



Photon C++ Client API 4.1.12.2

Overview >

Common-cpp

Table of data types

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Common-cpp

Table of data types

Photon defines a common set of serializable data types across all supported platforms, as listed in the table below.

Client (C++)	Server (C#)
nByte	Byte / byte
short	Int16 / short
int	Int32 / int
int64	Int64 / long
bool	Boolean / bool
float	Single / float
double	Double / double
JString	String / string
Hashtable	Hashtable
Dictionary	Dictionary

We also support arrays for **all** the above types:

Client (C++)	Server (C#)
type*	type[]

Multidimensional arrays are supported (however on the server side they will be interpreted as jagged array with all subarrays in the same dimension having the same size). Jagged arrays are not supported.

Example: int* - one dimension int** - two dimensions

Moreover we support object-arrays, which means arrays of elements of

different types, as long as the element-types themselves are supported. For example the first element of the array can be an int, the second one a string.

Client (C++)	Server (C#)
Object*	Object[]

Object is not supported for non-array data.

Finally we support custom types.

This means, if your need to send some custom data, which can not be represented easily by the Photon-builtin data-types, like for example a game specific container class, then you can simply implement Photon's custom type interface for it and this way supply Photon with the needed abilities to handle your type and then you can just send and receive your custom type with Photon.

All custom types are automatically supported as (multi-dimensional) arrays, too.

Please refer to the API doc for class [CustomType](#) for details.

All of the above types are supported as values in [Hashtable](#) and [Dictionary](#) instances.

As keys both these containers accept the following types:

nByte
short
int
int64
float
double
JString

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

How to set up your application Workflow

Operations

Events

Properties

Sending and receiving data

The Photon Server

Basics

Fragmentation and Channels

Using TCP

Troubleshooting

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

How to set up your application

The following chapter deals with all the settings that need to be done to include the Photon Client Library files in your project.

[Windows iOS](#)

[Marmalade](#)

[OS X](#)

[Android NDK](#)

[Blackberry NDK](#)

[Linux](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Windows

For Windows Photon supports Visual Studio as IDE.

1. In *C/C++ -> General -> Additional Include Directories*, add the parent-folder(s) of the following paths:

(...)/**Photon-cpp/inc**

(...)/**Common-cpp/inc**

2. At *Linker -> Input -> Additional Dependencies* add either **Common-cpp_release_windows.lib** and **Photon-cpp_release_windows.lib** or **Common-cpp_debug_windows.lib** and **Photon-cpp_debug_windows.lib**

3. Add the folders **Photon-cpp/lib** and **Common-cpp/lib** to *Linker -> General -> Additional Library Directories*

4. Add the following `#include` directive to your source-code:
`#include "Photon-cpp/inc/PhotonPeer.h"`



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

How to set up your application

iOS

For iPhone Photon supports Xcode on Mac OS X as IDE.

1. Add the parent-folder(s) of the folders **Photon-cpp/inc** and **Common-cpp/inc** to "Header Search Paths" in category "Search Paths" in your applications Project Settings

2. In your Project Settings at category "Linking" add to "Other Linker Flags" the following both entries:

-ICommon-cpp_\$(CONFIGURATION)_\$(PLATFORM_NAME)
-IPhoton-cpp_\$(CONFIGURATION)_\$(PLATFORM_NAME)

3. Add the folders **Photon-cpp/lib** and **Common-cpp/lib** to "Library Search Paths" in category "Search Paths" in your applications Project Settings

4. Add the following #include directive to your source-code:
#include "Photon-cpp/inc/PhotonPeer.h"

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

How to set up your application

Marmalade

For Marmalade Photon supports Visual Studio and Xcode as IDEs and Windows and OS X as development platforms.

1. Add the entries **Photon-cpp** and **Common-cpp** to category **subprojects** in your project's .mkb-file
2. Add the entries **"../Common-cpp,Common-cpp"** and **"../Photon-cpp,Photon-cpp"** (including the "") to category **librarys** in your projects' .mkb-file
3. Add the entries **../Common-cpp** and **../Photon-cpp** to category **librarypaths** in your projects' .mkb-file
4. Add the entry **module_path="../"** to category **options** in you project's .mkb-file
5. Add the following #include directive to your source-code:
#include "Photon-cpp/inc/PhotonPeer.h"

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

How to set up your application

OS X

For OS X Photon supports Xcode as IDE.

1. Add the parent-folder(s) of the folders **Photon-cpp/inc** and **Common-cpp/inc** to "Header Search Paths" in category "Search Paths" in your applications Project Settings

2. In your Project Settings at category "Linking" add to "Other Linker Flags" the following both entries:

```
-ICommon-cpp_$(CONFIGURATION)_$(PLATFORM_NAME)  
-IPhoton-cpp_$(CONFIGURATION)_$(PLATFORM_NAME)
```

3. Add the folders **Photon-cpp/lib** and **Common-cpp/lib** to "Library Search Paths" in category "Search Paths" in your applications Project Settings

4. Add the following #include directive to your source-code:
#include "Photon-cpp/inc/PhotonPeer.h"

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Android NDK

For Android NDK Photon supports Visual Studio with WinGDB plugin on Windows as IDE, but you can also use makefiles, which will work on Windows and OS X. Linux is currently not supported.

1. Add **common-cpp-static-prebuilt** and **photon-cpp-static-prebuilt** to "APP_CPPFLAGS" in your applications Android.mk file.

2. In your Projects Android.mk file add the following lines:

```
$(call import-add-path-optional, $(shell pwd)/../../../../Photon-cpp/src/android)  
$(call import-add-path-optional, $(shell pwd)/../../../../Photon-cpp)  
$(call import-module,photon-cpp-prebuilt)
```

3. Add **-frtti** to "LOCAL_STATIC_LIBRARIES" in your applications Application.mk file.

4. Set "APP_STL" in your applications Application.mk file to **stlport_static**, **stlport_shared**, **gnustl_static** or **gnustl_shared**.

5. Add the following #include directive to your source-code:

```
#include "Photon-cpp/inc/PhotonPeer.h"
```



Blackberry NDK

Photon supports Blackberry NDK version 10 and newer. Photon's IDE of choice for Blackberry currently is QNX Momentics.

1. Right-click on your project, go to "Configure" -> "Add Library", choose "Standard BlackBerry Platform Library" and add the following libraries to your project:

- a) **Common Services -> Math - libm**
- b) **Localization -> Iconv - libiconv**
- c) **Networking -> Socket - libsocket**

2. Right-click on your project, go to "Configure" -> "Add Library", choose "External Library" and add the Common C++ library to your project:

Device Library: **../../Common-cpp/libcommon-cpp-debug_blackberry.a**
or **../../Common-cpp/libcommon-cpp-release_blackberry.a**

Simulator Library: **../../Common-cpp/libcommon-cpp-debug_blackberry_simulator.a** or **../../Common-cpp/libcommon-cpp-release_blackberry_simulator.a**

Include folders: **../..**

3. Right-click on your project, go to "Configure" -> "Add Library", choose "External Library" and add the Photon C++ library to your project:

Device Library: **../../Photon-cpp/libphoton-cpp-debug_blackberry.a** or **../../Photon-cpp/libphoton-cpp-release_blackberry.a**

Simulator Library: **../../Photon-cpp/libphoton-cpp-debug_blackberry_simulator.a** or **../../Photon-cpp/libphoton-cpp-release_blackberry_simulator.a**

Include folders: **../..**

4. Right-click on your project, go to "Properties" -> "C/C++ General" -> "Paths and Symbols" -> "Libraries" and make sure, that the Photon lib is listed above the Common lib and the Common lib above all standard

blackberry platform libs

5. Add the following `#include` directive to your source-code:
`#include "Photon-cpp/inc/PhotonPeer.h"`

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[Overview](#)[Photon-cpp](#)[How to set up your application](#)

Linux

Use IDE of your choice or makefiles.

