

# Photon DotNet Client Reference

for Windows, Unity3D, PSM, Xamarin, Windows 8  
Store and Phone SDKs

# Table of Contents

<b>Overview</b>	<b>1</b>
<b>Photon Workflow</b>	<b>1</b>
<b>Operations</b>	<b>2</b>
<b>Events</b>	<b>3</b>
<b>Fragmentation and Channels</b>	<b>3</b>
<b>Using TCP</b>	<b>4</b>
<b>Network Simulation</b>	<b>5</b>
<b>Serializable Datatypes</b>	<b>6</b>
<b>The Photon Server</b>	<b>7</b>
<b>Lite Application</b>	<b>7</b>
Properties on Photon	7
<b>Custom Authentication</b>	<b>8</b>
<b>Further Help</b>	<b>9</b>
<b>Symbol Reference</b>	<b>10</b>
<b>Classes</b>	<b>10</b>
EventData Class	11
EventData Fields	11
EventData.Code Field	11
EventData.Parameters Field	11
EventData Methods	11
EventData.this Indexer	11
EventData.ToString Method	12
EventData.ToStringFull Method	12
IPhotonSocket Class	12
IPhotonSocket.IPhotonSocket Constructor	13
IPhotonSocket Fields	13
IPhotonSocket.PollReceive Field	13
IPhotonSocket Methods	13
IPhotonSocket.Connect Method	13
IPhotonSocket.Disconnect Method	13
IPhotonSocket.EnqueueDebugReturn Method	14
IPhotonSocket.HandleReceivedDatagram Method	14
IPhotonSocket.Receive Method	14

IPhotonSocket.ReportDebugOfLevel Method	14
IPhotonSocket.Send Method	14
IPhotonSocket Properties	14
IPhotonSocket.Connected Property	14
IPhotonSocket.Listener Property	15
IPhotonSocket.MTU Property	15
IPhotonSocket.Protocol Property	15
IPhotonSocket.ServerAddress Property	15
IPhotonSocket.ServerPort Property	15
IPhotonSocket.State Property	15
LiteEventCode Class	16
LiteEventCode Fields	16
LiteEventCode.Join Field	16
LiteEventCode.Leave Field	16
LiteEventCode.PropertiesChanged Field	16
LiteEventKey Class	16
LiteEventKey Fields	17
LiteEventKey.ActorList Field	17
LiteEventKey.ActorNr Field	17
LiteEventKey.ActorProperties Field	17
LiteEventKey.CustomContent Field	17
LiteEventKey.Data Field	17
LiteEventKey.GameProperties Field	18
LiteEventKey.Properties Field	18
LiteEventKey.TargetActorNr Field	18
LiteOpCode Class	18
LiteOpCode Fields	18
LiteOpCode.ChangeGroups Field	18
LiteOpCode.ExchangeKeysForEncryption Field	19
LiteOpCode.GetProperties Field	19
LiteOpCode.Join Field	19
LiteOpCode.Leave Field	19
LiteOpCode.RaiseEvent Field	19
LiteOpCode.SetProperties Field	19
LiteOpKey Class	19
LiteOpKey Fields	20
LiteOpKey.ActorList Field	20
LiteOpKey.ActorNr Field	20
LiteOpKey.ActorProperties Field	21
LiteOpKey.Add Field	21
LiteOpKey.Asid Field	21
LiteOpKey.Broadcast Field	21

LiteOpKey.Cache Field	21
LiteOpKey.Code Field	21
LiteOpKey.Data Field	21
LiteOpKey.GameId Field	21
LiteOpKey.GameProperties Field	22
LiteOpKey.Group Field	22
LiteOpKey.Properties Field	22
LiteOpKey.ReceiverGroup Field	22
LiteOpKey.Remove Field	22
LiteOpKey.RoomName Field	22
LiteOpKey.TargetActorNr Field	22
LitePeer Class	23
LitePeer Constructor	27
LitePeer.LitePeer Constructor ()	27
LitePeer.LitePeer Constructor (ConnectionProtocol)	27
LitePeer.LitePeer Constructor (IPhotonPeerListener)	27
LitePeer.LitePeer Constructor (IPhotonPeerListener, ConnectionProtocol)	27
LitePeer Methods	28
LitePeer.OpChangeGroups Method	28
LitePeer.OpGetProperties Method	28
OpGetPropertiesOfActor Method	28
OpGetPropertiesOfGame Method	29
OpJoin Method	30
LitePeer.OpLeave Method	31
OpRaiseEvent Method	31
LitePeer.OpSetPropertiesOfActor Method	35
LitePeer.OpSetPropertiesOfGame Method	36
NetworkSimulationSet Class	36
NetworkSimulationSet Fields	37
NetworkSimulationSet.NetSimManualResetEvent Field	37
NetworkSimulationSet Methods	37
NetworkSimulationSet.ToString Method	37
NetworkSimulationSet Properties	37
NetworkSimulationSet.IncomingJitter Property	37
NetworkSimulationSet.IncomingLag Property	37
NetworkSimulationSet.IncomingLossPercentage Property	38
NetworkSimulationSet.LostPackagesIn Property	38
NetworkSimulationSet.LostPackagesOut Property	38
NetworkSimulationSet.OutgoingJitter Property	38
NetworkSimulationSet.OutgoingLag Property	38
NetworkSimulationSet.OutgoingLossPercentage Property	38
OperationRequest Class	38

OperationRequest Fields	39
OperationRequest.OperationCode Field	39
OperationRequest.Parameters Field	39
OperationResponse Class	39
OperationResponse Fields	40
OperationResponse.DebugMessage Field	40
OperationResponse.OperationCode Field	40
OperationResponse.Parameters Field	40
OperationResponse.ReturnCode Field	40
OperationResponse Methods	40
OperationResponse.this Indexer	40
OperationResponse.ToString Method	41
OperationResponse.ToStringFull Method	41
PeerBase Class	41
PeerBase Enumerations	42
PeerBase.ConnectionStateValue Enumeration	42
PeerBase Fields	42
PeerBase.ByteCountCurrentDispatch Field	42
PeerBase.ByteCountLastOperation Field	42
PeerBase.CryptoProvider Field	43
PeerBase.SerializeMemStream Field	43
PeerBase.TrafficStatsGameLevel Field	43
PeerBase.TrafficStatsIncoming Field	43
PeerBase.TrafficStatsOutgoing Field	43
PeerBase Properties	43
PeerBase.NetworkSimulationSettings Property	43
PeerBase.PeerID Property	44
PeerBase.TrafficStatsEnabled Property	44
PeerBase.TrafficStatsEnabledTime Property	44
PhotonPeer Class	44
PhotonPeer Constructor	48
PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener)	48
PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener, ConnectionProtocol)	48
PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener, bool)	48
PhotonPeer Methods	48
PhotonPeer.Connect Method	49
PhotonPeer.Disconnect Method	49
PhotonPeer.DispatchIncomingCommands Method	49
PhotonPeer.EstablishEncryption Method	50
PhotonPeer.FetchServerTimestamp Method	50
OpCustom Method	50
PhotonPeer.RegisterType Method	52

PhotonPeer.SendAcksOnly Method	52
PhotonPeer.SendOutgoingCommands Method	53
PhotonPeer.Service Method	53
PhotonPeer.StopThread Method	54
PhotonPeer.TrafficStatsReset Method	54
PhotonPeer.VitalStatsToString Method	54
PhotonPeer Properties	54
PhotonPeer.ByteCountCurrentDispatch Property	54
PhotonPeer.ByteCountLastOperation Property	54
PhotonPeer.BytesIn Property	55
PhotonPeer.BytesOut Property	55
PhotonPeer.ChannelCount Property	55
PhotonPeer.CommandBufferSize Property	55
PhotonPeer.CrcEnabled Property	55
PhotonPeer.DebugOut Property	56
PhotonPeer.DisconnectTimeout Property	56
PhotonPeer.HttpUrlParameters Property	56
PhotonPeer.IsEncryptionAvailable Property	56
PhotonPeer.IsSendingOnlyAcks Property	56
PhotonPeer.IsSimulationEnabled Property	56
PhotonPeer.LimitOfUnreliableCommands Property	57
PhotonPeer.Listener Property	57
PhotonPeer.LocalMsTimestampDelegate Property	57
PhotonPeer.LocalTimeInMilliseconds Property	57
PhotonPeer.MaximumTransferUnit Property	58
PhotonPeer.NetworkSimulationSettings Property	58
PhotonPeer.OutgoingStreamBufferSize Property	58
PhotonPeer.PacketLossByCrc Property	58
PhotonPeer.PeerID Property	58
PhotonPeer.PeerState Property	58
PhotonPeer.QueuedIncomingCommands Property	59
PhotonPeer.QueuedOutgoingCommands Property	59
PhotonPeer.ResentReliableCommands Property	59
PhotonPeer.RoundTripTime Property	59
PhotonPeer.RoundTripTimeVariance Property	59
PhotonPeer.SentCountAllowance Property	59
PhotonPeer.ServerAddress Property	60
PhotonPeer.ServerTimeInMilliseconds Property	60
PhotonPeer.SocketImplementation Property	60
PhotonPeer.TcpConnectionPrefix Property	60
PhotonPeer.TimePingInterval Property	61
PhotonPeer.TimestampOfLastSocketReceive Property	61

PhotonPeer.TrafficStatsElapsedMs Property	61
PhotonPeer.TrafficStatsEnabled Property	61
PhotonPeer.TrafficStatsGameLevel Property	61
PhotonPeer.TrafficStatsIncoming Property	61
PhotonPeer.TrafficStatsOutgoing Property	61
PhotonPeer.UsedProtocol Property	62
PhotonPeer.WarningSize Property	62
Protocol Class	62
Protocol Methods	62
Deserialize Method	62
Serialize Method	63
SocketUdpNativeDynamic Class	64
SocketUdpNativeDynamic.SocketUdpNativeDynamic Constructor	65
SocketUdpNativeDynamic Methods	66
SocketUdpNativeDynamic.Connect Method	66
SocketUdpNativeDynamic.Disconnect Method	66
SocketUdpNativeDynamic.Receive Method	66
SocketUdpNativeDynamic.ReceiveLoop Method	66
SocketUdpNativeDynamic.Send Method	66
SocketUdpNativeStatic Class	66
SocketUdpNativeStatic.SocketUdpNativeStatic Constructor	67
SocketUdpNativeStatic Methods	68
SocketUdpNativeStatic.Connect Method	68
SocketUdpNativeStatic.Disconnect Method	68
SocketUdpNativeStatic.Receive Method	68
SocketUdpNativeStatic.ReceiveLoop Method	68
SocketUdpNativeStatic.Send Method	68
SupportClass Class	68
SupportClass Classes	69
SupportClass.ThreadSafeRandom Class	69
SupportClass Methods	69
SupportClass.ByteArrayToString Method	70
SupportClass.CalculateCrc Method	70
CallInBackground Method	70
DictionaryToString Method	70
SupportClass.GetMethods Method	71
SupportClass.GetTickCount Method	71
SupportClass.HashtableToString Method	71
NumberToByteArray Method	71
WriteStackTrace Method	72
SupportClass Delegates	72
SupportClass.IntegerMillisecondsDelegate Delegate	73

TrafficStats Class	73
TrafficStats Methods	73
TrafficStats.ToString Method	74
TrafficStats Properties	74
TrafficStats.ControlCommandBytes Property	74
TrafficStats.ControlCommandCount Property	74
TrafficStats.FragmentCommandBytes Property	74
TrafficStats.FragmentCommandCount Property	74
TrafficStats.PackageHeaderSize Property	74
TrafficStats.ReliableCommandBytes Property	75
TrafficStats.ReliableCommandCount Property	75
TrafficStats.TimestampOfLastAck Property	75
TrafficStats.TimestampOfLastReliableCommand Property	75
TrafficStats.TotalCommandBytes Property	75
TrafficStats.TotalCommandCount Property	75
TrafficStats.TotalCommandsInPackets Property	75
TrafficStats.TotalPacketBytes Property	76
TrafficStats.TotalPacketCount Property	76
TrafficStats.UnreliableCommandBytes Property	76
TrafficStats.UnreliableCommandCount Property	76
TrafficStatsGameLevel Class	76
TrafficStatsGameLevel Methods	77
TrafficStatsGameLevel.ResetMaximumCounters Method	77
TrafficStatsGameLevel.ToString Method	77
TrafficStatsGameLevel.ToStringVitalStats Method	78
TrafficStatsGameLevel Properties	78
TrafficStatsGameLevel.DispatchCalls Property	78
TrafficStatsGameLevel.DispatchIncomingCommandsCalls Property	78
TrafficStatsGameLevel.EventByteCount Property	78
TrafficStatsGameLevel.EventCount Property	78
TrafficStatsGameLevel.LongestDeltaBetweenDispatching Property	78
TrafficStatsGameLevel.LongestDeltaBetweenSending Property	78
TrafficStatsGameLevel.LongestEventCallback Property	79
TrafficStatsGameLevel.LongestEventCallbackCode Property	79
TrafficStatsGameLevel.LongestOpResponseCallback Property	79
TrafficStatsGameLevel.LongestOpResponseCallbackOpCode Property	79
TrafficStatsGameLevel.OperationByteCount Property	79
TrafficStatsGameLevel.OperationCount Property	79
TrafficStatsGameLevel.ResultByteCount Property	79
TrafficStatsGameLevel.ResultCount Property	80
TrafficStatsGameLevel.SendOutgoingCommandsCalls Property	80
TrafficStatsGameLevel.TotalByteCount Property	80



TrafficStatsGameLevel.TotalIncomingByteCount Property	80
TrafficStatsGameLevel.TotalIncomingMessageCount Property	80
TrafficStatsGameLevel.TotalMessageCount Property	80
TrafficStatsGameLevel.TotalOutgoingByteCount Property	80
TrafficStatsGameLevel.TotalOutgoingMessageCount Property	80
<b>Interfaces</b>	<b>81</b>
IPhotonPeerListener Interface	81
IPhotonPeerListener Methods	81
IPhotonPeerListener.DebugReturn Method	81
IPhotonPeerListener.OnEvent Method	82
IPhotonPeerListener.OnOperationResponse Method	82
IPhotonPeerListener.OnStatusChanged Method	83
<b>Structs, Records, Enums</b>	<b>83</b>
ConnectionProtocol Enumeration	84
DebugLevel Enumeration	84
EventCaching Enumeration	84
GpType Enumeration	85
LitePropertyTypes Enumeration	86
PeerStateValue Enumeration	86
PhotonDisconnectCause Enumeration	87
PhotonSocketError Enumeration	87
PhotonSocketState Enumeration	87
ReceiverGroup Enumeration	88
StatusCode Enumeration	88
<b>Types</b>	<b>89</b>
DeserializeMethod Type	90
SerializeMethod Type	90

## Index a

# 1 Overview

Photon is a development framework to build real-time multiplayer games and applications for various platforms. It consists of a Server SDK and Client SDKs for several platforms.

This is the documentation and reference for the Photon Client Library.

Additional documentation and help is available online. Visit: [doc.exitgames.com/photon-server](http://doc.exitgames.com/photon-server) and [forum.exitgames.com](http://forum.exitgames.com)

Photon provides a low-latency communication-layer based on UDP (or alternatively TCP). It enables reliable and unreliable transfer of data in "commands". On top of this, an operation- and event-framework is established to ease development of your own games.

Each game is different, so we developed several "server applications" which provide a basic logic and included them in the server SDK as example and code base.

- The "Lite" application offers the basic operations that we felt useful for most room-based multiplayer games. We included its operations and **events** are part of the client API.
- The "Loadbalancing" application extends the "Lite" application and allows you to run multiple servers for your game. A master server coordinates the games creation and joining. There is a special API in the SDK that makes use of this application.
- The "MMO" application focuses on seamless worlds and its client side api is available in source in the MMO application's code (as it shares code with the server side).

Photon Cloud is a hosted service for your Photon games. To use it, register at [cloud.exitgames.com](http://cloud.exitgames.com) and use the Loadbalancing API and demos to start your development.

---

## 1.1 Photon Workflow

To get an impression of how to work on the client, we will use the server's Lite logic. This application defines rooms which are created when users try to join them. Each user in a room becomes an actor with her own number.

A simplified workflow looks like this:

- create a **LitePeer** instance
- from now on: regularly call Service to get **events** and send commands (e.g. ten times a second)
- call Connect to connect the server
  - wait until the library calls **OnStatusChanged**
  - the returned status int should equal **StatusCode.Connect**
- call OpJoin to get into a game
  - wait until the library calls OnOperationResponse with opCode: **LiteOpCode.Join**
- send data in the game by calling **OpRaiseEvent**
- receive **events** in **OnEvent**

- The Lite Application defines several useful **events** for common situations: Someone joins or leaves the room.
- In Lite, **events** created by calling `OpRaiseEvent` will be received by others in the same room in this method.
- when you are done: call `LitePeer.OpLeave` to quit/leave the game
  - wait for "leave" return in `OnOperationResponse` with `opCode`: `LiteOpCode.Leave`
- disconnect with `Disconnect`
  - check "disconnect" return in `OnStatusChanged` with `statusCode`: `StatusCode.Disconnect`

Combined with the server's Lite application, this simple workflow would allow you to use rooms and send your game's **events**. The methods used could be broken down into three layers:

- **Low Level:** Service, Connect, Disconnect and the `OnStatusChanged` are directly referring to the connection to the server. This level works with UDP/TCP packets which transport commands (which in turn carry your operations). It keeps your connection alive and organizes your RPC calls and **events** into packages.
- **Logic Level:** **Operations**, results and **events** make up the logical level in Photon. Any operation defined on the server (think RPC call) and can have a result. **Events** are incoming from the server and update the client with some data.
- **Application Level:** Made up by a specific application and its features. In this case we use the operations and logic of the Lite application. In this specific case, we have rooms and actors and more. The `LitePeer` is matching the server side implementation and wraps it up for you.

You don't have to manage the low level communication in most cases. However, it makes sense to know that everything that goes from client to server (and the other way round) is put into "commands". Internally, commands are also used to establish and keep the connection between client and server alive (without carrying additional data).

All methods that are operations (RPC calls) are prefixed with "Op" to tell them apart from anything else. Other server side applications (like MMO or your own) will define different operations. These will have different parameters and return values. These operations are not part of the client library but can be implemented by calling `OpCustom()`.

The interface `IPhotonPeerListener` must be implemented for callbacks. They are:

- `OnStatusChanged` is for peer state-changes (connect, disconnect, errors, compare with `StatusCode Enumeration`)
- `OnOperationResponse` is the callback for operations (join, leave, etc.)
- `OnEvent` as callback for **events** coming in
- `DebugReturn` as callback to debug output (less frequently used by release builds)

The following properties in `PhotonPeer` are of special interest:

- `TimePingInterval` sets the time between ping-operations
- `RoundTripTime` of reliable operations to the server and back
- `RoundTripTimeVariance` shows the variability of the roundtrip time
- `ServerTimeInMilliseconds` is the continuously approximated server's time

## 1.2 Operations

Operation is our term for remote procedure calls (RPC) on Photon. This in turn can be described as methods that are implemented on the server-side and called by clients. As any method, they have parameters and return values. The Photon development framework takes care of getting your RPC calls from clients to server and results back.

Server-side, operations are part of an application running on top of Photon. The default application provided by Exit Games is called "Lite Application" or simply Lite. The **LitePeer** class extends the **PhotonPeer** by methods for each of the Lite Operations.

Examples for Lite Operations are "join" and "raise event". On the client side, they can be found in the **LitePeer** class as methods: **OpJoin** and **OpRaiseEvent**. They can be used right away with the default implementation of Photon and the Lite Application.

### Custom Operations

Photon is extendable with features that are specific to your game. You could persist world states or double check information from the clients. Any operation that is not in Lite or the MMO application logic is called Custom Operation. Creating those is primarily a server-side task, of course, but the clients have to use new functions / operations of the server.

So Operations are methods that can be called from the client side. They can have any number of parameters and any name. To preserve bandwidth, we assign byte-codes for every operation and each parameter. The definition is done server side. Each Operation has its own, unique number to identify it, known as the operation code (opCode). An operation class defines the expected parameters and assigns a parameter code for each. With this definition, the client side only has to fill in the values and let the server know the opCode of the Operation.

Photon uses Dictionaries to aggregate parameters for operation requests, responses and **events**. Use **OpCustom** to call any operation, providing the parameters in a Dictionary.

Client side, opCode and parameter-codes are currently of type byte (to minimize overhead). They need to match the definition of the server side to successfully call your operation.

## 1.3 Events

Unlike operations, events are "messages" that are rarely triggered by the client that receives them. Events come from outside: the server or other clients.

They are created as side effect of operations (e.g. when you join a room) or raised as main purpose of the operation **RaiseEvent**. Most events carry some form of data but in rare cases the type of event itself is the message.

Events are (once more) Dictionaries with arbitrary content. In the "top-level" of an event, bytes are used as keys for values. The values can be of any serializable type. The Lite Application, e.g., uses a Hashtable for custom event content in its operation **RaiseEvent**.

## 1.4 Fragmentation and Channels

### Fragmentation

Bigger data chunks of data won't fit into a single package, so they are fragmented and reassembled automatically. Depending on the data size, a single operation or event can be made up of multiple packages.

Be aware that this might stall other commands. Call **Service()** or **SendOutgoingCommands()** more often than absolutely necessary. You should check that **PhotonPeer.QueuedOutgoingCommands** is becoming zero regularly to make sure everything gets out. You can also check the debug output for "UDP package is full", which can happen from time to time but should not happen permanently.

### Maximum Transfer Unit

The maximum size for any UDP package can be configured by setting **PhotonPeer.MaximumTransferUnit Property**. By default, this is 1200 bytes. Some routers will fragment even this UDP package size. If you don't need bigger sizes, go for 512 bytes per package, which is more overhead per command but potentially safer.

This setting is ignored by TCP connections, which negotiate their MTU internally.

### Sequencing

The sequencing of the protocol makes sure that any receiving client will Dispatch your actions in the order you sent them. Unreliable data is considered replaceable and can be lost. Reliable **events** and operations will be repeated several times if needed but they will all be Dispatched in order without gaps. Unreliable actions are also related to the last reliable action and not Dispatched before that reliable data was Dispatched first. This can be useful, if the **events** are related to each other.

Example: Your FPS sends out unreliable movement updates and reliable chat messages. A lost package with movement updates would be left out as the next movement update is coming fast. At the receiving end, this would maybe show as a small jump. If a package with a chat message is lost, this is repeated and would introduce lag, even to all movement updates after the message was created. In this case, the data is unrelated and should be put into different channels.

### Channels

The DotNet clients and server are now supporting "channels". This allows you to separate information into multiple channels, each being sequenced independently. This means, that **Events** of one channel will not be stalled because **events** of another channel are not available.

By default an **PhotonPeer** has two channels and channel zero is the default to send operations. The operations join and leave are always sent in channel zero (for simplicity). There is a "background" channel 255 used internally for connect and disconnect messages. This is ignored for the channel count.

Channels are prioritized: the lowest channel number is put into a UDP package first. Data in a higher channel might be sent later when a UDP package is already full.

Example: The chat messages can now be sent in channel one, while movement is sent in channel zero. They are not related and if a chat message is delayed, it will no longer affect movement in channel zero. Also, channel zero has higher priority and is more likely to be sent (in case packages get filled up).

---

## 1.5 Using TCP

A **PhotonPeer** could be instantiated with TCP as underlying protocol if necessary. This is not best practice but some platforms don't support UDP sockets. This is why Silverlight (e.g.) uses TCP in all cases.

The Photon Client API is the same for both protocols but there are some differences in what goes on under the hood.

Everything sent over TCP is always reliable, even if you call your operations as unreliable!

If you use only TCP clients

Simply send any operation unreliable. It saves some work (and bandwidth) in the underlying protocols.

If you have TCP and UDP clients

Anything you send between the TCP clients will always be transferred reliable. But as you communicate with some clients that use UDP these will get your **events** reliable or unreliable.

Example:

A Silverlight client might send unreliable movement updates in channel # 1. This will be sent via TCP, which makes it reliable. Photon however also has connections with UDP clients (like a 3D downloadable game client). It will use your reliable / unreliable settings to forward your movement updates accordingly.

---

## 1.6 Network Simulation

During development, most tests will be done in a local network. Once released, the clients will communicate through the internet, which has a higher delay per message and in some cases even drops messages entirely.

To prepare a game for real-life conditions, the Photon client libraries let you simulate some effects of internet-communication: lag, jitter and packet loss.

- **Lag / Latency:** a more or less constant delay of messages between client and server. Either direction can be affected in a different way but usually the values are close to another. Affects the roundtrip time.
- **Jitter:** Is randomizes the Lag in the simulation. This affects the variance of the roundtrip time. Udp packages can get out of order this way, which also is simulated. The new lag will be:  $Lag + [-JitterValue..+JitterValue]$ . This keeps the mean Lag at the setting and some packages are actually faster than the Lag value implies.
- **Packet Loss:** UDP packages can become lost. In the Photon protocol, commands that are flagged as reliable will be repeated while other commands (operations) might get lost this way.

The lag simulation is running in its own Thread which tries to meet delays defined in the settings. In most cases, they can be met but actual delays will have a variance of up to +/- 20ms.

### Using Network Simulation

By default, Network Simulation is turned off. It can be turned on by setting `PhotonPeer.IsSimulationEnabled` and the settings are aggregated into a `NetworkSimulationSet`, also accessible by the peer class (e.g. the `LitePeer`).

Code Sample:

```
//Activate / Deactivate:
this.peer.IsSimulationEnabled = true;

//Raise Incoming Lag:
this.peer.NetworkSimulationSettings.IncomingLag = 300; //default is 100ms

//add 10% of outgoing loss:
this.peer.NetworkSimulationSettings.OutgoingLossPercentage = 10; //default is 1

//this property counts the actual simulated loss:
this.peer.NetworkSimulationSettings.LostPackagesOut;
```

## 1.7 Serializable Datatypes

Starting with Photon Server SDK 1.8.0 and DotNet SDK 5.6.0, the set of serializable datatypes is changed. The ArrayList was removed but aside from Arrays and Hashtables every serializable type can also be sent as array (e.g. String[], Float[]). Only one-dimensional arrays are supported currently.

### Photon 1.8.0 and higher

String / string

Boolean / bool

Byte / byte (unsigned! can be cast to SByte / sbyte to be equivalent to Java's byte)

Int16 / short (signed)

Int32 / int (signed)

Int64 / long

Single / float

Double / double

Array (of the types above, with a max number of Short.MaxValue entries).

