Once the group exchanged a roomName, any player can call JoinOrCreateRoom and it doesn't matter who actually joins or creates the room.

The parameters roomOptions and typedLobby are only used when the room actually gets created by this client. You know if this client created a room, if you get a callback OnCreatedRoom (before OnJoinedRoom gets called as well).

**Parameters**

<i>roomName</i>	Name of the room to join. Must be non null.
<i>roomOptions</i>	Options for the room, in case it does not exist yet. Else these values are ignored.
<i>typedLobby</i>	Lobby you want a new room to be listed in. Ignored if the room was existing and got joined.

**Returns**

If the operation got queued and will be sent.

**8.14.2.29 static bool PhotonNetwork.JoinRandomRoom ( ) [static]**

Joins any available room of the currently used lobby and fails if none is available.

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby. Use the JoinRandomRoom overload with [TypedLobby](#) parameter.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the currently active lobby or (if no lobby is joined) in the default lobby.

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

**8.14.2.30** `static bool PhotonNetwork.JoinRandomRoom ( Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers ) [static]`

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

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby. Use the JoinRandomRoom overload with [TypedLobby](#) parameter.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the currently active lobby or (if no lobby is joined) in the default lobby.

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

#### Parameters

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

**8.14.2.31** `static bool PhotonNetwork.JoinRandomRoom ( Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType, TypedLobby typedLobby, string sqlLobbyFilter ) [static]`

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

Rooms can be created in arbitrary lobbies which get created on demand. You can join rooms from any lobby without actually joining the lobby with this overload.

This method will only match rooms attached to one lobby! If you use many lobbies, you might have to repeat JoinRandomRoom, to find some fitting room. This method looks up a room in the specified lobby or the currently active lobby (if none specified) or in the default lobby (if none active).

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

#### Parameters

<i>expectedCustomRoomProperties</i>	Filters for rooms that match these custom properties (string keys and values). To ignore, pass null.
<i>expectedMaxPlayers</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.
<i>typedLobby</i>	The lobby in which you want to lookup a room. Pass null, to use the default lobby. This does not join that lobby and neither sets the lobby property.

<i>sqlLobbyFilter</i>	A filter-string for SQL-typed lobbies.
-----------------------	--

#### 8.14.2.32 static bool PhotonNetwork.JoinRoom ( string *roomName*, bool *createIfNotExists* ) [static]

Join room by roomName with an option to create it on the fly if not existing.

Join will try to enter a room by roomName. If this room is full or closed, this will fail. If the room is not existing, JoinRoom will also fail by default.

You can set createIfNotExists to true to make the server create the room if required. This makes it easier for groups of players to get into the same room. Once the group exchanged a roomName, any player can try to join or create the room in one step - it doesn't matter who's first.

[OnJoinedRoom\(\)](#) gets called if the room existed and was joined, [OnCreatedRoom\(\)](#) gets called if the room didn't exist and this client created it. [OnPhotonJoinRoomFailed\(\)](#) gets called if the room couldn't be joined or created. Implement either in any script in the scene to react to joining/creating a room.

To join a room from the lobby's listing, use [RoomInfo.name](#) as roomName here.

PhotonNetworkingMessage.OnPhotonJoinRoomFailed PhotonNetworkingMessage.OnJoinedRoom

##### Parameters

<i>roomName</i>	Unique name of the room to join (or create if createIfNotExists is true).
<i>createIfNotExists</i>	If true, the server will attempt to create a room, making the success callback <a href="#">OnCreatedRoom()</a> .

#### 8.14.2.33 static bool PhotonNetwork.JoinRoom ( string *roomName* ) [static]

Join room by roomname and on success calls [OnJoinedRoom\(\)](#). This is not affected by lobbies.

On success, the method [OnJoinedRoom\(\)](#) is called on any script. You can implement it to react to joining a room.

JoinRoom fails if the room is either full or no longer available (it might become empty while you attempt to join). Implement [OnPhotonJoinRoomFailed\(\)](#) to get a callback in error case.

To join a room from the lobby's listing, use [RoomInfo.name](#) as roomName here. Despite using multiple lobbies, a roomName is always "global" for your application and so you don't have to specify which lobby it's in. The Master Server will find the room. In the [Photon](#) Cloud, an application is defined by AppId, Game- and PUN-version.

PhotonNetworkingMessage.OnPhotonJoinRoomFailed PhotonNetworkingMessage.OnJoinedRoom

##### Parameters

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

#### 8.14.2.34 static bool PhotonNetwork.LeaveLobby ( ) [static]

Leave a lobby to stop getting updates about available rooms.

This does not reset [PhotonNetwork.lobby](#)! This allows you to join this particular lobby later easily.

The values countOfPlayers, countOfPlayersOnMaster, countOfPlayersInRooms and countOfRooms are received even without being in a lobby.

You can use JoinRandomRoom without being in a lobby. Use autoJoinLobby to not join a lobby when you connect.

#### 8.14.2.35 static bool PhotonNetwork.LeaveRoom ( ) [static]

Leave the current room and return to the Master Server where you can join or create rooms (see remarks).

This will clean up all (network) GameObjects with a [PhotonView](#), unless you changed `autoCleanUp` to false. Returns to the Master Server.

#### 8.14.2.36 `static void PhotonNetwork.LoadLevel ( int levelNumber ) [static]`

Wraps loading a level to pause the network message-queue. Optionally syncs the loaded level in a room.

While loading levels, it makes sense to not dispatch messages received by other players. This method takes care of that by setting [PhotonNetwork.isMessageQueueRunning](#) = false and enabling the queue when the level was loaded.

To sync the loaded level in a room, set [PhotonNetwork.automaticallySyncScene](#) to true. The Master Client of a room will then sync the loaded level with every other player in the room.

You should make sure you don't fire RPCs before you load another scene (which doesn't contain the same Game-Objects and PhotonViews). You can call this in `OnJoinedRoom`.

This uses `Application.LoadLevel`.

##### Parameters

<i>levelNumber</i>	Number of the level to load. When using level numbers, make sure they are identical on all clients.
--------------------	---

#### 8.14.2.37 `static void PhotonNetwork.LoadLevel ( string levelName ) [static]`

Wraps loading a level to pause the network message-queue. Optionally syncs the loaded level in a room.

While loading levels, it makes sense to not dispatch messages received by other players. This method takes care of that by setting [PhotonNetwork.isMessageQueueRunning](#) = false and enabling the queue when the level was loaded.

To sync the loaded level in a room, set [PhotonNetwork.automaticallySyncScene](#) to true. The Master Client of a room will then sync the loaded level with every other player in the room.

You should make sure you don't fire RPCs before you load another scene (which doesn't contain the same Game-Objects and PhotonViews). You can call this in `OnJoinedRoom`.

This uses `Application.LoadLevel`.

##### Parameters

<i>levelName</i>	Name of the level to load. Make sure it's available to all clients in the same room.
------------------	--

#### 8.14.2.38 `static void PhotonNetwork.NetworkStatisticsReset ( ) [static]`

Resets the traffic stats and re-enables them.

#### 8.14.2.39 `static string PhotonNetwork.NetworkStatisticsToString ( ) [static]`

Only available when `NetworkStatisticsEnabled` was used to gather some stats.

##### Returns

A string with vital networking statistics.

#### 8.14.2.40 `static void PhotonNetwork.OverrideBestCloudServer ( CloudServerRegion region ) [static]`

Overwrites the region that is used for [ConnectToBestCloudServer\(string gameVersion\)](#).

This will overwrite the result of pinging all cloud servers. Use this to allow your users to specify their region manually.

**8.14.2.41** `static bool PhotonNetwork.RaiseEvent ( byte eventCode, object eventContent, bool sendReliable, RaiseEventOptions options ) [static]`

Sends fully customizable events in a room. Events consist of at least an [EventCode](#) (1..199) and can have content.

To receive the events someone sends, register your handling method in [PhotonNetwork.OnEventCall](#).

Example: `private void OnEventHandler(byte eventCode, object content, PhotonPlayer sender) { Debug.Log("On-EventHandler"); }`

[PhotonNetwork.OnEventCall](#) += this.OnEventHandler;

The *eventContent* is optional. To be able to send something, it must be a "serializable type", something that the client can turn into a `byte[]` basically. Most basic types and arrays of them are supported, including Unity's `Vector2`, `Vector3`, `Quaternion`. Transforms or classes some project defines are NOT supported! You can make your own class a "serializable type" by following the example in `CustomTypes.cs`.

The [RaiseEventOptions](#) have some (less intuitive) combination rules: If you set *targetActors* (an array of [PhotonPlayer.ID](#) values), the *receivers* parameter gets ignored. When using event caching, the *targetActors*, *receivers* and *interestGroup* can't be used. Buffered events go to all. When using *cachingOption removeFromRoomCache*, the *eventCode* and *content* are actually not sent but used as filter.

#### Parameters

<i>eventCode</i>	A byte identifying the type of event. You might want to use a code per action or to signal which content can be expected. Allowed: 1..199.
<i>eventContent</i>	Some serializable object like string, byte, integer, float (etc) and arrays of those. Hashtables with byte keys are good to send variable content.
<i>sendReliable</i>	Makes sure this event reaches all players. It gets acknowledged, which requires bandwidth and it can't be skipped (might add lag in case of loss).
<i>options</i>	Allows more complex usage of events. If null, <a href="#">RaiseEventOptions.Default</a> will be used (which is fine).

#### Returns

False if event could not be sent

**8.14.2.42** `static void PhotonNetwork.RefreshCloudServerRating ( ) [static]`

Pings all cloud servers again to find the one with best ping (currently).

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

Remove all buffered RPCs from server that were sent by *targetPlayer*. Can only be called on local player (for "self") or Master Client (for anyone).

This method requires either:

- This is the *targetPlayer*'s client.
- This client is the Master Client (can remove any [PhotonPlayer](#)'s RPCs).

If the *targetPlayer* calls RPCs at the same time that this is called, network lag will determine if those get buffered or cleared like the rest.

#### Parameters

<i>targetPlayer</i>	This player's buffered RPCs get removed from server buffer.
---------------------	---

#### 8.14.2.44 `static void PhotonNetwork.RemoveRPCs ( PhotonView targetPhotonView ) [static]`

Remove all buffered RPCs from server that were sent via targetPhotonView. The Master Client and the owner of the targetPhotonView may call this.

This method requires either:

- The targetPhotonView is owned by this client (Instantiated by it).
- This client is the Master Client (can remove any PhotonView's RPCs).

##### Parameters

<i>targetPhoton-View</i>	RPCs buffered for this PhotonView get removed from server buffer.
--------------------------	---

#### 8.14.2.45 `static void PhotonNetwork.RemoveRPCsInGroup ( int targetGroup ) [static]`

Remove all buffered RPCs from server that were sent in the targetGroup, if this is the Master Client or if this controls the individual PhotonView.

This method requires either:

- This client is the Master Client (can remove any RPCs per group).
- Any other client: each PhotonView is checked if it is under this client's control. Only those RPCs are removed.