1. In Code::Blocks IDE right-click on your project, go to "Build Options" and go to "Search directories".
2. In the "Compiler" tab add paths to the parent-folder(s) of the **Common-cpp/inc** and **Photon-cpp/inc** directories.
3. In the "Linker" tab add paths to **Common-cpp** and **Photon-cpp** directories.
4. For each target of your project go to its "Linker settings" tab left to "Search directories". Add **PhotonDebug64** and **CommonDebug64** libraries for the debug or **PhotonRelease64** and **CommonRelease64** for the release configuration for 64 bit builds, or replace the "64" by "32" for 32bit builds. The order of the libraries is important for a successful build. The prefix 'lib' and the extension 'a' are added automatically.
5. In the projects "Linker settings" tab add **-pthread** to "Other linker options".
6. Add the following #include directive to your source-code:
#include "Photon-cpp/inc/PhotonPeer.h"

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

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 its own number.

A simplified workflow looks like this:

- create a LitePeer instance
- from now on: regularly call **service()** to get events and operation responses and to send operations (e.g. ten times a second)
- call **connect()** to connect to the server
- wait until the library calls **onStatusChanged()**
- the return code should equal `StatusCode::CONNECT`
- call **opJoin()** to get into a room
- wait until the library calls **onOperationResponse()** with operation code `OPC_RT_JOIN`
- send data in the game by calling **opRaiseEvent()**
- receive events in **onEvent()**
- when you are done: call **opLeave()** to quit/leave the game room
- wait for a response to "leave" in **onOperationResponse()** with operation code: `OPC_RT_LEAVE`
- disconnect with **disconnect()**
- wait for status code `StatusCode::DISCONNECT` in **onStatusChanged()**

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

- Low Level: **service()**, **connect()**, **disconnect()** and **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 is 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 functions that are operations (RPC calls) are prefixed with "Op" to tell them apart from anything else. Other server-side applications (like for example 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()`.

Aside from operations, there is a separate communication layer to make UDP reliable. Everything that goes from client to server (and the other way round) is put into "commands" and some commands establish and keep the connection between client and server (without carrying additional data).

Callbacks

PhotonPeer uses the virtual functions of the class `ExitGames::Photon::PhotonListener` to do callbacks. Each function is called in separate cases:

- `onStatusChanged()` is for peer state changes (connect, disconnect, errors)
- `onOperationResponse()` is for operation responses (join, leave, raiseEvent and custom operations, etc.)
- `onEvent()` gets called for events coming in
- `debugReturn()` is called to pass debug output to you (not used by release builds)

The calls to `onStatusChanged()` are of special interest, as they denote connection status changes and errors.

Getters/Setters

The following getter- and setter-functions in PhotonPeer are of special interest:

- **setTimePingInterval()** sets the time between ping operations
- **getRoundTripTime()** returns the ping between the Photon client and the server
- **getRoundTripTimeVariance()** shows the jitter (variability of the roundtrip time)
- **getServerTime()** is the continuously approximated server's time in milliseconds

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

Operations

Operation is our term for remote procedure calls (RPC) on Photon. This in turn can be described as functions that are implemented on the server-side and called by clients. As any function, 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 functions 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 functions: **opJoin()** and **opRaiseEvent()**. They can be used right away with the default implementation of Photon and the Lite Application.

Custom Operations

Photon is about being extendable with features that are specific to your game. You could persist states in a database or double check information from the clients on the server by implementing functions. If your new functions can be called from the client-side, we call them 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 functions that can be called from the client-side. They can have any number of parameters and any name. As you will be calling operations a lot, we avoid using strings and instead assign byte-codes for every operation and each parameter.

This 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 instances of **OperationRequest** and **OperationResponse** to aggregate the opCode and all parameters. Use **opCustom()** to send your Hashtable and call any operation.

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

Operation Codes: byte versus short

Currently, the server-side uses the short-type to define opCodes and parameter keys while the client-side uses bytes only. This is a remainder of Neutron, essentially, where we implemented more values of opCodes. But using short for each opCode and parameter is a lot of overhead in a realtime environment, so we decided to revert this in the protocol and just send bytes. This simply saves lots of bandwidth.

© [Exit Games](#)®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

Overview

Photon-cpp

Events

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

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

Events are instances of **EventData** with arbitrary content.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

Properties

Properties can be set by these functions:

- **opSetPropertiesOfActor()** sets a player's properties
- **opSetPropertiesOfGame()** sets a room's properties
- **opJoin()** also allows you to set properties on room creation

They can be fetched with these functions:

- **opGetProperties()**
- **opGetPropertiesOfActor()**
- **opGetPropertiesOfGame()**

Broadcast Events

Any change that uses the broadcast option will trigger a property update event `EV_RT_SETPROPERTIES`. This event carries the properties as value of key `EV_RT_KEY_PROPERTIES`.

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

Notes

You can delete properties by sending them with *NULL* as value. This also means, that you can't use *NULL* as a normal value for them. Lite currently 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.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview > Photon-cpp >

Sending and receiving data

How to send data to other players

In Photon you exchange data with other players by sending and receiving "Operations" and "Events". Please refer to [Operations](#) for more information about this concept.

The cross-platform communication ability of Photon implies the need for common data structures across all the different client versions. Please refer to [Datatypes](#) for a table of supported types and their equivalents on the server side.

When your Hashtable is complete, use [opRaiseEvent\(\)](#) (or [opCustom\(\)](#), if you implemented custom types of operations on the server) to initiate the transmission.

Receiving Data

Photon will interact with your application by calling the callback functions you implemented, thereby passing data structures as arguments. All these data structures belong to Photon, which means that Photon is responsible for deleting them. This will happen as soon as the callback function has returned.

So your application is responsible A) for ***extracting and copying any data needed from the arguments within the callback function***, and of course

B) for later freeing up the memory needed for those copies, as usual.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

Overview

Photon-cpp

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 UDP and TCP (TCP can be used by clients on platforms, that are not fully supporting UDP, and as for server-setups, that do not support UDP (like some cloud services) and in case of some paranoid firewall settings - use UDP (which can be sent reliable with Photon!) whenever you can and only use TCP as a fallback, as we can't guarantee feature completeness for TCP.

The Photon Server SDK includes a pre-built version that can be run out of the box. It also allows you to extend the server-side easily.

Get the Photon Server SDK at:

<https://www.photonengine.com/en/OnPremise/Download>

The Lite Application

The Lite Application is the default implementation for room-based games on Photon and (hopefully) a flexible base for your own, more game-specific, extensions. It offers rooms, joining and leaving them, sending events to the other players in a room and handles properties.

It basically does everything you came to expect of Photon.

So why is this done in a separate project? Because this way we can separate the low level C++ server core and the high level C# server API, which comes in multiple so called Applications, that can be used for different needs.

On the client-side LitePeer is the counterpart for the server-side Lite Application.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

Overview

Photon-cpp

Basics

Creating a PhotonPeer instance and connecting

When a PhotonPeer instance is created, the PhotonPeer is ready to connect to a Photon Server. To do that, call the function **connect()**

After initializing the connection, the application should wait for the **onStatusChanged()** callback function. If its returnCode is `StatusCode::CONNECT`, the connection has been established.

Joining a game

As soon as the client application is connected to Photon, use the function **opJoin()** to join or create a game. If there is no game with the given identifier, a new game will be created. If the call succeeds, the **onOperationResponse()** callback will be called with operation code OPC_RT_JOIN, and also an event will be raised, resulting in a call to **onEvent()** callback with event code EV_RT_JOIN .