Hashtable (not available as array)

The following data types can be used within [events](#) as keys and values on Neutron:

### Photon 1.6.0 and lower

String

Boolean

Byte / byte (unsigned! can be cast to SByte / sbyte to be equivalent to Java's byte)

Int16 (equals Short in Java)

Int32 (equals Integer in Java)

Int64 (equals Long in Java)

ArrayList (equals Vector in Java)

Hashtable

sbyte[]

int[]

String[]

## 1.8 The Photon Server

The Photon Server is the central hub for communication for all your clients. It is a service that can be run on any Windows machine, handling the UDP and TCP connections of clients and hosting a DotNet runtime layer with your own business logic, called application.

The Photon Server SDK includes several applications in source and pre-built. You can run them out of the box or develop your own server logic.

Get the Photon Server SDK at: [photon.exitgames.com](http://photon.exitgames.com)

---

## 1.9 Lite Application

The Lite Application is the example application for room-based games on Photon and (hopefully) a flexible basis for your own games. It offers rooms, joining and leaving them, sending **events** to the other players in a room and handles properties.

The Lite Application is tightly integrated with the client libraries and used as example throughout most documentation.

---

### 1.9.1 Properties on Photon

The Lite Application implements a general purpose mechanism to set and fetch key/value pairs on the server side (in memory). They are associated to a room/game or a player within a room and can be fetched or updated by anyone in that game.

Each entry in the properties Hashtable is considered a separate property and can be overwritten independently. The value of a property can be of any **serializable datatype**. The keys must be either of type string or byte. Bytes are preferred, as they mean the less overhead.

To avoid confusion, don't mix string and byte as key-types. Mixed types of keys, require separate requests to fetch them.

#### Property broadcasting and events

Property changes in a game can be "broadcasted", which triggers **events** for the other players to update them. The player who changed the property does not get the update (again).

Any change that uses the broadcast option will trigger a property update event. This event carries the changed properties (only), who changed the properties and where the properties belong to.

Your clients need to "merge" the changes (if properties are cached at all).

Properties can be set by these methods:

- **LitePeer.OpSetPropertiesOfActor Method** sets a player's properties
- **LitePeer.OpSetPropertiesOfGame Method** sets a game's properties
- **LitePeer.OpJoin Method** also allows you to set properties if the game did not exist yet



And fetched with these methods:

- `OpGetPropertiesOfActor`
- `OpGetPropertiesOfGame`

### Broadcast Events

Any change that uses the broadcast option will trigger a property update event `LiteEventCode.PropertiesChanged`. This event carries the properties as value of key `LiteEventKey.Properties`.

Additionally, there is information about who changed the properties in key `LiteEventKey.ActorNr`.

The key `LiteEventKey.TargetActorNr` will only be available if the property-set belongs to a certain player. If it's not present, the properties are game-properties.

### Notes

The current Lite application is not able to delete properties and does not support wildcard characters in string keys to fetch properties.

Other types of keys could be used but to keep things simple, we decided against adding those. If needed, we would help you with the implementation.

The property handling is likely to be updated and extended in the future.

---

## 1.10 Custom Authentication

By default, all users are allowed to connect to your Photon Cloud applications. With "Custom Authentication", you can combine Photon with more or less any community, check account credentials and customize who has access to your game.

The Photon Cloud Dashboard lets you configure a web service to check when a user connects. The client's credentials will be passed on to that service (via https).

Credentials in this case are anything you want to pass. It can even be a token/session provided by a third party service that the clients contact before connecting to Photon.

Values for CustomAuthentication are stored in the class `AuthenticationValues`. `LoadBalancingClient.Connect` has an overload with `AuthenticationValues`. Alternatively you can set the values as property: `LoadBalancingClient.CustomAuthenticationValues`.

Details about the workflow and the communication between Photon and the custom authentication service can be found online:

<http://doc.exitgames.com/photon-cloud/CustomAuthentication>

This is a LoadBalancing and Photon Cloud feature and it's not available in Lite.

---

## 1.11 Further Help

### Developer Network

Visit: [doc.exitgames.com/photon-server](http://doc.exitgames.com/photon-server)

### Developer Forum

Visit: [forum.exitgames.com](http://forum.exitgames.com)

### Mail Support



















Don't hesitate to mail us: [developer@exitgames.com](mailto:developer@exitgames.com)

# 2 Symbol Reference

## 2.1 Classes

The following table lists classes in this documentation.

### Classes

	Name	Description
	<a href="#">EventData</a>	Contains all components of a Photon <a href="#">Event</a> . <a href="#">Event Parameters</a> , like <a href="#">OperationRequests</a> and <a href="#">OperationResults</a> , consist of a Dictionary with byte-typed keys per value.
	<a href="#">IPhotonSocket</a>	This is class <a href="#">IPhotonSocket</a> .
	<a href="#">LiteEventCode</a>	Lite - <a href="#">Event</a> codes. These codes are defined by the Lite application's logic on the server side. Other application's won't necessarily use these.
	<a href="#">LiteEventKey</a>	Lite - Keys of event-parameters that are defined by the Lite application logic. To keep things lean (in terms of bandwidth), we use byte keys to identify values in <a href="#">events</a> within Photon. In Lite, you can send custom <a href="#">events</a> by defining a <a href="#">EventCode</a> and some content. This custom content is a Hashtable, which can use any type for keys and values. The parameter for operation <a href="#">RaiseEvent</a> and the resulting <a href="#">Events</a> use key (byte)245 for the custom content. The constant for this is: <a href="#">Data</a> or <a href="#">LiteEventKey.CustomContent Field</a> .
	<a href="#">LiteOpCode</a>	Lite - Operation Codes. This enumeration contains the codes that are given to the Lite Application's operations. Instead of sending " <a href="#">Join</a> ", this enables us to send the byte 255.
	<a href="#">LiteOpKey</a>	Lite - keys for parameters of operation requests and responses (short: <a href="#">OpKey</a> ).
	<a href="#">LitePeer</a>	A <a href="#">LitePeer</a> is an extended <a href="#">PhotonPeer</a> and implements the operations offered by the "Lite" Application of the Photon Server SDK.
	<a href="#">NetworkSimulationSet</a>	A set of network simulation settings, enabled (and disabled) by <a href="#">PhotonPeer.IsSimulationEnabled</a> .
	<a href="#">OperationRequest</a>	Container for an Operation request, which is a code and parameters.
	<a href="#">OperationResponse</a>	Contains the server's response for an operation called by this peer. The indexer of this class actually provides access to the <a href="#">Parameters</a> Dictionary.
	<a href="#">PeerBase</a>	This is class <a href="#">PeerBase</a> .
	<a href="#">PhotonPeer</a>	Instances of the <a href="#">PhotonPeer</a> class are used to connect to a Photon server and communicate with it.
	<a href="#">Protocol</a>	Provides tools for the Exit Games Protocol
	<a href="#">SocketUdpNativeDynamic</a>	This is class <a href="#">SocketUdpNativeDynamic</a> .
	<a href="#">SocketUdpNativeStatic</a>	This is class <a href="#">SocketUdpNativeStatic</a> .
	<a href="#">SupportClass</a>	Contains several (more or less) useful static methods, mostly used for debugging.
	<a href="#">TrafficStats</a>	This is class <a href="#">TrafficStats</a> .
	<a href="#">TrafficStatsGameLevel</a>	Only in use as long as <a href="#">PhotonPeer.TrafficStatsEnabled</a> = true;

## 2.1.1 EventData Class

Contains all components of a Photon [Event](#). [Event Parameters](#), like OperationRequests and OperationResults, consist of a Dictionary with byte-typed keys per value.



### C#

```
public class EventData;
```




### Remarks

The indexer of this class actually provides access to the [Parameters](#) Dictionary. The operation RaiseEvent of the Lite application allows you to provide custom event content. Defined in Lite, this CustomContent will be made the value of key LiteEventKey.OperationRaiseEvent which is (byte)42. Enums and constants for the Lite-Application codes are defined in the [LitePeer](#) namespace. Check: [LiteEventKey](#), etc.

### EventData Fields

	Name	Description
	<a href="#">Code</a>	The event code identifies the type of event.
	<a href="#">Parameters</a>	The Parameters of an event is a Dictionary.

### EventData Methods

	Name	Description
	<a href="#">this</a>	Alternative access to the <a href="#">Parameters</a> .
	<a href="#">ToString</a>	ToString() override.
	<a href="#">ToStringFull</a>	Extensive output of the event content.

### 2.1.1.1 EventData Fields

#### 2.1.1.1.1 EventData.Code Field

The event code identifies the type of event.

### C#

```
public byte Code;
```

#### 2.1.1.1.2 EventData.Parameters Field

The Parameters of an event is a Dictionary.

### C#

```
public Dictionary<byte, object> Parameters;
```

### 2.1.1.2 EventData Methods

#### 2.1.1.2.1 EventData.this Indexer

Alternative access to the [Parameters](#).

### C#

```
public object this[byte key];
```

**Parameters**

Parameters	Description
byte key	The key byte-code of a event value.

**Returns**

The **Parameters** value, or null if the key does not exist in **Parameters**.

### 2.1.1.2.2 EventData.ToString Method

ToString() override.

**C#**

```
public override string ToString();
```

**Returns**

Short output of "Event" and it's **Code**.

### 2.1.1.2.3 EventData.ToStringFull Method

Extensive output of the event content.

**C#**

```
public string ToStringFull();
```

**Returns**

To be used in debug situations only, as it returns a string for each value.

## 2.1.2 IPhotonSocket Class

**C#**

```
public abstract class IPhotonSocket;
```

**Description**

This is class IPhotonSocket.

**Methods**

	Name	Description
💎	<b>IPhotonSocket</b>	This is IPhotonSocket, a member of class IPhotonSocket.

**IPhotonSocket Fields**








	Name	Description
💎	<b>PollReceive</b>	This is PollReceive, a member of class IPhotonSocket.

**IPhotonSocket Methods**

	Name	Description
💎 V	<b>Connect</b>	This is Connect, a member of class IPhotonSocket.
💎 A	<b>Disconnect</b>	This is Disconnect, a member of class IPhotonSocket.
💎	<b>EnqueueDebugReturn</b>	This is EnqueueDebugReturn, a member of class IPhotonSocket.
💎	<b>HandleReceivedDatagram</b>	This is HandleReceivedDatagram, a member of class IPhotonSocket.
💎 A	<b>Receive</b>	This is Receive, a member of class IPhotonSocket.
💎	<b>ReportDebugOfLevel</b>	This is ReportDebugOfLevel, a member of class IPhotonSocket.

	Send	This is Send, a member of class IPhotonSocket.
---	------	--

IPhonSocket Properties

	Name	Description
	Connected	This is Connected, a member of class IPhotonSocket.
	Listener	This is Listener, a member of class IPhotonSocket.
	MTU	This is MTU, a member of class IPhotonSocket.
	Protocol	This is Protocol, a member of class IPhotonSocket.
	ServerAddress	This is ServerAddress, a member of class IPhotonSocket.
	ServerPort	This is ServerPort, a member of class IPhotonSocket.
	State	This is State, a member of class IPhotonSocket.

2.1.2.1 IPhotonSocket.IPhotonSocket Constructor

```
C#
public IPhotonSocket (PeerBase peerBase) ;
```

Description

This is IPhotonSocket, a member of class IPhotonSocket.

2.1.2.2 IPhotonSocket Fields

2.1.2.2.1 IPhotonSocket.PollReceive Field

```
C#
public bool PollReceive ;
```

Description

This is PollReceive, a member of class IPhotonSocket.

2.1.2.3 IPhotonSocket Methods

2.1.2.3.1 IPhotonSocket.Connect Method

```
C#
public virtual bool Connect () ;
```

Description

This is Connect, a member of class IPhotonSocket.

2.1.2.3.2 IPhotonSocket.Disconnect Method

```
C#
public abstract bool Disconnect () ;
```

Description

This is Disconnect, a member of class IPhotonSocket.

### 2.1.2.3.3 IPhotonSocket.EnqueueDebugReturn Method

**C#**

```
public void EnqueueDebugReturn(DebugLevel debugLevel, string message);
```

**Description**

This is EnqueueDebugReturn, a member of class IPhotonSocket.

### 2.1.2.3.4 IPhotonSocket.HandleReceivedDatagram Method

**C#**

```
public void HandleReceivedDatagram(byte[] inBuffer, int length, bool willBeReused);
```

**Description**

This is HandleReceivedDatagram, a member of class IPhotonSocket.

### 2.1.2.3.5 IPhotonSocket.Receive Method

**C#**

```
public abstract PhotonSocketError Receive(out byte[] data);
```

**Description**

This is Receive, a member of class IPhotonSocket.

### 2.1.2.3.6 IPhotonSocket.ReportDebugOfLevel Method

**C#**

```
public bool ReportDebugOfLevel(DebugLevel levelOfMessage);
```

**Description**

This is ReportDebugOfLevel, a member of class IPhotonSocket.

### 2.1.2.3.7 IPhotonSocket.Send Method

**C#**

```
public abstract PhotonSocketError Send(byte[] data, int length);
```

**Description**

This is Send, a member of class IPhotonSocket.

## 2.1.2.4 IPhotonSocket Properties

### 2.1.2.4.1 IPhotonSocket.Connected Property

**C#**

```
public bool Connected;
```

**Description**

This is Connected, a member of class IPhotonSocket.

#### 2.1.2.4.2 IPhotonSocket.Listener Property

C#

```
protected IPhotonPeerListener Listener;
```

##### Description

This is Listener, a member of class IPhotonSocket.

#### 2.1.2.4.3 IPhotonSocket.MTU Property

C#

```
public int MTU;
```

##### Description

This is MTU, a member of class IPhotonSocket.

#### 2.1.2.4.4 IPhotonSocket.Protocol Property

C#

```
public ConnectionProtocol Protocol;
```

##### Description

This is Protocol, a member of class IPhotonSocket.

#### 2.1.2.4.5 IPhotonSocket.ServerAddress Property

C#

```
public string ServerAddress;
```

##### Description

This is ServerAddress, a member of class IPhotonSocket.

#### 2.1.2.4.6 IPhotonSocket.ServerPort Property

C#

```
public int ServerPort;
```

##### Description

This is ServerPort, a member of class IPhotonSocket.

#### 2.1.2.4.7 IPhotonSocket.State Property

C#

```
public PhotonSocketState State;
```

##### Description

This is State, a member of class IPhotonSocket.



## 2.1.3 LiteEventCode Class

Lite - **Event** codes. These codes are defined by the Lite application's logic on the server side. Other application's won't necessarily use these.

**C#**

```
public static class LiteEventCode;
```

**Remarks**

If your game is built as extension of Lite, don't re-use these codes for your custom **events**.

**LiteEventCode Fields**

	Name	Description
◆	Join	(255) <b>Event</b> Join: someone joined the game
◆	Leave	(254) <b>Event</b> Leave: someone left the game
◆	PropertiesChanged	(253) <b>Event</b> PropertiesChanged

### 2.1.3.1 LiteEventCode Fields

#### 2.1.3.1.1 LiteEventCode.Join Field

(255) **Event** Join: someone joined the game

**C#**

```
public const byte Join = 255;
```

#### 2.1.3.1.2 LiteEventCode.Leave Field

(254) **Event** Leave: someone left the game

**C#**

```
public const byte Leave = 254;
```

#### 2.1.3.1.3 LiteEventCode.PropertiesChanged Field

(253) **Event** PropertiesChanged

**C#**

```
public const byte PropertiesChanged = 253;
```

## 2.1.4 LiteEventKey Class

Lite - Keys of event-parameters that are defined by the Lite application logic. To keep things lean (in terms of bandwidth), we use byte keys to identify values in **events** within Photon. In Lite, you can send custom **events** by defining a EventCode and some content. This custom content is a Hashtable, which can use any type for keys and values. The parameter for operation RaiseEvent and the resulting **Events** use key (byte)245 for the custom content. The constant for this is: **Data** or **LiteEventKey.CustomContent Field**.

**C#**

```
public static class LiteEventKey;
```

**Remarks**

If your game is built as extension of Lite, don't re-use these codes for your custom **events**.

**LiteEventKey Fields**

	Name	Description
◆	<b>ActorList</b>	(252) List of playernumbers currently in the room.
◆	<b>ActorNr</b>	(254) Playernumber of the player who triggered the event.
◆	<b>ActorProperties</b>	(249) Key for actor (player) property set (Hashtable).
◆	<b>CustomContent</b>	(245) The Lite operation RaiseEvent will place the Hashtable with your custom event-content under this key.
◆	<b>Data</b>	(245) Custom Content of an event (a Hashtable in Lite).
◆	<b>GameProperties</b>	(248) Key for game (room) property set (Hashtable).
◆	<b>Properties</b>	(251) Set of properties (a Hashtable).
◆	<b>TargetActorNr</b>	(253) Playernumber of the player who is target of an event (e.g. changed properties).

## 2.1.4.1 LiteEventKey Fields

### 2.1.4.1.1 LiteEventKey.ActorList Field

(252) List of playernumbers currently in the room.

**C#**

```
public const byte ActorList = 252;
```

### 2.1.4.1.2 LiteEventKey.ActorNr Field

(254) Playernumber of the player who triggered the event.

**C#**

```
public const byte ActorNr = 254;
```

### 2.1.4.1.3 LiteEventKey.ActorProperties Field

(249) Key for actor (player) property set (Hashtable).

**C#**

```
public const byte ActorProperties = 249;
```

### 2.1.4.1.4 LiteEventKey.CustomContent Field

(245) The Lite operation RaiseEvent will place the Hashtable with your custom event-content under this key.

**C#**

```
public const byte CustomContent = Data;
```

**Remarks**

Alternative for: **Data**!

### 2.1.4.1.5 LiteEventKey.Data Field

(245) Custom Content of an event (a Hashtable in Lite).

**C#**

```
public const byte Data = 245;
```

#### 2.1.4.1.6 LiteEventKey.GameProperties Field

(248) Key for game (room) property set (Hashtable).

**C#**

```
public const byte GameProperties = 248;
```

#### 2.1.4.1.7 LiteEventKey.Properties Field

(251) Set of properties (a Hashtable).

**C#**

```
public const byte Properties = 251;
```

#### 2.1.4.1.8 LiteEventKey.TargetActorNr Field

(253) Playernumber of the player who is target of an event (e.g. changed properties).

**C#**

```
public const byte TargetActorNr = 253;
```

## 2.1.5 LiteOpCode Class

Lite - Operation Codes. This enumeration contains the codes that are given to the Lite Application's operations. Instead of sending "Join", this enables us to send the byte 255.

**C#**

```
public static class LiteOpCode;
```

#### Remarks

Other applications (the MMO demo or your own) could define other operations and other codes. If your game is built as extension of Lite, don't re-use these codes for your custom [events](#).

#### LiteOpCode Fields

	Name	Description
◆	<a href="#">ChangeGroups</a>	(248) Operation code to change interest groups in Rooms (Lite application and extending ones).
◆	<a href="#">ExchangeKeysForEncryption</a>	This is ExchangeKeysForEncryption, a member of class LiteOpCode.
◆	<a href="#">GetProperties</a>	(251) Operation code for OpGetProperties.
◆	<a href="#">Join</a>	(255) Code for OpJoin, to get into a room.
◆	<a href="#">Leave</a>	(254) Code for OpLeave, to get out of a room.
◆	<a href="#">RaiseEvent</a>	(253) Code for OpRaiseEvent (not same as eventCode).
◆	<a href="#">SetProperties</a>	(252) Code for OpSetProperties.

### 2.1.5.1 LiteOpCode Fields

#### 2.1.5.1.1 LiteOpCode.ChangeGroups Field

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

**C#**

```
public const byte ChangeGroups = 248;
```

### 2.1.5.1.2 LiteOpCode.ExchangeKeysForEncryption Field

**C#**

```
[Obsolete("Exchanging encryption keys is done internally in the lib now. Don't expect this operation-result.")]  
public const byte ExchangeKeysForEncryption = 250;
```

#### Description

This is ExchangeKeysForEncryption, a member of class LiteOpCode.

### 2.1.5.1.3 LiteOpCode.GetProperties Field

(251) Operation code for OpGetProperties.

**C#**

```
public const byte GetProperties = 251;
```

### 2.1.5.1.4 LiteOpCode.Join Field

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

**C#**

```
public const byte Join = 255;
```

### 2.1.5.1.5 LiteOpCode.Leave Field

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

**C#**

```
public const byte Leave = 254;
```

### 2.1.5.1.6 LiteOpCode.RaiseEvent Field

(253) Code for OpRaiseEvent (not same as eventCode).

**C#**

```
public const byte RaiseEvent = 253;
```

### 2.1.5.1.7 LiteOpCode.SetProperties Field

(252) Code for OpSetProperties.

**C#**

```
public const byte SetProperties = 252;
```

---

## 2.1.6 LiteOpKey Class

Lite - keys for parameters of operation requests and responses (short: OpKey).

**C#**

```
public static class LiteOpKey;
```

**Remarks**

These keys match a definition in the Lite application (part of the server SDK). If your game is built as extension of Lite, don't re-use these codes for your custom **events**.

These keys are defined per application, so Lite has different keys than MMO or your custom application. This is why these are not an enumeration. Lite and Lite Lobby will use the keys 255 and lower, to give you room for your own codes.

Keys for operation-parameters could be assigned on a per operation basis, but it makes sense to have fixed keys for values which are used throughout the whole application.

**LiteOpKey Fields**

	Name	Description
◆	<b>ActorList</b>	(252) <b>Code</b> for list of players in a room. Currently not used.
◆	<b>ActorNr</b>	(254) <b>Code</b> of the Actor of an operation. Used for property get and set.
◆	<b>ActorProperties</b>	(249) <b>Code</b> for property set (Hashtable).
◆	<b>Add</b>	(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.
◆	<b>Asid</b>	(255) <b>Code</b> of the room name. Used in OpJoin (Asid = Application Session ID).
◆	<b>Broadcast</b>	(250) <b>Code</b> for broadcast parameter of OpSetProperties method.
◆	<b>Cache</b>	(247) <b>Code</b> for caching <b>events</b> while raising them.
◆	<b>Code</b>	(244) Code used when sending some code-related parameter, like OpRaiseEvent's event-code.
◆	<b>Data</b>	(245) <b>Code</b> of data of an event. Used in OpRaiseEvent.
◆	<b>Gameld</b>	(255) <b>Code</b> of the game id (a unique room name). Used in OpJoin.
◆	<b>GameProperties</b>	(248) <b>Code</b> for property set (Hashtable).
◆	<b>Group</b>	(240) <b>Code</b> for "group" operation-parameter (as used in Op RaiseEvent).
◆	<b>Properties</b>	(251) <b>Code</b> for property set (Hashtable). This key is used when sending only one set of properties. If either <b>ActorProperties</b> or <b>GameProperties</b> are used (or both), check those keys.
◆	<b>ReceiverGroup</b>	(246) <b>Code</b> to select the receivers of <b>events</b> (used in Lite, Operation RaiseEvent).
◆	<b>Remove</b>	(239) The "Remove" operation-parameter can be used to remove something from a list. E.g. remove groups from player's interest groups.
◆	<b>RoomName</b>	(255) <b>Code</b> of the room name. Used in OpJoin.
◆	<b>TargetActorNr</b>	(253) <b>Code</b> of the target Actor of an operation. Used for property set. Is 0 for game

## 2.1.6.1 LiteOpKey Fields

### 2.1.6.1.1 LiteOpKey.ActorList Field

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

**C#**

```
public const byte ActorList = 252;
```

### 2.1.6.1.2 LiteOpKey.ActorNr Field

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

**C#**

```
public const byte ActorNr = 254;
```

### 2.1.6.1.3 LiteOpKey.ActorProperties Field

(249) **Code** for property set (Hashtable).

**C#**

```
public const byte ActorProperties = 249;
```

### 2.1.6.1.4 LiteOpKey.Add Field

(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.

**C#**

```
public const byte Add = 238;
```

### 2.1.6.1.5 LiteOpKey.Asid Field

(255) **Code** of the room name. Used in OpJoin (Asid = Application Session ID).

**C#**

```
[Obsolete("Use GameId")]  
public const byte Asid = 255;
```

### 2.1.6.1.6 LiteOpKey.Broadcast Field

(250) **Code** for broadcast parameter of OpSetProperties method.

**C#**

```
public const byte Broadcast = 250;
```

### 2.1.6.1.7 LiteOpKey.Cache Field

(247) **Code** for caching **events** while raising them.

**C#**

```
public const byte Cache = 247;
```

### 2.1.6.1.8 LiteOpKey.Code Field

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

**C#**

```
public const byte Code = 244;
```

**Remarks**

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

### 2.1.6.1.9 LiteOpKey.Data Field

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

**C#**

```
public const byte Data = 245;
```

### 2.1.6.1.10 LiteOpKey.GameId Field

(255) **Code** of the game id (a unique room name). Used in OpJoin.

**C#**

```
public const byte GameId = 255;
```

### 2.1.6.1.11 LiteOpKey.GameProperties Field

(248) **Code** for property set (Hashtable).

**C#**

```
public const byte GameProperties = 248;
```

### 2.1.6.1.12 LiteOpKey.Group Field

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

**C#**

```
public const byte Group = 240;
```

### 2.1.6.1.13 LiteOpKey.Properties Field

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

**C#**

```
public const byte Properties = 251;
```

### 2.1.6.1.14 LiteOpKey.ReceiverGroup Field

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

**C#**

```
public const byte ReceiverGroup = 246;
```

### 2.1.6.1.15 LiteOpKey.Remove Field

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

**C#**

```
public const byte Remove = 239;
```

### 2.1.6.1.16 LiteOpKey.RoomName Field

(255) **Code** of the room name. Used in OpJoin.

**C#**

```
[Obsolete("Use GameId")]  
public const byte RoomName = 255;
```

#### Remarks

Alternative for: **Asid!**

### 2.1.6.1.17 LiteOpKey.TargetActorNr Field

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

**C#**

```
public const byte TargetActorNr = 253;
```

## 2.1.7 LitePeer Class

A LitePeer is an extended [PhotonPeer](#) and implements the operations offered by the "Lite" Application of the Photon Server SDK.

### C#


```
public class LitePeer : PhotonPeer;
```

### Remarks


This class is used by our samples and allows rapid development of simple games. You can use rooms and properties and send [events](#). For many games, this is a good start.

**Operations** are prefixed as "Op" and are always asynchronous. In most cases, an [OperationResult](#) is provided by a later call to [OnOperationResult](#).





### Methods

	Name	Description
	<a href="#">PhotonPeer</a>	Creates a new PhotonPeer instance to communicate with Photon. Connection is UDP based, except for Silverlight.