##### Parameters

<i>targetGroup</i>	Interest group that gets all RPCs removed.
--------------------	--

#### 8.14.2.46 `static void PhotonNetwork.SendOutgoingCommands ( ) [static]`

Can be used to immediately send the RPCs and Instantiates just called, so they are on their way to the other players.

This could be useful if you do a RPC to load a level and then load it yourself. While loading, no RPCs are sent to others, so this would delay the "load" RPC. You can send the RPC to "others", use this method, disable the message queue (by `isMessageQueueRunning`) and then load.

#### 8.14.2.47 `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
---------------	-------------------------------------

## 8.14.2.48 static bool PhotonNetwork.SetMasterClient ( PhotonPlayer masterClientPlayer ) [static]

Allows the current Master Client to assign someone else as MC - custom selection should pick the same user on any client.

The ReceiverGroup.MasterClient (usable in RPCs) is not affected by this (still points to lowest player.ID in room). Avoid using this enum value (and send to a specific player instead).

If the current Master Client leaves, PUN will detect a new one by "lowest player ID". Implement OnMasterClient-Switched to get a callback in this case. The PUN-selected Master Client might assign a new one.

Make sure you don't create an endless loop of Master-assigning! When selecting a custom Master Client, all clients should point to the same player, no matter who actually assigns this player.

Locally the Master Client is immediately switched, while remote clients get an event. This means the game is temporarily without Master Client like when a current Master Client leaves.

When switching the Master Client manually, keep in mind that this user might leave and not do it's work, just like any Master Client.

## Parameters

<i>masterClient- Player</i>	The player of the next Master Client.
---------------------------------	---------------------------------------

## Returns

False when this synced action couldn't be done. Must be online and Master Client.

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

Sets this (local) player's properties and synchronizes them to the other players (don't modify them directly).

While in a room, your properties are synced with the other players. CreateRoom, JoinRoom and JoinRandomRoom will all apply your player's custom properties when you enter the room. The whole Hashtable will get sent. Minimize the traffic by setting only updated key/values.

If the Hashtable is null, the custom properties will be cleared. Custom properties are never cleared automatically, so they carry over to the next room, if you don't change them.

Don't set properties by modifying PhotonNetwork.player.customProperties!

## Parameters

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

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

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

## Parameters

<i>group</i>	The interest group to affect.
<i>enabled</i>	Sets if receiving from group to enabled (or not).

**8.14.2.51** `static void PhotonNetwork.SetSendingEnabled ( int group, bool enabled )` `[static]`

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

Parameters

<i>group</i>	The interest group to affect.
<i>enabled</i>	Sets if sending to group is enabled (or not).

**8.14.2.52** `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.
---------------	---

### 8.14.3 Member Data Documentation

**8.14.3.1** `bool PhotonNetwork.InstantiateInRoomOnly = true` `[static]`

If true, Instantiate methods will check if you are in a room and fail if you are not.

Instantiating anything outside of a specific room is very likely to break things. Turn this off only if you know what you do.

**8.14.3.2** `PhotonLogLevel PhotonNetwork.logLevel = PhotonLogLevel.ErrorsOnly` `[static]`

Network log level. Controls how verbose PUN is.

**8.14.3.3** `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

**8.14.3.4** `EventCallback PhotonNetwork.OnEventCall` `[static]`

Register your RaiseEvent handling methods here by using "+=".

[RaiseEvent](#)

**8.14.3.5** `ServerSettings PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.serverSettingsAssetFile, typeof(ServerSettings))` `[static]`

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

**8.14.3.6** `float PhotonNetwork.precisionForFloatSynchronization = 0.01f` `[static]`

The minimum difference between floats before we send it via a [PhotonView](#)'s OnSerialize/ObservingComponent



**8.14.3.7** `float PhotonNetwork.precisionForQuaternionSynchronization = 1.0f` [static]

The minimum angle that a rotation needs to change before we send it via a [PhotonView](#)'s OnSerialize/Observing-Component

**8.14.3.8** `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`.

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

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

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

**8.14.3.10** `HashSet<GameObject> PhotonNetwork.SendMonoMessageTargets` [static]

If not null, this is the (exclusive) list of GameObjects that get called by `PUN SendMonoMessage()`.

For all callbacks defined in `PhotonNetworkingMessage`, PUN will use `SendMonoMessage` and call `FindObjectsOfType()` to find all scripts and GameObjects that might want a callback by PUN.

PUN callbacks are not very frequent (in-game, property updates are most frequent) but `FindObjectsOfType` is time consuming and with a large number of GameObjects, performance might suffer.

Optionally, `SendMonoMessageTargets` can be used to supply a list of target GameObjects. This skips the `FindObjectsOfType()` but any GameObject that needs callbacks will have to Add itself to this list.

If null, the default behaviour is to do a `SendMessage` on each GameObject with a `MonoBehaviour`.

**8.14.3.11** `const string PhotonNetwork.serverSettingsAssetFile = "PhotonServerSettings"`

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

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

Path to the `PhotonServerSettings` file (used by `PhotonEditor`).

**8.14.3.13** `bool PhotonNetwork.UseNameServer = false` [static]

**8.14.3.14** `bool PhotonNetwork.UsePrefabCache = true` [static]

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

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

#### 8.14.3.15 `const string PhotonNetwork.versionPUN = "1.25"`

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

### 8.14.4 Property Documentation

#### 8.14.4.1 `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`

#### 8.14.4.2 `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.

#### 8.14.4.3 `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).

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

Defines if PUN automatically synchronizes the loaded level per room. Default: false / disabled.

If the `MasterClient` loads a level, all clients will load the new scene too. This also takes care of smooth loading of the game scene after joining a game.

In best case, use [PhotonNetwork.LoadLevel](#) on the Master Client to change levels.

true if automatically sync scene; otherwise, false.

#### 8.14.4.5 `bool PhotonNetwork.connected` `[static]`, `[get]`

False until you connected to [Photon](#) initially. True in offline mode, while connected to any server and even while switching servers but

**8.14.4.6 bool PhotonNetwork.connectedAndReady** [static], [get]

A refined version of connected which is true only if your connection to the server is ready to accept operations like join, leave, etc.

**8.14.4.7 bool PhotonNetwork.connecting** [static], [get]

True when you called ConnectUsingSettings (or similar) until the low level connection to [Photon](#) gets established.

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

Simplified connection state

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

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

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

The count of players currently using this application (available on MasterServer in 5sec intervals).

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

Count of users currently playing your app in some room (sent every 5sec by Master Server). Use `playerList.Count` to get the count of players in the room you're in!

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

The count of players currently looking for a room (available on MasterServer in 5sec intervals).

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

The count of rooms currently in use (available on MasterServer in 5sec intervals).

While inside the lobby you can also check the count of listed rooms as: `PhotonNetwork.GetRoomList().Length`. Since PUN v1.25 this is only based on the statistic event [Photon](#) sends (counting all rooms).

**8.14.4.14 bool PhotonNetwork.CrcCheckEnabled** [static], [get], [set]

Crc checks can be useful to detect and avoid issues with broken datagrams. Can be enabled while not connected.

**8.14.4.15 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).

**8.14.4.16** `int PhotonNetwork.FriendsListAge` `[static], [get]`

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

**8.14.4.17** `bool PhotonNetwork.inRoom` `[static], [get]`

Is true while being in a room (connectionStateDetailed == PeerState.Joined).

Many actions can only be executed in a room, like Instantiate or Leave, etc. You can join a room in offline mode, too.

**8.14.4.18** `bool PhotonNetwork.insideLobby` `[static], [get]`

True while this client is in a lobby.

You are automatically leaving any lobby when you join a room! Lobbies only exist on the Master Server (whereas rooms are handled by Game Servers).

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

Are we the master client?

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

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

While IsMessageQueueRunning == false, the OnPhotonSerializeView calls are not done and nothing is sent by a client. Also, incoming messages will be queued until you re-activate the message queue.

This can be useful if you first want to load a level, then go on receiving data of PhotonViews and RPCs. The client will go on receiving and sending acknowledgements for incoming packages and your RPCs/Events. This adds "lag" and can cause issues when the pause is longer, as all incoming messages are just queued.

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

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

**8.14.4.22** `TypedLobby PhotonNetwork.lobby` `[static], [get], [set]`

**8.14.4.23** `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.

**8.14.4.24** `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.

**8.14.4.25** `int PhotonNetwork.MaxResendsBeforeDisconnect` `[static], [get], [set]`

Defines the number of times a reliable message can be resent before not getting an ACK for it will trigger a disconnect. Default: 5.

Less resends mean quicker disconnects, while more can lead to much more lag without helping. Min: 3. Max: 10.

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

Enables or disables the collection of statistics about this client's traffic.

If you encounter issues with clients, the traffic stats are a good starting point to find solutions. Only with enabled stats, you can use `GetVitalStats`

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

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

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

The other PhotonPlayers, not including our local player.

**8.14.4.29** `int PhotonNetwork.PacketLossByCrcCheck` `[static], [get]`

If `CrcCheckEnabled`, this counts the incoming packages that don't have a valid CRC checksum and got rejected.

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

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

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

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

**8.14.4.32** `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.

**8.14.4.33** `int PhotonNetwork.ResentReliableCommands` `[static], [get]`

Count of commands that got repeated (due to local repeat-timing before an ACK was received).

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

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

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

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

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

**8.14.4.36** `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.

**8.14.4.37** `ServerConnection PhotonNetwork.Server` `[static], [get]`

The server this client is currently connected or connecting to.

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

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

**8.14.4.39** `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.

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

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

## 8.15 PhotonPlayer Class Reference

### Public Member Functions

- [PhotonPlayer](#) (bool [isLocal](#), int actorID, string [name](#))
- override string [ToString](#) ()
- override bool [Equals](#) (object p)
- override int [GetHashCode](#) ()
- void [SetCustomProperties](#) (Hashtable propertiesToSet)

### Static Public Member Functions

- static [PhotonPlayer Find](#) (int ID)

### Public Attributes

- readonly bool [isLocal](#) = false

## Protected Member Functions

- [PhotonPlayer](#) (bool [isLocal](#), int actorID, Hashtable properties)

## Properties

- int [ID](#) [get]
- string [name](#) [get, set]
- bool [isMasterClient](#) [get]
- Hashtable [customProperties](#) [get, set]
- Hashtable [allProperties](#) [get]

### 8.15.1 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.

### 8.15.2 Constructor & Destructor Documentation

#### 8.15.2.1 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").

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

Internally used to create players from event Join

### 8.15.3 Member Function Documentation

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

Makes [PhotonPlayer](#) comparable

#### 8.15.3.2 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.

8.15.3.3 `override int PhotonPlayer.GetHashCode ( )`

8.15.3.4 `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.
------------------------	--

8.15.3.5 `override string PhotonPlayer.ToString ( )`

Gives the name.