Raising custom events in game

In addition to the events raised by Photon you can also define and raise events needed for your game. E.g. you could define a event named "EV_SHOOT" to broadcast the information that the local actor has just fired a weapon at the position stored in the variables `pMe->fireX`, `pMe->fireY`. First pick and define an operation code for your "shoot" event. Make sure it won't collide with the Event codes #defined in `PhotonConstants.h`

To keep your code more readable and maintainable, you should also define key codes for your corresponding Hashtable entries, as shown below:

```
const nByte EV_SHOOT = 101;
const nByte KEY_FIRE_X = 1;
const nByte KEY_FIRE_Y = 2;
```

In the game we can now create an Hashtable for the shoot event and include the fire-coordinates as Key/Value pairs.

```
HashTable event;

event.put(KEY_FIE_X, pMe->fireX);
event.put(KEY_FIE_Y, pMe->fireY);

mPeer.opRaiseEvent(TRUE, event, EV_SHOOT);
```

As soon as Photon has delivered this operation, the **onEvent()** callback will be called at all the other players inside the same room, with event code beeing `EV_SHOOT`. Use a switch case on the event code to handle the different events accordingly.

Leaving a room

Use the `opLeave()` function to leave the currently joined room.

It sends an operation to the server and other players will receive the event `EV_RT_LEAVE`. When the operation is completed successfully, the `ExitGames::Photon::PhotonListener::onOperationResponse()` callback will be called at the local peer with the opCode `OPC_RT_LEAVE`.

Disconnecting from the server

Disconnecting should be done using `disconnect()`.

When disconnecting is finished, the `onStatusChanged()` callback will be called and the status code should be `StatusCode::DISCONNECT`.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

Fragmentation and Channels

Fragmentation

Bigger chunks of data (more than about 1kB) are not fitting into a single packet, so they are fragmented and reassembled automatically. Depending on the data size, this takes up multiple packages.

Be aware that this might stall other commands. Call `service()` or `sendOutgoingCommands()` more often than absolutely necessary. You can also check the debug output for **"WARNING! There are x outgoing messages waiting in the local sendQueue !"**, which is triggered, if a sendqueue contains an unusual big amount of elements and means, that you probably do not call `service()` or `sendOutgoingCommands()` often enough to let Photon send all the packets out, which you are creating by triggering operations.

Sequencing

The sequencing of the protocol makes sure that any receiving client will dispatch your actions in the order, in which you have 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 in the same channel and do not get dispatched before that reliable data has been dispatched first. This can be useful, if the actions 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. On the receiving end, this would maybe show as a small jump. If a package with a chat message is lost, it would be resent and would introduce lag, even to all movement updates, created after that chat-message. In this case, the data is unrelated and should be put into different channels, to avoid that a needed resent of a chat message introduces lag into the movement updates.

Channels

Photon is supporting "channels". This allows you to separate information into multiple channels, each being sequenced independently. This means, that operations and events of one channel will not be stalled because events of another channel are not available yet.

By default a PhotonPeer has an amount of `getChannelCountUserChannels()` user channels and channel zero is the default channel, which will be used, when not explicitly specifying a channel. Operations join and leave are always sent in channel zero. There is a "system" channel 255 used internally for connect and disconnect messages. This channel is ignored for the user channel count.

Channels are prioritized: Data, to be send on the lowest channel number is put into an UDP package first. Data, which will be sent through a channel with a higher number might be sent later when an UDP package is already full.

Example: The chat messages could be sent in channel one, while movement is sent in channel zero. They are not related to the movement 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 immediately (in case packages get filled up).

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview

Photon-cpp

Using TCP

A PhotonPeer could be instanced with TCP as underlying protocol if necessary. This is not best practice but some client platforms and some clouds don't support UDP sockets and some end users' firewall or router settings may not allow it. However UDP is the preferable protocol for Photon, whenever you have the choice.

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 according to your specifications.

Example: A client, which has been initialized to use TCP, might send unreliable movement updates in channel 1. These will be sent via TCP, which makes it reliable. Photon however also has connections with UDP clients. It will use your reliable / unreliable settings to forward your movement updates accordingly.



Troubleshooting

This section contains suggestions for common problems developers using Photon might come across. As always: if the solution at hand is not fitting your needs, please contact us: developer@photonengine.com.

I get a message "WARNING! There are x outgoing messages waiting in the local sendQueue !" What is the problem ?

This message means that you are generating more Photon operations/events than you are sending. Photon can only send reliable operations one after another, and will wait for the response from the Photon server before the next operation will be sent. Of course, Photon only can send operations at all, if you call `service()`. If you do not call it often enough, it will not be able to send all the operations, which you are generating.

Solutions:

1. Make sure to call `service()` in a sufficiently high frequency (like ten times a second).
2. If you call call `service()` frequently enough, the problem lies in the underlying Network not being able to transmit the information quickly enough. Especially mobile networks are not able to transfer something like 50 or even more operations per second. The only solution for this is to create fewer operations over the same period of time. Try to sum up your ingame data and send it in bigger time intervals.

Note: There is no certain limit for the size of a queue. Photon will fail as soon as there is no more memory available for new messages.

Troubleshooting Windows

Including multiple versions of WinSock It is common for developers using multiple libraries to have a conflict around multiple versions of WinSock. For instance, a developer may use a game engine that uses WinSock and Photon which uses WinSock2. By adding **`_WINSOCK_`** to the preprocessor definitions the conflict is resolved, but the system will throw a warning which can be ignored.

You can also try re-arranging the order in which you're including the header files. You can see a good example of this in someone's application using a library called Allegro. Including the files in this order solved their compilation problems:

```
\#include <allegro.h>
#define _WINSOCKAPI_
\#include <winalleg.h>
\#include <winsock2.h>
```

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview >

LoadBalancing-cpp

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Overview >

Chat-cpp

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

Namespace List

Here is a list of all documented namespaces with brief descriptions:

[detail level 1 2 3]

▼  ExitGames	
▼  Chat	
 ClientState	
 CustomAuthenticationType	
 DisconnectCause	
 ErrorCode	
 UserStatus	
▼  Common	
 DebugLevel	
 MemoryManagement	
 TypeCode	
▼  Lite	
 EventCache	
 EventCode	
 EventKey	
 OperationCode	
 ParameterCode	
 ReceiverGroup	
▼  LoadBalancing	
 CustomAuthenticationType	
 DirectMode	
 DisconnectCause	

Ⓝ ErrorCode	
Ⓝ LobbyType	
Ⓝ MatchmakingMode	
Ⓝ PeerStates	
▼ Ⓝ Photon	
Ⓝ ConnectionProtocol	
Ⓝ ErrorCode	
Ⓝ NetworkPort	
Ⓝ PeerState	
Ⓝ Punchthrough	
Ⓝ StatusCode	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

ExitGames Namespace Reference

Namespaces

Chat

Common

Lite

LoadBalancing

Photon

Detailed Description

ExitGames

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Chat >

[Namespaces](#) | [Classes](#) | [Variables](#)

ExitGames::Chat Namespace Reference

Namespaces

ClientState

CustomAuthenticationType

DisconnectCause

ErrorCode

UserStatus

Classes

class **AuthenticationValues**

class **Channel**

class **Client**

class **Listener**

class **Peer**

Variables

```
const EG_CHAR *const REGION
```

Detailed Description

Chat

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Chat > ClientState >

Variables

ExitGames::Chat::ClientState Namespace Reference

Variables

static const int **Uninitialized**
Peer is created but not used yet.

static const int **ConnectingToNameServer**
Connecting to Name Server (includes connect
authenticate and joining the lobby)

static const int **ConnectedToNameServer**
Connected to Name Server.

static const int **Authenticating**
Authenticating.

static const int **Authenticated**
Authenticated.

static const int **DisconnectingFromNameServer**
Transition from Name to **Chat** Server.

static const int **ConnectingToFrontEnd**
Transition to **Chat** Server.

static const int **ConnectedToFrontEnd**
Connected to **Chat** Server. Subscribe to channels and
chat here.

static const int **Disconnecting**
The client disconnects (from any server).

static const int **Disconnected**
The client is no longer connected (to any server).
Connect to Name Server to go on.

Detailed Description

Possible states for a **Client**.