### LitePeer Class

	Name	Description
	<a href="#">LitePeer</a>	Creates a LitePeer instance to connect and communicate with a Photon server. Uses UDP as protocol (except in the Silverlight library).










### PhotonPeer Methods

	Name	Description
	<a href="#">Connect</a>	This method does a DNS lookup (if necessary) and connects to the given serverAddress. The return value gives you feedback if the address has the correct format. If so, this starts the process to establish the connection itself, which might take a few seconds. When the connection is established, a callback to <a href="#">IPhotonPeerListener.OnStatusChanged</a> will be done. If the connection can't be established, despite having a valid address, the OnStatusChanged is called with an error-value. The applicationName defines the application logic to use server-side and it should match the name of one of the apps in your server's config. By default,... <a href="#">more</a>
	<a href="#">Disconnect</a>	This method initiates a mutual disconnect between this client and the server.
	<a href="#">DispatchIncomingCommands</a>	This method directly causes the callbacks for <a href="#">events</a> , responses and state changes within a <a href="#">IPhotonPeerListener</a> . DispatchIncomingCommands only executes a single received command per call. If a command was dispatched, the return value is true and the method should be called again. This method is called by <a href="#">Service()</a> until currently available commands are dispatched.
	<a href="#">EstablishEncryption</a>	This method creates a public key for this client and exchanges it with the server.





















	<b>FetchServerTimestamp</b>	This will fetch the server's timestamp and update the approximation for property <b>ServerTimeInMilliseconds</b> . The server time approximation will NOT become more accurate by repeated calls. Accuracy currently depends on a single roundtrip which is done as fast as possible. The command used for this is immediately acknowledged by the server. This makes sure the roundtrip time is low and the timestamp + roundtrip time / 2 is close to the original value.
	<b>OpCustom</b>	Allows the client to send any operation to the Photon Server by setting any opCode and the operation's parameters.
	<b>RegisterType</b>	Registers new types/classes for de/serialization and the fitting methods to call for this type.
	<b>SendAcksOnly</b>	This is <b>SendAcksOnly</b> , a member of class <b>PhotonPeer</b> .
	<b>SendOutgoingCommands</b>	This method creates a UDP/TCP package for outgoing commands (operations and acknowledgements) and sends them to the server. This method is also called by <b>Service()</b> .
	<b>Service</b>	This method excutes <b>DispatchIncomingCommands</b> and <b>SendOutgoingCommands</b> in your application Thread-context.
	<b>StopThread</b>	This method immediately closes a connection (pure client side) and ends related listening Threads.
	<b>TrafficStatsReset</b>	Creates new instances of <b>TrafficStats</b> and starts a new timer for those.
	<b>VitalStatsToString</b>	Returns a string of the most interesting connection statistics. When you have issues on the client side, these might contain hints about the issue's cause.


**LitePeer Class**






	Name	Description
	<b>OpChangeGroups</b>	Operation to handle this client's interest groups (for <b>events</b> in room).
	<b>OpGetProperties</b>	Gets all properties of the game and each actor.
	<b>OpGetPropertiesOfActor</b>	Gets selected properties of some actors.
	<b>OpGetPropertiesOfGame</b>	Gets selected properties of current game.
	<b>OpJoin</b>	This operation will join an existing room by name or create one if the name is not in use yet. Rooms (or games) are simply identified by name. We assume that users always want to get into a room - no matter if it existed before or not, so it might be a new one. If you want to make sure a room is created (new, empty), the client side might come up with a unique name for it (make sure the name was not taken yet). The application "Lite Lobby" lists room names and effectively allows the user to... <b>more</b>
	<b>OpLeave</b>	Leave operation of the Lite Application (also in Lite Lobby). Leaves a room / game, but keeps the connection. This operations triggers the event <b>LiteEventCode.Leave</b> for the remaining clients. The event includes the actorNumber of the player who left in key <b>LiteEventKey.ActorNr</b> .
	<b>OpRaiseEvent</b>	RaiseEvent tells the server to send an event to the other players within the same room.
	<b>OpSetPropertiesOfActor</b>	Attaches or updates properties of the specified actor.
	<b>OpSetPropertiesOfGame</b>	Attaches or updates properties of the current game.

**PhotonPeer Properties**

	Name	Description
	<b>ByteCountCurrentDispatch</b>	Gets the size of the dispatched event or operation-result in bytes. This value is set before <b>OnEvent()</b> or <b>OnOperationResponse()</b> is called (within <b>DispatchIncomingCommands()</b> ).

	<b>ByteCountLastOperation</b>	Gets the size of the last serialized operation call in bytes. The value includes all headers for this single operation but excludes those of UDP, Enet Package Headers and TCP.
	<b>BytesIn</b>	Gets count of all bytes coming in (including headers, excluding UDP/TCP overhead)
	<b>BytesOut</b>	Gets count of all bytes going out (including headers, excluding UDP/TCP overhead)
	<b>ChannelCount</b>	Gets / sets the number of channels available in UDP connections with Photon. Photon Channels are only supported for UDP. The default ChannelCount is 2. Channel IDs start with 0 and 255 is a internal channel.
	<b>CommandBufferSize</b>	Initial size internal lists for incoming/outgoing commands (reliable and unreliable).
	<b>CrcEnabled</b>	While not connected, this controls if the next connection(s) should use a per-package CRC checksum.
	<b>DebugOut</b>	Sets the level (and amount) of debug output provided by the library.
	<b>DisconnectTimeout</b>	<p>Milliseconds after which a reliable UDP command triggers a timeout disconnect, unless acknowledged by server. This value currently only affects UDP connections. DisconnectTimeout is not an exact value for a timeout. The exact timing of the timeout depends on the frequency of <a href="#">Service()</a> calls and commands that are sent with long roundtrip-times and variance are checked less often for re-sending!</p> <p>DisconnectTimeout and <a href="#">SentCountAllowance</a> are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time. Default: 10000 ms.</p>
	<b>HttpUrlParameters</b>	This string is added as simple GET parameters to the end of the server url + peerID combination. Include a starting "&" in your parameters as the peerID is always added to the <a href="#">ServerAddress</a> . This value will be added to all following (http) operations, so make sure to clear it if needed. Can be applied even to "connect" (set before calling connect, clear / reset when connection is established). Set before making a Operation call and don't forget to clear or set it again before the next operation.
	<b>IsEncryptionAvailable</b>	This property is set internally, when OpExchangeKeysForEncryption successfully finished. While it's true, encryption can be used for operations.
	<b>IsSendingOnlyAcks</b>	While true, the peer will not send any other commands except ACKs (used in UDP connections).
 V	<b>IsSimulationEnabled</b>	Gets or sets the network simulation "enabled" setting. Changing this value also locks this peer's sending and when setting false, the internally used queues are executed (so setting to false can take some cycles).
	<b>LimitOfUnreliableCommands</b>	Limits the queue of received unreliable commands within <a href="#">DispatchIncomingCommands</a> before dispatching them. This works only in UDP. This limit is applied when you call <a href="#">DispatchIncomingCommands</a> . If this client (already) received more than LimitOfUnreliableCommands, it will throw away the older ones instead of dispatching them. This can produce bigger gaps for unreliable commands but your client catches up faster.
	<b>Listener</b>	Gets the <a href="#">IPhotonPeerListener</a> of this instance (set in constructor). Can be used in derived classes for Listener.DebugReturn().
	<b>LocalMsTimestampDelegate</b>	This setter for the (local-) timestamp delegate replaces the default Environment.TickCount with any equal function.
	<b>LocalTimeInMilliseconds</b>	Gets a local timestamp in milliseconds by calling <a href="#">SupportClass.GetTickCount()</a> . See <a href="#">LocalMsTimestampDelegate</a> .
	<b>MaximumTransferUnit</b>	The Maximum Trasfer Unit (MTU) defines the (network-level) packet-content size that is guaranteed to arrive at the server in one piece. The Photon <a href="#">Protocol</a> uses this size to split larger data into packets and for receive-buffers of packets.

	<b>NetworkSimulationSettings</b>	Gets the settings for built-in Network Simulation for this peer instance while <b>IsSimulationEnabled</b> will enable or disable them. Once obtained, the settings can be modified by changing the properties.
	<b>OutgoingStreamBufferSize</b>	Defines the initial size of an internally used MemoryStream for Tcp. The MemoryStream is used to aggregate operation into (less) send calls, which uses less resources.
	<b>PacketLossByCrc</b>	Count of packages dropped due to failed CRC checks for this connection.
	<b>PeerID</b>	This peer's ID as assigned by the server or 0 if not using UDP. Will be 0xFFFF before the client connects.
	<b>PeerState</b>	This is the (low level) state of the connection to the server of a <b>PhotonPeer</b> . It is managed internally and read-only.
	<b>QueuedIncomingCommands</b>	Count of all currently received but not-yet-Dispatched reliable commands ( <b>events</b> and operation results) from all channels.
	<b>QueuedOutgoingCommands</b>	Count of all commands currently queued as outgoing, including all channels and reliable, unreliable.
	<b>ResentReliableCommands</b>	Count of commands that got repeated (due to local repeat-timing before an ACK was received).
	<b>RoundTripTime</b>	Time until a reliable command is acknowledged by the server. The value measures network latency and for UDP it includes the server's ACK-delay (setting in config). In TCP, there is no ACK-delay, so the value is slightly lower (if you use default settings for Photon). RoundTripTime is updated constantly. Every reliable command will contribute a fraction to this value. This is also the approximate time until a raised event reaches another client or until an operation result is available.
	<b>RoundTripTimeVariance</b>	Changes of the roundtrip time as variance value. Gives a hint about how much the time is changing.
	<b>SentCountAllowance</b>	Number of send retries before a peer is considered lost/disconnected. Default: 5. The initial timeout countdown of a command is calculated by the current roundTripTime + 4 * roundTripTimeVariance. Please note that the timeout span until a command will be resent is not constant, but based on the roundtrip time at the initial sending, which will be doubled with every failed retry. <b>DisconnectTimeout</b> and <b>SentCountAllowance</b> are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time.
	<b>ServerAddress</b>	The server address which was used in <b>PhotonPeer.Connect()</b> or null (before <b>Connect()</b> was called).
	<b>ServerTimeInMilliseconds</b>	Approximated Environment.TickCount value of server (while connected).
	<b>SocketImplementation</b>	Can be used to set a UDP <b>IPhotonSocket</b> implementation at runtime (before connecting) in compatible DLL builds.
	<b>TcpConnectionPrefix</b>	The bytes set with this property will be sent as first bytes in any established connection.
	<b>TimePingInterval</b>	Sets the milliseconds without reliable command before a ping command (reliable) will be sent (Default: 1000ms). The ping command is used to keep track of the connection in case the client does not send reliable commands by itself. A ping (or reliable commands) will update the <b>RoundTripTime</b> calculation.
	<b>TimestampOfLastSocketReceive</b>	Stores timestamp of the last time anything (!) was received from the server (including low level Ping and ACKs but also <b>events</b> and operation-returns). This is not the time when something was dispatched. If you enable NetworkSimulation, this value is affected as well.
	<b>TrafficStatsElapsedMs</b>	Returns the count of milliseconds the stats are enabled for tracking.
	<b>TrafficStatsEnabled</b>	Enables the traffic statistics of a peer: <b>TrafficStatsIncoming</b> , <b>TrafficStatsOutgoing</b> and <b>TrafficStatsGameLevel</b> (nothing else). Default value: false (disabled).

	<b>TrafficStatsGameLevel</b>	Gets a statistic of incoming and outgoing traffic, split by operation, operation-result and event. <b>Operations</b> are outgoing traffic, results and <b>events</b> are incoming. Includes the per-command header sizes (Udp: Enet Command Header or Tcp: Message Header).
	<b>TrafficStatsIncoming</b>	Gets the byte-count of incoming "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.
	<b>TrafficStatsOutgoing</b>	Gets the byte-count of outgoing "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.
	<b>UsedProtocol</b>	The protocol this Peer uses to connect to Photon.
	<b>WarningSize</b>	The WarningSize is used test all message queues for congestion (in and out, reliable and unreliable). OnStatusChanged will be called with a warning if a queue holds WarningSize commands or a multiple of it. Default: 100.

## 2.1.7.1 LitePeer Constructor

### 2.1.7.1.1 LitePeer.LitePeer Constructor ()

Creates a LitePeer instance to connect and communicate with a Photon server.

Uses UDP as protocol (except in the Silverlight library).

C#

```
protected LitePeer();
```

### 2.1.7.1.2 LitePeer.LitePeer Constructor (ConnectionProtocol)

Creates a LitePeer instance to connect and communicate with a Photon server.

C#

```
protected LitePeer(ConnectionProtocol protocolType);
```

### 2.1.7.1.3 LitePeer.LitePeer Constructor (IPhotonPeerListener)

Creates a LitePeer instance to connect and communicate with a Photon server.

Uses UDP as protocol (except in the Silverlight library).

C#

```
public LitePeer(IPhotonPeerListener listener);
```

Parameters

Parameters	Description
IPhotonPeerListener listener	Your <a href="#">IPhotonPeerListener</a> implementation.

### 2.1.7.1.4 LitePeer.LitePeer Constructor (IPhotonPeerListener, ConnectionProtocol)

Creates a LitePeer instance to communicate with Photon with your selection of protocol. We recommend UDP.

C#

```
public LitePeer(IPhotonPeerListener listener, ConnectionProtocol protocolType);
```

**Parameters**

Parameters	Description
<code>IPhotonPeerListener listener</code>	Your <a href="#">IPhotonPeerListener</a> implementation.
<code>ConnectionProtocol protocolType</code>	<a href="#">Protocol</a> to use to connect to Photon.

## 2.1.7.2 LitePeer Methods

### 2.1.7.2.1 LitePeer.OpChangeGroups Method

Operation to handle this client's interest groups (for [events](#) in room).

**C#**

```
public virtual bool OpChangeGroups(byte[] groupsToRemove, byte[] groupsToAdd);
```

**Parameters**

Parameters	Description
<code>byte[] groupsToRemove</code>	Groups to remove from interest. Null will not leave any. A <code>byte[0]</code> will remove all.
<code>byte[] groupsToAdd</code>	Groups to add to interest. Null will not add any. A <code>byte[0]</code> will add all current.

**Remarks**

Note the difference between passing null and `byte[0]`: null won't add/remove any groups. `byte[0]` will add/remove all (existing) groups. First, removing groups is executed. This way, you could leave all groups and join only the ones provided.

### 2.1.7.2.2 LitePeer.OpGetProperties Method

Gets all properties of the game and each actor.

**C#**

```
public virtual bool OpGetProperties(byte channelId);
```

**Parameters**

Parameters	Description
<code>byte channelId</code>	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

### 2.1.7.2.3 OpGetPropertiesOfActor Method

#### 2.1.7.2.3.1 LitePeer.OpGetPropertiesOfActor Method (int[], byte[], byte)

Gets selected properties of some actors.

**C#**

```
public virtual bool OpGetPropertiesOfActor(int[] actorNrList, byte[] properties, byte channelId);
```

**Parameters**

Parameters	Description
int[] actorNrList	optional, a list of actornumbers to get the properties of
byte[] properties	array of property keys to fetch. optional (can be null).
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

**2.1.7.2.3.2 LitePeer.OpGetPropertiesOfActor Method (int[], string[], byte)**

Gets selected properties of an actor.

**C#**

```
public virtual bool OpGetPropertiesOfActor(int[] actorNrList, string[] properties, byte channelId);
```

**Parameters**

Parameters	Description
int[] actorNrList	optional, a list of actornumbers to get the properties of
string[] properties	optional, array of property keys to fetch
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

**2.1.7.2.4 OpGetPropertiesOfGame Method****2.1.7.2.4.1 LitePeer.OpGetPropertiesOfGame Method (byte[], byte)**

Gets selected properties of current game.

**C#**

```
public virtual bool OpGetPropertiesOfGame(byte[] properties, byte channelId);
```

**Parameters**

Parameters	Description
byte[] properties	array of property keys to fetch. optional (can be null).
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

**2.1.7.2.4.2 LitePeer.OpGetPropertiesOfGame Method (string[], byte)**

Gets selected properties of current game.

**C#**

```
public virtual bool OpGetPropertiesOfGame(string[] properties, byte channelId);
```

**Parameters**

Parameters	Description
string[] properties	array of property keys to fetch. optional (can be null).
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

## 2.1.7.2.5 OpJoin Method

### 2.1.7.2.5.1 LitePeer.OpJoin Method (string)

This operation will join an existing room by name or create one if the name is not in use yet.

Rooms (or games) are simply identified by name. We assume that users always want to get into a room - no matter if it existed before or not, so it might be a new one. If you want to make sure a room is created (new, empty), the client side might come up with a unique name for it (make sure the name was not taken yet).

The application "Lite Lobby" lists room names and effectively allows the user to select a distinct one.

Each actor (a.k.a. player) in a room will get [events](#) that are raised for the room by any player.

To distinguish the actors, each gets a consecutive actornumber. This is used in [events](#) to mark who triggered the event. A client finds out it's own actornumber in the return callback for operation Join. Number 1 is the lowest actornumber in each room and the client with that actornumber created the room.

Each client could easily send custom data around. If the data should be available to newcomers, it makes sense to use Properties.

Joining a room will trigger the event [LiteEventCode.Join](#), which contains the list of actorNumbers of current players inside the room ([LiteEventKey.ActorList](#)). This also gives you a count of current players.

**C#**

```
public virtual bool OpJoin(string gameName);
```

**Parameters**

Parameters	Description
string gameName	Any identifying name for a room / game.

**Returns**

If operation could be enqueued for sending

### 2.1.7.2.5.2 LitePeer.OpJoin Method (string, Hashtable, Hashtable, bool)

This operation will join an existing room by name or create one if the name is not in use yet.

Rooms (or games) are simply identified by name. We assume that users always want to get into a room - no matter if it existed before or not, so it might be a new one. If you want to make sure a room is created (new, empty), the client side might come up with a unique name for it (make sure the name was not taken yet).

The application "Lite Lobby" lists room names and effectively allows the user to select a distinct one.

Each actor (a.k.a. player) in a room will get [events](#) that are raised for the room by any player.



To distinguish the actors, each gets a consecutive actornumber. This is used in **events** to mark who triggered the event. A client finds out it's own actornumber in the return callback for operation Join. Number 1 is the lowest actornumber in each room and the client with that actornumber created the room.

Each client could easily send custom data around. If the data should be available to newcomers, it makes sense to use Properties.

Joining a room will trigger the event **LiteEventCode.Join**, which contains the list of actorNumbers of current players inside the room (**LiteEventKey.ActorList**). This also gives you a count of current players.

#### C#

```
public virtual bool OpJoin(string gameName, Hashtable gameProperties, Hashtable actorProperties, bool broadcastActorProperties);
```

#### Parameters

Parameters	Description
string gameName	Any identifying name for a room / game.
Hashtable gameProperties	optional, set of game properties, by convention: only used if game is new/created
Hashtable actorProperties	optional, set of actor properties
bool broadcastActorProperties	optional, broadcast actor properties in join-event

#### Returns

If operation could be enqueued for sending

### 2.1.7.2.6 LitePeer.OpLeave Method

Leave operation of the Lite Application (also in Lite Lobby). Leaves a room / game, but keeps the connection. This operations triggers the event **LiteEventCode.Leave** for the remaining clients. The event includes the actorNumber of the player who left in key **LiteEventKey.ActorNr**.

#### C#

```
public virtual bool OpLeave();
```

#### Returns

Consecutive invocationID of the OP. Will throw Exception if not connected.

### 2.1.7.2.7 OpRaiseEvent Method

#### 2.1.7.2.7.1 LitePeer.OpRaiseEvent Method (byte, Hashtable, bool)

RaiseEvent tells the server to send an event to the other players within the same room.

#### C#

```
public virtual bool OpRaiseEvent(byte eventCode, Hashtable customEventContent, bool sendReliable);
```

#### Parameters

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
Hashtable customEventContent	Custom data you want to send along (use null, if none).
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).

#### Returns

If operation could be enqueued for sending



**Remarks**

This method is described in one of its overloads.

**2.1.7.2.7.2 LitePeer.OpRaiseEvent Method (byte, Hashtable, bool, byte)**

RaiseEvent tells the server to send an event to the other players within the same room.

**C#**

```
public virtual bool OpRaiseEvent(byte eventCode, Hashtable customEventContent, bool
sendReliable, byte channelId);
```

**Parameters**

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
Hashtable customEventContent	Custom data you want to send along (use null, if none).
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).
byte channelId	Number of channel (sequence) to use (starting with 0).

**Returns**

If operation could be enqueued for sending.

**Remarks**

Type and content of the event can be defined by the client side at will. The server only forwards the content and eventCode to others in the same room.

The eventCode should be used to define the event's type and content respectively./// Lite and Loadbalancing are using a few eventCode values already but those start with 255 and go down. Your eventCodes can start at 1, going up.

The customEventContent is a Hashtable with any number of key-value pairs of [serializable datatypes](#) or null. Receiving clients can access this Hashtable as Parameter [LiteEventKey.Data](#) (see below).

RaiseEvent can be used reliable or unreliable. Both result in ordered [events](#) but the unreliable ones might be lost and allow gaps in the resulting event sequence. On the other hand, they cause less overhead and are optimal for data that is replaced soon.

Like all operations, RaiseEvent is not done immediately but when you call [SendOutgoingCommands](#).

It is recommended to keep keys (and data) as simple as possible (e.g. byte or short as key), as the data is typically sent multiple times per second. This easily adds up to a huge amount of data otherwise.

**Example**

```
//send some position data (using byte-keys, as they are small):

Hashtable evInfo = new Hashtable();
Player local = (Player)players[playerLocalID];
evInfo.Add((byte)STATUS_PLAYER_POS_X, (int)local.posX);
evInfo.Add((byte)STATUS_PLAYER_POS_Y, (int)local.posY);

peer.OpRaiseEvent(EV_MOVE, evInfo, true); //EV_MOVE = (byte)1

//receive this custom event in OnEvent():
Hashtable data = (Hashtable)photonEvent[LiteEventKey.Data];
switch (eventCode) {
    case EV_MOVE: //1 in this sample
        p = (Player)players[actorNr];
        if (p != null) {
            p.posX = (int)data[(byte)STATUS_PLAYER_POS_X];
            p.posY = (int)data[(byte)STATUS_PLAYER_POS_Y];
        }
        break;
```

**Events** from the Photon Server are internally buffered until they are **Dispatched**, just like **OperationResults**.

### 2.1.7.2.7.3 LitePeer.OpRaiseEvent Method (byte, Hashtable, bool, byte, EventCaching, ReceiverGroup)

Calls operation **RaiseEvent** on the server, with full control of event-caching and the target receivers.

**C#**

```
public virtual bool OpRaiseEvent(byte eventCode, Hashtable customEventContent, bool sendReliable, byte channelId, EventCaching cache, ReceiverGroup receivers);
```

**Parameters**

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
Hashtable customEventContent	Custom data you want to send along (use null, if none).
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).
byte channelId	Number of channel to use (starting with 0).
EventCaching cache	<b>Events</b> can be cached (merged and removed) for players joining later on.
ReceiverGroup receivers	Controls who should get this event.

**Returns**

If operation could be enqueued for sending.

**Remarks**

This method is described in one of its overloads.

The cache parameter defines if and how this event will be cached server-side. Per event-code, your client can store **events** and update them and will send cached **events** to players joining the same room.

The option **EventCaching.DoNotCache** matches the default behaviour of **RaiseEvent**. The option **EventCaching.MergeCache** will merge the customEventContent into existing one. Values in the customEventContent Hashtable can be null to remove existing values.

With the receivers parameter, you can chose who gets this event: Others (default), All (includes you as sender) or MasterClient. The MasterClient is the connected player with the lowest ActorNumber in this room. This player could get some privileges, if needed.

Read more about Cached **Events** in the DevNet: <http://doc.exitgames.com>

### 2.1.7.2.7.4 LitePeer.OpRaiseEvent Method (byte, Hashtable, bool, byte, int[])

**RaiseEvent** tells the server to send an event to the other players within the same room.

**C#**

```
public virtual bool OpRaiseEvent(byte eventCode, Hashtable customEventContent, bool sendReliable, byte channelId, int[] targetActors);
```

**Parameters**

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
Hashtable customEventContent	Custom data you want to send along (use null, if none).
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).
byte channelId	Number of channel to use (starting with 0).

<code>int[] targetActors</code>	List of actorNumbers that receive this event.
---------------------------------	---

### Returns

If operation could be enqueued for sending.

### Remarks

This method is described in one of its overloads.

This variant has an optional list of targetActors. Use this to send the event only to specific actors in the same room, each identified by an actorNumber (or ID).

This can be useful to implement private messages inside a room or similar.

## 2.1.7.2.7.5 LitePeer.OpRaiseEvent Method (byte, bool, object)

Send an event with custom code/type and any content to the other players in the same room.

### C#

```
public virtual bool OpRaiseEvent(byte eventCode, bool sendReliable, object customEventContent);
```

### Parameters

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).
object customEventContent	Any serializable datatype (including Hashtable like the other OpRaiseEvent overloads).

### Returns

If operation could be enqueued for sending. Sent when calling: [Service](#) or [SendOutgoingCommands](#).

### Remarks

This override explicitly uses another parameter order to not mix it up with the implementation for Hashtable only.

## 2.1.7.2.7.6 LitePeer.OpRaiseEvent Method (byte, bool, object, byte, EventCaching, int[], ReceiverGroup, byte)

Send an event with custom code/type and any content to the other players in the same room.

### C#

```
public virtual bool OpRaiseEvent(byte eventCode, bool sendReliable, object customEventContent, byte channelId, EventCaching cache, int[] targetActors, ReceiverGroup receivers, byte interestGroup);
```

### Parameters

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).
object customEventContent	Any serializable datatype (including Hashtable like the other OpRaiseEvent overloads).
byte channelId	Command sequence in which this command belongs. Must be less than value of <a href="#">ChannelCount</a> property. Default: 0.
EventCaching cache	Affects how the server will treat the event caching-wise. Can cache <a href="#">events</a> for players joining later on or remove previously cached <a href="#">events</a> . Default: DoNotCache.

int[] targetActors	List of ActorNumbers (in this room) to send the event to. Overrides caching. Default: null.
ReceiverGroup receivers	Defines a target-player group. Default: Others.
byte interestGroup	Defines to which interest group the event is sent. Players can subscribe or unsubscribe to groups. Group 0 is always sent to all. Default: 0.

**Returns**

If operation could be enqueued for sending. Sent when calling: [Service](#) or [SendOutgoingCommands](#).