## 8.15.4 Member Data Documentation

8.15.4.1 `readonly bool PhotonPlayer.isLocal = false`

Only one player is controlled by each client. Others are not local.

## 8.15.5 Property Documentation

8.15.5.1 `Hashtable PhotonPlayer.allProperties` `[get]`

Creates a Hashtable with all properties (custom and "well known" ones).

If used more often, this should be cached.

8.15.5.2 `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.

8.15.5.3 `int PhotonPlayer.ID` `[get]`

This player's actorID

8.15.5.4 `bool PhotonPlayer.isMasterClient` `[get]`

The player with the lowest actorID is the master and could be used for special tasks.



8.15.5.5 `string PhotonPlayer.name` `[get]`, `[set]`

Nickname of this player.

## 8.16 PhotonStatsGui Class Reference

Inherits `MonoBehaviour`.

### Public Member Functions

- void `Start` ()
- void `Update` ()
- void `OnGUI` ()
- void `TrafficStatsWindow` (int windowID)

### Public Attributes

- bool `statsWindowOn` = true
- bool `statsOn` = true
- bool `healthStatsVisible`
- bool `trafficStatsOn`
- bool `buttonsOn`
- Rect `statsRect` = new Rect(0, 100, 200, 50)
- int `WindowId` = 100

### 8.16.1 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).

### 8.16.2 Member Function Documentation

8.16.2.1 `void PhotonStatsGui.OnGUI ( )`

8.16.2.2 `void PhotonStatsGui.Start ( )`

8.16.2.3 `void PhotonStatsGui.TrafficStatsWindow ( int windowID )`

8.16.2.4 `void PhotonStatsGui.Update ( )`

Checks for shift+tab input combination (to toggle statsOn).

### 8.16.3 Member Data Documentation

8.16.3.1 `bool PhotonStatsGui.buttonsOn`

Show buttons to control stats and reset them.

#### 8.16.3.2 bool PhotonStatsGui.healthStatsVisible

Shows additional "health" values of connection.

#### 8.16.3.3 bool PhotonStatsGui.statsOn = true

Option to turn collecting stats on or off (used in [Update\(\)](#)).

#### 8.16.3.4 Rect PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)

Positioning rect for window.

#### 8.16.3.5 bool PhotonStatsGui.statsWindowOn = true

Shows or hides GUI (does not affect if stats are collected).

#### 8.16.3.6 bool PhotonStatsGui.trafficStatsOn

Shows additional "lower level" traffic stats.

#### 8.16.3.7 int PhotonStatsGui.WindowId = 100

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

## 8.17 PhotonStream Class Reference

### 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]

### 8.17.1 Detailed Description

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

The `isWriting` property will be true if this client is the "owner" of the [PhotonView](#) (and thus the `GameObject`). Add data to the stream and it's sent via the server to the other players in a room. On the receiving side, `isWriting` is false and the data should be read.

Send as few data as possible to keep connection quality up. An empty [PhotonStream](#) will not be sent.

Use either [Serialize\(\)](#) for reading and writing or [SendNext\(\)](#) and [ReceiveNext\(\)](#). The latter two are just explicit read and write methods but do about the same work as [Serialize\(\)](#). It's a matter of preference which methods you use.

See Also

[PhotonNetworkingMessage](#)

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 `PhotonStream.PhotonStream ( bool write, object[] incomingData )`

Creates a stream and initializes it. Used by PUN internally.

### 8.17.3 Member Function Documentation

#### 8.17.3.1 `object PhotonStream.ReceiveNext ( )`

Read next piece of data from the stream when `isReading` is true.

#### 8.17.3.2 `void PhotonStream.SendNext ( object obj )`

Add another piece of data to send it when `isWriting` is true.

#### 8.17.3.3 `void PhotonStream.Serialize ( ref bool myBool )`

Will read or write the value, depending on the stream's `isWriting` value.

#### 8.17.3.4 `void PhotonStream.Serialize ( ref int myInt )`

Will read or write the value, depending on the stream's `isWriting` value.

#### 8.17.3.5 `void PhotonStream.Serialize ( ref string value )`

Will read or write the value, depending on the stream's `isWriting` value.

#### 8.17.3.6 `void PhotonStream.Serialize ( ref char value )`

Will read or write the value, depending on the stream's `isWriting` value.

#### 8.17.3.7 `void PhotonStream.Serialize ( ref short value )`

Will read or write the value, depending on the stream's `isWriting` value.

**8.17.3.8 void PhotonStream.Serialize ( ref float *obj* )**

Will read or write the value, depending on the stream's isWriting value.

**8.17.3.9 void PhotonStream.Serialize ( ref PhotonPlayer *obj* )**

Will read or write the value, depending on the stream's isWriting value.

**8.17.3.10 void PhotonStream.Serialize ( ref Vector3 *obj* )**

Will read or write the value, depending on the stream's isWriting value.

**8.17.3.11 void PhotonStream.Serialize ( ref Vector2 *obj* )**

Will read or write the value, depending on the stream's isWriting value.

**8.17.3.12 void PhotonStream.Serialize ( ref Quaternion *obj* )**

Will read or write the value, depending on the stream's isWriting value.

**8.17.3.13 object [] PhotonStream.ToArray ( )**

Turns the stream into a new object[].

**8.17.4 Property Documentation****8.17.4.1 int PhotonStream.Count [get]**

Count of items in the stream.

**8.17.4.2 bool PhotonStream.isReading [get]**

If true, this client should read data send by another client.

**8.17.4.3 bool PhotonStream.isWriting [get]**

If true, this client should add data to the stream to send it.

**8.18 PhotonView Class Reference**

Inherits [Photon.MonoBehaviour](#).

**Public Member Functions**

- void [Awake](#) ()
- 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](#)

## Protected Member Functions

- void [ExecuteOnSerialize](#) (PhotonStream pStream, PhotonMessageInfo info)

## Properties

- int [prefix](#) [get, set]
- object[] [instantiationData](#) [get, set]
- int [viewID](#) [get, set]
- bool [isSceneView](#) [get]
- [PhotonPlayer](#) owner [get]
- int [OwnerActorNr](#) [get]
- bool [isMine](#) [get]

### 8.18.1 Detailed Description

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

### 8.18.2 Member Function Documentation

#### 8.18.2.1 void PhotonView.Awake ( )

Called by Unity on start of the application and does a setup the [PhotonView](#).

#### 8.18.2.2 void PhotonView.ExecuteOnSerialize ( PhotonStream pStream, PhotonMessageInfo info ) [protected]

#### 8.18.2.3 static PhotonView PhotonView.Find ( int viewID ) [static]

#### 8.18.2.4 static PhotonView PhotonView.Get ( Component component ) [static]

#### 8.18.2.5 static PhotonView PhotonView.Get ( GameObject gameObj ) [static]

#### 8.18.2.6 void PhotonView.OnApplicationQuit ( )

8.18.2.7 void PhotonView.OnDestroy ( )

8.18.2.8 void PhotonView.RPC ( string *methodName*, PhotonTargets *target*, params object[] *parameters* )

8.18.2.9 void PhotonView.RPC ( string *methodName*, PhotonPlayer *targetPlayer*, params object[] *parameters* )

8.18.2.10 override string PhotonView.ToString ( )

### 8.18.3 Member Data Documentation

8.18.3.1 int PhotonView.group = 0

8.18.3.2 int PhotonView.instantiationId

8.18.3.3 Component PhotonView.observed

8.18.3.4 OnSerializeRigidBody PhotonView.onSerializeRigidBodyOption = OnSerializeRigidBody.All

8.18.3.5 OnSerializeTransform PhotonView.onSerializeTransformOption = OnSerializeTransform.PositionAndRotation

8.18.3.6 int PhotonView.ownerId

8.18.3.7 int PhotonView.prefixBackup = -1

8.18.3.8 int PhotonView.subId

8.18.3.9 ViewSynchronization PhotonView.synchronization

### 8.18.4 Property Documentation

8.18.4.1 object [] PhotonView.instantiationData [get], [set]

This is the instantiationData that was passed when calling [PhotonNetwork.Instantiate\\*](#) (if that was used to spawn this prefab)

8.18.4.2 bool PhotonView.isMine [get]

True if the [PhotonView](#) is "mine" and can be controlled by this client.

PUN has an ownership concept that defines who can control and destroy each [PhotonView](#). True in case the owner matches the local [PhotonPlayer](#). True if this is a scene photonview on the Master client.

8.18.4.3 bool PhotonView.isSceneView [get]

True if the [PhotonView](#) was loaded with the scene (game object) or instantiated with InstantiateSceneObject.

Scene objects are not owned by a particular player but belong to the scene. Thus they don't get destroyed when their creator leaves the game and the current Master Client can control them (whoever that is). The ownerId is 0 (player IDs are 1 and up).

8.18.4.4 PhotonPlayer PhotonView.owner [get]

8.18.4.5 int PhotonView.OwnerActorNr [get]

8.18.4.6 int PhotonView.prefix [get], [set]

8.18.4.7 `int PhotonView.viewID` `[get]`, `[set]`

## 8.19 PingCloudRegions Class Reference

Inherits MonoBehaviour.

### Public Member Functions

- void [Awake](#) ()

### Static Public Member Functions

- static void [RefreshCloudServerRating](#) ()
- static void [ConnectToBestRegion](#) ()
- static void [DeleteRegionPref](#) ()
- static void [OverrideRegion](#) (CloudServerRegion region)
- static string [ResolveHost](#) (string hostName)

### Static Public Attributes

- static string [ClosestRegion](#)
- static [PingCloudRegions](#) Instance

### Properties

- static bool [ClosestRegionAvailable](#) `[get]`

#### 8.19.1 Detailed Description

This script is automatically added to the PhotonHandler gameobject by PUN. It will auto-ping the ExitGames cloud regions via Awake. This is done only once per client and the result is saved in PlayerPrefs. Use PhotonNetwork.ConnectToBestCloudServer(gameVersion) to connect to cloud region with best ping.

#### 8.19.2 Member Function Documentation

8.19.2.1 `void PingCloudRegions.Awake ( )`

8.19.2.2 `static void PingCloudRegions.ConnectToBestRegion ( )` `[static]`

8.19.2.3 `static void PingCloudRegions.DeleteRegionPref ( )` `[static]`

Removes the PlayerPref setting for "best region".

8.19.2.4 `static void PingCloudRegions.OverrideRegion ( CloudServerRegion region )` `[static]`

8.19.2.5 `static void PingCloudRegions.RefreshCloudServerRating ( )` `[static]`

8.19.2.6 `static string PingCloudRegions.ResolveHost ( string hostName )` `[static]`

Attempts to resolve a hostname into an IP string or returns empty string if that fails.