ClientState

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

 Photon C++
Client API 4.1.12.2

ExitGames > Chat > CustomAuthenticationType >

Variables

ExitGames::Chat::CustomAuthenticationType Namespace Reference

Variables

static const nByte **CUSTOM**
Use a custom authentication service.

static const nByte **STEAM**
Authenticates users by their Steam Account. Set auth values accordingly!

static const nByte **FACEBOOK**
Authenticates users by their Facebook Account. Set auth values accordingly!

static const nByte **OCULUS**
Authenticates users by their Oculus Account. Set auth values accordingly!

static const nByte **PLAYSTATION**
Authenticates users by their PSN Account. Set auth values accordingly!

static const nByte **XBOX**
Authenticates users by their XBox Network Account. Set auth values accordingly!

static const nByte **NONE**
Disables custom authentication.

Detailed Description

Options for optional "Custom Authentication" services used with **Photon**.
Used when the client sends an authentication request to the server.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Chat > DisconnectCause >

Variables

ExitGames::Chat::DisconnectCause Namespace Reference

Variables

static const int **NONE**
No error was tracked.

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
OnStatusChanged: The CCUs count of your **Photon** Server License is exhausted (temporarily).

static const int **EXCEPTION_ON_CONNECT**
OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

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

static const int **DISCONNECT_BY_SERVER_LOGIC**
OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).

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

static const int **EXCEPTION**
OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

static const int **INVALID_AUTHENTICATION**
OnOperationResponse: Authenticate in the **Photon** Cloud with invalid AppId. Update your subscription or contact Exit Games.

static const int **MAX_CCU_REACHED**

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

static const int **INVALID_REGION**

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

static const int **OPERATION_NOT_ALLOWED_IN_CURRENT_STATE**

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

static const int **CUSTOM_AUTHENTICATION_FAILED**

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

Detailed Description

Enumeration of causes for Disconnects (used in Chat.DisconnectedCause). Read the individual descriptions to find out what to do about this type of disconnect.

DisconnectCause

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Chat > ErrorCode >

Variables

ExitGames::Chat::ErrorCode Namespace Reference

Variables

static const int **OPERATION_DENIED**

static const int **OPERATION_INVALID**

static const int **INTERNAL_SERVER_ERROR**

static const int **OK**

static const int **INVALID_AUTHENTICATION**

static const int **GAME_ID_ALREADY_EXISTS**

static const int **GAME_FULL**

static const int **GAME_CLOSED**

static const int **ALREADY_MATCHED**

static const int **SERVER_FULL**

static const int **USER_BLOCKED**

static const int **NO_MATCH_FOUND**

static const int **GAME_DOES_NOT_EXIST**

static const int **MAX_CCU_REACHED**

static const int **INVALID_REGION**

static const int **CUSTOM_AUTHENTICATION_FAILED**

Detailed Description

CustomAuthenticationType

ErrorCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Chat > UserStatus >

Variables

ExitGames::Chat::UserStatus Namespace Reference

Variables

static const int **OFFLINE**
Offline.

static const int **INVISIBLE**
Be invisible to everyone. Sends no message.

static const int **ONLINE**
Online and available.

static const int **AWAY**
Online but not available.

static const int **DND**
Do not disturb.

static const int **LFG**
Looking For Game/Group. Could be used when you want to be invited or do matchmaking.

static const int **PLAYING**
Could be used when in a room, playing.

Detailed Description

Contains commonly used status values for SetOnlineStatus. You can define your own. While "online" (value 2 and up), the status message will be sent to anyone who has you on his friend list.

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

UserStatus

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Common >

[Namespaces](#) | [Classes](#) | [Functions](#)

ExitGames::Common Namespace Reference

Namespaces

DebugLevel

MemoryManagement

TypeCode

Classes

class **ANSIString**

class **Base**

class **BaseCharString**

class **BaseListener**

class **CustomType**

class **CustomTypeBase**

class **CustomTypeFactory**

class **DeSerializer**

class **Dictionary**

class **DictionaryBase**

class **EGTime**

class **Hashtable**

class **JString**

class **JVector**

class **KeyObject**

class **LogFormatOptions**

class **Logger**

class **Object**

class **Serializer**

class **ToString**

class **UTF8String**

class **ValueObject**

Detailed Description

Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Common > DebugLevel >

Variables

ExitGames::Common::DebugLevel Namespace Reference

Variables

static const int **OFF**
No debug out.

static const int **ERRORS**
Only error descriptions.

static const int **WARNINGS**
Warnings and errors.

static const int **INFO**
Information about internal workflows, warnings and errors.

static const int **ALL**
Most complete workflow description (but lots of debug output), info, warnings and errors.

Detailed Description

Amount of DebugReturn callbacks. Each debug level includes output for lower ones: OFF, ERRORS, WARNINGS, INFO, ALL.

DebugLevel

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames

Common

MemoryManagement

[Classes](#) | [Functions](#)

ExitGames::Common::MemoryManagement Namespace Reference

Classes

class **AllocatorInterface**

Functions

`__attribute__((weak)) AllocatorInterface *AllocatorInterface`

HighLevelMemoryManagement

The template functions in this section are an alternative for the C++ dynamic memory management operators `new`, `new[]`, `delete` and `delete[]`.

They are implemented in terms of enhancing the **Low Level Memory Management macros** and for this reason offer similar advantages over `new` and `co` like those macros offer over `malloc` and `co`.

However same as `new` and `co` they also construct and destruct the object that they allocate and deallocate.

```
void setMaxAllocSize (size_t maxAllocSize)
```

```
void setMaxSizeForAllocatorUsage (size_t maxSizeForAllocatorUs
```

```
void setAllocator  
    (ExitGames::Common::MemoryManagement::AllocatorInter  
    &allocator)
```

```
void setAllocatorToDefault (void)
```

```
template<typename Ftype >  
Ftype * allocate (void)
```

```
template<typename Ftype >  
Ftype * allocateArray (size_t count)
```

```
template<typename Ftype >  
Ftype * reallocateArray (Ftype *p, size_t count)
```

```
template<typename Ftype >  
void deallocate (const Ftype *p)
```

```
template<typename Ftype >  
void deallocateArray (const Ftype *p)
```


Detailed Description

MemoryManagement

Function Documentation

§ setMaxAllocSize()

void

```
ExitGames::Common::MemoryManagement::setMaxAllocSize ( size_t m
```

Sets the max size of memory that might get allocated ahead of time as a single memory request.

Requesting memory from the OS is an expensive operation. This is why memory manager might choose to request bigger amounts of memory at give out smaller chunks of them to the application code. This way it can r amount of necessary memory requests to the OS. Depending on the me requests that come in from the application code, a memory manager mig scale up its own requests to the OS.

You can set an upper limit for how much the currently active memory ma allowed to scale up through this function.

Example: Consider a pool-based memory manager that uses multiple me where each serves requests for memory of different sizes. There could b tiny memory requests, one for small requests, one for medium requests,

Now let's imagine that there is a pool that serves requests between 65 a in size each and for this purpose keeps a bunch of 128 byte blocks arou out to requesters. In the beginning it might just keep very few such block the memory manager does not know, how many blocks of this size an ap need to use in parallel. When an app requests lots of those blocks, the p scale accordingly and to not need to do a request to the OS too often, it i increase the size of it's own requests. i.e. at first it could have just 4 bloc then when it resizes, it would allocate memory for another 4 blocks, then then for another, 16, then 32, 64, 128, 256, 512, 1024 blocks more, and :

Now if you set an upper limit of 8192 bytes, then the pool would not incre of its requests to the OS beyond that limit. For that 128 byte blocks pool mean that it would request at max $8192/128==64$ blocks at once. So the pattern from above would change to 4, 8, 16, 32, 64, 64, 64, 64, and so c

Accordingly with the same 8192 bytes limit in place a pool that holds 102

blocks would not allocate memory for more than 8 such blocks at once.