**Remarks**

This override explicitly uses another parameter order to not mix it up with the implementation for Hashtable only.

**2.1.7.2.7.7 LitePeer.OpRaiseEvent Method (byte, byte, Hashtable, bool)**

Send your custom data as event to an "interest group" in the current Room.

**C#**

```
public virtual bool OpRaiseEvent(byte eventCode, byte interestGroup, Hashtable customEventContent, bool sendReliable);
```

**Parameters**

Parameters	Description
byte eventCode	Identifies this type of event (and the content). Your game's event codes can start with 0.
byte interestGroup	The ID of the interest group this event goes to (exclusively). Group 0 sends to all.
Hashtable customEventContent	Custom data you want to send along (use null, if none).
bool sendReliable	If this event has to arrive reliably (potentially repeated if it's lost).

**Returns**

If operation could be enqueued for sending

**Remarks**

No matter if reliable or not, when an event is sent to a interest Group, some users won't get this data. Clients can control the groups they are interested in by using [OpChangeGroups](#).

**2.1.7.2.8 LitePeer.OpSetPropertiesOfActor Method**

Attaches or updates properties of the specified actor.

**C#**

```
public virtual bool OpSetPropertiesOfActor(int actorNr, Hashtable properties, bool broadcast, byte channelId);
```

**Parameters**

Parameters	Description
int actorNr	the actorNr is used to identify a player/peer in a game
Hashtable properties	Hashtable containing the properties to add or update.
bool broadcast	true will trigger an event LiteEventKey.PropertiesChanged with the updated properties in it
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

### 2.1.7.2.9 LitePeer.OpSetPropertiesOfGame Method

Attaches or updates properties of the current game.

**C#**

```
public virtual bool OpSetPropertiesOfGame(Hashtable properties, bool broadcast, byte channelId);
```

**Parameters**

Parameters	Description
Hashtable properties	hashtable containing the properties to add or overwrite
bool broadcast	true will trigger an event LiteEventKey.PropertiesChanged with the updated properties in it
byte channelId	Number of channel to use (starting with 0).

**Returns**

If operation could be enqueued for sending

**Remarks**

Please read the general description of [Properties on Photon](#).

## 2.1.8 NetworkSimulationSet Class

A set of network simulation settings, enabled (and disabled) by [PhotonPeer.IsSimulationEnabled](#).

**C#**


```
public class NetworkSimulationSet;
```

**Remarks**

For performance reasons, the lag and jitter settings can't be produced exactly. In some cases, the resulting lag will be up to 20ms bigger than the lag settings. Even if all settings are 0, simulation will be used. Set [PhotonPeer.IsSimulationEnabled](#) to false to disable it if no longer needed.

All lag, jitter and loss is additional to the current, real network conditions. If the network is slow in reality, this will add even more lag. The jitter values will affect the lag positive and negative, so the lag settings describe the medium lag even with jitter. The jitter influence is: [-jitter..+jitter]. Packets "lost" due to [OutgoingLossPercentage](#) count for BytesOut and [LostPackagesOut](#). Packets "lost" due to [IncomingLossPercentage](#) count for BytesIn and [LostPackagesIn](#).









**NetworkSimulationSet Fields**

	Name	Description
	<a href="#">NetSimManualResetEvent</a>	This is NetSimManualResetEvent, a member of class NetworkSimulationSet.

**NetworkSimulationSet Methods**

	Name	Description
	<a href="#">ToString</a>	This is ToString, a member of class NetworkSimulationSet.

**NetworkSimulationSet Properties**

	Name	Description
	<b>IncomingJitter</b>	Randomizes <b>IncomingLag</b> by [-IncomingJitter..+IncomingJitter]. Default: 0.
	<b>IncomingLag</b>	Incoming packages delay in ms. Default: 100.
	<b>IncomingLossPercentage</b>	Percentage of incoming packets that should be lost. Between 0..100. Default: 1. TCP ignores this setting.
	<b>LostPackagesIn</b>	Counts how many incoming packages actually got lost. TCP connections ignore loss and this stays 0.
	<b>LostPackagesOut</b>	Counts how many outgoing packages actually got lost. TCP connections ignore loss and this stays 0.
	<b>OutgoingJitter</b>	Randomizes <b>OutgoingLag</b> by [-OutgoingJitter..+OutgoingJitter]. Default: 0.
	<b>OutgoingLag</b>	Outgoing packages delay in ms. Default: 100.
	<b>OutgoingLossPercentage</b>	Percentage of outgoing packets that should be lost. Between 0..100. Default: 1. TCP ignores this setting.

**2.1.8.1 NetworkSimulationSet Fields****2.1.8.1.1 NetworkSimulationSet.NetSimManualResetEvent Field****C#**

```
public readonly ManualResetEvent NetSimManualResetEvent = new ManualResetEvent(false);
```

**Description**

This is NetSimManualResetEvent, a member of class NetworkSimulationSet.

**2.1.8.2 NetworkSimulationSet Methods****2.1.8.2.1 NetworkSimulationSet.ToString Method****C#**

```
public override string ToString();
```

**Description**

This is ToString, a member of class NetworkSimulationSet.

**2.1.8.3 NetworkSimulationSet Properties****2.1.8.3.1 NetworkSimulationSet.IncomingJitter Property**

Randomizes **IncomingLag** by [-IncomingJitter..+IncomingJitter]. Default: 0.

**C#**

```
public int IncomingJitter;
```

**2.1.8.3.2 NetworkSimulationSet.IncomingLag Property**

Incoming packages delay in ms. Default: 100.

**C#**

```
public int IncomingLag;
```

### 2.1.8.3.3 NetworkSimulationSet.IncomingLossPercentage Property

Percentage of incoming packets that should be lost. Between 0..100. Default: 1. TCP ignores this setting.

**C#**

```
public int IncomingLossPercentage;
```

### 2.1.8.3.4 NetworkSimulationSet.LostPackagesIn Property

Counts how many incoming packages actually got lost. TCP connections ignore loss and this stays 0.

**C#**

```
public int LostPackagesIn;
```

### 2.1.8.3.5 NetworkSimulationSet.LostPackagesOut Property

Counts how many outgoing packages actually got lost. TCP connections ignore loss and this stays 0.

**C#**

```
public int LostPackagesOut;
```

### 2.1.8.3.6 NetworkSimulationSet.OutgoingJitter Property

Randomizes [OutgoingLag](#) by [-OutgoingJitter..+OutgoingJitter]. Default: 0.

**C#**

```
public int OutgoingJitter;
```

### 2.1.8.3.7 NetworkSimulationSet.OutgoingLag Property

Outgoing packages delay in ms. Default: 100.

**C#**

```
public int OutgoingLag;
```

### 2.1.8.3.8 NetworkSimulationSet.OutgoingLossPercentage Property

Percentage of outgoing packets that should be lost. Between 0..100. Default: 1. TCP ignores this setting.

**C#**

```
public int OutgoingLossPercentage;
```

---

## 2.1.9 OperationRequest Class

Container for an Operation request, which is a code and parameters.

**C#**

```
public class OperationRequest;
```

#### Remarks

On the lowest level, Photon only allows byte-typed keys for operation parameters. The values of each such parameter can be any serializable datatype: byte, int, hashtable and many more.

**OperationRequest Fields**

	Name	Description
◆	<a href="#">OperationCode</a>	Byte-typed code for an operation - the short identifier for the server's method to call.
◆	<a href="#">Parameters</a>	The parameters of the operation - each identified by a byte-typed code in Photon.

## 2.1.9.1 OperationRequest Fields

### 2.1.9.1.1 OperationRequest.OperationCode Field

Byte-typed code for an operation - the short identifier for the server's method to call.

**C#**

```
public byte OperationCode;
```

### 2.1.9.1.2 OperationRequest.Parameters Field

The parameters of the operation - each identified by a byte-typed code in Photon.

**C#**

```
public Dictionary<byte, object> Parameters;
```

## 2.1.10 OperationResponse Class

Contains the server's response for an operation called by this peer. The indexer of this class actually provides access to the [Parameters](#) Dictionary.

**C#**

```
public class OperationResponse;
```

**Remarks**

The [OperationCode](#) defines the type of operation called on Photon and in turn also the [Parameters](#) that are set in the request. Those are provided as Dictionary with byte-keys. There are pre-defined constants for various codes defined in the Lite application. Check: [LiteOpCode](#), [LiteOpKey](#), etc.

An operation's request is summarized by the [ReturnCode](#): a short typed code for "Ok" or some different result. The code's meaning is specific per operation. An optional [DebugMessage](#) can be provided to simplify debugging.




Each call of an operation gets an ID, called the "invocID". This can be matched to the IDs returned with any operation calls. This way, an application could track if a certain OpRaiseEvent call was successful.

**OperationResponse Fields**

	Name	Description
◆	<a href="#">DebugMessage</a>	An optional string sent by the server to provide readable feedback in error-cases. Might be null.
◆	<a href="#">OperationCode</a>	The code for the operation called initially (by this peer).
◆	<a href="#">Parameters</a>	A Dictionary of values returned by an operation, using byte-typed keys per value.
◆	<a href="#">ReturnCode</a>	A code that "summarizes" the operation's success or failure. Specific per operation. 0 usually means "ok".



OperationResponse Methods

	Name	Description
	<a href="#">this</a>	Alternative access to the <a href="#">Parameters</a> , which wraps up a TryGetValue() call on the <a href="#">Parameters</a> Dictionary.
	<a href="#">ToString</a>	ToString() override.
	<a href="#">ToStringFull</a>	Extensive output of operation results.

2.1.10.1 OperationResponse Fields

2.1.10.1.1 OperationResponse.DebugMessage Field

An optional string sent by the server to provide readable feedback in error-cases. Might be null.

C#

```
public string DebugMessage;
```

2.1.10.1.2 OperationResponse.OperationCode Field

The code for the operation called initially (by this peer).

C#

```
public byte OperationCode;
```

Remarks

Use enums or constants to be able to handle those codes, like [LiteOpCode](#) does.

2.1.10.1.3 OperationResponse.Parameters Field

A Dictionary of values returned by an operation, using byte-typed keys per value.

C#

```
public Dictionary<byte, object> Parameters;
```

2.1.10.1.4 OperationResponse.ReturnCode Field

A code that "summarizes" the operation's success or failure. Specific per operation. 0 usually means "ok".

C#

```
public short ReturnCode;
```

2.1.10.2 OperationResponse Methods

2.1.10.2.1 OperationResponse.this Indexer

Alternative access to the [Parameters](#), which wraps up a TryGetValue() call on the [Parameters](#) Dictionary.

C#

```
public object this[byte parameterCode];
```

Parameters

Parameters	Description
byte parameterCode	The byte-code of a returned value.

**Returns**

The value returned by the server, or null if the key does not exist in [Parameters](#).

### 2.1.10.2.2 OperationResponse.ToString Method

ToString() override.

**C#**

```
public override string ToString();
```

**Returns**

Relatively short output of OpCode and returnCode.

### 2.1.10.2.3 OperationResponse.ToStringFull Method

Extensive output of operation results.

**C#**

```
public string ToStringFull();
```

**Returns**

To be used in debug situations only, as it returns a string for each value.

## 2.1.11 PeerBase Class


**C#**

```
public abstract class PeerBase;
```








**Description**

This is class PeerBase.



**PeerBase Enumerations**



	Name	Description
	<a href="#">ConnectionStateValue</a>	This is the replacement for the const values used in eNet like: PS_DISCONNECTED, PS_CONNECTED, etc.

**PeerBase Fields**

	Name	Description
	<a href="#">ByteCountCurrentDispatch</a>	This is ByteCountCurrentDispatch, a member of class PeerBase.
	<a href="#">ByteCountLastOperation</a>	This is ByteCountLastOperation, a member of class PeerBase.
	<a href="#">CryptoProvider</a>	This is CryptoProvider, a member of class PeerBase.
	<a href="#">SerializeMemStream</a>	This is SerializeMemStream, a member of class PeerBase.
	<a href="#">TrafficStatsGameLevel</a>	This is TrafficStatsGameLevel, a member of class PeerBase.
	<a href="#">TrafficStatsIncoming</a>	This is TrafficStatsIncoming, a member of class PeerBase.
	<a href="#">TrafficStatsOutgoing</a>	This is TrafficStatsOutgoing, a member of class PeerBase.

**PeerBase Properties**

	Name	Description
	<a href="#">NetworkSimulationSettings</a>	Gets the currently used settings for the built-in network simulation. Please check the description of <a href="#">NetworkSimulationSet</a> for more details.
	<a href="#">PeerID</a>	This is PeerID, a member of class PeerBase.

	<b>TrafficStatsEnabled</b>	Enables or disables collection of statistics. Setting this to true, also starts the stopwatch to measure the timespan the stats are collected.
	<b>TrafficStatsEnabledTime</b>	This is TrafficStatsEnabledTime, a member of class PeerBase.

## 2.1.11.1 PeerBase Enumerations

### 2.1.11.1.1 PeerBase.ConnectionStateValue Enumeration

This is the replacement for the const values used in eNet like: PS\_DISCONNECTED, PS\_CONNECTED, etc.

**C#**

```
public enum ConnectionStateValue : byte {
    Disconnected = 0,
    Connecting = 1,
    Connected = 3,
    Disconnecting = 4,
    AcknowledgingDisconnect = 5,
    Zombie = 6
}
```

**Members**

Members	Description
Disconnected = 0	No connection is available. Use connect.
Connecting = 1	Establishing a connection already. The app should wait for a status callback.
Connected = 3	The low level connection with Photon is established. On connect, the library will automatically send an Init package to select the application it connects to (see also <a href="#">PhotonPeer.Connect()</a> ). When the Init is done, <a href="#">IPhotonPeerListener.OnStatusChanged()</a> is called with connect.
Disconnecting = 4	Connection going to be ended. Wait for status callback.
AcknowledgingDisconnect = 5	Acknowledging a disconnect from Photon. Wait for status callback.
Zombie = 6	Connection not properly disconnected.

## 2.1.11.2 PeerBase Fields

### 2.1.11.2.1 PeerBase.ByteCountCurrentDispatch Field

**C#**

```
public int ByteCountCurrentDispatch;
```

**Description**

This is ByteCountCurrentDispatch, a member of class PeerBase.

### 2.1.11.2.2 PeerBase.ByteCountLastOperation Field

**C#**

```
public int ByteCountLastOperation;
```

**Description**

This is ByteCountLastOperation, a member of class PeerBase.

### 2.1.11.2.3 PeerBase.CryptoProvider Field

C#

```
public DiffieHellmanCryptoProvider CryptoProvider;
```

#### Description

This is CryptoProvider, a member of class PeerBase.

### 2.1.11.2.4 PeerBase.SerializeMemStream Field

C#

```
protected MemoryStream SerializeMemStream = new MemoryStream();
```

#### Description

This is SerializeMemStream, a member of class PeerBase.

### 2.1.11.2.5 PeerBase.TrafficStatsGameLevel Field

C#

```
public TrafficStatsGameLevel TrafficStatsGameLevel;
```

#### Description

This is TrafficStatsGameLevel, a member of class PeerBase.

### 2.1.11.2.6 PeerBase.TrafficStatsIncoming Field

C#

```
public TrafficStats TrafficStatsIncoming;
```

#### Description

This is TrafficStatsIncoming, a member of class PeerBase.

### 2.1.11.2.7 PeerBase.TrafficStatsOutgoing Field

C#

```
public TrafficStats TrafficStatsOutgoing;
```

#### Description

This is TrafficStatsOutgoing, a member of class PeerBase.

## 2.1.11.3 PeerBase Properties

### 2.1.11.3.1 PeerBase.NetworkSimulationSettings Property

Gets the currently used settings for the built-in network simulation. Please check the description of [NetworkSimulationSet](#) for more details.

C#

```
public NetworkSimulationSet NetworkSimulationSettings;
```

### 2.1.11.3.2 PeerBase.PeerID Property

C#

```
public virtual string PeerID;
```

#### Description

This is PeerID, a member of class PeerBase.

### 2.1.11.3.3 PeerBase.TrafficStatsEnabled Property

Enables or disables collection of statistics. Setting this to true, also starts the stopwatch to measure the timespan the stats are collected.

C#

```
public bool TrafficStatsEnabled;
```

### 2.1.11.3.4 PeerBase.TrafficStatsEnabledTime Property

C#

```
public long TrafficStatsEnabledTime;
```

#### Description

This is TrafficStatsEnabledTime, a member of class PeerBase.

---

## 2.1.12 PhotonPeer Class

Instances of the PhotonPeer class are used to connect to a Photon server and communicate with it.

C#


```
public class PhotonPeer;
```

#### Remarks














A PhotonPeer instance allows communication with the Photon Server, which in turn distributes messages to other PhotonPeer clients.

An application can use more than one PhotonPeer instance, which are treated as separate users on the server. Each should have its own listener instance, to separate the operations, callbacks and [events](#).



#### Methods



















	Name	Description
	<a href="#">PhotonPeer</a>	Creates a new PhotonPeer instance to communicate with Photon. Connection is UDP based, except for Silverlight.




















## PhotonPeer Methods

	Name	Description
	<b>Connect</b>	<p>This method does a DNS lookup (if necessary) and connects to the given serverAddress.</p> <p>The return value gives you feedback if the address has the correct format. If so, this starts the process to establish the connection itself, which might take a few seconds.</p> <p>When the connection is established, a callback to <b>IPhotonPeerListener.OnStatusChanged</b> will be done. If the connection can't be established, despite having a valid address, the OnStatusChanged is called with an error-value.</p> <p>The applicationName defines the application logic to use server-side and it should match the name of one of the apps in your server's config.</p> <p>By default,... <a href="#">more</a></p>
	<b>Disconnect</b>	This method initiates a mutual disconnect between this client and the server.
	<b>DispatchIncomingCommands</b>	<p>This method directly causes the callbacks for <a href="#">events</a>, responses and state changes within a <b>IPhotonPeerListener</b>.</p> <p>DispatchIncomingCommands only executes a single received command per call. If a command was dispatched, the return value is true and the method should be called again. This method is called by <b>Service()</b> until currently available commands are dispatched.</p>
	<b>EstablishEncryption</b>	This method creates a public key for this client and exchanges it with the server.
	<b>FetchServerTimestamp</b>	<p>This will fetch the server's timestamp and update the approximation for property ServerTimeInMilliseconds.</p> <p>The server time approximation will NOT become more accurate by repeated calls. Accuracy currently depends on a single roundtrip which is done as fast as possible.</p> <p>The command used for this is immediately acknowledged by the server. This makes sure the roundtrip time is low and the timestamp + roundtrip / 2 is close to the original value.</p>
	<b>OpCustom</b>	Allows the client to send any operation to the Photon Server by setting any opCode and the operation's parameters.
	<b>RegisterType</b>	Registers new types/classes for de/serialization and the fitting methods to call for this type.
	<b>SendAcksOnly</b>	This is SendAcksOnly, a member of class PhotonPeer.
	<b>SendOutgoingCommands</b>	This method creates a UDP/TCP package for outgoing commands (operations and acknowledgements) and sends them to the server. This method is also called by <b>Service()</b> .
	<b>Service</b>	This method excutes <b>DispatchIncomingCommands</b> and <b>SendOutgoingCommands</b> in your application Thread-context.
	<b>StopThread</b>	This method immediately closes a connection (pure client side) and ends related listening Threads.
	<b>TrafficStatsReset</b>	Creates new instances of <b>TrafficStats</b> and starts a new timer for those.
	<b>VitalStatsToString</b>	Returns a string of the most interesting connection statistics. When you have issues on the client side, these might contain hints about the issue's cause.





## PhotonPeer Properties

	Name	Description
	<b>ByteCountCurrentDispatch</b>	Gets the size of the dispatched event or operation-result in bytes. This value is set before OnEvent() or OnOperationResponse() is called (within <b>DispatchIncomingCommands()</b> ).
	<b>ByteCountLastOperation</b>	Gets the size of the last serialized operation call in bytes. The value includes all headers for this single operation but excludes those of UDP, Enet Package Headers and TCP.

	<b>BytesIn</b>	Gets count of all bytes coming in (including headers, excluding UDP/TCP overhead)
	<b>BytesOut</b>	Gets count of all bytes going out (including headers, excluding UDP/TCP overhead)
	<b>ChannelCount</b>	Gets / sets the number of channels available in UDP connections with Photon. Photon Channels are only supported for UDP. The default ChannelCount is 2. Channel IDs start with 0 and 255 is a internal channel.
	<b>CommandBufferSize</b>	Initial size internal lists for incoming/outgoing commands (reliable and unreliable).
	<b>CrcEnabled</b>	While not connected, this controls if the next connection(s) should use a per-package CRC checksum.
	<b>DebugOut</b>	Sets the level (and amount) of debug output provided by the library.
	<b>DisconnectTimeout</b>	<p>Milliseconds after which a reliable UDP command triggers a timeout disconnect, unless acknowledged by server. This value currently only affects UDP connections. DisconnectTimeout is not an exact value for a timeout. The exact timing of the timeout depends on the frequency of <b>Service()</b> calls and commands that are sent with long roundtrip-times and variance are checked less often for re-sending!</p> <p>DisconnectTimeout and <b>SentCountAllowance</b> are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time. Default: 10000 ms.</p>
	<b>HttpUriParameters</b>	This string is added as simple GET parameters to the end of the server url + peerID combination. Include a starting "&" in your parameters as the peerID is always added to the <b>ServerAddress</b> . This value will be added to all following (http) operations, so make sure to clear it if needed. Can be applied even to "connect" (set before calling connect, clear / reset when connection is established). Set before making a Operation call and don't forget to clear or set it again before the next operation.
	<b>IsEncryptionAvailable</b>	This property is set internally, when OpExchangeKeysForEncryption successfully finished. While it's true, encryption can be used for operations.
	<b>IsSendingOnlyAcks</b>	While true, the peer will not send any other commands except ACKs (used in UDP connections).
 	<b>IsSimulationEnabled</b>	Gets or sets the network simulation "enabled" setting. Changing this value also locks this peer's sending and when setting false, the internally used queues are executed (so setting to false can take some cycles).
	<b>LimitOfUnreliableCommands</b>	Limits the queue of received unreliable commands within <b>DispatchIncomingCommands</b> before dispatching them. This works only in UDP. This limit is applied when you call <b>DispatchIncomingCommands</b> . If this client (already) received more than LimitOfUnreliableCommands, it will throw away the older ones instead of dispatching them. This can produce bigger gaps for unreliable commands but your client catches up faster.
	<b>Listener</b>	Gets the <b>IPhotonPeerListener</b> of this instance (set in constructor). Can be used in derived classes for Listener.DebugReturn().
	<b>LocalMsTimestampDelegate</b>	This setter for the (local-) timestamp delegate replaces the default Environment.TickCount with any equal function.
	<b>LocalTimeInMilliseconds</b>	Gets a local timestamp in milliseconds by calling <b>SupportClass.GetTickCount()</b> . See <b>LocalMsTimestampDelegate</b> .
	<b>MaximumTransferUnit</b>	The Maximum Trasfer Unit (MTU) defines the (network-level) packet-content size that is guaranteed to arrive at the server in one piece. The Photon <b>Protocol</b> uses this size to split larger data into packets and for receive-buffers of packets.
	<b>NetworkSimulationSettings</b>	Gets the settings for built-in Network Simulation for this peer instance while <b>IsSimulationEnabled</b> will enable or disable them. Once obtained, the settings can be modified by changing the properties.

	<b>OutgoingStreamBufferSize</b>	Defines the initial size of an internally used MemoryStream for Tcp. The MemoryStream is used to aggregate operation into (less) send calls, which uses less resources.
	<b>PacketLossByCrc</b>	Count of packages dropped due to failed CRC checks for this connection.
	<b>PeerID</b>	This peer's ID as assigned by the server or 0 if not using UDP. Will be 0xFFFF before the client connects.
	<b>PeerState</b>	This is the (low level) state of the connection to the server of a PhotonPeer. It is managed internally and read-only.
	<b>QueuedIncomingCommands</b>	Count of all currently received but not-yet-Dispatched reliable commands ( <b>events</b> and operation results) from all channels.
	<b>QueuedOutgoingCommands</b>	Count of all commands currently queued as outgoing, including all channels and reliable, unreliable.
	<b>ResentReliableCommands</b>	Count of commands that got repeated (due to local repeat-timing before an ACK was received).
	<b>RoundTripTime</b>	Time until a reliable command is acknowledged by the server. The value measures network latency and for UDP it includes the server's ACK-delay (setting in config). In TCP, there is no ACK-delay, so the value is slightly lower (if you use default settings for Photon). RoundTripTime is updated constantly. Every reliable command will contribute a fraction to this value. This is also the approximate time until a raised event reaches another client or until an operation result is available.
	<b>RoundTripTimeVariance</b>	Changes of the roundtrip time as variance value. Gives a hint about how much the time is changing.
	<b>SentCountAllowance</b>	Number of send retries before a peer is considered lost/disconnected. Default: 5. The initial timeout countdown of a command is calculated by the current roundTripTime + 4 * roundTripTimeVariance. Please note that the timeout span until a command will be resent is not constant, but based on the roundtrip time at the initial sending, which will be doubled with every failed retry. <b>DisconnectTimeout</b> and <b>SentCountAllowance</b> are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time.
	<b>ServerAddress</b>	The server address which was used in <b>PhotonPeer.Connect()</b> or null (before <b>Connect()</b> was called).
	<b>ServerTimeInMilliseconds</b>	Approximated Environment.TickCount value of server (while connected).
	<b>SocketImplementation</b>	Can be used to set a UDP <b>IPhotonSocket</b> implementation at runtime (before connecting) in compatible DLL builds.
	<b>TcpConnectionPrefix</b>	The bytes set with this property will be sent as first bytes in any established connection.
	<b>TimePingInterval</b>	Sets the milliseconds without reliable command before a ping command (reliable) will be sent (Default: 1000ms). The ping command is used to keep track of the connection in case the client does not send reliable commands by itself. A ping (or reliable commands) will update the <b>RoundTripTime</b> calculation.
	<b>TimestampOfLastSocketReceive</b>	Stores timestamp of the last time anything (!) was received from the server (including low level Ping and ACKs but also <b>events</b> and operation-returns). This is not the time when something was dispatched. If you enable NetworkSimulation, this value is affected as well.
	<b>TrafficStatsElapsedMs</b>	Returns the count of milliseconds the stats are enabled for tracking.
	<b>TrafficStatsEnabled</b>	Enables the traffic statistics of a peer: <b>TrafficStatsIncoming</b> , <b>TrafficStatsOutgoing</b> and TrafficstatsGameLevel (nothing else). Default value: false (disabled).
	<b>TrafficStatsGameLevel</b>	Gets a statistic of incoming and outgoing traffic, split by operation, operation-result and event. <b>Operations</b> are outgoing traffic, results and <b>events</b> are incoming. Includes the per-command header sizes (Udp: Enet Command Header or Tcp: Message Header).