## Parameters

<i>hostName</i>	Hostname to resolve.
-----------------	----------------------

## Returns

IP string or empty string if resolution fails

### 8.19.3 Member Data Documentation

8.19.3.1 `string PingCloudRegions.ClosestRegion` `[static]`

8.19.3.2 `PingCloudRegions PingCloudRegions.Instance` `[static]`

### 8.19.4 Property Documentation

8.19.4.1 `bool PingCloudRegions.ClosestRegionAvailable` `[static], [get]`

## 8.20 RaiseEventOptions Class Reference

### Public Attributes

- EventCaching [CachingOption](#)
- byte [InterestGroup](#)
- int[] [TargetActors](#)
- ReceiverGroup [Receivers](#)
- byte [SequenceChannel](#)
- bool [ForwardToWebhook](#)
- int [CacheSliceIndex](#)

### Static Public Attributes

- static readonly [RaiseEventOptions Default](#) = new [RaiseEventOptions\(\)](#)

### 8.20.1 Detailed Description

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

### 8.20.2 Member Data Documentation

8.20.2.1 `int RaiseEventOptions.CacheSliceIndex`

Used along with `CachingOption` `SliceSetIndex`, `SlicePurgeIndex` or `SlicePurgeUpToIndex` if you want to set or purge a specific cache-slice.

8.20.2.2 `EventCaching RaiseEventOptions.CachingOption`

Defines if the server should simply send the event, put it in the cache or remove events that are like this one.

When using option: `SliceSetIndex`, `SlicePurgeIndex` or `SlicePurgeUpToIndex`, set a `CacheSliceIndex`. All other options except `SequenceChannel` get ignored.



8.20.2.3 `readonly RaiseEventOptions RaiseEventOptions.Default = new RaiseEventOptions()` `[static]`

Default options: CachingOption: DoNotCache, InterestGroup: 0, targetActors: null, receivers: Others, sequenceChannel: 0.

8.20.2.4 `bool RaiseEventOptions.ForwardToWebhook`

Events can be forwarded to Webhooks, which can evaluate and use the events to follow the game's state.

8.20.2.5 `byte RaiseEventOptions.InterestGroup`

The number of the Interest Group to send this to. 0 goes to all users but to get 1 and up, clients must subscribe to the group first.

8.20.2.6 `ReceiverGroup RaiseEventOptions.Receivers`

Sends the event to All, MasterClient or Others (default). Be careful with MasterClient, as the client might disconnect before it got the event and it gets lost.

8.20.2.7 `byte RaiseEventOptions.SequenceChannel`

Events are ordered per "channel". If you have events that are independent of others, they can go into another sequence or channel.

8.20.2.8 `int [] RaiseEventOptions.TargetActors`

A list of PhotonPlayer.IDs to send this event to. You can implement events that just go to specific users this way.

## 8.21 Room Class Reference

Inherits [RoomInfo](#).

### Public Member Functions

- void [SetCustomProperties](#) (Hashtable propertiesToSet)
- void [SetPropertiesListedInLobby](#) (string[] propsListedInLobby)

### Properties

- new int [playerCount](#) `[get]`
- new string [name](#) `[get, set]`
- new int [maxPlayers](#) `[get, set]`
- new bool [open](#) `[get, set]`
- new bool [visible](#) `[get, set]`
- string[] [propertiesListedInLobby](#) `[get, set]`
- bool [autoCleanUp](#) `[get]`

## Additional Inherited Members

### 8.21.1 Detailed Description

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

### 8.21.2 Member Function Documentation

#### 8.21.2.1 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!).

To reduce network traffic, set only values that actually changed.

New properties are added, existing values are updated. Other values will not be changed, so only provide values that changed or are new. To delete a named (custom) property of this room, use null as value. 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.

#### Parameters

<i>propertiesToSet</i>	Hashtable of props to update, set and sync. See description.
------------------------	--

#### 8.21.2.2 void Room.SetPropertiesListedInLobby ( string[] *propsListedInLobby* )

Enables you to define the properties available in the lobby if not all properties are needed to pick a room.

It makes sense to limit the amount of properties sent to users in the lobby as this improves speed and stability.

#### Parameters

<i>propsListedInLobby</i>	An array of custom room property names to forward to the lobby.
---------------------------	---

### 8.21.3 Property Documentation

#### 8.21.3.1 bool Room.autoCleanUp [get]

Gets if this room uses `autoCleanUp` to remove all (buffered) RPCs and instantiated GameObjects when a player leaves.

#### 8.21.3.2 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.

#### 8.21.3.3 new string Room.name [get], [set]

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

#### 8.21.3.4 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.

#### 8.21.3.5 new int Room.playerCount [get]

Count of players in this room.

#### 8.21.3.6 string[] Room.propertiesListedInLobby [get], [set]

A list of custom properties that should be forwarded to the lobby and listed there.

#### 8.21.3.7 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.

## 8.22 RoomInfo Class Reference

Inherited by [Room](#).

### Public Member Functions

- override bool [Equals](#) (object p)
- override int [GetHashCode](#) ()
- override string [ToString](#) ()

### Protected Attributes

- byte [maxPlayersField](#) = 0
- bool [openField](#) = true
- bool [visibleField](#) = true
- bool [autoCleanUpField](#) = [PhotonNetwork.autoCleanUpPlayerObjects](#)
- string [nameField](#)

### Properties

- bool [removedFromList](#) [get, set]
- Hashtable [customProperties](#) [get]
- string [name](#) [get]
- int [playerCount](#) [get, set]
- bool [isLocalClientInside](#) [get, set]
- byte [maxPlayers](#) [get]
- bool [open](#) [get]
- bool [visible](#) [get]

### 8.22.1 Detailed Description

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

This class resembles info about available rooms, as sent by the Master server's lobby. Consider all values as readonly. None are synced (only updated by events by server).

### 8.22.2 Member Function Documentation

#### 8.22.2.1 `override bool RoomInfo.Equals ( object p )`

Makes [RoomInfo](#) comparable (by name).

#### 8.22.2.2 `override int RoomInfo.GetHashCode ( )`

Accompanies Equals, using the name's GetHashCode as return.

Returns

#### 8.22.2.3 `override string RoomInfo.ToString ( )`

Simple printingin method.

Returns

String showing the [RoomInfo](#).

### 8.22.3 Member Data Documentation

#### 8.22.3.1 `bool RoomInfo.autoCleanUpField = PhotonNetwork.autoCleanUpPlayerObjects` [protected]

Backing field for property. False unless the GameProperty is set to true (else it's not sent).

#### 8.22.3.2 `byte RoomInfo.maxPlayersField = 0` [protected]

Backing field for property.

#### 8.22.3.3 `string RoomInfo.nameField` [protected]

Backing field for property.

#### 8.22.3.4 `bool RoomInfo.openField = true` [protected]

Backing field for property.

#### 8.22.3.5 `bool RoomInfo.visibleField = true` [protected]

Backing field for property.

## 8.22.4 Property Documentation

### 8.22.4.1 `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.

### 8.22.4.2 `bool RoomInfo.isLocalClientInside` `[get]`, `[set]`

State if the local client is already in the game or still going to join it on gameserver (in lobby always false).

### 8.22.4.3 `byte RoomInfo.maxPlayers` `[get]`

Sets a limit of players to this room. This property is shown in lobby, too. If the room is full (players count == maxplayers), joining this room will fail.

As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

### 8.22.4.4 `string RoomInfo.name` `[get]`

The name of a room. Unique identifier (per Loadbalancing group) for a room/match.

### 8.22.4.5 `bool RoomInfo.open` `[get]`

Defines if the room can be joined. This does not affect listing in a lobby but joining the room will fail if not open. If not open, the room is excluded from random matchmaking. Due to racing conditions, found matches might become closed before they are joined. Simply re-connect to master and find another. Use property "IsVisible" to not list the room.

As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

### 8.22.4.6 `int RoomInfo.playerCount` `[get]`, `[set]`

Only used internally in lobby, to display number of players in room (while you're not in).

### 8.22.4.7 `bool RoomInfo.removedFromList` `[get]`, `[set]`

Used internally in lobby, to mark rooms that are no longer listed.

### 8.22.4.8 `bool RoomInfo.visible` `[get]`

Defines if the room is listed in its lobby. Rooms can be created invisible, or changed to invisible. To change if a room can be joined, use property: open.

As part of [RoomInfo](#) this can't be set. As part of a [Room](#) (which the player joined), the setter will update the server and all clients.

## 8.23 RoomOptions Class Reference

## Public Attributes

- bool [isVisible](#) = true
- bool [isOpen](#) = true
- int [maxPlayers](#)
- bool [cleanupCacheOnLeave](#) = [PhotonNetwork.autoCleanUpPlayerObjects](#)
- Hashtable [customRoomProperties](#)
- string[] [customRoomPropertiesForLobby](#) = new string[0]

### 8.23.1 Detailed Description

Wraps up common room properties needed when you create rooms.

This directly maps to what the fields in the [Room](#) class.

### 8.23.2 Member Data Documentation

8.23.2.1 bool [RoomOptions.cleanupCacheOnLeave](#) = [PhotonNetwork.autoCleanUpPlayerObjects](#)

8.23.2.2 Hashtable [RoomOptions.customRoomProperties](#)

8.23.2.3 string [] [RoomOptions.customRoomPropertiesForLobby](#) = new string[0]

8.23.2.4 bool [RoomOptions.isOpen](#) = true

8.23.2.5 bool [RoomOptions.isVisible](#) = true

8.23.2.6 int [RoomOptions.maxPlayers](#)

## 8.24 ServerSettings Class Reference

Inherits [ScriptableObject](#).

## Public Types

- enum [HostingOption](#) { [HostingOption.NotSet](#), [HostingOption.PhotonCloud](#), [HostingOption.SelfHosted](#), [HostingOption.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 [ExtractRegionFromAddress](#) (string address)
- static string [FindServerAddressForRegion](#) (int regionIndex)
- static string [FindServerAddressForRegion](#) (CloudServerRegion regionIndex)
- static bool [TryParseCloudServerRegion](#) (string regionShortcut, out CloudServerRegion region)

## Public Attributes

- const string [DefaultCloudServerUrl](#) = "app-eu.exitgamescloud.com"
- const string [DefaultServerAddress](#) = "127.0.0.1"
- const int [DefaultMasterPort](#) = 5055
- const int [DefaultNameServerPort](#) = 5058
- const string [DefaultAppID](#) = "Master"
- [HostingOption](#) [HostType](#) = HostingOption.NotSet
- string [ServerAddress](#) = [DefaultServerAddress](#)
- int [ServerPort](#) = 5055
- string [AppID](#) = ""
- bool [PingCloudServersOnAwake](#) = false
- List< string > [RpcList](#)
- bool [DisableAutoOpenWizard](#)

## Static Public Attributes

- static readonly string[] [CloudServerRegionPrefixes](#) = new string[] { "app-eu", "app-us", "app-asia", "app-jp" }

## Properties

- string [Region](#) [get]

### 8.24.1 Detailed Description

Collection of connection-relevant settings, used internally by [PhotonNetwork.ConnectUsingSettings](#).