Note

This does not set a limit to the overall memory that might get allocated to the memory that gets allocated as a direct result of a single memory request. The very next request might already lead to another allocation if the manager decides so (for example a pool based memory manager might handle differently sized requests from different pools that resize independently of each other).

Remarks

This function forwards the passed in value to the currently set allocator (`setAllocator()`) and does not store it itself. For this reason a call to `setAllocator()` only affects the settings of the currently set allocator and not those of another allocator, that might be set by `setAllocator()` at any point in time after this function got called.

It is the responsibility of the allocator to honor the the setting that the user applied through this function.

Parameters

maxAllocSize the max size for a single memory request to the OS

See also

`setAllocator()`, `AllocatorInterface`

§ setMaxSizeForAllocatorUsage()

void

ExitGames::Common::MemoryManagement::setMaxSizeForAllocatorUsage

Sets a limit up to which memory requests get forwarded to the set allocator. Requests above the limit will be forwarded directly to the OS instead.

Requesting memory from the OS is an expensive operation.

For frequent requests of small amounts of memory it is usually more efficient to use the set allocator instead, which requests bigger amounts of memory from the OS and returns those smaller blocks to the requester.

However this is effectively a trade of reduced execution time bought with usually a good deal for frequent small requests, but a bad deal for infrequent requests of large amounts of memory.

For this reason from a certain request size on requests get forwarded directly to the OS instead of the set allocator.

This function lets you set the upper limit up to which the set allocator is used.

Requests above the limit will be forwarded directly to the OS.

Remarks

The value that is set through this function affects all allocators, not just the set allocator.

Parameters

maxSizeForAllocatorUsage the max size for a memory request up to which the set allocator is used.

See also

[setAllocator\(\)](#), [AllocatorInterface](#)

§ setAllocator()

void

ExitGames::Common::MemoryManagement::setAllocator (**ExitGames::**

Sets the allocator that will be used by future memory requests to the pro

All dynamic memory allocation requests by the Photon Client libraries go through one of the **Low Level Memory Management macros**. The appl memory requests if its developer chooses so.

Each request for an amount of memory that does not exceed the limit se allocator. Photon provides a default general-purpose allocator that uses j applications.

However you can set your own allocator through this function and Photon afterwards.

Regarding potential reasons for writing your own custom allocator please [https://en.wikipedia.org/wiki/Allocator_\(C%2B%2B\)#Custom_allocators](https://en.wikipedia.org/wiki/Allocator_(C%2B%2B)#Custom_allocators).

Remarks

Photons memory management stores the address of the allocator th free memory to the same allocator that allocated that memory.

This means a) that you can set a different allocator as often as you l once set allocator available even when it is no longer set as the curr memory that once got requested from it, got returned to it and non o

If you want to already set an initial custom allocator before any glob: then you need to replace **AllocatorInterface::get()**.

Parameters

allocator an instance of a subclass of **AllocatorInterface**

See also

setMaxSizeForAllocatorUsage(), **AllocatorInterface**, **AllocatorInt**

§ setAllocatorToDefault()

```
void  
ExitGames::Common::MemoryManagement::setAllocatorToDefault ( void
```

Calls [setAllocator\(\)](#) with Photons default allocator as parameter.

See also

[setAllocator\(\)](#)

§ allocate()

```
Ftype* ExitGames::Common::MemoryManagement::allocate ( void )
```

This function allocates a new instance of the type, that has been specified as first template parameter, on dynamic memory and properly initializes it. For an instance of a class type this includes calling a constructor on the instance.

Instances, that have been allocated with **allocate()**, have to be deallocated with **deallocate()**, when they are no longer needed.

Up to 10 optional arguments can be passed to **allocate()** and **allocate()** will call a constructor with the matching number of parameters and matching parameter types. If the class of the object that is to be constructed, doesn't provide a constructor with a matching signature, if that constructor isn't publicly accessible or if it is ambiguous, which constructor to choose, then the call to **allocate()** will trigger an error from the compiler..

The allocation is implemented via a call to **EG_MALLOC()**.

§ allocateArray()

Ftype*

ExitGames::Common::MemoryManagement::allocateArray (size_t count

This function allocates an array of new instances of the type, that has been specified as first template parameter, on dynamic memory and properly initializes all of them. For arrays of class types this includes constructing each element via a constructor with matching parameter list

Instances, that have been allocated with **allocateArray()**, have to be deallocated with **deallocateArray()**, when they are no longer needed.

The passed element count is allowed to be 0. In that case this function still allocates storage to store the element count of 0 in, so the returned address still has to be deallocated later.

Up to 10 optional arguments can be passed to **allocateArray()** and **allocateArray()** will call a constructor with the matching number of parameters and matching parameter types. If the class of the elements that are to be constructed, doesn't provide a constructor with a matching signature, if that constructor isn't publicly accessible or if it is ambiguous, which constructor to choose, then the call to **allocateArray()** will trigger an error from the compiler.

The allocation is implemented via a call to **EG_MALLOC()**.

Parameters

count the amount of elements that the new array should have

§ `reallocateArray()`

```
Ftype*
ExitGames::Common::MemoryManagement::reallocateArray ( Ftype * p,
                                                         size_t  cc
                                                         )
```

This function resizes an array, that has previously been allocated with `allocateArray()`.

The function allocates a new array of the same type as the provided one but with the requested element count. Afterwards it copies all elements of the old array that fit into the new array into the new array by calling the constructor of the class of the elements.

If the new element count is lower than the old one, then the corresponding elements at the end of the old array don't get copied over to the new one but are just destructed.

If the new requested element count is higher than the old one, then the remaining uninitialized elements in the new array get constructed by choosing the constructor that matches the provided optional arguments to `reallocateArray()` best (no optional arguments means the default constructor gets called).

Finally the old array gets deallocated via `deallocateArray()` and the new array gets returned.

The returned address will most likely not match the passed one.

The passed address is allowed to be NULL. In that case this function behaves like `allocateArray()`.

The passed element count is allowed to be 0. In that case this function still allocates storage to store the element count of 0 in, so the returned address still has to be deallocated later.

If the passed address has not previously been returned by a call to

allocateArray() or **reallocateArray()** and also isn't NULL or if it has already been passed to **deallocateArray()**, then the behavior is undefined.

Up to 10 optional arguments can be passed to **reallocateArray()** and **reallocateArray()** will call a constructor with the matching number of parameters and matching parameter types on each element of the new array, which hasn't already been copy-constructed from the corresponding element in the old array. If the class of the elements that are to be constructed, doesn't provide a constructor with a matching signature or if it doesn't provide a copy constructor, if that constructor or copy constructor isn't publicly accessible or if it is ambiguous, which constructor to choose then the call to **reallocateArray()** will trigger an error from the compiler.

Parameters

- p** the address of the array, that is to be resized
- count** the new amount of elements that the array should have

§ deallocate()

```
void  
ExitGames::Common::MemoryManagement::deallocate ( const Ftype * p
```

Call this function to destruct and deallocate an instance, that has previously been allocated and constructed by a call to **allocate()**.

The passed address is allowed to be NULL. In that case the call doesn't have any effect.

If the passed address has not previously been returned by a call to **allocate()** and also isn't NULL, then the behavior is undefined.

Parameters

p the address of the instance, that should be deallocated

§ deallocateArray()

void

```
ExitGames::Common::MemoryManagement::deallocateArray ( const Fty
```

Call this function to destruct and deallocate an array, that has previously allocated and constructed by a call to [allocateArray\(\)](#).

This function will call their destructor on all elements of the array and the deallocate the memory of the array.

The passed address is allowed to be NULL. In that case the call doesn't any effect.

If the passed address has not previously been returned by a call to [allocateArray\(\)](#) or [reallocateArray\(\)](#) and also isn't NULL, then the beha undefined.

Parameters

p the address of the array, that should be deallocated.