	<b>TrafficStatsIncoming</b>	Gets the byte-count of incoming "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.
	<b>TrafficStatsOutgoing</b>	Gets the byte-count of outgoing "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.
	<b>UsedProtocol</b>	The protocol this Peer uses to connect to Photon.
	<b>WarningSize</b>	The WarningSize is used test all message queues for congestion (in and out, reliable and unreliable). OnStatusChanged will be called with a warning if a queue holds WarningSize commands or a multiple of it. Default: 100.

## 2.1.12.1 PhotonPeer Constructor

### 2.1.12.1.1 PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener)

Creates a new PhotonPeer instance to communicate with Photon.

Connection is UDP based, except for Silverlight.

**C#**

```
[Obsolete("Use the constructor with ConnectionProtocol instead.")]
public PhotonPeer(IPhotonPeerListener listener);
```

**Parameters**

Parameters	Description
IPhotonPeerListener listener	a <a href="#">IPhotonPeerListener</a> implementation

### 2.1.12.1.2 PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener, ConnectionProtocol)

Creates a new PhotonPeer instance to communicate with Photon and selects either UDP or TCP as protocol. We recommend UDP.

**C#**

```
public PhotonPeer(IPhotonPeerListener listener, ConnectionProtocol protocolType);
```

**Parameters**

Parameters	Description
IPhotonPeerListener listener	a <a href="#">IPhotonPeerListener</a> implementation
ConnectionProtocol protocolType	<a href="#">Protocol</a> to use to connect to Photon.

### 2.1.12.1.3 PhotonPeer.PhotonPeer Constructor (IPhotonPeerListener, bool)

Deprecated. Please use: PhotonPeer([IPhotonPeerListener](#) listener, [ConnectionProtocol](#) protocolType).

**C#**

```
[Obsolete("Use the constructor with ConnectionProtocol instead.")]
public PhotonPeer(IPhotonPeerListener listener, bool useTcp);
```

## 2.1.12.2 PhotonPeer Methods

### 2.1.12.2.1 PhotonPeer.Connect Method

This method does a DNS lookup (if necessary) and connects to the given serverAddress.

The return value gives you feedback if the address has the correct format. If so, this starts the process to establish the connection itself, which might take a few seconds.

When the connection is established, a callback to [IPhotonPeerListener.OnStatusChanged](#) will be done. If the connection can't be established, despite having a valid address, the OnStatusChanged is called with an error-value.

The applicationName defines the application logic to use server-side and it should match the name of one of the apps in your server's config.

By default, the applicationName is "Lite" but other samples use "LiteLobby" and "MmoDemo" in Connect(). You can setup your own application and name it any way you like.

**C#**

```
public virtual bool Connect(String serverAddress, String applicationName);
```

**Parameters**

Parameters	Description
String serverAddress	Address of the Photon server. Format: ip:port (e.g. 127.0.0.1:5055) or hostname:port (e.g. localhost:5055)
String applicationName	The name of the application to use within Photon or the appld of PhotonCloud. Should match a "Name" for an application, as setup in your PhotonServer.config.

**Returns**

true if IP is available (DNS name is resolved) and server is being connected. false on error.

### 2.1.12.2.2 PhotonPeer.Disconnect Method

This method initiates a mutual disconnect between this client and the server.

**C#**

```
public virtual void Disconnect();
```

**Remarks**

Calling this method does not immediately close a connection. Disconnect lets the server know that this client is no longer listening. For the server, this is a much faster way to detect that the client is gone but it requires the client to send a few final messages.

On completion, OnStatusChanged is called with the StatusCode.Disconnect.

If the client is disconnected already or the connection thread is stopped, then there is no callback.

Lite: The default server logic will leave any joined game and trigger the respective event ([LiteEventCode.Leave](#)) for the remaining players.

### 2.1.12.2.3 PhotonPeer.DispatchIncomingCommands Method

This method directly causes the callbacks for [events](#), responses and state changes within a [IPhotonPeerListener](#). DispatchIncomingCommands only executes a single received command per call. If a command was dispatched, the return value is true and the method should be called again. This method is called by [Service\(\)](#) until currently available commands are dispatched.

**C#**

```
public virtual bool DispatchIncomingCommands();
```

### Remarks

In general, this method should be called until it returns false. In a few cases, it might make sense to pause dispatching (if a certain state is reached and the app needs to load data, before it should handle new [events](#)).

The callbacks to the peer's [IPhotonPeerListener](#) are executed in the same thread that is calling `DispatchIncomingCommands`. This makes things easier in a game loop: [Event](#) execution won't clash with painting objects or the game logic.

## 2.1.12.2.4 PhotonPeer.EstablishEncryption Method

This method creates a public key for this client and exchanges it with the server.

### C#

```
public bool EstablishEncryption();
```

### Returns

If operation could be enqueued for sending

### Remarks

Encryption is not instantly available but calls `OnStatusChanged` when it finishes. Check for [StatusCode](#) `EncryptionEstablished` and `EncryptionFailedToEstablish`.

Calling this method sets [IsEncryptionAvailable](#) to false. This method must be called before the "encrypt" parameter of [OpCustom](#) can be used.

## 2.1.12.2.5 PhotonPeer.FetchServerTimestamp Method

This will fetch the server's timestamp and update the approximation for property `ServerTimeInMilliseconds`.

The server time approximation will NOT become more accurate by repeated calls. Accuracy currently depends on a single roundtrip which is done as fast as possible.

The command used for this is immediately acknowledged by the server. This makes sure the roundtrip time is low and the timestamp + rountripime / 2 is close to the original value.

### C#

```
public virtual void FetchServerTimestamp();
```

## 2.1.12.2.6 OpCustom Method

### 2.1.12.2.6.1 PhotonPeer.OpCustom Method (OperationRequest, bool, byte, bool)

Allows the client to send any operation to the Photon Server by setting any `opCode` and the operation's parameters.

### C#

```
public virtual bool OpCustom(OperationRequest operationRequest, bool sendReliable, byte channelId, bool encrypt);
```

### Parameters

Parameters	Description
OperationRequest operationRequest	The operation to call on Photon.
bool sendReliable	Use unreliable (false) if the call might get lost (when it's content is soon outdated).
byte channelId	Defines the sequence of requests this operation belongs to.
bool encrypt	Encrypt request before sending. Depends on <a href="#">IsEncryptionAvailable</a> .

**Returns**

If operation could be enqueued for sending

**Remarks**

Variant with an [OperationRequest](#) object.

This variant offers an alternative way to describe a operation request. Operation code and it's parameters are wrapped up in a object. Still, the parameters are a Dictionary.

**2.1.12.2.6.2 PhotonPeer.OpCustom Method (byte, Dictionary<byte, object>, bool)**

Channel-less wrapper for OpCustom().

**C#**

```
public virtual bool OpCustom(byte customOpCode, Dictionary<byte, object>
customOpParameters, bool sendReliable);
```

**Parameters**

Parameters	Description
byte customOpCode	<a href="#">Operations</a> are handled by their byte-typed code. The codes of the "Lite" application are in the struct <a href="#">LiteOpCode</a> .
Dictionary<byte, object> customOpParameters	Containing parameters as key-value pair. The key is byte-typed, while the value is any serializable datatype.
bool sendReliable	Selects if the operation must be acknowledged or not. If false, the operation is not guaranteed to reach the server.

**Returns**

If operation could be enqueued for sending

**2.1.12.2.6.3 PhotonPeer.OpCustom Method (byte, Dictionary<byte, object>, bool, byte)**

Allows the client to send any operation to the Photon Server by setting any opCode and the operation's parameters.

**C#**

```
public virtual bool OpCustom(byte customOpCode, Dictionary<byte, object>
customOpParameters, bool sendReliable, byte channelId);
```

**Parameters**

Parameters	Description
byte customOpCode	<a href="#">Operations</a> are handled by their byte-typed code. The codes of the "Lite" application are in the struct <a href="#">LiteOpCode</a> .
Dictionary<byte, object> customOpParameters	Containing parameters as key-value pair. The key is byte-typed, while the value is any serializable datatype.
bool sendReliable	Selects if the operation must be acknowledged or not. If false, the operation is not guaranteed to reach the server.
byte channelId	The channel in which this operation should be sent.

**Returns**

If operation could be enqueued for sending

**Description**

Photon can be extended with new operations which are identified by a single byte, defined server side and known as operation code (opCode). Similarly, the operation's parameters are defined server side as byte keys of values, which a client sends as customOpParameters accordingly.

This is explained in more detail as "[Custom Operations](#)".

#### 2.1.12.2.6.4 PhotonPeer.OpCustom Method (byte, Dictionary<byte, object>, bool, byte, bool)

Allows the client to send any operation to the Photon Server by setting any opCode and the operation's parameters.

Variant with encryption parameter.

Use this only after encryption was established by [EstablishEncryption](#) and waiting for the OnStateChanged callback.

##### C#

```
public virtual bool OpCustom(byte customOpCode, Dictionary<byte, object> customOpParameters, bool sendReliable, byte channelId, bool encrypt);
```

##### Parameters

Parameters	Description
byte customOpCode	<a href="#">Operations</a> are handled by their byte-typed code. The codes of the "Lite" application are in the struct <a href="#">LiteOpCode</a> .
Dictionary<byte, object> customOpParameters	Containing parameters as key-value pair. The key is byte-typed, while the value is any serializable datatype.
bool sendReliable	Selects if the operation must be acknowledged or not. If false, the operation is not guaranteed to reach the server.
byte channelId	The channel in which this operation should be sent.
bool encrypt	Can only be true, while <a href="#">IsEncryptionAvailable</a> is true, too.

##### Returns

If operation could be enqueued for sending

#### 2.1.12.2.7 PhotonPeer.RegisterType Method

Registers new types/classes for de/serialization and the fitting methods to call for this type.

##### C#

```
public static bool RegisterType(Type customType, byte code, SerializeMethod serializeMethod, DeserializeMethod constructor);
```

##### Parameters

Parameters	Description
Type customType	Type (class) to register.
byte code	A byte-code used as shortcut during transfer of this Type.
SerializeMethod serializeMethod	Method delegate to create a byte[] from a customType instance.
DeserializeMethod constructor	Method delegate to create instances of customType's from byte[].

##### Returns

If the Type was registered successfully.

##### Remarks

[SerializeMethod](#) and [DeserializeMethod](#) are complementary: Feed the product of serializeMethod to the constructor, to get a comparable instance of the object.

After registering a Type, it can be used in [events](#) and operations and will be serialized like built-in types.

#### 2.1.12.2.8 PhotonPeer.SendAcksOnly Method

##### C#

```
public virtual bool SendAcksOnly();
```

**Description**

This is SendAcksOnly, a member of class PhotonPeer.

### 2.1.12.2.9 PhotonPeer.SendOutgoingCommands Method

This method creates a UDP/TCP package for outgoing commands (operations and acknowledgements) and sends them to the server. This method is also called by [Service\(\)](#).

**C#**

```
public virtual bool SendOutgoingCommands() ;
```

**Returns**

The if commands are not yet sent. Udp limits it's package size, Tcp doesnt.

**Remarks**

As the Photon library does not create any UDP/TCP packages by itself. Instead, the application fully controls how many packages are sent and when. A tradeoff, an application will lose connection, if it is no longer calling SendOutgoingCommands or [Service](#).

If multiple operations and ACKs are waiting to be sent, they will be aggregated into one package. The package fills in this order: ACKs for received commands A "Ping" - only if no reliable data was sent for a while Starting with the lowest Channel-Nr: Reliable Commands in channel Unreliable Commands in channel

This gives a higher priority to lower channels.

A longer interval between sends will lower the overhead per sent operation but increase the internal delay (which adds "lag").

Call this 2..20 times per second (depending on your target platform).

### 2.1.12.2.10 PhotonPeer.Service Method

This method excutes [DispatchIncomingCommands](#) and [SendOutgoingCommands](#) in your application Thread-context.

**C#**

```
public virtual void Service() ;
```

**Remarks**

The Photon client libraries are designed to fit easily into a game or application. The application is in control of the context (thread) in which incoming [events](#) and responses are executed and has full control of the creation of UDP/TCP packages.

Sending packages and dispatching received messages are two separate tasks. Service combines them into one method at the cost of control. It calls [DispatchIncomingCommands](#) and [SendOutgoingCommands](#).

Call this method regularly (2..20 times a second).

This will Dispatch ANY remaining buffered responses and [events](#) AND will send queued outgoing commands. Fewer calls might be more effective if a device cannot send many packets per second, as multiple operations might be combined into one package.

**See Also**

[PhotonPeer.DispatchIncomingCommands](#), [PhotonPeer.SendOutgoingCommands](#)

**Example**

You could replace Service by:

```
while (DispatchIncomingCommands\(\)); //Dispatch until everything is Dispatched... SendOutgoingCommands\(\); //Send a UDP/TCP package with outgoing messages
```

### 2.1.12.2.11 PhotonPeer.StopThread Method

This method immediately closes a connection (pure client side) and ends related listening Threads.

**C#**

```
public virtual void StopThread();
```

**Remarks**

Unlike [Disconnect](#), this method will simply stop to listen to the server. Udp connections will timeout. If the connections was open, this will trigger a callback to OnStatusChanged with code [StatusCode.Disconnect](#).

### 2.1.12.2.12 PhotonPeer.TrafficStatsReset Method

Creates new instances of [TrafficStats](#) and starts a new timer for those.

**C#**

```
public void TrafficStatsReset();
```

### 2.1.12.2.13 PhotonPeer.VitalStatsToString Method

Returns a string of the most interesting connection statistics. When you have issues on the client side, these might contain hints about the issue's cause.

**C#**

```
public string VitalStatsToString(bool all);
```

**Parameters**

Parameters	Description
bool all	If true, Incoming and Outgoing low-level stats are included in the string.

**Returns**

Stats as string.

## 2.1.12.3 PhotonPeer Properties

### 2.1.12.3.1 PhotonPeer.ByteCountCurrentDispatch Property

Gets the size of the dispatched event or operation-result in bytes. This value is set before [OnEvent\(\)](#) or [OnOperationResponse\(\)](#) is called (within [DispatchIncomingCommands\(\)](#)).

**C#**

```
public int ByteCountCurrentDispatch;
```

**Remarks**

Get this value directly in [OnEvent\(\)](#) or [OnOperationResponse\(\)](#). Example: void OnEvent(...) { int eventSizeInBytes = this.peer.ByteCountCurrentDispatch; //...

void OnOperationResponse(...) { int resultSizeInBytes = this.peer.ByteCountCurrentDispatch; //...

### 2.1.12.3.2 PhotonPeer.ByteCountLastOperation Property

Gets the size of the last serialized operation call in bytes. The value includes all headers for this single operation but excludes those of UDP, Enet Package Headers and TCP.

**C#**

```
public int ByteCountLastOperation;
```

**Remarks**

Get this value immediately after calling an operation. Example: `this.litepeer.OpJoin("myroom"); int opjoinByteCount = this.peer.ByteCountLastOperation;`

### 2.1.12.3.3 PhotonPeer.BytesIn Property

Gets count of all bytes coming in (including headers, excluding UDP/TCP overhead)

**C#**

```
public long BytesIn;
```

### 2.1.12.3.4 PhotonPeer.BytesOut Property

Gets count of all bytes going out (including headers, excluding UDP/TCP overhead)

**C#**

```
public long BytesOut;
```

### 2.1.12.3.5 PhotonPeer.ChannelCount Property

Gets / sets the number of channels available in UDP connections with Photon. Photon Channels are only supported for UDP. The default ChannelCount is 2. Channel IDs start with 0 and 255 is a internal channel.

**C#**

```
public byte ChannelCount;
```

### 2.1.12.3.6 PhotonPeer.CommandBufferSize Property

Initial size internal lists for incoming/outgoing commands (reliable and unreliable).

**C#**

```
public int CommandBufferSize;
```

**Remarks**

This sets only the initial size. All lists simply grow in size as needed. This means that incoming or outgoing commands can pile up and consume heap size if [Service](#) is not called often enough to handle the messages in either direction.

Configure the [WarningSize](#), to get callbacks when the lists reach a certain size.

UDP: Incoming and outgoing commands each have separate buffers for reliable and unreliable sending. There are additional buffers for "sent commands" and "ACKs". TCP: Only two buffers exist: incoming and outgoing commands.

### 2.1.12.3.7 PhotonPeer.CrcEnabled Property

While not connected, this controls if the next connection(s) should use a per-package CRC checksum.

**C#**

```
public bool CrcEnabled;
```

**Remarks**

While turned on, the client and server will add a CRC checksum to every sent package. The checksum enables both sides to detect and ignore packages that were corrupted during transfer. Corrupted packages have the same impact as lost packages: They require a re-send, adding a delay and could lead to timeouts.

Building the checksum has a low processing overhead but increases integrity of sent and received data. Packages discarded



due to failed CRC checks are counted in [PhotonPeer.PacketLossByCrc](#).

### 2.1.12.3.8 PhotonPeer.DebugOut Property

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

**C#**

```
public DebugLevel DebugOut;
```

**Remarks**

This affects the callbacks to [IPhotonPeerListener.DebugReturn](#). Default Level: Error.

### 2.1.12.3.9 PhotonPeer.DisconnectTimeout Property

Milliseconds after which a reliable UDP command triggers a timeout disconnect, unless acknowledged by server. This value currently only affects UDP connections. DisconnectTimeout is not an exact value for a timeout. The exact timing of the timeout depends on the frequency of [Service\(\)](#) calls and commands that are sent with long roundtrip-times and variance are checked less often for re-sending!

DisconnectTimeout and [SentCountAllowance](#) are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time. Default: 10000 ms.

**C#**

```
public int DisconnectTimeout;
```

### 2.1.12.3.10 PhotonPeer.HttpUrlParameters Property

This string is added as simple GET parameters to the end of the server url + peerID combination. Include a starting "&" in your parameters as the peerID is always added to the [ServerAddress](#). This value will be added to all following (http) operations, so make sure to clear it if needed. Can be applied even to "connect" (set before calling connect, clear / reset when connection is established). Set before making a Operation call and don't forget to clear or set it again before the next operation.

**C#**

```
public string HttpUrlParameters;
```

### 2.1.12.3.11 PhotonPeer.IsEncryptionAvailable Property

This property is set internally, when OpExchangeKeysForEncryption successfully finished. While it's true, encryption can be used for operations.

**C#**

```
public bool IsEncryptionAvailable;
```

### 2.1.12.3.12 PhotonPeer.IsSendingOnlyAcks Property

While true, the peer will not send any other commands except ACKs (used in UDP connections).

**C#**

```
public bool IsSendingOnlyAcks;
```

### 2.1.12.3.13 PhotonPeer.IsSimulationEnabled Property

Gets or sets the network simulation "enabled" setting. Changing this value also locks this peer's sending and when setting false, the internally used queues are executed (so setting to false can take some cycles).

**C#**

```
public virtual bool IsSimulationEnabled;
```

### 2.1.12.3.14 PhotonPeer.LimitOfUnreliableCommands Property

Limits the queue of received unreliable commands within [DispatchIncomingCommands](#) before dispatching them. This works only in UDP. This limit is applied when you call [DispatchIncomingCommands](#). If this client (already) received more than [LimitOfUnreliableCommands](#), it will throw away the older ones instead of dispatching them. This can produce bigger gaps for unreliable commands but your client catches up faster.

**C#**

```
public int LimitOfUnreliableCommands;
```

#### Remarks

This can be useful when the client couldn't dispatch anything for some time (cause it was in a room but loading a level). If set to 20, the incoming unreliable queues are truncated to 20. If 0, all received unreliable commands will be dispatched. This is a "per channel" value, so each channel can hold up to [LimitOfUnreliableCommands](#) commands. This value interacts with [DispatchIncomingCommands](#): If that is called less often, more commands get skipped.

### 2.1.12.3.15 PhotonPeer.Listener Property

Gets the [IPhotonPeerListener](#) of this instance (set in constructor). Can be used in derived classes for [Listener.DebugReturn\(\)](#).

**C#**

```
public IPhotonPeerListener Listener;
```

### 2.1.12.3.16 PhotonPeer.LocalMsTimestampDelegate Property

This setter for the (local-) timestamp delegate replaces the default [Environment.TickCount](#) with any equal function.

**C#**

```
public SupportClass.IntegerMillisecondsDelegate LocalMsTimestampDelegate;
```

#### Remarks

About [Environment.TickCount](#): The value of this property is derived from the system timer and is stored as a 32-bit signed integer. Consequently, if the system runs continuously, [TickCount](#) will increment from zero to [Int32.MaxValue](#) for approximately 24.9 days, then jump to [Int32.MinValue](#), which is a negative number, then increment back to zero during the next 24.9 days.

#### Exceptions

Exceptions	Description
Exception	Exception is thrown peer. <a href="#">PeerState</a> is not PS_DISCONNECTED.

### 2.1.12.3.17 PhotonPeer.LocalTimeInMilliseconds Property

Gets a local timestamp in milliseconds by calling [SupportClass.GetTickCount\(\)](#). See [LocalMsTimestampDelegate](#).

**C#**

```
[Obsolete("Should be replaced by: SupportClass.GetTickCount(). Internally this is used, too.")]  
public int LocalTimeInMilliseconds;
```

### 2.1.12.3.18 PhotonPeer.MaximumTransferUnit Property

The Maximum Transfer Unit (MTU) defines the (network-level) packet-content size that is guaranteed to arrive at the server in one piece. The Photon **Protocol** uses this size to split larger data into packets and for receive-buffers of packets.

**C#**

```
public int MaximumTransferUnit;
```

**Remarks**

This value affects the Packet-content. The resulting UDP packages will have additional headers that also count against the package size (so it's bigger than this limit in the end) Setting this value while being connected is not allowed and will throw an Exception. Minimum is 520. Huge values won't speed up connections in most cases!

### 2.1.12.3.19 PhotonPeer.NetworkSimulationSettings Property

Gets the settings for built-in Network Simulation for this peer instance while **IsSimulationEnabled** will enable or disable them. Once obtained, the settings can be modified by changing the properties.

**C#**

```
public NetworkSimulationSet NetworkSimulationSettings;
```

### 2.1.12.3.20 PhotonPeer.OutgoingStreamBufferSize Property

Defines the initial size of an internally used MemoryStream for Tcp. The MemoryStream is used to aggregate operation into (less) send calls, which uses less resources.

**C#**

```
public static int OutgoingStreamBufferSize;
```

**Remarks**

The size is not restricting the buffer and does not affect when outgoing data is actually sent.

### 2.1.12.3.21 PhotonPeer.PacketLossByCrc Property

Count of packages dropped due to failed CRC checks for this connection.

**C#**

```
public int PacketLossByCrc;
```

**Description**

**CrcEnabled**

### 2.1.12.3.22 PhotonPeer.PeerID Property

This peer's ID as assigned by the server or 0 if not using UDP. Will be 0xFFFF before the client connects.

**C#**

```
public string PeerID;
```

**Remarks**

Used for debugging only. This value is not useful in everyday Photon usage.

### 2.1.12.3.23 PhotonPeer.PeerState Property

This is the (low level) state of the connection to the server of a **PhotonPeer**. It is managed internally and read-only.

**C#**

```
public PeerStateValue PeerState;
```

**Remarks**

Don't mix this up with the [StatusCode](#) provided in `IPhotonListener.OnStatusChanged()`. Applications should use the [StatusCode](#) of `OnStatusChanged()` to track their state, as it also covers the higher level initialization between a client and Photon.

### 2.1.12.3.24 PhotonPeer.QueuedIncomingCommands Property

Count of all currently received but not-yet-Dispatched reliable commands ([events](#) and operation results) from all channels.

**C#**

```
public int QueuedIncomingCommands;
```

### 2.1.12.3.25 PhotonPeer.QueuedOutgoingCommands Property

Count of all commands currently queued as outgoing, including all channels and reliable, unreliable.

**C#**

```
public int QueuedOutgoingCommands;
```

### 2.1.12.3.26 PhotonPeer.ResentReliableCommands Property

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

**C#**

```
public int ResentReliableCommands;
```

### 2.1.12.3.27 PhotonPeer.RoundTripTime Property

Time until a reliable command is acknowledged by the server.

The value measures network latency and for UDP it includes the server's ACK-delay (setting in config). In TCP, there is no ACK-delay, so the value is slightly lower (if you use default settings for Photon).

RoundTripTime is updated constantly. Every reliable command will contribute a fraction to this value.

This is also the approximate time until a raised event reaches another client or until an operation result is available.

**C#**

```
public int RoundTripTime;
```

### 2.1.12.3.28 PhotonPeer.RoundTripTimeVariance Property

Changes of the roundtriptime as variance value. Gives a hint about how much the time is changing.

**C#**

```
public int RoundTripTimeVariance;
```

### 2.1.12.3.29 PhotonPeer.SentCountAllowance Property

Number of send retries before a peer is considered lost/disconnected. Default: 5. The initial timeout countdown of a command is calculated by the current `roundTripTime` + 4 \* `roundTripTimeVariance`. Please note that the timeout span until a command will be resent is not constant, but based on the roundtrip time at the initial sending, which will be doubled with every failed retry.

[DisconnectTimeout](#) and `SentCountAllowance` are competing settings: either might trigger a disconnect on the client first, depending on the values and Roundtrip Time.

**C#**

```
public int SentCountAllowance;
```

### 2.1.12.3.30 PhotonPeer.ServerAddress Property

The server address which was used in [PhotonPeer.Connect\(\)](#) or null (before [Connect\(\)](#) was called).

**C#**

```
public string ServerAddress;
```

**Remarks**

The ServerAddress can only be changed for HTTP connections (to replace one that goes through a Loadbalancer with a direct URL).

### 2.1.12.3.31 PhotonPeer.ServerTimeInMilliseconds Property

Approximated Environment.TickCount value of server (while connected).

**C#**

```
public int ServerTimeInMilliseconds;
```

**Description**

0 until connected. While connected, the value is an approximation of the server's current timestamp.

**Remarks**

UDP: The server's timestamp is automatically fetched after connecting (once). This is done internally by a command which is acknowledged immediately by the server. TCP: The server's timestamp fetched with each ping but set only after connecting (once).

The approximation will be off by +/- 10ms in most cases. Per peer/client and connection, the offset will be constant (unless [FetchServerTimestamp\(\)](#) is used). A constant offset should be better to adjust for. Unfortunately there is no way to find out how much the local value differs from the original.