### 8.24.2 Member Enumeration Documentation

#### 8.24.2.1 enum ServerSettings.HostingOption

Enumerator

***NotSet***

***PhotonCloud***

***SelfHosted***

***OfflineMode***

### 8.24.3 Member Function Documentation

8.24.3.1 static string ServerSettings.ExtractRegionFromAddress ( string *address* ) [static]

8.24.3.2 static int ServerSettings.FindRegionForServerAddress ( string *server* ) [static]

8.24.3.3 static string ServerSettings.FindServerAddressForRegion ( int *regionIndex* ) [static]

8.24.3.4 static string ServerSettings.FindServerAddressForRegion ( CloudServerRegion *regionIndex* ) [static]

8.24.3.5 override string ServerSettings.ToString ( )

8.24.3.6 `static bool ServerSettings.TryParseCloudServerRegion ( string regionShortcut, out CloudServerRegion region )`  
`[static]`

Tries to convert a region shortcut to a value of CloudServerRegion enum. Defaults to CloudServerRegion.US, if that fails.

#### Returns

If conversion is successful.

8.24.3.7 `void ServerSettings.UseCloud ( string cloudAppid, int regionIndex )`

8.24.3.8 `void ServerSettings.UseMyServer ( string serverAddress, int serverPort, string application )`

### 8.24.4 Member Data Documentation

8.24.4.1 `string ServerSettings.AppID = ""`

8.24.4.2 `readonly string [] ServerSettings.CloudServerRegionPrefixes = new string[] { "app-eu", "app-us", "app-asia", "app-jp" }`  
`[static]`

8.24.4.3 `const string ServerSettings.DefaultAppID = "Master"`

8.24.4.4 `const string ServerSettings.DefaultCloudServerUrl = "app-eu.exitgamescloud.com"`

8.24.4.5 `const int ServerSettings.DefaultMasterPort = 5055`

8.24.4.6 `const int ServerSettings.DefaultNameServerPort = 5058`

8.24.4.7 `const string ServerSettings.DefaultServerAddress = "127.0.0.1"`

8.24.4.8 `bool ServerSettings.DisableAutoOpenWizard`

8.24.4.9 `HostingOption ServerSettings.HostType = HostingOption.NotSet`

8.24.4.10 `bool ServerSettings.PingCloudServersOnAwake = false`

8.24.4.11 `List<string> ServerSettings.RpcList`

8.24.4.12 `string ServerSettings.ServerAddress = DefaultServerAddress`

8.24.4.13 `int ServerSettings.ServerPort = 5055`

### 8.24.5 Property Documentation

8.24.5.1 `string ServerSettings.Region` `[get]`

## 8.25 TypedLobby Class Reference

### Public Member Functions

- [TypedLobby](#) ()
- [TypedLobby](#) (string name, LobbyType type)
- override string [ToString](#) ()



## Public Attributes

- string [Name](#)
- LobbyType [Type](#)

## Static Public Attributes

- static readonly [TypedLobby Default](#) = new [TypedLobby\(\)](#)

## Properties

- bool [IsDefault](#) [get]

### 8.25.1 Detailed Description

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

### 8.25.2 Constructor & Destructor Documentation

8.25.2.1 [TypedLobby.TypedLobby \( \)](#)

8.25.2.2 [TypedLobby.TypedLobby \( string name, LobbyType type \)](#)

### 8.25.3 Member Function Documentation

8.25.3.1 override string [TypedLobby.ToString \( \)](#)

### 8.25.4 Member Data Documentation

8.25.4.1 readonly [TypedLobby TypedLobby.Default](#) = new [TypedLobby\(\)](#) [static]

8.25.4.2 string [TypedLobby.Name](#)

8.25.4.3 LobbyType [TypedLobby.Type](#)

### 8.25.5 Property Documentation

8.25.5.1 bool [TypedLobby.IsDefault](#) [get]

# Index

- ActorList
  - ParameterCode, [39](#)
- ActorNr
  - ParameterCode, [39](#)
- ActorProperties, [27](#)
  - PlayerName, [27](#)
- Add
  - ParameterCode, [39](#)
  - PBitStream, [43](#)
- Address
  - ParameterCode, [39](#)
- All
  - Public API, [24](#)
- AllBuffered
  - Public API, [24](#)
- AllBufferedViaServer
  - Public API, [24](#)
- AllViaServer
  - Public API, [24](#)
- allProperties
  - PhotonPlayer, [72](#)
- AllocateViewID
  - PhotonNetwork, [48](#)
- AlmostEquals
  - Extensions, [33](#)
- AlreadyMatched
  - ErrorCode, [29](#)
- AppID
  - ServerSettings, [88](#)
- AppStats
  - EventCode, [31](#)
- AppVersion
  - ParameterCode, [39](#)
- ApplicationId
  - ParameterCode, [39](#)
- AuthParameters
  - AuthenticationValues, [28](#)
- AuthPostData
  - AuthenticationValues, [29](#)
- AuthType
  - AuthenticationValues, [28](#)
- AuthValues
  - PhotonNetwork, [66](#)
- Authenticate
  - OperationCode, [37](#)
- Authenticated
  - Public API, [22](#)
- Authenticating
  - Public API, [22](#)
- AuthenticationValues, [27](#)
  - AuthParameters, [28](#)
  - AuthPostData, [29](#)
  - AuthType, [28](#)
  - Secret, [28](#)
  - SetAuthParameters, [28](#)
  - SetAuthPostData, [28](#)
  - ToString, [28](#)
- autoCleanUp
  - Room, [82](#)
- autoCleanUpField
  - RoomInfo, [84](#)
- autoCleanUpPlayerObjects
  - PhotonNetwork, [66](#)
- autoJoinLobby
  - PhotonNetwork, [66](#)
- automaticallySyncScene
  - PhotonNetwork, [66](#)
- Awake
  - PhotonView, [77](#)
  - PingCloudRegions, [79](#)
- AzureNodeInfo
  - EventCode, [31](#)
- BitCount
  - PBitStream, [43](#)
- Broadcast
  - ParameterCode, [39](#)
- buttonsOn
  - PhotonStatsGui, [73](#)
- ByteCount
  - PBitStream, [43](#)
- BytesForBits
  - PBitStream, [43](#)
- Cache
  - ParameterCode, [39](#)
- CacheSliceIndex
  - RaiseEventOptions, [80](#)
- CachingOption
  - RaiseEventOptions, [80](#)
- ChangeGroups
  - OperationCode, [37](#)
- CleanupCacheOnLeave
  - GameProperties, [35](#)
  - ParameterCode, [40](#)
- cleanupCacheOnLeave
  - RoomOptions, [86](#)
- ClientAuthenticationData
  - ParameterCode, [40](#)

- ClientAuthenticationParams
  - ParameterCode, [40](#)
- ClientAuthenticationType
  - ParameterCode, [40](#)
- CloseConnection
  - PhotonNetwork, [48](#)
- ClosestRegion
  - PingCloudRegions, [80](#)
- ClosestRegionAvailable
  - PingCloudRegions, [80](#)
- CloudServerRegionPrefixes
  - ServerSettings, [88](#)
- Code
  - ParameterCode, [40](#)
- ConnectToBestCloudServer
  - PhotonNetwork, [48](#)
- ConnectToBestRegion
  - PingCloudRegions, [79](#)
- ConnectToMaster
  - PhotonNetwork, [49](#)
- ConnectUsingSettings
  - PhotonNetwork, [49](#)
- connected
  - PhotonNetwork, [66](#)
- ConnectedToGameserver
  - Public API, [22](#)
- ConnectedToMaster
  - Public API, [22](#)
- ConnectedToNameServer
  - Public API, [22](#)
- connectedAndReady
  - PhotonNetwork, [66](#)
- connecting
  - PhotonNetwork, [67](#)
- ConnectingToGameserver
  - Public API, [22](#)
- ConnectingToMasterserver
  - Public API, [22](#)
- ConnectingToNameServer
  - Public API, [22](#)
- connectionState
  - PhotonNetwork, [67](#)
- connectionStateDetailed
  - PhotonNetwork, [67](#)
- Contains
  - Extensions, [33](#)
- Count
  - PhotonStream, [76](#)
- countOfPlayers
  - PhotonNetwork, [67](#)
- countOfPlayersInRooms
  - PhotonNetwork, [67](#)
- countOfPlayersOnMaster
  - PhotonNetwork, [67](#)
- countOfRooms
  - PhotonNetwork, [67](#)
- CrcCheckEnabled
  - PhotonNetwork, [67](#)
- CreateGame
  - OperationCode, [37](#)
- CreateIfNotExists
  - ParameterCode, [40](#)
- CreateRoom
  - PhotonNetwork, [50](#), [51](#)
- CustomAuthenticationFailed
  - ErrorCode, [29](#)
- CustomEventContent
  - ParameterCode, [40](#)
- customProperties
  - PhotonPlayer, [72](#)
  - RoomInfo, [85](#)
- customRoomProperties
  - RoomOptions, [86](#)
- customRoomPropertiesForLobby
  - RoomOptions, [86](#)
- Data
  - ParameterCode, [40](#)
- Default
  - RaiseEventOptions, [80](#)
  - TypedLobby, [89](#)
- DefaultAppID
  - ServerSettings, [88](#)
- DefaultCloudServerUrl
  - ServerSettings, [88](#)
- DefaultMasterPort
  - ServerSettings, [88](#)
- DefaultNameServerPort
  - ServerSettings, [88](#)
- DefaultServerAddress
  - ServerSettings, [88](#)
- DeleteRegionPref
  - PingCloudRegions, [79](#)
- Destroy
  - PhotonNetwork, [52](#)
- DestroyAll
  - PhotonNetwork, [53](#)
- DestroyPlayerObjects
  - PhotonNetwork, [53](#)
- DisableAutoOpenWizard
  - ServerSettings, [88](#)
- Disconnect
  - PhotonNetwork, [54](#)
- DisconnectByClientTimeout
  - Public API, [21](#)
- DisconnectByServer
  - Public API, [21](#)
- DisconnectByServerLogic
  - Public API, [21](#)
- DisconnectByServerTimeout
  - Public API, [21](#)
- DisconnectByServerUserLimit
  - Public API, [21](#)
- DisconnectCause
  - Public API, [21](#)
- Disconnected
  - Public API, [22](#)