Photon C++

Client API 4.1.12.2

ExitGames > Common > TypeCode >

[Variables](#)

ExitGames::Common::TypeCode Namespace Reference

Variables

static const nByte **BYTE**
nByte

static const nByte **SHORT**
short

static const nByte **INTEGER**
int

static const nByte **LONG**
int64

static const nByte **FLOAT**
float

static const nByte **DOUBLE**
double

static const nByte **BOOLEAN**
bool

static const nByte **STRING**
JString.

static const nByte **HASHTABLE**
Hashtable.

static const nByte **DICTIONARY**
Dictionary.

static const nByte **OBJECT**
Object, only allowed for arrays!

static const nByte **ARRAY**
internal only

static const nByte **BYTEARRAY**
internal only

static const nByte **PHOTON_COMMAND**
internal only

static const nByte **EG_NULL**
internal only

static const nByte **CUSTOM**
internal only

static const nByte **UNKNOWN**
internal only

Detailed Description

TypeCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Lite >

[Namespaces](#) | [Classes](#)

ExitGames::Lite Namespace Reference

Namespaces

EventCache

EventCode

EventKey

OperationCode

ParameterCode

ReceiverGroup

Classes

class **LitePeer**

Detailed Description

Lite

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Lite > EventCache >

Variables

ExitGames::Lite::EventCache Namespace Reference

Variables

static const nByte **DO_NOT_CACHE**

static const nByte **MERGE_CACHE**

static const nByte **REPLACE_CACHE**

static const nByte **REMOVE_CACHE**

static const nByte **ADD_TO_ROOM_CACHE**

static const nByte **ADD_TO_ROOM_CACHE_GLOBAL**

static const nByte **REMOVE_FROM_ROOM_CACHE**

static const nByte **REMOVE_FROM_ROOM_CACHE_FOR_ACTORS_**

static const nByte **SLICE_INC_INDEX**

static const nByte **SLICE_SET_INDEX**

static const nByte **SLICE_PURGE_INDEX**

static const nByte **SLICE_PURGE_UP_TO_INDEX**

Detailed Description

EventCache

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Lite > EventCode >

Variables

ExitGames::Lite::EventCode Namespace Reference

Variables

static const nByte **JOIN**

static const nByte **LEAVE**

static const nByte **PROPERTIES_CHANGED**

Detailed Description

EventCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Lite > EventKey >

Variables

ExitGames::Lite::EventKey Namespace Reference

Variables

static const nByte **ACTORNR**

static const nByte **TARGET_ACTORNR**

static const nByte **ACTORLIST**

static const nByte **PROPERTIES**

static const nByte **ACTORPROPERTIES**

static const nByte **GAMEPROPERTIES**

static const nByte **DATA**

Detailed Description

EventKey

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Lite > OperationCode >

[Variables](#)

ExitGames::Lite::OperationCode Namespace Reference

Variables

static const nByte **JOIN**

static const nByte **LEAVE**

static const nByte **RAISE_EV**

static const nByte **SETPROPERTIES**

static const nByte **GETPROPERTIES**

static const nByte **CHANGE_GROUPS**

Detailed Description

OperationCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Lite > ParameterCode >

[Variables](#)

ExitGames::Lite::ParameterCode Namespace Reference

Variables

static const nByte **GAMEID**

static const nByte **ACTORNR**

static const nByte **TARGET_ACTORNR**

static const nByte **ACTOR_LIST**

static const nByte **PROPERTIES**

static const nByte **BROADCAST**

static const nByte **ACTOR_PROPERTIES**

static const nByte **GAME_PROPERTIES**

static const nByte **CACHE**

static const nByte **RECEIVER_GROUP**

static const nByte **DATA**

static const nByte **CODE**

static const nByte **GROUP**

static const nByte **REMOVE**

static const nByte **ADD**

Detailed Description

ParameterCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Lite > ReceiverGroup >

Variables

ExitGames::Lite::ReceiverGroup Namespace Reference

Variables

static const nByte **OTHERS**

static const nByte **ALL**

static const nByte **MASTER_CLIENT**

Detailed Description

ReceiverGroup

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) >

[Namespaces](#) | [Classes](#)

ExitGames::LoadBalancing Namespace Reference

Namespaces

CustomAuthenticationType

DirectMode

DisconnectCause

ErrorCode

LobbyType

MatchmakingMode

PeerStates

Classes

class **AuthenticationValues**

class **Client**

class **FriendInfo**

class **Listener**

class **LobbyStatsRequest**

class **LobbyStatsResponse**

class **MutablePlayer**

class **MutableRoom**

class **Peer**

class **Player**

class **RaiseEventOptions**

class **Room**

class **RoomOptions**

class **WebFlags**

Detailed Description

LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) > [CustomAuthenticationType](#) >

[Variables](#)

ExitGames::LoadBalancing::CustomAuthenticat Namespace Reference

Variables

static const nByte **CUSTOM**
Use a custom authentication service.

static const nByte **STEAM**
Authenticates users by their Steam Account. Set auth values accordingly!

static const nByte **FACEBOOK**
Authenticates users by their Facebook Account. Set auth values accordingly!

static const nByte **OCULUS**
Authenticates users by their Oculus Account. Set auth values accordingly!

static const nByte **PLAYSTATION**
Authenticates users by their PSN Account. Set auth values accordingly!

static const nByte **XBOX**
Authenticates users by their XBox Network Account. Set auth values accordingly!

static const nByte **NONE**
Disables custom authentication.

Detailed Description

Options for optional "Custom Authentication" services used with **Photon**. Used when the client sends an authentication request to the server.

CustomAuthenticationType

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > DirectMode >

[Variables](#)

ExitGames::LoadBalancing::DirectMode Namespace Reference

Variables

static const nByte **NONE**
Do not create any 2p2 connections between the clients. This is the default.

static const nByte **ALL_TO_ALL**
Each client establishes a direct connection with every other client inside the room.

static const nByte **MASTER_TO_ALL**
The master client establishes a direct connection with every other client inside the room. All other clients only establish a direct connection with the master client but not with each other.

Detailed Description

Options for optional client to client direct connections - set in **RoomOptions** during room creation.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

 Photon C++
Client API 4.1.12.2

ExitGames > LoadBalancing > DisconnectCause >

Variables

ExitGames::LoadBalancing::DisconnectCause Namespace Reference

Variables

static const int **NONE**
No error was tracked.

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
OnStatusChanged: The CCUs count of your **Photon** Server License is exhausted (temporarily).

static const int **EXCEPTION_ON_CONNECT**
OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

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

static const int **DISCONNECT_BY_SERVER_LOGIC**
OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).

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

static const int **EXCEPTION**
OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

static const int **INVALID_AUTHENTICATION**
OnOperationResponse: Authenticate in the **Photon** Cloud with invalid AppId. Update your subscription or contact Exit Games.

static const int **MAX_CCU_REACHED**

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

static const int **INVALID_REGION**

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

static const int **OPERATION_NOT_ALLOWED_IN_CURRENT_STATE**

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

static const int **CUSTOM_AUTHENTICATION_FAILED**

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

Detailed Description

Enumeration of causes for Disconnects (used in `LoadBalancingClient.DisconnectedCause`). Read the individual descriptions to find out what to do about this type of disconnect.