The approximation adds RoundtripTime / 2 and uses this [LocalTimeInMilliseconds](#) to calculate in-between values (this property returns a new value per tick).

The value sent by Photon equals Environment.TickCount in the logic layer (e.g. Lite).

### 2.1.12.3.32 PhotonPeer.SocketImplementation Property

Can be used to set a UDP [IPhotonSocket](#) implementation at runtime (before connecting) in compatible DLL builds.

**C#**

```
public Type SocketImplementation;
```

**Remarks**

This won't work for TCP and regular C# libraries. Get in contact with us.

### 2.1.12.3.33 PhotonPeer.TcpConnectionPrefix Property

The bytes set with this property will be sent as first bytes in any established connection.

**C#**

```
public byte TcpConnectionPrefix;
```

**Remarks**

This property is only used for specific carriers to mark traffic. Don't set it unless you are specifically asked to.

### 2.1.12.3.34 PhotonPeer.TimePingInterval Property

Sets the milliseconds without reliable command before a ping command (reliable) will be sent (Default: 1000ms). The ping command is used to keep track of the connection in case the client does not send reliable commands by itself. A ping (or reliable commands) will update the [RoundTripTime](#) calculation.

C#

```
public int TimePingInterval;
```

### 2.1.12.3.35 PhotonPeer.TimestampOfLastSocketReceive Property

Stores timestamp of the last time anything (!) was received from the server (including low level Ping and ACKs but also [events](#) and operation-returns). This is not the time when something was dispatched. If you enable NetworkSimulation, this value is affected as well.

C#

```
public int TimestampOfLastSocketReceive;
```

### 2.1.12.3.36 PhotonPeer.TrafficStatsElapsedMs Property

Returns the count of milliseconds the stats are enabled for tracking.

C#

```
public long TrafficStatsElapsedMs;
```

### 2.1.12.3.37 PhotonPeer.TrafficStatsEnabled Property

Enables the traffic statistics of a peer: [TrafficStatsIncoming](#), [TrafficStatsOutgoing](#) and [TrafficStatsGameLevel](#) (nothing else). Default value: false (disabled).

C#

```
public bool TrafficStatsEnabled;
```

### 2.1.12.3.38 PhotonPeer.TrafficStatsGameLevel Property

Gets a statistic of incoming and outgoing traffic, split by operation, operation-result and event. [Operations](#) are outgoing traffic, results and [events](#) are incoming. Includes the per-command header sizes (Udp: Enet Command Header or Tcp: Message Header).

C#

```
public TrafficStatsGameLevel TrafficStatsGameLevel;
```

### 2.1.12.3.39 PhotonPeer.TrafficStatsIncoming Property

Gets the byte-count of incoming "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.

C#

```
public TrafficStats TrafficStatsIncoming;
```

### 2.1.12.3.40 PhotonPeer.TrafficStatsOutgoing Property

Gets the byte-count of outgoing "low level" messages, which are either Enet Commands or Tcp Messages. These include all headers, except those of the underlying internet protocol Udp or Tcp.

C#

```
public TrafficStats TrafficStatsOutgoing;
```

### 2.1.12.3.41 PhotonPeer.UsedProtocol Property

The protocol this Peer uses to connect to Photon.

C#

```
public ConnectionProtocol UsedProtocol;
```

### 2.1.12.3.42 PhotonPeer.WarningSize Property

The WarningSize is used test all message queues for congestion (in and out, reliable and unreliable). OnStatusChanged will be called with a warning if a queue holds WarningSize commands or a multiple of it. Default: 100.

C#

```
public int WarningSize;
```

Example

If command is received, OnStatusChanged will be called when the respective command queue has 100, 200, 300 ... items.

## 2.1.13 Protocol Class

Provides tools for the Exit Games Protocol

C#

```
public class Protocol;
```

Protocol Methods

	Name	Description
💎💰	<a href="#">Deserialize</a>	Deserialize returns an object reassembled from the given byte-array.
💎💰	<a href="#">Serialize</a>	Serializes an float typed value into a byte-array (target) starting at the also given targetOffset. The altered offset is known to the caller, because it is given via a referenced parameter.

### 2.1.13.1 Protocol Methods

#### 2.1.13.1.1 Deserialize Method

##### 2.1.13.1.1.1 Protocol.Deserialize Method (byte[])

Deserialize returns an object reassembled from the given byte-array.

C#

```
public static object Deserialize(byte[] serializedData);
```

Parameters

Parameters	Description
byte[] serializedData	The byte-array to be Deserialized

Returns

The Deserialized object

### 2.1.13.1.1.2 Protocol.Deserialize Method (out float, byte[], int)

Deserialize fills the given float typed value with the given byte-array (source) starting at the also given offset. The result is placed in a variable (value). There is no need to return a value because the parameter value is given by reference. The altered offset is this way also known to the caller.

C#

```
public static void Deserialize(out float value, byte[] source, ref int offset);
```

Parameters

Parameters	Description
out float value	The float value to deserialize
byte[] source	The byte-array to deserialize from
ref int offset	The offset in the byte-array

### 2.1.13.1.1.3 Protocol.Deserialize Method (out int, byte[], int)

Deserialize fills the given int typed value with the given byte-array (source) starting at the also given offset. The result is placed in a variable (value). There is no need to return a value because the parameter value is given by reference. The altered offset is this way also known to the caller.

C#

```
public static void Deserialize(out int value, byte[] source, ref int offset);
```

Parameters

Parameters	Description
out int value	The int value to deserialize into
byte[] source	The byte-array to deserialize from
ref int offset	The offset in the byte-array

### 2.1.13.1.1.4 Protocol.Deserialize Method (out short, byte[], int)

Deserialize fills the given short typed value with the given byte-array (source) starting at the also given offset. The result is placed in a variable (value). There is no need to return a value because the parameter value is given by reference. The altered offset is this way also known to the caller.

C#

```
public static void Deserialize(out short value, byte[] source, ref int offset);
```

Parameters

Parameters	Description
out short value	The short value to deserialized into
byte[] source	The byte-array to deserialize from
ref int offset	The offset in the byte-array

## 2.1.13.1.2 Serialize Method

### 2.1.13.1.2.1 Protocol.Serialize Method (float, byte[], int)

Serializes an float typed value into a byte-array (target) starting at the also given targetOffset. The altered offset is known to the caller, because it is given via a referenced parameter.

C#

```
public static void Serialize(float value, byte[] target, ref int targetOffset);
```



**Parameters**

Parameters	Description
float value	The float value to be serialized
byte[] target	The byte-array to serialize the short to
ref int targetOffset	The offset in the byte-array

**2.1.13.1.2.2 Protocol.Serialize Method (int, byte[], int)**

Serializes an int typed value into a byte-array (target) starting at the also given targetOffset. The altered offset is known to the caller, because it is given via a referenced parameter.

**C#**

```
public static void Serialize(int value, byte[] target, ref int targetOffset);
```

**Parameters**

Parameters	Description
int value	The int value to be serialized
byte[] target	The byte-array to serialize the short to
ref int targetOffset	The offset in the byte-array

**2.1.13.1.2.3 Protocol.Serialize Method (object)**

Serialize creates a byte-array from the given object and returns it.

**C#**

```
public static byte[] Serialize(object obj);
```

**Parameters**

Parameters	Description
object obj	The object to serialize

**Returns**

The serialized byte-array

**2.1.13.1.2.4 Protocol.Serialize Method (short, byte[], int)**

Serializes a short typed value into a byte-array (target) starting at the also given targetOffset. The altered offset is known to the caller, because it is given via a referenced parameter.

**C#**

```
public static void Serialize(short value, byte[] target, ref int targetOffset);
```

**Parameters**

Parameters	Description
short value	The short value to be serialized
byte[] target	The byte-array to serialize the short to
ref int targetOffset	The offset in the byte-array

---


**2.1.14 SocketUdpNativeDynamic Class****C#**

```
public class SocketUdpNativeDynamic : IPhotonSocket;
```


**Description**

This is class SocketUdpNativeDynamic.


**Methods**

	Name	Description
	<b>IPhotonSocket</b>	This is IPhotonSocket, a member of class IPhotonSocket.







**SocketUdpNativeDynamic Class**

	Name	Description
	<b>SocketUdpNativeDynamic</b>	This is SocketUdpNativeDynamic, a member of class SocketUdpNativeDynamic.






**IPhotonSocket Fields**

	Name	Description
	<b>PollReceive</b>	This is PollReceive, a member of class IPhotonSocket.








**IPhotonSocket Methods**

	Name	Description
	<b>Connect</b>	This is Connect, a member of class IPhotonSocket.
	<b>Disconnect</b>	This is Disconnect, a member of class IPhotonSocket.
	<b>EnqueueDebugReturn</b>	This is EnqueueDebugReturn, a member of class IPhotonSocket.
	<b>HandleReceivedDatagram</b>	This is HandleReceivedDatagram, a member of class IPhotonSocket.
	<b>Receive</b>	This is Receive, a member of class IPhotonSocket.
	<b>ReportDebugOfLevel</b>	This is ReportDebugOfLevel, a member of class IPhotonSocket.
	<b>Send</b>	This is Send, a member of class IPhotonSocket.

**SocketUdpNativeDynamic Class**

	Name	Description
	<b>Connect</b>	This is Connect, a member of class SocketUdpNativeDynamic.
	<b>Disconnect</b>	This is Disconnect, a member of class SocketUdpNativeDynamic.
	<b>Receive</b>	This is Receive, a member of class SocketUdpNativeDynamic.
	<b>ReceiveLoop</b>	This is ReceiveLoop, a member of class SocketUdpNativeDynamic.
	<b>Send</b>	used by <b>PhotonPeer</b> *

**IPhotonSocket Properties**

	Name	Description
	<b>Connected</b>	This is Connected, a member of class IPhotonSocket.
	<b>Listener</b>	This is Listener, a member of class IPhotonSocket.
	<b>MTU</b>	This is MTU, a member of class IPhotonSocket.
	<b>Protocol</b>	This is Protocol, a member of class IPhotonSocket.
	<b>ServerAddress</b>	This is ServerAddress, a member of class IPhotonSocket.
	<b>ServerPort</b>	This is ServerPort, a member of class IPhotonSocket.
	<b>State</b>	This is State, a member of class IPhotonSocket.

## 2.1.14.1 SocketUdpNativeDynamic.SocketUdpNativeDynamic Constructor

**C#**

```
public SocketUdpNativeDynamic(PeerBase npeer);
```

**Description**

This is SocketUdpNativeDynamic, a member of class SocketUdpNativeDynamic.

## 2.1.14.2 SocketUdpNativeDynamic Methods

### 2.1.14.2.1 SocketUdpNativeDynamic.Connect Method

**C#**

```
public override bool Connect();
```

**Description**

This is Connect, a member of class SocketUdpNativeDynamic.

### 2.1.14.2.2 SocketUdpNativeDynamic.Disconnect Method

**C#**

```
public override bool Disconnect();
```

**Description**

This is Disconnect, a member of class SocketUdpNativeDynamic.

### 2.1.14.2.3 SocketUdpNativeDynamic.Receive Method

**C#**

```
public override PhotonSocketError Receive(out byte[] data);
```

**Description**

This is Receive, a member of class SocketUdpNativeDynamic.

### 2.1.14.2.4 SocketUdpNativeDynamic.ReceiveLoop Method

**C#**

```
public void ReceiveLoop();
```

**Description**

This is ReceiveLoop, a member of class SocketUdpNativeDynamic.

### 2.1.14.2.5 SocketUdpNativeDynamic.Send Method

used by [PhotonPeer\\*](#)

**C#**

```
public override PhotonSocketError Send(byte[] data, int length);
```

---

## 2.1.15 SocketUdpNativeStatic Class

**C#**

```
public class SocketUdpNativeStatic : IPhotonSocket;
```


**Description**

This is class SocketUdpNativeStatic.


**Methods**

	Name	Description
	<a href="#">IPhotonSocket</a>	This is IPhotonSocket, a member of class IPhotonSocket.








**SocketUdpNativeStatic Class**

	Name	Description
	<a href="#">SocketUdpNativeStatic</a>	This is SocketUdpNativeStatic, a member of class SocketUdpNativeStatic.






**IPhotonSocket Fields**

	Name	Description
	<a href="#">PollReceive</a>	This is PollReceive, a member of class IPhotonSocket.








**IPhotonSocket Methods**

	Name	Description
	<a href="#">Connect</a>	This is Connect, a member of class IPhotonSocket.
	<a href="#">Disconnect</a>	This is Disconnect, a member of class IPhotonSocket.
	<a href="#">EnqueueDebugReturn</a>	This is EnqueueDebugReturn, a member of class IPhotonSocket.
	<a href="#">HandleReceivedDatagram</a>	This is HandleReceivedDatagram, a member of class IPhotonSocket.
	<a href="#">Receive</a>	This is Receive, a member of class IPhotonSocket.
	<a href="#">ReportDebugOfLevel</a>	This is ReportDebugOfLevel, a member of class IPhotonSocket.
	<a href="#">Send</a>	This is Send, a member of class IPhotonSocket.

**SocketUdpNativeStatic Class**

	Name	Description
	<a href="#">Connect</a>	This is Connect, a member of class SocketUdpNativeStatic.
	<a href="#">Disconnect</a>	This is Disconnect, a member of class SocketUdpNativeStatic.
	<a href="#">Receive</a>	This is Receive, a member of class SocketUdpNativeStatic.
	<a href="#">ReceiveLoop</a>	This is ReceiveLoop, a member of class SocketUdpNativeStatic.
	<a href="#">Send</a>	used by <a href="#">PhotonPeer</a> *

**IPhotonSocket Properties**

	Name	Description
	<a href="#">Connected</a>	This is Connected, a member of class IPhotonSocket.
	<a href="#">Listener</a>	This is Listener, a member of class IPhotonSocket.
	<a href="#">MTU</a>	This is MTU, a member of class IPhotonSocket.
	<a href="#">Protocol</a>	This is Protocol, a member of class IPhotonSocket.
	<a href="#">ServerAddress</a>	This is ServerAddress, a member of class IPhotonSocket.
	<a href="#">ServerPort</a>	This is ServerPort, a member of class IPhotonSocket.
	<a href="#">State</a>	This is State, a member of class IPhotonSocket.

## 2.1.15.1 SocketUdpNativeStatic.SocketUdpNativeStatic Constructor

**C#**

```
public SocketUdpNativeStatic(PeerBase npeer);
```

**Description**

This is SocketUdpNativeStatic, a member of class SocketUdpNativeStatic.

## 2.1.15.2 SocketUdpNativeStatic Methods

### 2.1.15.2.1 SocketUdpNativeStatic.Connect Method

**C#**

```
public override bool Connect();
```

**Description**

This is Connect, a member of class SocketUdpNativeStatic.

### 2.1.15.2.2 SocketUdpNativeStatic.Disconnect Method

**C#**

```
public override bool Disconnect();
```

**Description**

This is Disconnect, a member of class SocketUdpNativeStatic.

### 2.1.15.2.3 SocketUdpNativeStatic.Receive Method

**C#**

```
public override PhotonSocketError Receive(out byte[] data);
```

**Description**

This is Receive, a member of class SocketUdpNativeStatic.

### 2.1.15.2.4 SocketUdpNativeStatic.ReceiveLoop Method

**C#**

```
public void ReceiveLoop();
```

**Description**

This is ReceiveLoop, a member of class SocketUdpNativeStatic.

### 2.1.15.2.5 SocketUdpNativeStatic.Send Method

used by [PhotonPeer](#)\*

**C#**

```
public override PhotonSocketError Send(byte[] data, int length);
```

---


## 2.1.16 SupportClass Class

Contains several (more or less) useful static methods, mostly used for debugging.

**C#**

```
public class SupportClass;
```










**SupportClass Classes**

	Name	Description
	<a href="#">ThreadSafeRandom</a>	Class to wrap static access to the random. <a href="#">Next()</a> call in a thread safe manner.

**SupportClass Delegates**

Name	Description
<a href="#">IntegerMillisecondsDelegate</a>	This is nested type SupportClass.IntegerMillisecondsDelegate.

**SupportClass Methods**

	Name	Description
	<a href="#">ByteArrayToString</a>	Converts a byte-array to string (useful as debugging output). Uses BitConverter.ToString(list) internally after a null-check of list.
	<a href="#">CalculateCrc</a>	This is CalculateCrc, a member of class SupportClass.
	<a href="#">CallInBackground</a>	Creates a background thread that calls the passed function in 100ms intervals, as long as that returns true.
	<a href="#">DictionaryToString</a>	This method returns a string, representing the content of the given IDictionary. Returns "null" if parameter is null.
	<a href="#">GetMethods</a>	This is GetMethods, a member of class SupportClass.
	<a href="#">GetTickCount</a>	Gets the local machine's "milliseconds since start" value (precision is described in remarks).
	<a href="#">HashtableToString</a>	This is HashtableToString, a member of class SupportClass.
	<a href="#">NumberToByteArray</a>	Inserts the number's value into the byte array, using Big-Endian order (a.k.a. Network-byte-order).
	<a href="#">WriteStackTrace</a>	Writes the exception's stack trace to the received stream. Writes to: System.Diagnostics.Debug.

## 2.1.16.1 SupportClass Classes


### 2.1.16.1.1 SupportClass.ThreadSafeRandom Class

Class to wrap static access to the random.[Next\(\)](#) call in a thread safe manner.

**C#**

```
public class ThreadSafeRandom;
```

**ThreadSafeRandom Methods**

	Name	Description
	<a href="#">Next</a>	This is Next, a member of class ThreadSafeRandom.

#### 2.1.16.1.1.1 ThreadSafeRandom Methods

##### 2.1.16.1.1.1.1 SupportClass.ThreadSafeRandom.Next Method

**C#**

```
public static int Next();
```

**Description**

This is Next, a member of class ThreadSafeRandom.

## 2.1.16.2 SupportClass Methods

### 2.1.16.2.1 SupportClass.ByteArrayToString Method

Converts a byte-array to string (useful as debugging output). Uses BitConverter.ToString(list) internally after a null-check of list.

C#

```
public static string ByteArrayToString(byte[] list);
```

Parameters

Parameters	Description
byte[] list	Byte-array to convert to string.

Returns

List of bytes as string.

### 2.1.16.2.2 SupportClass.CalculateCrc Method

C#

```
public static uint CalculateCrc(byte[] buffer, int length);
```

Description

This is CalculateCrc, a member of class SupportClass.

### 2.1.16.2.3 CallInBackground Method

#### 2.1.16.2.3.1 SupportClass.CallInBackground Method (Func<bool>)

Creates a background thread that calls the passed function in 100ms intervals, as long as that returns true.

C#

```
public static void CallInBackground(Func<bool> myThread);
```

Parameters

Parameters	Description
Func<bool> myThread	

#### 2.1.16.2.3.2 SupportClass.CallInBackground Method (Func<bool>, int)

Creates a background thread that calls the passed function in 100ms intervals, as long as that returns true.

C#

```
public static void CallInBackground(Func<bool> myThread, int millisecondsInterval);
```

Parameters

Parameters	Description
Func<bool> myThread	
int millisecondsInterval	Milliseconds to sleep between calls of myThread.

### 2.1.16.2.4 DictionaryToString Method

#### 2.1.16.2.4.1 SupportClass.DictionaryToString Method (IDictionary)

This method returns a string, representing the content of the given IDictionary. Returns "null" if parameter is null.

**C#**

```
public static string DictionaryToString(IDictionary dictionary);
```

**Parameters**

Parameters	Description
IDictionary dictionary	IDictionary to return as string.

**Returns**

The string representation of keys and values in IDictionary.

**2.1.16.2.4.2 SupportClass.DictionaryToString Method (IDictionary, bool)**

This method returns a string, representing the content of the given IDictionary. Returns "null" if parameter is null.

**C#**

```
public static string DictionaryToString(IDictionary dictionary, bool includeTypes);
```

**Parameters**

Parameters	Description
IDictionary dictionary	IDictionary to return as string.
bool includeTypes	

**2.1.16.2.5 SupportClass.GetMethods Method****C#**

```
public static List<MethodInfo> GetMethods(Type type, Type attribute);
```

**Description**

This is GetMethods, a member of class SupportClass.

**2.1.16.2.6 SupportClass.GetTickCount Method**

Gets the local machine's "milliseconds since start" value (precision is described in remarks).

**C#**

```
public static int GetTickCount();
```

**Returns**

Fraction of the current time in Milliseconds (this is not a proper datetime timestamp).

**Remarks**

This method uses Environment.TickCount (cheap but with only 16ms precision). [PhotonPeer.LocalMsTimestampDelegate](#) is available to set the delegate (unless already connected).

**2.1.16.2.7 SupportClass.HashtableToString Method****C#**

```
[Obsolete("Use DictionaryToString() instead.")]  
public static string HashtableToString(Hashtable hash);
```

**Description**

This is HashtableToString, a member of class SupportClass.

**2.1.16.2.8 NumberToByteArray Method**



### 2.1.16.2.8.1 SupportClass.NumberToByteArray Method (byte[], int, int)

Inserts the number's value into the byte array, using Big-Endian order (a.k.a. Network-byte-order).

**C#**

```
[Obsolete("Use Protocol.Serialize() instead.")]  
public static void NumberToByteArray(byte[] buffer, int index, int number);
```

**Parameters**

Parameters	Description
byte[] buffer	Byte array to write into.
int index	Index of first position to write to.
int number	Number to write.

### 2.1.16.2.8.2 SupportClass.NumberToByteArray Method (byte[], int, short)

Inserts the number's value into the byte array, using Big-Endian order (a.k.a. Network-byte-order).

**C#**

```
[Obsolete("Use Protocol.Serialize() instead.")]  
public static void NumberToByteArray(byte[] buffer, int index, short number);
```

**Parameters**

Parameters	Description
byte[] buffer	Byte array to write into.
int index	Index of first position to write to.
short number	Number to write.

## 2.1.16.2.9 WriteStackTrace Method

### 2.1.16.2.9.1 SupportClass.WriteStackTrace Method (System.Exception)

Writes the exception's stack trace to the received stream. Writes to: System.Diagnostics.Debug.

**C#**

```
public static void WriteStackTrace(System.Exception throwable);
```

**Parameters**

Parameters	Description
System.Exception throwable	Exception to obtain information from.

### 2.1.16.2.9.2 SupportClass.WriteStackTrace Method (System.Exception, System.IO.TextWriter)

Writes the exception's stack trace to the received stream.

**C#**

```
public static void WriteStackTrace(System.Exception throwable, System.IO.TextWriter stream);
```

**Parameters**

Parameters	Description
System.Exception throwable	Exception to obtain information from.
System.IO.TextWriter stream	Output stream used to write to.

## 2.1.16.3 SupportClass Delegates

### 2.1.16.3.1 SupportClass.IntegerMillisecondsDelegate Delegate

C#

```
public delegate int IntegerMillisecondsDelegate();
```

#### Description

This is nested type SupportClass.IntegerMillisecondsDelegate.

## 2.1.17 TrafficStats Class


C#

```
public class TrafficStats;
```

















#### Description

This is class TrafficStats.

#### TrafficStats Methods

	Name	Description
	<a href="#">ToString</a>	This is ToString, a member of class TrafficStats.

#### TrafficStats Properties

	Name	Description
	<a href="#">ControlCommandBytes</a>	This is ControlCommandBytes, a member of class TrafficStats.
	<a href="#">ControlCommandCount</a>	This is ControlCommandCount, a member of class TrafficStats.
	<a href="#">FragmentCommandBytes</a>	This is FragmentCommandBytes, a member of class TrafficStats.
	<a href="#">FragmentCommandCount</a>	This is FragmentCommandCount, a member of class TrafficStats.
	<a href="#">PackageHeaderSize</a>	Gets the byte-size of per-package headers.
	<a href="#">ReliableCommandBytes</a>	This is ReliableCommandBytes, a member of class TrafficStats.
	<a href="#">ReliableCommandCount</a>	Counts commands created/received by this client, ignoring repeats (out command count can be higher due to repeats).
	<a href="#">TimestampOfLastAck</a>	Timestamp of the last incoming ACK read (every second this client sends a PING which must be ACKd).
	<a href="#">TimestampOfLastReliableCommand</a>	Timestamp of last incoming reliable command (every second we expect a PING).
	<a href="#">TotalCommandBytes</a>	This is TotalCommandBytes, a member of class TrafficStats.
	<a href="#">TotalCommandCount</a>	This is TotalCommandCount, a member of class TrafficStats.
	<a href="#">TotalCommandsInPackets</a>	This is TotalCommandsInPackets, a member of class TrafficStats.
	<a href="#">TotalPacketBytes</a>	Gets count of bytes as traffic, excluding UDP/TCP headers (42 bytes / x bytes).
	<a href="#">TotalPacketCount</a>	This is TotalPacketCount, a member of class TrafficStats.
	<a href="#">UnreliableCommandBytes</a>	This is UnreliableCommandBytes, a member of class TrafficStats.
	<a href="#">UnreliableCommandCount</a>	This is UnreliableCommandCount, a member of class TrafficStats.

### 2.1.17.1 TrafficStats Methods

### 2.1.17.1.1 TrafficStats.ToString Method

C#

```
public override string ToString();
```

#### Description

This is ToString, a member of class TrafficStats.

## 2.1.17.2 TrafficStats Properties

### 2.1.17.2.1 TrafficStats.ControlCommandBytes Property

C#

```
public int ControlCommandBytes;
```

#### Description

This is ControlCommandBytes, a member of class TrafficStats.

### 2.1.17.2.2 TrafficStats.ControlCommandCount Property

C#

```
public int ControlCommandCount;
```

#### Description

This is ControlCommandCount, a member of class TrafficStats.

### 2.1.17.2.3 TrafficStats.FragmentCommandBytes Property

C#

```
public int FragmentCommandBytes;
```

#### Description

This is FragmentCommandBytes, a member of class TrafficStats.

### 2.1.17.2.4 TrafficStats.FragmentCommandCount Property

C#

```
public int FragmentCommandCount;
```

#### Description

This is FragmentCommandCount, a member of class TrafficStats.

### 2.1.17.2.5 TrafficStats.PackageHeaderSize Property

Gets the byte-size of per-package headers.

C#

```
public int PackageHeaderSize;
```

### 2.1.17.2.6 TrafficStats.ReliableCommandBytes Property

C#

```
public int ReliableCommandBytes;
```

#### Description

This is ReliableCommandBytes, a member of class TrafficStats.

### 2.1.17.2.7 TrafficStats.ReliableCommandCount Property

Counts commands created/received by this client, ignoring repeats (out command count can be higher due to repeats).