- Disconnecting
  - Public API, [22](#)
- DisconnectingFromGameserver
  - Public API, [22](#)
- DisconnectingFromMasterserver
  - Public API, [22](#)
- DisconnectingFromNameServer
  - Public API, [22](#)
- Equals
  - PhotonPlayer, [71](#)
  - RoomInfo, [84](#)
- ErrorCode, [29](#)
  - AlreadyMatched, [29](#)
  - CustomAuthenticationFailed, [29](#)
  - GameClosed, [29](#)
  - GameDoesNotExist, [29](#)
  - GameFull, [30](#)
  - GameIdAlreadyExists, [30](#)
  - InternalServerError, [30](#)
  - InvalidAuthentication, [30](#)
  - InvalidOperationCode, [30](#)
  - InvalidRegion, [30](#)
  - MaxCcuReached, [30](#)
  - NoRandomMatchFound, [30](#)
  - Ok, [31](#)
  - OperationNotAllowedInCurrentState, [31](#)
  - ServerFull, [31](#)
  - UserBlocked, [31](#)
- ErrorsOnly
  - Public API, [22](#)
- EventCallback
  - PhotonNetwork, [54](#)
- EventCode, [31](#)
  - AppStats, [31](#)
  - AzureNodeInfo, [31](#)
  - GameList, [32](#)
  - GameListUpdate, [32](#)
  - Join, [32](#)
  - Leave, [32](#)
  - Match, [32](#)
  - PropertiesChanged, [32](#)
  - QueueState, [32](#)
  - SetProperties, [32](#)
- EventForward
  - ParameterCode, [40](#)
- Exception
  - Public API, [21](#)
- ExceptionOnConnect
  - Public API, [21](#)
- ExecuteOnSerialize
  - PhotonView, [77](#)
- Extensions, [32](#)
  - AlmostEquals, [33](#)
  - Contains, [33](#)
  - GetPhotonView, [33](#)
  - GetPhotonViewsInChildren, [33](#)
  - Merge, [33](#)
  - MergeStringKeys, [34](#)
  - StripKeysWithNullValues, [34](#)
  - StripToStringKeys, [34](#)
  - ToStringFull, [34](#)
- ExtractRegionFromAddress
  - ServerSettings, [87](#)
- FetchServerTimestamp
  - PhotonNetwork, [54](#)
- Find
  - PhotonPlayer, [71](#)
  - PhotonView, [77](#)
- FindFriends
  - OperationCode, [37](#)
  - PhotonNetwork, [54](#)
- FindFriendsRequestList
  - ParameterCode, [40](#)
- FindFriendsResponseOnlineList
  - ParameterCode, [40](#)
- FindFriendsResponseRoomIdList
  - ParameterCode, [41](#)
- FindRegionForServerAddress
  - ServerSettings, [87](#)
- FindServerAddressForRegion
  - ServerSettings, [87](#)
- ForwardToWebhook
  - RaiseEventOptions, [81](#)
- FriendInfo, [34](#)
  - IsInRoom, [35](#)
  - IsOnline, [35](#)
  - Name, [35](#)
  - Room, [35](#)
  - ToString, [35](#)
- Friends
  - PhotonNetwork, [67](#)
- FriendsListAge
  - PhotonNetwork, [67](#)
- Full
  - Public API, [22](#)
- GameClosed
  - ErrorCode, [29](#)
- GameCount
  - ParameterCode, [41](#)
- GameDoesNotExist
  - ErrorCode, [29](#)
- GameFull
  - ErrorCode, [30](#)
- GameIdAlreadyExists
  - ErrorCode, [30](#)
- GameList
  - EventCode, [32](#)
  - ParameterCode, [41](#)
- GameListUpdate
  - EventCode, [32](#)
- GameProperties, [35](#)
  - CleanupCacheOnLeave, [35](#)
  - IsOpen, [35](#)
  - IsVisible, [35](#)
  - MaxPlayers, [36](#)

- ParameterCode, [41](#)
- PlayerCount, [36](#)
- PropsListedInLobby, [36](#)
- Removed, [36](#)
- Get
  - PBitStream, [43](#)
  - PhotonView, [77](#)
- GetHashCode
  - PhotonPlayer, [71](#)
  - RoomInfo, [84](#)
- GetNext
  - PBitStream, [43](#)
- GetPhotonView
  - Extensions, [33](#)
- GetPhotonViewsInChildren
  - Extensions, [33](#)
- GetPing
  - PhotonNetwork, [54](#)
- GetProperties
  - OperationCode, [37](#)
- GetRegions
  - OperationCode, [37](#)
- GetRoomList
  - PhotonNetwork, [55](#)
- Group
  - ParameterCode, [41](#)
- group
  - PhotonView, [78](#)
- healthStatsVisible
  - PhotonStatsGui, [73](#)
- HostType
  - ServerSettings, [88](#)
- HostingOption
  - ServerSettings, [87](#)
- ID
  - PhotonPlayer, [72](#)
- inRoom
  - PhotonNetwork, [68](#)
- Informational
  - Public API, [22](#)
- InitializeSecurity
  - PhotonNetwork, [55](#)
- insideLobby
  - PhotonNetwork, [68](#)
- Instance
  - PingCloudRegions, [80](#)
- Instantiate
  - PhotonNetwork, [55](#)
- InstantiateInRoomOnly
  - PhotonNetwork, [64](#)
- InstantiateSceneObject
  - PhotonNetwork, [56](#)
- instantiationData
  - PhotonView, [78](#)
- instantiationId
  - PhotonView, [78](#)
- InterestGroup
  - RaiseEventOptions, [81](#)
- InternalReceiveException
  - Public API, [21](#)
- InternalCleanPhotonMonoFromScenelfStuck
  - PhotonNetwork, [56](#)
- InternalServerError
  - ErrorCode, [30](#)
- InvalidAuthentication
  - Public API, [21](#)
- InvalidRegion
  - Public API, [21](#)
- InvalidAuthentication
  - ErrorCode, [30](#)
- InvalidOperationCode
  - ErrorCode, [30](#)
- InvalidRegion
  - ErrorCode, [30](#)
- IsDefault
  - TypedLobby, [89](#)
- IsInRoom
  - FriendInfo, [35](#)
- isLocal
  - PhotonPlayer, [72](#)
- isLocalClientInside
  - RoomInfo, [85](#)
- isMasterClient
  - PhotonNetwork, [68](#)
  - PhotonPlayer, [72](#)
- isMessageQueueRunning
  - PhotonNetwork, [68](#)
- isMine
  - PhotonView, [78](#)
- isNonMasterClientInRoom
  - PhotonNetwork, [68](#)
- IsOnline
  - FriendInfo, [35](#)
- IsOpen
  - GameProperties, [35](#)
- isOpen
  - RoomOptions, [86](#)
- isReading
  - PhotonStream, [76](#)
- isSceneView
  - PhotonView, [78](#)
- IsVisible
  - GameProperties, [35](#)
- isVisible
  - RoomOptions, [86](#)
- isWriting
  - PhotonStream, [76](#)
- Join
  - EventCode, [32](#)
- JoinGame
  - OperationCode, [37](#)
- JoinLobby
  - OperationCode, [37](#)
  - PhotonNetwork, [56](#)
- JoinOrCreateRoom

- PhotonNetwork, 57
- JoinRandomGame
  - OperationCode, 37
- JoinRandomRoom
  - PhotonNetwork, 57, 58
- JoinRoom
  - PhotonNetwork, 59
- Joined
  - Public API, 22
- JoinedLobby
  - Public API, 22
- Joining
  - Public API, 22
- Leave
  - EventCode, 32
  - OperationCode, 38
- LeaveLobby
  - OperationCode, 38
  - PhotonNetwork, 59
- LeaveRoom
  - PhotonNetwork, 59
- Leaving
  - Public API, 22
- LoadLevel
  - PhotonNetwork, 60
- lobby
  - PhotonNetwork, 68
- LobbyName
  - ParameterCode, 41
- LobbyType
  - ParameterCode, 41
- logLevel
  - PhotonNetwork, 64
- MAX\_VIEW\_IDS
  - PhotonNetwork, 64
- MasterClient
  - Public API, 24
- masterClient
  - PhotonNetwork, 68
- MasterPeerCount
  - ParameterCode, 41
- Match
  - EventCode, 32
- MatchMakingType
  - ParameterCode, 41
- MaxCcuReached
  - Public API, 21
- MaxCcuReached
  - ErrorCode, 30
- maxConnections
  - PhotonNetwork, 68
- MaxPlayers
  - GameProperties, 36
- maxPlayers
  - Room, 82
  - RoomInfo, 85
  - RoomOptions, 86
- maxPlayersField
  - RoomInfo, 84
- MaxResendsBeforeDisconnect
  - PhotonNetwork, 68
- Merge
  - Extensions, 33
- MergeStringKeys
  - Extensions, 34
- Name
  - FriendInfo, 35
  - TypedLobby, 89
- name
  - PhotonPlayer, 72
  - Room, 82
  - RoomInfo, 85
- nameField
  - RoomInfo, 84
- NetworkStatisticsEnabled
  - PhotonNetwork, 69
- NetworkStatisticsReset
  - PhotonNetwork, 60
- NetworkStatisticsToString
  - PhotonNetwork, 60
- networkView
  - Photon::MonoBehaviour, 36
- NoRandomMatchFound
  - ErrorCode, 30
- NotSet
  - ServerSettings, 87
- observed
  - PhotonView, 78
- OfflineMode
  - ServerSettings, 87
- offlineMode
  - PhotonNetwork, 69
- Ok
  - ErrorCode, 31
- OnConnectedToMaster
  - Public API, 23
- OnConnectedToPhoton
  - Public API, 23
- OnConnectionFail
  - Public API, 23
- OnCreatedRoom
  - Public API, 23
- OnCustomAuthenticationFailed
  - Public API, 24
- OnDisconnectedFromPhoton
  - Public API, 23
- OnFailedToConnectToPhoton
  - Public API, 23
- OnJoinedLobby
  - Public API, 23
- OnJoinedRoom
  - Public API, 23
- OnLeftLobby
  - Public API, 23