DisconnectCause

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) > [ErrorCode](#) >

[Variables](#)

ExitGames::LoadBalancing::ErrorCode

Namespace Reference

Variables

static const int **OPERATION_DENIED**

static const int **OPERATION_INVALID**

static const int **INTERNAL_SERVER_ERROR**

static const int **OK**

static const int **INVALID_AUTHENTICATION**

static const int **GAME_ID_ALREADY_EXISTS**

static const int **GAME_FULL**

static const int **GAME_CLOSED**

static const int **ALREADY_MATCHED**

static const int **SERVER_FULL**

static const int **USER_BLOCKED**

static const int **NO_MATCH_FOUND**

static const int **GAME_DOES_NOT_EXIST**

static const int **MAX_CCU_REACHED**

static const int **INVALID_REGION**

static const int **CUSTOM_AUTHENTICATION_FAILED**

static const int **AUTHENTICATION_TOKEN_EXPIRED**

static const int **PLUGIN_REPORTED_ERROR**

static const int **PLUGIN_MISMATCH**

static const int **JOIN_FAILED_PEER_ALREADY_JOINED**

static const int **JOIN_FAILED_FOUND_INACTIVE_JOINER**

static const int **JOIN_FAILED_WITH_REJOINER_NOT_FOUND**

static const int **JOIN_FAILED_FOUND_EXCLUDED_USER_ID**

static const int **JOIN_FAILED_FOUND_ACTIVE_JOINER**

static const int **HTTP_LIMIT_REACHED**

static const int **EXTERNAL_HTTP_CALL_FAILED**

static const int **SLOT_ERROR**

static const int **INVALID_ENCRYPTION_PARAMETERS**

Detailed Description

ErrorCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) > [LobbyType](#)

[Variables](#)

ExitGames::LoadBalancing::LobbyType Namespace Reference

Variables

static const nByte **DEFAULT**
This lobby type is used unless another lobby type is specified. **Room** lists will be sent and **Client::opJoinRandomRoom()** can filter by matching properties.

static const nByte **SQL_LOBBY**
This lobby type lists rooms like type DEFAULT but SQL-like "where" clauses for filtering can be used with **Client::opJoinRandomRoom()**. This allows 'bigger', 'less', 'or' and 'and' combinations.

static const nByte **ASYNC_RANDOM_LOBBY**
This lobby does not send room lists. It is only used for **Client::opJoinRandomRoom()**. It keeps rooms available for matchmaking for a while even when there are only inactive users left.

Detailed Description

Options of lobby types available. Lobby types might be implemented in certain **Photon** versions and won't be available on older servers.

LobbyType

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) > [MatchmakingMode](#)

[Variables](#)

ExitGames::LoadBalancing::MatchmakingMode Namespace Reference

Variables

static const nByte **FILL_ROOM**

Fills up rooms (oldest first) to get players together as fast as possible. Default. Makes most sense with MaxPlayers > 0 and games that can only start with more players.

static const nByte **SERIAL_MATCHING**

Distributes players across available rooms sequentially but takes filters into account. Without filters, rooms get players evenly distributed.

static const nByte **RANDOM_MATCHING**

Joins a (fully) random room. Expected properties must match, but aside from this, any available room might be selected.

Detailed Description

Options for matchmaking rules for OpJoinRandomRoom.

MatchmakingMode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [LoadBalancing](#) > [PeerStates](#) >

[Variables](#)

ExitGames::LoadBalancing::PeerStates Namespace Reference

Variables

static const int **Uninitialized**

static const int **PeerCreated**

static const int **ConnectingToNameserver**

static const int **ConnectedToNameserver**

static const int **DisconnectingFromNameserver**

static const int **Connecting**

static const int **Connected**

static const int **WaitingForCustomAuthenticationNextStepCall**

static const int **Authenticated**

static const int **JoinedLobby**

static const int **DisconnectingFromMasterserver**

static const int **ConnectingToGameserver**

static const int **ConnectedToGameserver**

static const int **AuthenticatedOnGameServer**

static const int **Joining**

static const int **Joined**

static const int **Leaving**

static const int **Left**

static const int **DisconnectingFromGameserver**

static const int **ConnectingToMasterserver**

static const int **ConnectedComingFromGameserver**

static const int **AuthenticatedComingFromGameserver**

static const int **Disconnecting**

static const int **Disconnected**

Detailed Description

PeerStates

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#)

[Namespaces](#) | [Classes](#) | [Typedefs](#)

ExitGames::Photon Namespace Reference

Namespaces

ConnectionProtocol

ErrorCode

NetworkPort

PeerState

Punchthrough

StatusCode

Classes

class **EventData**

class **OperationRequest**

class **OperationResponse**

class **PhotonListener**

class **PhotonPeer**

class **TrafficStats**

class **TrafficStatsGameLevel**

Typedefs

```
typedef Common::Dictionary< nByte, Common::Object > OperationF
```

Detailed Description

Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames >

Photon >

ConnectionProtocol >

[Functions](#) | [Variables](#)

ExitGames::Photon::ConnectionProtocol Namespace Reference

Functions

bool **getIsUDP** (nByte connectionProtocol)

bool **getIsTCP** (nByte connectionProtocol)

bool **getIsWebSocket** (nByte connectionProtocol)

bool **getIsSecure** (nByte connectionProtocol)

Variables

static const nByte **UDP**
Use UDP to connect to **Photon**, which allows you to send operations reliable or unreliable on demand.

static const nByte **TCP**
Use TCP to connect to **Photon**.

static const nByte **WS**
Use websockets to connect to **Photon**.

static const nByte **WSS**
Use secure websockets to connect to **Photon**.

static const nByte **DEFAULT**

Detailed Description

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

ConnectionProtocol

Function Documentation

§ getIsUDP()

```
bool getIsUDP ( nByte connectionProtocol )
```

Parameters

connectionProtocol one of the constants in **ConnectionProtocol**

Returns

true if the passed in value matches **ConnectionProtocol::UDP**, false otherwise.

§ getIsTCP()

```
bool getIsTCP ( nByte connectionProtocol )
```

Parameters

connectionProtocol one of the constants in **ConnectionProtocol**

Returns

true if the passed in value matches **ConnectionProtocol::TCP**, false otherwise.

§ getIsWebSocket()

```
bool getIsWebSocket ( nByte connectionProtocol )
```

Parameters

connectionProtocol one of the constants in **ConnectionProtocol**

Returns

true if the passed in value matches either **ConnectionProtocol::WS** or **ConnectionProtocol::WSS**, false otherwise.

§ getIsSecure()

```
bool getIsSecure ( nByte connectionProtocol )
```

Parameters

connectionProtocol one of the constants in [ConnectionProtocol](#)

Returns

true if the passed in value matches a connection protocol that uses secure sockets (like HTTPS or WSS), false otherwise.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames > Photon > ErrorCode >

[Variables](#)

ExitGames::Photon::ErrorCode Namespace Reference

Variables

static const int **SUCCESS**
No error.

static const int **EFAILED**
General failure.

static const int **ENOMEMORY**
Out of memory.

static const int **EBADCLASS**
NULL class object.

static const int **EBADPARAM**
Invalid parameter.

static const int **EITEMBUSY**
Context (system, interface, etc.) is busy.

static const int **NET_SUCCESS**
No network error, successful operation.

static const int **NET_ERROR**
Unsuccessful operation.

static const int **NET_ENETNONET**
Network subsystem unavailable.

static const int **NET_MSGSIZE**
Message too long. A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.

```
static const int NET_ENOTCONN
```

Detailed Description

Photon library error codes - can be returned as operationcode in callbacks, if the returncode indicates an error

ErrorCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Photon > NetworkPort >

Classes

ExitGames::Photon::NetworkPort Namespace Reference

Classes

struct **Protocol**

struct **TCP**

struct **UDP**

struct **UDPAlternative**

struct **WS**

struct **WSS**

Detailed Description

NetworkPort

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Photon > PeerState >

Variables

ExitGames::Photon::PeerState Namespace Reference

Variables

static const int **DISCONNECTED**

The peer is disconnected and can't call Operations. Call PhotonPeer_connect().

static const int **CONNECTING**

The peer is establishing the connection: opening a socket, exchanging packages with **Photon**.

static const int **INITIALIZING_APPLICATION**

The connection is established and now sends the application name to **Photon**. You set the "application name" by calling PhotonPeer_connect().

static const int **CONNECTED**

The peer is connected and initialized (selected an application). You can now use operations.

static const int **DISCONNECTING**

The peer is disconnecting. It sent a disconnect to the server, which will acknowledge closing the connection.