C#

```
public int ReliableCommandCount;
```

### 2.1.17.2.8 TrafficStats.TimestampOfLastAck Property

Timestamp of the last incoming ACK read (every second this client sends a PING which must be ACKd).

C#

```
public int TimestampOfLastAck;
```

### 2.1.17.2.9 TrafficStats.TimestampOfLastReliableCommand Property

Timestamp of last incoming reliable command (every second we expect a PING).

C#

```
public int TimestampOfLastReliableCommand;
```

### 2.1.17.2.10 TrafficStats.TotalCommandBytes Property

C#

```
public int TotalCommandBytes;
```

#### Description

This is TotalCommandBytes, a member of class TrafficStats.

### 2.1.17.2.11 TrafficStats.TotalCommandCount Property

C#

```
public int TotalCommandCount;
```

#### Description

This is TotalCommandCount, a member of class TrafficStats.

### 2.1.17.2.12 TrafficStats.TotalCommandsInPackets Property

C#

```
public int TotalCommandsInPackets;
```

#### Description

This is TotalCommandsInPackets, a member of class TrafficStats.

### 2.1.17.2.13 TrafficStats.TotalPacketBytes Property

Gets count of bytes as traffic, excluding UDP/TCP headers (42 bytes / x bytes).

C#

```
public int TotalPacketBytes;
```

### 2.1.17.2.14 TrafficStats.TotalPacketCount Property

C#

```
public int TotalPacketCount;
```

#### Description

This is TotalPacketCount, a member of class TrafficStats.

### 2.1.17.2.15 TrafficStats.UnreliableCommandBytes Property

C#

```
public int UnreliableCommandBytes;
```

#### Description

This is UnreliableCommandBytes, a member of class TrafficStats.

### 2.1.17.2.16 TrafficStats.UnreliableCommandCount Property

C#

```
public int UnreliableCommandCount;
```

#### Description

This is UnreliableCommandCount, a member of class TrafficStats.




## 2.1.18 TrafficStatsGameLevel Class

Only in use as long as `PhotonPeer.TrafficStatsEnabled` = true;





C#


















```
public class TrafficStatsGameLevel;
```

#### TrafficStatsGameLevel Methods

	Name	Description
	<a href="#">ResetMaximumCounters</a>	Resets the values that can be maxed out, like <a href="#">LongestDeltaBetweenDispatching</a> . See remarks.
	<a href="#">ToString</a>	This is ToString, a member of class TrafficStatsGameLevel.
	<a href="#">ToStringVitalStats</a>	This is ToStringVitalStats, a member of class TrafficStatsGameLevel.

#### TrafficStatsGameLevel Properties

	Name	Description
	<a href="#">DispatchCalls</a>	Gets number of calls of DispatchIncomingCommands.
	<a href="#">DispatchIncomingCommandsCalls</a>	Gets number of calls of DispatchIncomingCommands.
	<a href="#">EventByteCount</a>	Gets sum of byte-cost of incoming <a href="#">events</a> .
	<a href="#">EventCount</a>	Gets count of incoming <a href="#">events</a> .

	<b>LongestDeltaBetweenDispatching</b>	Gets longest time between subsequent calls to DispatchIncomingCommands in milliseconds.
	<b>LongestDeltaBetweenSending</b>	Gets longest time between subsequent calls to SendOutgoingCommands in milliseconds.
	<b>LongestEventCallback</b>	Gets longest time a call to OnEvent (in your code) took. If such a callback takes long, it will lower the network performance and might lead to timeouts.
	<b>LongestEventCallbackCode</b>	Gets EventCode that caused the <b>LongestEventCallback</b> . See that description.
	<b>LongestOpResponseCallback</b>	Gets longest time it took to complete a call to OnOperationResponse (in your code). If such a callback takes long, it will lower the network performance and might lead to timeouts.
	<b>LongestOpResponseCallbackOpCode</b>	Gets OperationCode that causes the <b>LongestOpResponseCallback</b> . See that description.
	<b>OperationByteCount</b>	Gets sum of outgoing operations in bytes.
	<b>OperationCount</b>	Gets count of outgoing operations.
	<b>ResultByteCount</b>	Gets sum of byte-cost of incoming operation-results.
	<b>ResultCount</b>	Gets count of incoming operation-results.
	<b>SendOutgoingCommandsCalls</b>	Gets number of calls of SendOutgoingCommands.
	<b>TotalByteCount</b>	Gets sum of byte-cost of all "logic level" messages.
	<b>TotalIncomingByteCount</b>	Gets sum of byte-cost of all incoming "logic level" messages.
	<b>TotalIncomingMessageCount</b>	Gets sum of counted incoming "logic level" messages.
	<b>TotalMessageCount</b>	Gets sum of counted "logic level" messages.
	<b>TotalOutgoingByteCount</b>	Gets sum of byte-cost of all outgoing "logic level" messages (= <b>OperationByteCount</b> ).
	<b>TotalOutgoingMessageCount</b>	Gets sum of counted outgoing "logic level" messages (= <b>OperationCount</b> ).

## 2.1.18.1 TrafficStatsGameLevel Methods

### 2.1.18.1.1 TrafficStatsGameLevel.ResetMaximumCounters Method

Resets the values that can be maxed out, like **LongestDeltaBetweenDispatching**. See remarks.

**C#**

```
public void ResetMaximumCounters();
```

**Remarks**

Set to 0: **LongestDeltaBetweenDispatching**, **LongestDeltaBetweenSending**, **LongestEventCallback**, **LongestEventCallbackCode**, **LongestOpResponseCallback**, **LongestOpResponseCallbackOpCode**. Also resets internal values: timeOfLastDispatchCall and timeOfLastSendCall (so intervals are tracked correctly).

### 2.1.18.1.2 TrafficStatsGameLevel.ToString Method

**C#**

```
public override string ToString();
```

**Description**

This is ToString, a member of class TrafficStatsGameLevel.

### 2.1.18.1.3 TrafficStatsGameLevel.ToStringVitalStats Method

C#

```
public string ToStringVitalStats();
```

#### Description

This is ToStringVitalStats, a member of class TrafficStatsGameLevel.

## 2.1.18.2 TrafficStatsGameLevel Properties

### 2.1.18.2.1 TrafficStatsGameLevel.DispatchCalls Property

Gets number of calls of DispatchIncomingCommands.

C#

```
[Obsolete("Use DispatchIncomingCommandsCalls, which has proper naming.")]  
public int DispatchCalls;
```

### 2.1.18.2.2 TrafficStatsGameLevel.DispatchIncomingCommandsCalls Property

Gets number of calls of DispatchIncomingCommands.

C#

```
public int DispatchIncomingCommandsCalls;
```

### 2.1.18.2.3 TrafficStatsGameLevel.EventByteCount Property

Gets sum of byte-cost of incoming [events](#).

C#

```
public int EventByteCount;
```

### 2.1.18.2.4 TrafficStatsGameLevel.EventCount Property

Gets count of incoming [events](#).

C#

```
public int EventCount;
```

### 2.1.18.2.5 TrafficStatsGameLevel.LongestDeltaBetweenDispatching Property

Gets longest time between subsequent calls to DispatchIncomingCommands in milliseconds.

C#

```
public int LongestDeltaBetweenDispatching;
```

#### Notes

This is not a crucial timing for the networking. Long gaps just add "local lag" to [events](#) that are available already.

### 2.1.18.2.6 TrafficStatsGameLevel.LongestDeltaBetweenSending Property

Gets longest time between subsequent calls to SendOutgoingCommands in milliseconds.

**C#**

```
public int LongestDeltaBetweenSending;
```

**Notes**

This is a crucial value for network stability. Without calling `SendOutgoingCommands`, nothing will be sent to the server, who might time out this client.

### 2.1.18.2.7 TrafficStatsGameLevel.LongestEventCallback Property

Gets longest time a call to `OnEvent` (in your code) took. If such a callback takes long, it will lower the network performance and might lead to timeouts.

**C#**

```
public int LongestEventCallback;
```

### 2.1.18.2.8 TrafficStatsGameLevel.LongestEventCallbackCode Property

Gets `EventCode` that caused the [LongestEventCallback](#). See that description.

**C#**

```
public byte LongestEventCallbackCode;
```

### 2.1.18.2.9 TrafficStatsGameLevel.LongestOpResponseCallback Property

Gets longest time it took to complete a call to `OnOperationResponse` (in your code). If such a callback takes long, it will lower the network performance and might lead to timeouts.

**C#**

```
public int LongestOpResponseCallback;
```

### 2.1.18.2.10 TrafficStatsGameLevel.LongestOpResponseCallbackOpCode Property

Gets `OperationCode` that causes the [LongestOpResponseCallback](#). See that description.

**C#**

```
public byte LongestOpResponseCallbackOpCode;
```

### 2.1.18.2.11 TrafficStatsGameLevel.OperationByteCount Property

Gets sum of outgoing operations in bytes.

**C#**

```
public int OperationByteCount;
```

### 2.1.18.2.12 TrafficStatsGameLevel.OperationCount Property

Gets count of outgoing operations.

**C#**

```
public int OperationCount;
```

### 2.1.18.2.13 TrafficStatsGameLevel.ResultByteCount Property

Gets sum of byte-cost of incoming operation-results.



**C#**

```
public int ResultByteCount;
```

### 2.1.18.2.14 TrafficStatsGameLevel.ResultCount Property

Gets count of incoming operation-results.

**C#**

```
public int ResultCount;
```

### 2.1.18.2.15 TrafficStatsGameLevel.SendOutgoingCommandsCalls Property

Gets number of calls of SendOutgoingCommands.

**C#**

```
public int SendOutgoingCommandsCalls;
```

### 2.1.18.2.16 TrafficStatsGameLevel.TotalByteCount Property

Gets sum of byte-cost of all "logic level" messages.

**C#**

```
public int TotalByteCount;
```

### 2.1.18.2.17 TrafficStatsGameLevel.TotalIncomingByteCount Property

Gets sum of byte-cost of all incoming "logic level" messages.

**C#**

```
public int TotalIncomingByteCount;
```

### 2.1.18.2.18 TrafficStatsGameLevel.TotalIncomingMessageCount Property

Gets sum of counted incoming "logic level" messages.

**C#**

```
public int TotalIncomingMessageCount;
```

### 2.1.18.2.19 TrafficStatsGameLevel.TotalMessageCount Property

Gets sum of counted "logic level" messages.

**C#**

```
public int TotalMessageCount;
```

### 2.1.18.2.20 TrafficStatsGameLevel.TotalOutgoingByteCount Property

Gets sum of byte-cost of all outgoing "logic level" messages (= [OperationByteCount](#)).

**C#**

```
public int TotalOutgoingByteCount;
```

### 2.1.18.2.21 TrafficStatsGameLevel.TotalOutgoingMessageCount Property

Gets sum of counted outgoing "logic level" messages (= [OperationCount](#)).


**C#**

```
public int TotalOutgoingMessageCount;
```

## 2.2 Interfaces

The following table lists interfaces in this documentation.

### Interfaces

	Name	Description
	<a href="#">IPhotonPeerListener</a>	Callback interface for the Photon client side. Must be provided to a new <a href="#">PhotonPeer</a> in its constructor.

### 2.2.1 IPhotonPeerListener Interface

Callback interface for the Photon client side. Must be provided to a new [PhotonPeer](#) in its constructor.





#### C#

```
public interface IPhotonPeerListener;
```

#### Remarks

These methods are used by your [PhotonPeer](#) instance to keep your app updated. Read each method's description and check out the samples to see how to use them.

#### IPhotonPeerListener Methods

	Name	Description
	<a href="#">DebugReturn</a>	Provides textual descriptions for various error conditions and noteworthy situations. In cases where the application needs to react, a call to <a href="#">OnStatusChanged</a> is used. <a href="#">OnStatusChanged</a> gives "feedback" to the game, DebugReturn provies human readable messages on the background.
	<a href="#">OnEvent</a>	Called whenever an event from the Photon Server is dispatched.
	<a href="#">OnOperationResponse</a>	Callback method which gives you (async) responses for called operations.
	<a href="#">OnStatusChanged</a>	OnStatusChanged is called to let the game know when asynchronous actions finished or when errors happen.

#### 2.2.1.1 IPhotonPeerListener Methods

##### 2.2.1.1.1 IPhotonPeerListener.DebugReturn Method

Provides textual descriptions for various error conditions and noteworthy situations. In cases where the application needs to react, a call to [OnStatusChanged](#) is used. [OnStatusChanged](#) gives "feedback" to the game, DebugReturn provies human readable messages on the background.

#### C#

```
void DebugReturn(DebugLevel level, string message);
```

#### Parameters

Parameters	Description
DebugLevel level	<a href="#">DebugLevel</a> (severity) of the message.
string message	Debug text. Print to System.Console or screen.

### Remarks

All debug output of the library will be reported through this method. Print it or put it in a buffer to use it on-screen. Use [PhotonPeer.DebugOut](#) to select how verbose the output is.

## 2.2.1.1.2 IPhotonPeerListener.OnEvent Method

Called whenever an event from the Photon Server is dispatched.

### C#

```
void OnEvent(EventData eventData);
```

### Parameters

Parameters	Description
EventData eventData	The event currently being dispatched.

### Remarks

[Events](#) are used for communication between clients and allow the server to update clients over time. The creation of an event is often triggered by an operation (called by this client or an other).

Each event carries its specific content in its Parameters. Your application knows which content to expect by checking the event's 'type', given by the event's Code.

[Events](#) can be defined and extended server-side.

If you use the Lite application as base, several [events](#) like EvJoin and EvLeave are already defined. For these [events](#) and their Parameters, the library provides constants, so check: [LiteEventCode](#) and [LiteEventKey](#) classes. Lite also allows you to come up with custom [events](#) on the fly, purely client-side. To do so, use [LitePeer.OpRaiseEvent](#).

[Events](#) are buffered on the client side and must be Dispatched. This way, OnEvent is always taking place in the same thread as a [PhotonPeer.DispatchIncomingCommands](#) call.

## 2.2.1.1.3 IPhotonPeerListener.OnOperationResponse Method

Callback method which gives you (async) responses for called operations.

### C#

```
void OnOperationResponse(OperationResponse operationResponse);
```

### Parameters

Parameters	Description
OperationResponse operationResponse	The response to an operation-call.

### Remarks

Like methods, operations can have a result. As operation-calls are non-blocking, the response for any operation of this peer is provided by this method. As example: Joining a room on a Lite-based server will return a list of players currently in game, your actorNumber and some other values.

This method is used as general callback for all operations. Each response corresponds to a certain "type" of operation by its OperationCode (see: [Operations](#)).

The "Lite Application" uses these OpCodes:

- [LiteOpCode.Join](#) for OpJoin, contains the actorNr of "this" player
- [LiteOpCode.Leave](#) when leaving a room
- [LiteOpCode.RaiseEvent](#) for OpRaiseEvent, if this was sent as reliable command
- [LiteOpCode.SetProperties](#) for OpSetPropertiesOfActor and OpSetPropertiesOfGame

### Example

When you join a room, the server will assign a consecutive number to each client: the "actorNr" or "player number". This is sent back in the OperationResult's Parameters as value of key `LiteEventKey.ActorNr`.

Fetch your actorNr of a Join response like this:

```
int actorNr = (int)operationResponse[(byte)LiteOpKey.ActorNr];
```

#### 2.2.1.1.4 IPhotonPeerListener.OnStatusChanged Method

OnStatusChanged is called to let the game know when asynchronous actions finished or when errors happen.

### C#

```
void OnStatusChanged(StatusCode statusCode);
```

### Parameters

Parameters	Description
StatusCode statusCode	A code to identify the situation.

### Remarks












Not all of the many `StatusCode` values will apply to your game. Example: If you don't use encryption, the respective status changes are never made.

The values are all part of the `StatusCode` enumeration and described value-by-value.

## 2.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

### Enumerations

	Name	Description
	<code>ConnectionProtocol</code>	These are the options that can be used as underlying transport protocol.
	<code>DebugLevel</code>	Level / amount of DebugReturn callbacks. Each debug level includes output for lower ones: OFF, ERROR, WARNING, INFO, ALL.
	<code>EventCaching</code>	Lite - OpRaiseEvent allows you to cache <b>events</b> and automatically send them to joining players in a room. <b>Events</b> are cached per event code and player: <b>Event</b> 100 (example!) can be stored once per player. Cached <b>events</b> can be modified, replaced and removed.
	<code>GpType</code>	The gp type.
	<code>LitePropertyTypes</code>	Lite - Flags for "types of properties", being used as filter in OpGetProperties.
	<code>PeerStateValue</code>	Value range for a Peer's connection and initialization state, as returned by the PeerState property.
	<code>PhotonDisconnectCause</code>	This is record PhotonDisconnectCause.
	<code>PhotonSocketError</code>	This is record PhotonSocketError.
	<code>PhotonSocketState</code>	This is record PhotonSocketState.
	<code>ReceiverGroup</code>	Lite - OpRaiseEvent lets you chose which actors in the room should receive <b>events</b> . By default, <b>events</b> are sent to "Others" but you can overrule this.
	<code>StatusCode</code>	Enumeration of situations that change the peers internal status. Used in calls to OnStatusChanged to inform your application of various situations that might happen.

## 2.3.1 ConnectionProtocol Enumeration

These are the options that can be used as underlying transport protocol.

C#

```
public enum ConnectionProtocol : byte {  
    Udp = 0,  
    Tcp = 1,  
    RHttp = 3  
}
```

Members

Members	Description
Udp = 0	Use UDP to connect to Photon, which allows you to send operations reliable or unreliable on demand.
Tcp = 1	Use TCP to connect to Photon.
RHttp = 3	Use Realtime HTTP connections to connect a Photon Master (not available in regular Photon SDK).

## 2.3.2 DebugLevel Enumeration

Level / amount of DebugReturn callbacks. Each debug level includes output for lower ones: OFF, ERROR, WARNING, INFO, ALL.

C#

```
public enum DebugLevel : byte {  
    OFF = 0,  
    ERROR = 1,  
    WARNING = 2,  
    INFO = 3,  
    ALL = 5  
}
```

Members

Members	Description
OFF = 0	No debug out.
ERROR = 1	Only error descriptions.
WARNING = 2	Warnings and errors.
INFO = 3	Information about internal workflows, warnings and errors.
ALL = 5	Most complete workflow description (but lots of debug output), info, warnings and errors.

## 2.3.3 EventCaching Enumeration

Lite - OpRaiseEvent allows you to cache [events](#) and automatically send them to joining players in a room. [Events](#) are cached per event code and player: [Event](#) 100 (example!) can be stored once per player. Cached [events](#) can be modified, replaced and removed.

C#

```
public enum EventCaching : byte {  
    DoNotCache = 0,  
}
```

```

MergeCache = 1,
ReplaceCache = 2,
RemoveCache = 3,
AddToRoomCache = 4,
AddToRoomCacheGlobal = 5,
RemoveFromRoomCache = 6,
RemoveFromRoomCacheForActorsLeft = 7
}

```

#### Members

Members	Description
DoNotCache = 0	Default value (not sent).
MergeCache = 1	Will merge this event's keys with those already cached.
ReplaceCache = 2	Replaces the event cache for this eventCode with this event's content.
RemoveCache = 3	Removes this event (by eventCode) from the cache.
AddToRoomCache = 4	Adds an event to the room's cache.
AddToRoomCacheGlobal = 5	Adds this event to the cache for actor 0 (becoming a "globally owned" event in the cache).
RemoveFromRoomCache = 6	Remove fitting event from the room's cache.
RemoveFromRoomCacheForActorsLeft = 7	Removes <b>events</b> of players who already left the room (cleaning up).

#### Remarks

Caching works only combination with **ReceiverGroup** options Others and All.

## 2.3.4 GpType Enumeration

The gp type.

#### C#

```

internal enum GpType : byte {
    Unknown = 0,
    Array = (byte)'y',
    Boolean = (byte)'o',
    Byte = (byte)'b',
    ByteArray = (byte)'x',
    ObjectArray = (byte)'z',
    Short = (byte)'k',
    Float = (byte)'f',
    Dictionary = (byte)'D',
    Double = (byte)'d',
    Hashtable = (byte)'h',
    Integer = (byte)'i',
    IntegerArray = (byte)'n',
    Long = (byte)'l',
    String = (byte)'s',
    StringArray = (byte)'a',
    Custom = (byte)'c',
    Null = (byte)*',
    EventData = (byte)'e',
    OperationRequest = (byte)'q',
    OperationResponse = (byte)'p'
}

```

#### Members

Members	Description
Unknown = 0	Unkown type.
Array = (byte)'y'	0x79 (121)

Boolean = (byte)'o'	0x6F
Byte = (byte)'b'	A byte value.
ByteArray = (byte)'x'	An array of bytes.
ObjectArray = (byte)'z'	An array of objects.
Short = (byte)'k'	A 16-bit integer value.
Float = (byte)'f'	A 32-bit floating-point value.
Dictionary = (byte)'D'	0x44 (68)
Double = (byte)'d'	A 64-bit floating-point value.
Hashtable = (byte)'h'	A Hashtable.
Integer = (byte)'i'	0x69 (105)
IntegerArray = (byte)'n'	An array of 32-bit integer values.
Long = (byte)'l'	A 64-bit integer value.
String = (byte)'s'	A string value.
StringArray = (byte)'a'	An array of string values.
Custom = (byte)'c'	A costum type
Null = (byte)''*	Null value don't have types.

## 2.3.5 LitePropertyTypes Enumeration

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

**C#**

```
[Flags]
public enum LitePropertyTypes : byte {
    None = 0x00,
    Game = 0x01,
    Actor = 0x02,
    GameAndActor = Game | Actor
}
```

**Members**

Members	Description
None = 0x00	(0x00) Flag type for no property type.
Game = 0x01	(0x01) Flag type for game-attached properties.
Actor = 0x02	(0x02) Flag type for actor related propeties.
GameAndActor = Game Actor	(0x01) Flag type for game AND actor properties. Equal to 'Game'

## 2.3.6 PeerStateValue Enumeration

Value range for a Peer's connection and initialization state, as returned by the PeerState property.

**C#**

```
public enum PeerStateValue : byte {
    Disconnected = 0,
    Connecting = 1,
    InitializingApplication = 10,
    Connected = 3,
    Disconnecting = 4
}
```

**Members**

Members	Description
Disconnected = 0	The peer is disconnected and can't call <b>Operations</b> . Call <b>Connect()</b> .
Connecting = 1	The peer is establishing the connection: opening a socket, exchanging packages with Photon.
InitializingApplication = 10	The connection is established and now sends the application name to Photon.
Connected = 3	The peer is connected and initialized (selected an application). You can now use operations.
Disconnecting = 4	The peer is disconnecting. It sent a disconnect to the server, which will acknowledge closing the connection.

**Remarks**

While this is not the same as the **StatusCode** of **IPhotonPeerListener.OnStatusChanged()**, it directly relates to it. In most cases, it makes more sense to build a game's state on top of the **OnStatusChanged()** as you get changes.

---

## 2.3.7 PhotonDisconnectCause Enumeration

**C#**

```
public enum PhotonDisconnectCause {  
    SecurityExceptionOnConnect,  
    ExceptionOnConnect,  
    Exception,  
    ReadException,  
    WriteException  
}
```

**Description**

This is record PhotonDisconnectCause.

---

## 2.3.8 PhotonSocketError Enumeration

**C#**

```
public enum PhotonSocketError {  
    Success,  
    Skipped,  
    NoData,  
    Exception  
}
```

**Description**

This is record PhotonSocketError.

---

## 2.3.9 PhotonSocketState Enumeration

**C#**

```
public enum PhotonSocketState {  
    Disconnected,  
    Connecting,  
}
```



```
    Connected,  
    Disconnecting  
}
```

#### Description

This is record PhotonSocketState.

## 2.3.10 ReceiverGroup Enumeration

Lite - OpRaiseEvent lets you chose which actors in the room should receive **events**. By default, **events** are sent to "Others" but you can overrule this.

#### C#

```
public enum ReceiverGroup : byte {  
    Others = 0,  
    All = 1,  
    MasterClient = 2  
}
```

#### Members

Members	Description
Others = 0	Default value (not sent). Anyone else gets my event.
All = 1	Everyone in the current room (including this peer) will get this event.
MasterClient = 2	The server sends this event only to the actor with the lowest actorNumber.

## 2.3.11 StatusCode Enumeration

Enumeration of situations that change the peers internal status. Used in calls to OnStatusChanged to inform your application of various situations that might happen.

#### C#

```
public enum StatusCode : int {  
    Connect = 1024,  
    Disconnect = 1025,  
    Exception = 1026,  
    ExceptionOnConnect = 1023,  
    SecurityExceptionOnConnect = 1022,  
    QueueOutgoingReliableWarning = 1027,  
    QueueOutgoingUnreliableWarning = 1029,  
    SendError = 1030,  
    QueueOutgoingAcksWarning = 1031,  
    QueueIncomingReliableWarning = 1033,  
    QueueIncomingUnreliableWarning = 1035,  
    QueueSentWarning = 1037,  
    ExceptionOnReceive = 1039,  
    InternalReceiveException = 1039,  
    TimeoutDisconnect = 1040,  
    DisconnectByServer = 1041,  
    DisconnectByServerUserLimit = 1042,  
    DisconnectByServerLogic = 1043,  
    TcpRouterResponseOk = 1044,  
    TcpRouterResponseNodeIdUnknown = 1045,  
    TcpRouterResponseEndpointUnknown = 1046,  
    TcpRouterResponseNodeNotReady = 1047,  
    EncryptionEstablished = 1048,  
    EncryptionFailedToEstablish = 1049  
}
```

**Members**

Members	Description
Connect = 1024	the <b>PhotonPeer</b> is connected. See {@link PhotonListener#OnStatusChanged}*
Disconnect = 1025	the <b>PhotonPeer</b> just disconnected. See {@link PhotonListener#OnStatusChanged}*
Exception = 1026	the <b>PhotonPeer</b> encountered an exception and will disconnect, too. See {@link PhotonListener#OnStatusChanged}*
ExceptionOnConnect = 1023	the <b>PhotonPeer</b> encountered an exception while opening the incoming connection to the server. The server could be down / not running or the client has no network or a misconfigured DNS. See {@link PhotonListener#OnStatusChanged}*
SecurityExceptionOnConnect = 1022	Used on platforms that throw a security exception on connect. Unity3d does this, e.g., if a webplayer build could not fetch a policy-file from a remote server.
QueueOutgoingReliableWarning = 1027	<b>PhotonPeer</b> outgoing queue is filling up. send more often.
QueueOutgoingUnreliableWarning = 1029	<b>PhotonPeer</b> outgoing queue is filling up. send more often.
SendError = 1030	Sending command failed. Either not connected, or the requested channel is bigger than the number of initialized channels.
QueueOutgoingAcksWarning = 1031	<b>PhotonPeer</b> outgoing queue is filling up. send more often.
QueueIncomingReliableWarning = 1033	<b>PhotonPeer</b> incoming queue is filling up. Dispatch more often.
QueueIncomingUnreliableWarning = 1035	<b>PhotonPeer</b> incoming queue is filling up. Dispatch more often.
QueueSentWarning = 1037	<b>PhotonPeer</b> incoming queue is filling up. Dispatch more often.
ExceptionOnReceive = 1039	Exception, if a server cannot be connected. Most likely, the server is not responding. Ask user to try again later.
TimeoutDisconnect = 1040	Disconnection due to a timeout (client did no longer receive ACKs from server).
DisconnectByServer = 1041	Disconnect by server due to timeout (received a disconnect command, cause server misses ACKs of client).
DisconnectByServerUserLimit = 1042	Disconnect by server due to concurrent user limit reached (received a disconnect command).
DisconnectByServerLogic = 1043	Disconnect by server due to server's logic (received a disconnect command).
EncryptionEstablished = 1048	(1048) Value for OnStatusChanged()-call, when the encryption-setup for secure communication finished successfully.
EncryptionFailedToEstablish = 1049	(1049) Value for OnStatusChanged()-call, when the encryption-setup failed for some reason. Check debug logs.