- OnLeftRoom
  - Public API, [23](#)
- OnMasterClientSwitched
  - Public API, [23](#)
- OnPhotonCreateRoomFailed
  - Public API, [23](#)
- OnPhotonCustomRoomPropertiesChanged
  - Public API, [24](#)
- OnPhotonInstantiate
  - Public API, [23](#)
- OnPhotonJoinRoomFailed
  - Public API, [23](#)
- OnPhotonMaxCccuReached
  - Public API, [24](#)
- OnPhotonPlayerConnected
  - Public API, [23](#)
- OnPhotonPlayerDisconnected
  - Public API, [23](#)
- OnPhotonPlayerPropertiesChanged
  - Public API, [24](#)
- OnPhotonRandomJoinFailed
  - Public API, [23](#)
- OnPhotonSerializeView
  - Public API, [23](#)
- OnReceivedRoomListUpdate
  - Public API, [23](#)
- OnUpdatedFriendList
  - Public API, [24](#)
- OnApplicationQuit
  - PhotonView, [77](#)
- OnDestroy
  - PhotonView, [77](#)
- OnEventCall
  - PhotonNetwork, [64](#)
- OnGUI
  - PhotonLagSimulationGui, [44](#)
  - PhotonStatsGui, [73](#)
- onSerializeRigidBodyOption
  - PhotonView, [78](#)
- onSerializeTransformOption
  - PhotonView, [78](#)
- open
  - Room, [82](#)
  - RoomInfo, [85](#)
- openField
  - RoomInfo, [84](#)
- OperationCode, [36](#)
  - Authenticate, [37](#)
  - ChangeGroups, [37](#)
  - CreateGame, [37](#)
  - FindFriends, [37](#)
  - GetProperties, [37](#)
  - GetRegions, [37](#)
  - JoinGame, [37](#)
  - JoinLobby, [37](#)
  - JoinRandomGame, [37](#)
  - Leave, [38](#)
  - LeaveLobby, [38](#)
  - RaiseEvent, [38](#)
  - SetProperties, [38](#)
- OperationNotAllowedInCurrentState
  - ErrorCode, [31](#)
- Optional Gui Elements, [19](#)
- otherPlayers
  - PhotonNetwork, [69](#)
- Others
  - Public API, [24](#)
- OthersBuffered
  - Public API, [24](#)
- OverrideBestCloudServer
  - PhotonNetwork, [60](#)
- OverrideRegion
  - PingCloudRegions, [79](#)
- owner
  - PhotonView, [78](#)
- OwnerActorNr
  - PhotonView, [78](#)
- ownerId
  - PhotonView, [78](#)
- PBitStream, [43](#)
  - Add, [43](#)
  - BitCount, [43](#)
  - ByteCount, [43](#)
  - BytesForBits, [43](#)
  - Get, [43](#)
  - GetNext, [43](#)
  - PBitStream, [43](#)
  - PBitStream, [43](#)
  - Position, [43](#)
  - Set, [43](#)
  - ToBytes, [43](#)
- PacketLossByCrcCheck
  - PhotonNetwork, [69](#)
- ParameterCode, [38](#)
  - ActorList, [39](#)
  - ActorNr, [39](#)
  - Add, [39](#)
  - Address, [39](#)
  - AppVersion, [39](#)
  - ApplicationId, [39](#)
  - Broadcast, [39](#)
  - Cache, [39](#)
  - CleanupCacheOnLeave, [40](#)
  - ClientAuthenticationData, [40](#)
  - ClientAuthenticationParams, [40](#)
  - ClientAuthenticationType, [40](#)
  - Code, [40](#)
  - CreateIfNotExists, [40](#)
  - CustomEventContent, [40](#)
  - Data, [40](#)
  - EventForward, [40](#)
  - FindFriendsRequestList, [40](#)
  - FindFriendsResponseOnlineList, [40](#)
  - FindFriendsResponseRoomIdList, [41](#)
  - GameCount, [41](#)
  - GameList, [41](#)

- GameProperties, 41
- Group, 41
- LobbyName, 41
- LobbyType, 41
- MasterPeerCount, 41
- MatchMakingType, 41
- PeerCount, 41
- PlayerProperties, 41
- Position, 42
- Properties, 42
- ReceiverGroup, 42
- Region, 42
- Remove, 42
- RoomName, 42
- Secret, 42
- SuppressRoomEvents, 42
- TargetActorNr, 42
- UserId, 42
- Peer
  - PhotonLagSimulationGui, 44
- PeerCreated
  - Public API, 22
- PeerCount
  - ParameterCode, 41
- PeerState
  - Public API, 21
- Photon, 25
- Photon.MonoBehaviour, 36
- PhotonCloud
  - ServerSettings, 87
- Photon::MonoBehaviour
  - networkView, 36
  - photonView, 36
- PhotonLagSimulationGui, 44
  - OnGUI, 44
  - Peer, 44
  - Start, 44
  - Visible, 44
  - WindowId, 44
  - WindowRect, 44
- PhotonLogLevel
  - Public API, 22
- PhotonMessageInfo, 45
  - PhotonMessageInfo, 45
  - photonView, 45
  - PhotonMessageInfo, 45
  - sender, 45
  - timestamp, 45
  - ToString, 45
- PhotonNetwork, 45
  - AllocateViewID, 48
  - AuthValues, 66
  - autoCleanUpPlayerObjects, 66
  - autoJoinLobby, 66
  - automaticallySyncScene, 66
  - CloseConnection, 48
  - ConnectToBestCloudServer, 48
  - ConnectToMaster, 49
  - ConnectUsingSettings, 49
  - connected, 66
  - connectedAndReady, 66
  - connecting, 67
  - connectionState, 67
  - connectionStateDetailed, 67
  - countOfPlayers, 67
  - countOfPlayersInRooms, 67
  - countOfPlayersOnMaster, 67
  - countOfRooms, 67
  - CrcCheckEnabled, 67
  - CreateRoom, 50, 51
  - Destroy, 52
  - DestroyAll, 53
  - DestroyPlayerObjects, 53
  - Disconnect, 54
  - EventCallback, 54
  - FetchServerTimestamp, 54
  - FindFriends, 54
  - Friends, 67
  - FriendsListAge, 67
  - GetPing, 54
  - GetRoomList, 55
  - inRoom, 68
  - InitializeSecurity, 55
  - insideLobby, 68
  - Instantiate, 55
  - InstantiateInRoomOnly, 64
  - InstantiateSceneObject, 56
  - InternalCleanPhotonMonoFromScenelfStuck, 56
  - isMasterClient, 68
  - isMessageQueueRunning, 68
  - isNonMasterClientInRoom, 68
  - JoinLobby, 56
  - JoinOrCreateRoom, 57
  - JoinRandomRoom, 57, 58
  - JoinRoom, 59
  - LeaveLobby, 59
  - LeaveRoom, 59
  - LoadLevel, 60
  - lobby, 68
  - logLevel, 64
  - MAX\_VIEW\_IDS, 64
  - masterClient, 68
  - maxConnections, 68
  - MaxResendsBeforeDisconnect, 68
  - NetworkStatisticsEnabled, 69
  - NetworkStatisticsReset, 60
  - NetworkStatisticsToString, 60
  - offlineMode, 69
  - OnEventCall, 64
  - otherPlayers, 69
  - OverrideBestCloudServer, 60
  - PacketLossByCrcCheck, 69
  - PhotonServerSettings, 64
  - player, 69
  - playerList, 69
  - playerName, 69



- precisionForFloatSynchronization, [64](#)
- precisionForQuaternionSynchronization, [64](#)
- precisionForVectorSynchronization, [65](#)
- PrefabCache, [65](#)
- RaiseEvent, [60](#)
- RefreshCloudServerRating, [61](#)
- RemoveRPCs, [61](#), [62](#)
- RemoveRPCsInGroup, [62](#)
- ResentReliableCommands, [69](#)
- room, [69](#)
- SendMonoMessageTargets, [65](#)
- SendOutgoingCommands, [62](#)
- sendRate, [69](#)
- sendRateOnSerialize, [70](#)
- Server, [70](#)
- ServerAddress, [70](#)
- serverSettingsAssetFile, [65](#)
- serverSettingsAssetPath, [65](#)
- SetLevelPrefix, [62](#)
- SetMasterClient, [63](#)
- SetPlayerCustomProperties, [63](#)
- SetReceivingEnabled, [63](#)
- SetSendingEnabled, [64](#)
- time, [70](#)
- UnAllocateViewID, [64](#)
- unreliableCommandsLimit, [70](#)
- UseNameServer, [65](#)
- UsePrefabCache, [65](#)
- versionPUN, [65](#)
- PhotonNetworkingMessage
  - Public API, [22](#)
- PhotonPlayer, [70](#)
  - allProperties, [72](#)
  - customProperties, [72](#)
  - Equals, [71](#)
  - Find, [71](#)
  - GetHashCode, [71](#)
  - ID, [72](#)
  - isLocal, [72](#)
  - isMasterClient, [72](#)
  - name, [72](#)
  - PhotonPlayer, [71](#)
  - PhotonPlayer, [71](#)
  - SetCustomProperties, [72](#)
  - ToString, [72](#)
- PhotonServerSettings
  - PhotonNetwork, [64](#)
- PhotonStatsGui, [73](#)
  - buttonsOn, [73](#)
  - healthStatsVisible, [73](#)
  - OnGUI, [73](#)
  - Start, [73](#)
  - statsOn, [74](#)
  - statsRect, [74](#)
  - statsWindowOn, [74](#)
  - trafficStatsOn, [74](#)
  - TrafficStatsWindow, [73](#)
  - Update, [73](#)
- WindowId, [74](#)
- PhotonStream, [74](#)
  - Count, [76](#)
  - isReading, [76](#)
  - isWriting, [76](#)
  - PhotonStream, [75](#)
  - PhotonStream, [75](#)
  - ReceiveNext, [75](#)
  - SendNext, [75](#)
  - Serialize, [75](#), [76](#)
  - ToArray, [76](#)
- PhotonTargets
  - Public API, [24](#)
- PhotonView, [76](#)
  - Awake, [77](#)
  - ExecuteOnSerialize, [77](#)
  - Find, [77](#)
  - Get, [77](#)
  - group, [78](#)
  - instantiationData, [78](#)
  - instantiationId, [78](#)
  - isMine, [78](#)
  - isSceneView, [78](#)
  - observed, [78](#)
  - OnApplicationQuit, [77](#)
  - OnDestroy, [77](#)
  - onSerializeRigidBodyOption, [78](#)
  - onSerializeTransformOption, [78](#)
  - owner, [78](#)
  - OwnerActorNr, [78](#)
  - ownerId, [78](#)
  - prefix, [78](#)
  - prefixBackup, [78](#)
  - RPC, [78](#)
  - subId, [78](#)
  - synchronization, [78](#)
  - ToString, [78](#)
  - viewID, [78](#)
- photonView
  - Photon::MonoBehaviour, [36](#)
  - PhotonMessageInfo, [45](#)
- PingCloudRegions, [79](#)
  - Awake, [79](#)
  - ClosestRegion, [80](#)
  - ClosestRegionAvailable, [80](#)
  - ConnectToBestRegion, [79](#)
  - DeleteRegionPref, [79](#)
  - Instance, [80](#)
  - OverrideRegion, [79](#)
  - RefreshCloudServerRating, [79](#)
  - ResolveHost, [79](#)
- PingCloudServersOnAwake
  - ServerSettings, [88](#)
- player
  - PhotonNetwork, [69](#)
- PlayerCount
  - GameProperties, [36](#)
- playerCount

- Room, [83](#)
- RoomInfo, [85](#)
- playerList
  - PhotonNetwork, [69](#)
- PlayerName
  - ActorProperties, [27](#)
- playerName
  - PhotonNetwork, [69](#)
- PlayerProperties
  - ParameterCode, [41](#)
- Position
  - ParameterCode, [42](#)
  - PBitStream, [43](#)
- precisionForFloatSynchronization
  - PhotonNetwork, [64](#)
- precisionForQuaternionSynchronization
  - PhotonNetwork, [64](#)
- precisionForVectorSynchronization
  - PhotonNetwork, [65](#)
- PrefabCache
  - PhotonNetwork, [65](#)
- prefix
  - PhotonView, [78](#)
- prefixBackup
  - PhotonView, [78](#)
- Properties
  - ParameterCode, [42](#)
- PropertiesChanged
  - EventCode, [32](#)
- propertiesListedInLobby
  - Room, [83](#)
- PropsListedInLobby
  - GameProperties, [36](#)
- Public API
  - All, [24](#)
  - AllBuffered, [24](#)
  - AllBufferedViaServer, [24](#)
  - AllViaServer, [24](#)
  - Authenticated, [22](#)
  - Authenticating, [22](#)
  - ConnectedToGameserver, [22](#)
  - ConnectedToMaster, [22](#)
  - ConnectedToNameServer, [22](#)
  - ConnectingToGameserver, [22](#)
  - ConnectingToMasterserver, [22](#)
  - ConnectingToNameServer, [22](#)
  - DisconnectByClientTimeout, [21](#)
  - DisconnectByServer, [21](#)
  - DisconnectByServerLogic, [21](#)
  - DisconnectByServerTimeout, [21](#)
  - DisconnectByServerUserLimit, [21](#)
  - Disconnected, [22](#)
  - Disconnecting, [22](#)
  - DisconnectingFromGameserver, [22](#)
  - DisconnectingFromMasterserver, [22](#)
  - DisconnectingFromNameServer, [22](#)
  - ErrorsOnly, [22](#)
  - Exception, [21](#)
  - ExceptionOnConnect, [21](#)
  - Full, [22](#)
  - Informational, [22](#)
  - InternalReceiveException, [21](#)
  - InvalidAuthentication, [21](#)
  - InvalidRegion, [21](#)
  - Joined, [22](#)
  - JoinedLobby, [22](#)
  - Joining, [22](#)
  - Leaving, [22](#)
  - MasterClient, [24](#)
  - MaxCcuReached, [21](#)
  - OnConnectedToMaster, [23](#)
  - OnConnectedToPhoton, [23](#)
  - OnConnectionFail, [23](#)
  - OnCreatedRoom, [23](#)
  - OnCustomAuthenticationFailed, [24](#)
  - OnDisconnectedFromPhoton, [23](#)
  - OnFailedToConnectToPhoton, [23](#)
  - OnJoinedLobby, [23](#)
  - OnJoinedRoom, [23](#)
  - OnLeftLobby, [23](#)
  - OnLeftRoom, [23](#)
  - OnMasterClientSwitched, [23](#)
  - OnPhotonCreateRoomFailed, [23](#)
  - OnPhotonCustomRoomPropertiesChanged, [24](#)
  - OnPhotonInstantiate, [23](#)
  - OnPhotonJoinRoomFailed, [23](#)
  - OnPhotonMaxCccuReached, [24](#)
  - OnPhotonPlayerConnected, [23](#)
  - OnPhotonPlayerDisconnected, [23](#)
  - OnPhotonPlayerPropertiesChanged, [24](#)
  - OnPhotonRandomJoinFailed, [23](#)
  - OnPhotonSerializeView, [23](#)
  - OnReceivedRoomListUpdate, [23](#)
  - OnUpdatedFriendList, [24](#)
  - Others, [24](#)
  - OthersBuffered, [24](#)
  - PeerCreated, [22](#)
  - Queued, [22](#)
  - QueuedComingFromGameserver, [22](#)
  - SecurityExceptionOnConnect, [21](#)
  - TimeoutDisconnect, [21](#)
  - Uninitialized, [22](#)
- Public API, [20](#)
  - DisconnectCause, [21](#)
  - PeerState, [21](#)
  - PhotonLogLevel, [22](#)
  - PhotonNetworkingMessage, [22](#)
  - PhotonTargets, [24](#)
- QueueState
  - EventCode, [32](#)
- Queued
  - Public API, [22](#)
- QueuedComingFromGameserver
  - Public API, [22](#)
- RPC

- PhotonView, 78
- RaiseEvent
  - OperationCode, 38
  - PhotonNetwork, 60
- RaiseEventOptions, 80
  - CacheSliceIndex, 80
  - CachingOption, 80
  - Default, 80
  - ForwardToWebhook, 81
  - InterestGroup, 81
  - Receivers, 81
  - SequenceChannel, 81
  - TargetActors, 81
- ReceiveNext
  - PhotonStream, 75
- ReceiverGroup
  - ParameterCode, 42
- Receivers
  - RaiseEventOptions, 81
- RefreshCloudServerRating
  - PhotonNetwork, 61
  - PingCloudRegions, 79
- Region
  - ParameterCode, 42
  - ServerSettings, 88
- Remove
  - ParameterCode, 42
- RemoveRPCs
  - PhotonNetwork, 61, 62
- RemoveRPCsInGroup
  - PhotonNetwork, 62
- Removed
  - GameProperties, 36
- removedFromList
  - RoomInfo, 85
- ResentReliableCommands
  - PhotonNetwork, 69
- ResolveHost
  - PingCloudRegions, 79
- Room, 81
  - autoCleanUp, 82
  - FriendInfo, 35
  - maxPlayers, 82
  - name, 82
  - open, 82
  - playerCount, 83
  - propertiesListedInLobby, 83
  - SetCustomProperties, 82
  - SetPropertiesListedInLobby, 82
  - visible, 83
- room
  - PhotonNetwork, 69
- RoomInfo, 83
  - autoCleanUpField, 84
  - customProperties, 85
  - Equals, 84
  - GetHashCode, 84
  - isLocalClientInside, 85
  - maxPlayers, 85
  - maxPlayersField, 84
  - name, 85
  - nameField, 84
  - open, 85
  - openField, 84
  - playerCount, 85
  - removedFromList, 85
  - ToString, 84
  - visible, 85
  - visibleField, 84
- RoomName
  - ParameterCode, 42
- RoomOptions, 85
  - cleanupCacheOnLeave, 86
  - customRoomProperties, 86
  - customRoomPropertiesForLobby, 86
  - isOpen, 86
  - isVisible, 86
  - maxPlayers, 86
- RpcList
  - ServerSettings, 88
- Secret
  - AuthenticationValues, 28
  - ParameterCode, 42
- SecurityExceptionOnConnect
  - Public API, 21
- SelfHosted
  - ServerSettings, 87
- SendMonoMessageTargets
  - PhotonNetwork, 65
- SendNext
  - PhotonStream, 75
- SendOutgoingCommands
  - PhotonNetwork, 62
- sendRate
  - PhotonNetwork, 69
- sendRateOnSerialize
  - PhotonNetwork, 70
- sender
  - PhotonMessageInfo, 45
- SequenceChannel
  - RaiseEventOptions, 81
- Serialize
  - PhotonStream, 75, 76
- Server
  - PhotonNetwork, 70
- ServerSettings
  - NotSet, 87
  - OfflineMode, 87
  - PhotonCloud, 87
  - SelfHosted, 87
- ServerAddress
  - PhotonNetwork, 70
  - ServerSettings, 88
- ServerFull
  - ErrorCode, 31
- ServerPort

- ServerSettings, 88
- ServerSettings, 86
  - AppID, 88
  - CloudServerRegionPrefixes, 88
  - DefaultAppID, 88
  - DefaultCloudServerUrl, 88
  - DefaultMasterPort, 88
  - DefaultNameServerPort, 88
  - DefaultServerAddress, 88
  - DisableAutoOpenWizard, 88
  - ExtractRegionFromAddress, 87
  - FindRegionForServerAddress, 87
  - FindServerAddressForRegion, 87
  - HostType, 88
  - HostingOption, 87
  - PingCloudServersOnAwake, 88
  - Region, 88
  - RpcList, 88
  - ServerAddress, 88
  - ServerPort, 88
  - ToString, 87
  - TryParseCloudServerRegion, 87
  - UseCloud, 88
  - UseMyServer, 88
- serverSettingsAssetFile
  - PhotonNetwork, 65
- serverSettingsAssetPath
  - PhotonNetwork, 65
- Set
  - PBitStream, 43
- SetAuthParameters
  - AuthenticationValues, 28
- SetAuthPostData
  - AuthenticationValues, 28
- SetCustomProperties
  - PhotonPlayer, 72
  - Room, 82
- SetLevelPrefix
  - PhotonNetwork, 62
- SetMasterClient
  - PhotonNetwork, 63
- SetPlayerCustomProperties
  - PhotonNetwork, 63
- SetProperties
  - EventCode, 32
  - OperationCode, 38
- SetPropertiesListedInLobby
  - Room, 82
- SetReceivingEnabled
  - PhotonNetwork, 63
- SetSendingEnabled
  - PhotonNetwork, 64
- Start
  - PhotonLagSimulationGui, 44
  - PhotonStatsGui, 73
- statsOn
  - PhotonStatsGui, 74
- statsRect
  - PhotonStatsGui, 74
- statsWindowOn
  - PhotonStatsGui, 74
- StripKeysWithNullValues
  - Extensions, 34
- StripToStringKeys
  - Extensions, 34
- subId
  - PhotonView, 78
- SuppressRoomEvents
  - ParameterCode, 42
- synchronization
  - PhotonView, 78
- TargetActorNr
  - ParameterCode, 42
- TargetActors
  - RaiseEventOptions, 81
- time
  - PhotonNetwork, 70
- TimeoutDisconnect
  - Public API, 21
- timestamp
  - PhotonMessageInfo, 45
- ToArray
  - PhotonStream, 76
- ToBytes
  - PBitStream, 43
- ToString
  - AuthenticationValues, 28
  - FriendInfo, 35
  - PhotonMessageInfo, 45
  - PhotonPlayer, 72
  - PhotonView, 78
  - RoomInfo, 84
  - ServerSettings, 87
  - TypedLobby, 89
- ToStringFull
  - Extensions, 34
- trafficStatsOn
  - PhotonStatsGui, 74
- TrafficStatsWindow
  - PhotonStatsGui, 73
- TryParseCloudServerRegion
  - ServerSettings, 87
- Type
  - TypedLobby, 89
- TypedLobby, 88
  - Default, 89
  - IsDefault, 89
  - Name, 89
  - ToString, 89
  - Type, 89
  - TypedLobby, 89
  - TypedLobby, 89
- UnAllocateViewID
  - PhotonNetwork, 64
- Uninitialized

- Public API, [22](#)
- unreliableCommandsLimit
  - PhotonNetwork, [70](#)
- Update
  - PhotonStatsGui, [73](#)
- UseCloud
  - ServerSettings, [88](#)
- UseMyServer
  - ServerSettings, [88](#)
- UseNameServer
  - PhotonNetwork, [65](#)
- UsePrefabCache
  - PhotonNetwork, [65](#)
- UserBlocked
  - ErrorCode, [31](#)
- UserId
  - ParameterCode, [42](#)
- versionPUN
  - PhotonNetwork, [65](#)
- viewID
  - PhotonView, [78](#)
- Visible
  - PhotonLagSimulationGui, [44](#)
- visible
  - Room, [83](#)
  - RoomInfo, [85](#)
- visibleField
  - RoomInfo, [84](#)
- WindowId
  - PhotonLagSimulationGui, [44](#)
  - PhotonStatsGui, [74](#)
- WindowRect
  - PhotonLagSimulationGui, [44](#)