**Remarks**

Most of these codes are referenced somewhere else in the documentation when they are relevant to methods.

## 2.4 Types

The following table lists types in this documentation.

## Types

Name	Description
<a href="#">DeserializeMethod</a>	Type of deserialization methods to add custom type support. Use <a href="#">PhotonPeer.RegisterType()</a> to register new types with serialization and deserialization methods.
<a href="#">SerializeMethod</a>	Type of serialization methods to add custom type support. Use <a href="#">PhotonPeer.ReisterType()</a> to register new types with serialization and deserialization methods.

## 2.4.1 DeserializeMethod Type

Type of deserialization methods to add custom type support. Use [PhotonPeer.RegisterType\(\)](#) to register new types with serialization and deserialization methods.

### C#

```
public delegate object DeserializeMethod(byte[] serializedCustomObject);
```

### Parameters

Parameters	Description
serializedCustomObject	The framwork passes in the data it got by the associated <a href="#">SerializeMethod</a> . The type code and length are stripped and applied before a DeserializeMethod is called.

### Returns

Return a object of the type that was associated with this method through RegisterType().

## 2.4.2 SerializeMethod Type

Type of serialization methods to add custom type support. Use [PhotonPeer.ReisterType\(\)](#) to register new types with serialization and deserialization methods.

### C#

```
public delegate byte SerializeMethod(object customObject);
```

### Parameters

Parameters	Description
customObject	The method will get objects passed that were registered with it in RegisterType().

### Returns

Return a byte[] that resembles the object passed in. The framework will surround it with length and type info, so don't include it.

## Index

### A

AcknowledgingDisconnect enumeration member 42  
Actor enumeration member 86  
AddToRoomCache enumeration member 84  
AddToRoomCacheGlobal enumeration member 84  
All enumeration member 88  
ALL enumeration member 84  
Array enumeration member 85

### B

Boolean enumeration member 85  
Byte enumeration member 85  
ByteArray enumeration member 85

### C

Classes 10  
Connect enumeration member 88  
Connected enumeration member 42, 86, 87  
Connecting enumeration member 42, 86, 87  
ConnectionProtocol 84  
ConnectionProtocol enumeration 84  
Custom Authentication 8  
Custom enumeration member 85

### D

DebugLevel 84  
DebugLevel enumeration 84  
DeserializeMethod 90  
DeserializeMethod type 90  
Dictionary enumeration member 85  
Disconnect enumeration member 88  
DisconnectByServer enumeration member 88  
DisconnectByServerLogic enumeration member 88  
DisconnectByServerUserLimit enumeration member 88  
Disconnected enumeration member 42, 86, 87  
Disconnecting enumeration member 42, 86, 87  
DoNotCache enumeration member 84  
Double enumeration member 85

### E

EncryptionEstablished enumeration member 88  
EncryptionFailedToEstablish enumeration member 88  
ERROR enumeration member 84  
EventCaching 84  
EventCaching enumeration 84  
EventData 11  
EventData class 11  
    about EventData class 11  
    Code 11  
    Parameters 11  
    this 11  
    ToString 12  
    ToStringFull 12  
EventData enumeration member 85  
EventData.Code 11  
EventData.Parameters 11  
EventData.this 11  
EventData.ToString 12  
EventData.ToStringFull 12  
Events 3  
Exception enumeration member 87, 88  
ExceptionOnConnect enumeration member 87, 88  
ExceptionOnReceive enumeration member 88

### F

Float enumeration member 85  
Fragmentation and Channels 3  
Further Help 9

### G

Game enumeration member 86  
GameAndActor enumeration member 86  
GpType 85  
GpType enumeration 85

### H

Hashtable enumeration member 85

## I

- INFO enumeration member 84
- InitializingApplication enumeration member 86
- Integer enumeration member 85
- IntegerArray enumeration member 85
- Interfaces 81
- InternalReceiveException enumeration member 88
- IPhotonPeerListener 81
- IPhotonPeerListener interface 81
  - about IPhotonPeerListener interface 81
  - DebugReturn 81
  - OnEvent 82
  - OnOperationResponse 82
  - OnStatusChanged 83
- IPhotonPeerListener.DebugReturn 81
- IPhotonPeerListener.OnEvent 82
- IPhotonPeerListener.OnOperationResponse 82
- IPhotonPeerListener.OnStatusChanged 83
- IPhotonSocket 12
- IPhotonSocket class 12
  - about IPhotonSocket class 12
  - Connect 13
  - Connected 14
  - Disconnect 13
  - EnqueueDebugReturn 14
  - HandleReceivedDatagram 14
  - IPhotonSocket 13
  - Listener 15
  - MTU 15
  - PollReceive 13
  - Protocol 15
  - Receive 14
  - ReportDebugOfLevel 14
  - Send 14
  - ServerAddress 15
  - ServerPort 15
  - State 15
- IPhotonSocket.Connect 13
- IPhotonSocket.Connected 14
- IPhotonSocket.Disconnect 13
- IPhotonSocket.EnqueueDebugReturn 14

- IPhotonSocket.HandleReceivedDatagram 14
- IPhotonSocket.IPhotonSocket 13
- IPhotonSocket.Listener 15
- IPhotonSocket.MTU 15
- IPhotonSocket.PollReceive 13
- IPhotonSocket.Protocol 15
- IPhotonSocket.Receive 14
- IPhotonSocket.ReportDebugOfLevel 14
- IPhotonSocket.Send 14
- IPhotonSocket.ServerAddress 15
- IPhotonSocket.ServerPort 15
- IPhotonSocket.State 15

## L

- Lite Application 7
- LiteEventCode 16
- LiteEventCode class 16
  - about LiteEventCode class 16
  - Join 16
  - Leave 16
  - PropertiesChanged 16
- LiteEventCode.Join 16
- LiteEventCode.Leave 16
- LiteEventCode.PropertiesChanged 16
- LiteEventKey 16
- LiteEventKey class 16
  - about LiteEventKey class 16
  - ActorList 17
  - ActorNr 17
  - ActorProperties 17
  - CustomContent 17
  - Data 17
  - GameProperties 18
  - Properties 18
  - TargetActorNr 18
- LiteEventKey.ActorList 17
- LiteEventKey.ActorNr 17
- LiteEventKey.ActorProperties 17
- LiteEventKey.CustomContent 17
- LiteEventKey.Data 17
- LiteEventKey.GameProperties 18
- LiteEventKey.Properties 18

---

LiteEventKey.TargetActorNr 18  
 LiteOpCode 18  
 LiteOpCode class 18  
     about LiteOpCode class 18  
     ChangeGroups 18  
     ExchangeKeysForEncryption 19  
     GetProperties 19  
     Join 19  
     Leave 19  
     RaiseEvent 19  
     SetProperties 19  
 LiteOpCode.ChangeGroups 18  
 LiteOpCode.ExchangeKeysForEncryption 19  
 LiteOpCode.GetProperties 19  
 LiteOpCode.Join 19  
 LiteOpCode.Leave 19  
 LiteOpCode.RaiseEvent 19  
 LiteOpCode.SetProperties 19  
 LiteOpKey 19  
 LiteOpKey class 19  
     about LiteOpKey class 19  
     ActorList 20  
     ActorNr 20  
     ActorProperties 21  
     Add 21  
     Asid 21  
     Broadcast 21  
     Cache 21  
     Code 21  
     Data 21  
     Gameld 21  
     GameProperties 22  
     Group 22  
     Properties 22  
     ReceiverGroup 22  
     Remove 22  
     RoomName 22  
     TargetActorNr 22  
 LiteOpKey.ActorList 20  
 LiteOpKey.ActorNr 20  
 LiteOpKey.ActorProperties 21  
 LiteOpKey.Add 21  
 LiteOpKey.Asid 21  
 LiteOpKey.Broadcast 21  
 LiteOpKey.Cache 21  
 LiteOpKey.Code 21  
 LiteOpKey.Data 21  
 LiteOpKey.Gameld 21  
 LiteOpKey.GameProperties 22  
 LiteOpKey.Group 22  
 LiteOpKey.Properties 22  
 LiteOpKey.ReceiverGroup 22  
 LiteOpKey.Remove 22  
 LiteOpKey.RoomName 22  
 LiteOpKey.TargetActorNr 22  
 LitePeer 23  
 LitePeer class 23  
     about LitePeer class 23  
     LitePeer 27  
     OpChangeGroups 28  
     OpGetProperties 28  
     OpGetPropertiesOfActor 28, 29  
     OpGetPropertiesOfGame 29  
     OpJoin 30  
     OpLeave 31  
     OpRaiseEvent 31, 32, 33, 34, 35  
     OpSetPropertiesOfActor 35  
     OpSetPropertiesOfGame 36  
 LitePeer.LitePeer 27  
 LitePeer.OpChangeGroups 28  
 LitePeer.OpGetProperties 28  
 LitePeer.OpGetPropertiesOfActor 28, 29  
 LitePeer.OpGetPropertiesOfGame 29  
 LitePeer.OpJoin 30  
 LitePeer.OpLeave 31  
 LitePeer.OpRaiseEvent 31, 32, 33, 34, 35  
 LitePeer.OpSetPropertiesOfActor 35  
 LitePeer.OpSetPropertiesOfGame 36  
 LitePropertyTypes 86  
 LitePropertyTypes enumeration 86  
 Long enumeration member 85  
  
**M**  
 MasterClient enumeration member 88

---

MergeCache enumeration member 84

## N

Network Simulation 5

NetworkSimulationSet 36

NetworkSimulationSet class 36

about NetworkSimulationSet class 36

IncomingJitter 37

IncomingLag 37

IncomingLossPercentage 38

LostPackagesIn 38

LostPackagesOut 38

NetSimManualResetEvent 37

OutgoingJitter 38

OutgoingLag 38

OutgoingLossPercentage 38

ToString 37

NetworkSimulationSet.IncomingJitter 37

NetworkSimulationSet.IncomingLag 37

NetworkSimulationSet.IncomingLossPercentage 38

NetworkSimulationSet.LostPackagesIn 38

NetworkSimulationSet.LostPackagesOut 38

NetworkSimulationSet.NetSimManualResetEvent 37

NetworkSimulationSet.OutgoingJitter 38

NetworkSimulationSet.OutgoingLag 38

NetworkSimulationSet.OutgoingLossPercentage 38

NetworkSimulationSet.ToString 37

NoData enumeration member 87

None enumeration member 86

Null enumeration member 85

## O

ObjectArray enumeration member 85

OFF enumeration member 84

OperationRequest 38

OperationRequest class 38

about OperationRequest class 38

OperationCode 39

Parameters 39

OperationRequest enumeration member 85

OperationRequest.OperationCode 39

OperationRequest.Parameters 39

OperationResponse 39

OperationResponse class 39

about OperationResponse class 39

DebugMessage 40

OperationCode 40

Parameters 40

ReturnCode 40

this 40

ToString 41

ToStringFull 41

OperationResponse enumeration member 85

OperationResponse.DebugMessage 40

OperationResponse.OperationCode 40

OperationResponse.Parameters 40

OperationResponse.ReturnCode 40

OperationResponse.this 40

OperationResponse.ToString 41

OperationResponse.ToStringFull 41

Operations 2

Others enumeration member 88

Overview 1

## P

PeerBase 41

PeerBase class 41

about PeerBase class 41

ByteCountCurrentDispatch 42

ByteCountLastOperation 42

CryptoProvider 43

NetworkSimulationSettings 43

PeerID 44

SerializeMemStream 43

TrafficStatsEnabled 44

TrafficStatsEnabledTime 44

TrafficStatsGameLevel 43

TrafficStatsIncoming 43

TrafficStatsOutgoing 43

PeerBase.ByteCountCurrentDispatch 42

PeerBase.ByteCountLastOperation 42

PeerBase.ConnectionStateValue 42

PeerBase.ConnectionStateValue enumeration 42

PeerBase.CryptoProvider 43

PeerBase.NetworkSimulationSettings 43	PacketLossByCrc 58
PeerBase.PeerID 44	PeerID 58
PeerBase.SerializeMemStream 43	PeerState 58
PeerBase.TrafficStatsEnabled 44	PhotonPeer 48
PeerBase.TrafficStatsEnabledTime 44	QueuedIncomingCommands 59
PeerBase.TrafficStatsGameLevel 43	QueuedOutgoingCommands 59
PeerBase.TrafficStatsIncoming 43	RegisterType 52
PeerBase.TrafficStatsOutgoing 43	ResentReliableCommands 59
PeerStateValue 86	RoundTripTime 59
PeerStateValue enumeration 86	RoundTripTimeVariance 59
Photon Workflow 1	SendAcksOnly 52
PhotonDisconnectCause 87	SendOutgoingCommands 53
PhotonDisconnectCause enumeration 87	SentCountAllowance 59
PhotonPeer 44	ServerAddress 60
PhotonPeer class 44	ServerTimeInMilliseconds 60
about PhotonPeer class 44	Service 53
ByteCountCurrentDispatch 54	SocketImplementation 60
ByteCountLastOperation 54	StopThread 54
BytesIn 55	TcpConnectionPrefix 60
BytesOut 55	TimePingInterval 61
ChannelCount 55	TimestampOfLastSocketReceive 61
CommandBufferSize 55	TrafficStatsElapsedMs 61
Connect 49	TrafficStatsEnabled 61
CrcEnabled 55	TrafficStatsGameLevel 61
DebugOut 56	TrafficStatsIncoming 61
Disconnect 49	TrafficStatsOutgoing 61
DisconnectTimeout 56	TrafficStatsReset 54
DispatchIncomingCommands 49	UsedProtocol 62
EstablishEncryption 50	VitalStatsToString 54
FetchServerTimestamp 50	WarningSize 62
HttpUrlParameters 56	PhotonPeer.ByteCountCurrentDispatch 54
IsEncryptionAvailable 56	PhotonPeer.ByteCountLastOperation 54
IsSendingOnlyAcks 56	PhotonPeer.BytesIn 55
IsSimulationEnabled 56	PhotonPeer.BytesOut 55
LimitOfUnreliableCommands 57	PhotonPeer.ChannelCount 55
Listener 57	PhotonPeer.CommandBufferSize 55
LocalMsTimestampDelegate 57	PhotonPeer.Connect 49
LocalTimeInMilliseconds 57	PhotonPeer.CrcEnabled 55
MaximumTransferUnit 58	PhotonPeer.DebugOut 56
NetworkSimulationSettings 58	PhotonPeer.Disconnect 49
OpCustom 50, 51, 52	PhotonPeer.DisconnectTimeout 56
OutgoingStreamBufferSize 58	PhotonPeer.DispatchIncomingCommands 49



PhotonPeer.EstablishEncryption 50  
 PhotonPeer.FetchServerTimestamp 50  
 PhotonPeer.HttpUrlParameters 56  
 PhotonPeer.IsEncryptionAvailable 56  
 PhotonPeer.IsSendingOnlyAcks 56  
 PhotonPeer.IsSimulationEnabled 56  
 PhotonPeer.LimitOfUnreliableCommands 57  
 PhotonPeer.Listener 57  
 PhotonPeer.LocalMsTimestampDelegate 57  
 PhotonPeer.LocalTimeInMilliseconds 57  
 PhotonPeer.MaximumTransferUnit 58  
 PhotonPeer.NetworkSimulationSettings 58  
 PhotonPeer.OpCustom 50, 51, 52  
 PhotonPeer.OutgoingStreamBufferSize 58  
 PhotonPeer.PacketLossByCrc 58  
 PhotonPeer.PeerID 58  
 PhotonPeer.PeerState 58  
 PhotonPeer.PhotonPeer 48  
 PhotonPeer.QueuedIncomingCommands 59  
 PhotonPeer.QueuedOutgoingCommands 59  
 PhotonPeer.RegisterType 52  
 PhotonPeer.ResentReliableCommands 59  
 PhotonPeer.RoundTripTime 59  
 PhotonPeer.RoundTripTimeVariance 59  
 PhotonPeer.SendAcksOnly 52  
 PhotonPeer.SendOutgoingCommands 53  
 PhotonPeer.SentCountAllowance 59  
 PhotonPeer.ServerAddress 60  
 PhotonPeer.ServerTimeInMilliseconds 60  
 PhotonPeer.Service 53  
 PhotonPeer.SocketImplementation 60  
 PhotonPeer.StopThread 54  
 PhotonPeer.TcpConnectionPrefix 60  
 PhotonPeer.TimePingInterval 61  
 PhotonPeer.TimestampOfLastSocketReceive 61  
 PhotonPeer.TrafficStatsElapsedMs 61  
 PhotonPeer.TrafficStatsEnabled 61  
 PhotonPeer.TrafficStatsGameLevel 61  
 PhotonPeer.TrafficStatsIncoming 61  
 PhotonPeer.TrafficStatsOutgoing 61  
 PhotonPeer.TrafficStatsReset 54  
 PhotonPeer.UsedProtocol 62

PhotonPeer.VitalStatsToString 54  
 PhotonPeer.WarningSize 62  
 PhotonSocketError 87  
 PhotonSocketError enumeration 87  
 PhotonSocketState 87  
 PhotonSocketState enumeration 87  
 Properties on Photon 7  
 Protocol 62  
 Protocol class 62  
     about Protocol class 62  
     Deserialize 62, 63  
     Serialize 63, 64  
 Protocol.Deserialize 62, 63  
 Protocol.Serialize 63, 64

## Q

QueueIncomingReliableWarning enumeration member 88  
 QueueIncomingUnreliableWarning enumeration member 88  
 QueueOutgoingAcksWarning enumeration member 88  
 QueueOutgoingReliableWarning enumeration member 88  
 QueueOutgoingUnreliableWarning enumeration member 88  
 QueueSentWarning enumeration member 88

## R

ReadException enumeration member 87  
 ReceiverGroup 88  
 ReceiverGroup enumeration 88  
 RemoveCache enumeration member 84  
 RemoveFromRoomCache enumeration member 84  
 RemoveFromRoomCacheForActorsLeft enumeration member 84  
 ReplaceCache enumeration member 84  
 RHttp enumeration member 84

## S

SecurityExceptionOnConnect enumeration member 87, 88  
 SendError enumeration member 88  
 Serializable Datatypes 6  
 SerializeMethod 90  
 SerializeMethod type 90  
 Short enumeration member 85  
 Skipped enumeration member 87

SocketUdpNativeDynamic 64  
 SocketUdpNativeDynamic class 64  
     about SocketUdpNativeDynamic class 64  
     Connect 66  
     Disconnect 66  
     Receive 66  
     ReceiveLoop 66  
     Send 66  
     SocketUdpNativeDynamic 65  
 SocketUdpNativeDynamic.Connect 66  
 SocketUdpNativeDynamic.Disconnect 66  
 SocketUdpNativeDynamic.Receive 66  
 SocketUdpNativeDynamic.ReceiveLoop 66  
 SocketUdpNativeDynamic.Send 66  
 SocketUdpNativeDynamic.SocketUdpNativeDynamic 65  
 SocketUdpNativeStatic 66  
 SocketUdpNativeStatic class 66  
     about SocketUdpNativeStatic class 66  
     Connect 68  
     Disconnect 68  
     Receive 68  
     ReceiveLoop 68  
     Send 68  
     SocketUdpNativeStatic 67  
 SocketUdpNativeStatic.Connect 68  
 SocketUdpNativeStatic.Disconnect 68  
 SocketUdpNativeStatic.Receive 68  
 SocketUdpNativeStatic.ReceiveLoop 68  
 SocketUdpNativeStatic.Send 68  
 SocketUdpNativeStatic.SocketUdpNativeStatic 67  
 StatusCode 88  
 StatusCode enumeration 88  
 String enumeration member 85  
 StringArray enumeration member 85  
 Structs, Records, Enums 83  
 Success enumeration member 87  
 SupportClass 68  
 SupportClass class 68  
     about SupportClass class 68  
     ByteArrayToString 70  
     CalculateCrc 70  
     CallInBackground 70

    DictionaryToString 70, 71  
     GetMethods 71  
     GetTickCount 71  
     HashtableToString 71  
     IntegerMillisecondsDelegate 73  
     NumberToByteArray 72  
     WriteStackTrace 72  
 SupportClass.ByteArrayToString 70  
 SupportClass.CalculateCrc 70  
 SupportClass.CallInBackground 70  
 SupportClass.DictionaryToString 70, 71  
 SupportClass.GetMethods 71  
 SupportClass.GetTickCount 71  
 SupportClass.HashtableToString 71  
 SupportClass.IntegerMillisecondsDelegate 73  
 SupportClass.NumberToByteArray 72  
 SupportClass.ThreadSafeRandom 69  
 SupportClass.ThreadSafeRandom class 69  
     about SupportClass.ThreadSafeRandom class 69  
     Next 69  
 SupportClass.ThreadSafeRandom.Next 69  
 SupportClass.WriteStackTrace 72

## T

Tcp enumeration member 84  
 TcpRouterResponseEndpointUnknown enumeration member 88  
 TcpRouterResponseNodeIdUnknown enumeration member 88  
 TcpRouterResponseNodeNotReady enumeration member 88  
 TcpRouterResponseOk enumeration member 88  
 The Photon Server 7  
 TimeoutDisconnect enumeration member 88  
 TrafficStats 73  
 TrafficStats class 73  
     about TrafficStats class 73  
     ControlCommandBytes 74  
     ControlCommandCount 74  
     FragmentCommandBytes 74  
     FragmentCommandCount 74  
     PackageHeaderSize 74  
     ReliableCommandBytes 75  
     ReliableCommandCount 75

TimestampOfLastAck 75	ResetMaximumCounters 77
TimestampOfLastReliableCommand 75	ResultByteCount 79
ToString 74	ResultCount 80
TotalCommandBytes 75	SendOutgoingCommandsCalls 80
TotalCommandCount 75	ToString 77
TotalCommandsInPackets 75	ToStringVitalStats 78
TotalPacketBytes 76	TotalByteCount 80
TotalPacketCount 76	TotalIncomingByteCount 80
UnreliableCommandBytes 76	TotalIncomingMessageCount 80
UnreliableCommandCount 76	TotalMessageCount 80
TrafficStats.ControlCommandBytes 74	TotalOutgoingByteCount 80
TrafficStats.ControlCommandCount 74	TotalOutgoingMessageCount 80
TrafficStats.FragmentCommandBytes 74	TrafficStatsGameLevel.DispatchCalls 78
TrafficStats.FragmentCommandCount 74	TrafficStatsGameLevel.DispatchIncomingCommandsCalls 78
TrafficStats.PackageHeaderSize 74	TrafficStatsGameLevel.EventByteCount 78
TrafficStats.ReliableCommandBytes 75	TrafficStatsGameLevel.EventCount 78
TrafficStats.ReliableCommandCount 75	TrafficStatsGameLevel.LongestDeltaBetweenDispatching 78
TrafficStats.TimestampOfLastAck 75	TrafficStatsGameLevel.LongestDeltaBetweenSending 78
TrafficStats.TimestampOfLastReliableCommand 75	TrafficStatsGameLevel.LongestEventCallback 79
TrafficStats.ToString 74	TrafficStatsGameLevel.LongestEventCallbackCode 79
TrafficStats.TotalCommandBytes 75	TrafficStatsGameLevel.LongestOpResponseCallback 79
TrafficStats.TotalCommandCount 75	TrafficStatsGameLevel.LongestOpResponseCallbackOpCode 79
TrafficStats.TotalCommandsInPackets 75	TrafficStatsGameLevel.OperationByteCount 79
TrafficStats.TotalPacketBytes 76	TrafficStatsGameLevel.OperationCount 79
TrafficStats.TotalPacketCount 76	TrafficStatsGameLevel.ResetMaximumCounters 77
TrafficStats.UnreliableCommandBytes 76	TrafficStatsGameLevel.ResultByteCount 79
TrafficStats.UnreliableCommandCount 76	TrafficStatsGameLevel.ResultCount 80
TrafficStatsGameLevel 76	TrafficStatsGameLevel.SendOutgoingCommandsCalls 80
TrafficStatsGameLevel class 76	TrafficStatsGameLevel.ToString 77
about TrafficStatsGameLevel class 76	TrafficStatsGameLevel.ToStringVitalStats 78
DispatchCalls 78	TrafficStatsGameLevel.TotalByteCount 80
DispatchIncomingCommandsCalls 78	TrafficStatsGameLevel.TotalIncomingByteCount 80
EventByteCount 78	TrafficStatsGameLevel.TotalIncomingMessageCount 80
EventCount 78	TrafficStatsGameLevel.TotalMessageCount 80
LongestDeltaBetweenDispatching 78	TrafficStatsGameLevel.TotalOutgoingByteCount 80
LongestDeltaBetweenSending 78	TrafficStatsGameLevel.TotalOutgoingMessageCount 80
LongestEventCallback 79	Types 89
LongestEventCallbackCode 79	
LongestOpResponseCallback 79	
LongestOpResponseCallbackOpCode 79	
OperationByteCount 79	
OperationCount 79	
	<b>U</b>
	Udp enumeration member 84
	Unknown enumeration member 85
	Using TCP 4

## W

WARNING enumeration member 84

WriteException enumeration member 87

## Z

Zombie enumeration member 42