Detailed Description

PeerState

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Photon > Punchthrough >

Classes

ExitGames::Photon::Punchthrough Namespace Reference

Classes

class **Puncher**

class **PunchListener**

class **RelayClient**

Detailed Description

Punchthrough

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

ExitGames > Photon > StatusCode >

Variables

ExitGames::Photon::StatusCode Namespace Reference

Variables

static const int **EXCEPTION_ON_CONNECT**
the **PhotonPeer** encountered an exception while opening the incoming connection to the server. The server could be down / not running.

static const int **CONNECT**
the **PhotonPeer** is connected.

static const int **DISCONNECT**
the **PhotonPeer** just disconnected.

static const int **EXCEPTION**
the **PhotonPeer** encountered an exception and will disconnect, too.

static const int **QUEUE_OUTGOING_RELIABLE_WARNING**
PhotonPeer outgoing queue is filling up. send more often.

static const int **QUEUE_OUTGOING_UNRELIABLE_WARNING**
PhotonPeer outgoing queue is filling up. send more often.

static const int **SEND_ERROR**
Sending command failed. Either not connected, or the requested channel is bigger than the number of initialized channels.

static const int **QUEUE_OUTGOING_ACKS_WARNING**
PhotonPeer outgoing queue is filling up. Send more often.

static const int **QUEUE_INCOMING_RELIABLE_WARNING**
PhotonPeer incoming reliable queue is filling up.

Dispatch more often.

static const int **QUEUE_INCOMING_UNRELIABLE_WARNING**
PhotonPeer incoming unreliable queue is filling up.
Dispatch more often.

static const int **QUEUE_SENT_WARNING**
PhotonPeer sent queue is filling up. Check, why the server does not acknowledge your sent reliable commands.

static const int **INTERNAL_RECEIVE_EXCEPTION**
Exception, if a server cannot be connected. Most likely, the server is not responding. Ask the user to try again later.

static const int **TIMEOUT_DISCONNECT**
Disconnection due to a timeout (client did no longer receive ACKs from server).

static const int **DISCONNECT_BY_SERVER**
Disconnect by server due to timeout (received a disconnect command, cause server misses ACKs of client).

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
Disconnect by server due to concurrent user limit reached (received a disconnect command).

static const int **DISCONNECT_BY_SERVER_LOGIC**
Disconnect by server due to server's logic (received a disconnect command).

static const int **ENCRYPTION_ESTABLISHED**
The encryption-setup for secure communication finished successfully.

static const int **ENCRYPTION_FAILED_TO_ESTABLISH**

The encryption-setup failed for some reason. Check debug logs.

Detailed Description

StatusCode

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented namespace members with links to the namespaces they belong to:

- a -

- ALL : [ExitGames::Common::DebugLevel](#)
- ALL_TO_ALL : [ExitGames::LoadBalancing::DirectMode](#)
- allocate() : [ExitGames::Common::MemoryManagement](#)
- allocateArray() : [ExitGames::Common::MemoryManagement](#)
- ARRAY : [ExitGames::Common::TypeCode](#)
- ASYNC_RANDOM_LOBBY :
[ExitGames::LoadBalancing::LobbyType](#)
- Authenticated : [ExitGames::Chat::ClientState](#)
- Authenticating : [ExitGames::Chat::ClientState](#)
- AWAY : [ExitGames::Chat::UserStatus](#)

- b -

- BOOLEAN : [ExitGames::Common::TypeCode](#)
- BYTE : [ExitGames::Common::TypeCode](#)
- BYTEARRAY : [ExitGames::Common::TypeCode](#)

- c -

- CONNECT : [ExitGames::Photon::StatusCode](#)
- CONNECTED : [ExitGames::Photon::PeerState](#)
- ConnectedToFrontEnd : [ExitGames::Chat::ClientState](#)
- ConnectedToNameServer : [ExitGames::Chat::ClientState](#)
- CONNECTING : [ExitGames::Photon::PeerState](#)
- ConnectingToFrontEnd : [ExitGames::Chat::ClientState](#)
- ConnectingToNameServer : [ExitGames::Chat::ClientState](#)
- CUSTOM : [ExitGames::Chat::CustomAuthenticationType](#) ,
[ExitGames::Common::TypeCode](#) ,
[ExitGames::LoadBalancing::CustomAuthenticationType](#)
- CUSTOM_AUTHENTICATION_FAILED :

ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause

- d -

- deallocate() : **ExitGames::Common::MemoryManagement**
- deallocateArray() : **ExitGames::Common::MemoryManagement**
- DEFAULT : **ExitGames::LoadBalancing::LobbyType**
- DICTIONARY : **ExitGames::Common::TypeCode**
- DISCONNECT : **ExitGames::Photon::StatusCode**
- DISCONNECT_BY_SERVER :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- DISCONNECT_BY_SERVER_LOGIC :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- DISCONNECT_BY_SERVER_USER_LIMIT :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- Disconnected : **ExitGames::Chat::ClientState**
- DISCONNECTED : **ExitGames::Photon::PeerState**
- Disconnecting : **ExitGames::Chat::ClientState**
- DISCONNECTING : **ExitGames::Photon::PeerState**
- DisconnectingFromNameServer : **ExitGames::Chat::ClientState**
- DND : **ExitGames::Chat::UserStatus**
- DOUBLE : **ExitGames::Common::TypeCode**

- e -

- EBADCLASS : **ExitGames::Photon::ErrorCode**
- EBADPARAM : **ExitGames::Photon::ErrorCode**
- EFAILED : **ExitGames::Photon::ErrorCode**
- EG_NULL : **ExitGames::Common::TypeCode**
- EITEMBUSY : **ExitGames::Photon::ErrorCode**
- ENCRYPTION_ESTABLISHED : **ExitGames::Photon::StatusCode**
- ENCRYPTION_FAILED_TO_ESTABLISH :
ExitGames::Photon::StatusCode

- ENOMEMORY : **ExitGames::Photon::ErrorCode**
- ERRORS : **ExitGames::Common::DebugLevel**
- EXCEPTION : **ExitGames::Chat::DisconnectCause** ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- EXCEPTION_ON_CONNECT :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode

- f -

- FACEBOOK : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType
- FILL_ROOM : **ExitGames::LoadBalancing::MatchmakingMode**
- FLOAT : **ExitGames::Common::TypeCode**

- g -

- getIsSecure() : **ExitGames::Photon::ConnectionProtocol**
- getIsTCP() : **ExitGames::Photon::ConnectionProtocol**
- getIsUDP() : **ExitGames::Photon::ConnectionProtocol**
- getIsWebSocket() : **ExitGames::Photon::ConnectionProtocol**

- h -

- HASHTABLE : **ExitGames::Common::TypeCode**

- i -

- INFO : **ExitGames::Common::DebugLevel**
- INITIALIZING_APPLICATION : **ExitGames::Photon::PeerState**
- INTEGER : **ExitGames::Common::TypeCode**
- INTERNAL_RECEIVE_EXCEPTION :
ExitGames::Photon::StatusCode
- INVALID_AUTHENTICATION :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause
- INVALID_REGION : **ExitGames::Chat::DisconnectCause** ,

ExitGames::LoadBalancing::DisconnectCause

- INVISIBLE : **ExitGames::Chat::UserStatus**

- l -

- LFG : **ExitGames::Chat::UserStatus**
- LONG : **ExitGames::Common::TypeCode**

- m -

- MASTER_TO_ALL : **ExitGames::LoadBalancing::DirectMode**
- MAX_CCU_REACHED : **ExitGames::Chat::DisconnectCause** ,
ExitGames::LoadBalancing::DisconnectCause

- n -

- NET_ENETNONET : **ExitGames::Photon::ErrorCode**
- NET_ERROR : **ExitGames::Photon::ErrorCode**
- NET_MSGSIZE : **ExitGames::Photon::ErrorCode**
- NET_SUCCESS : **ExitGames::Photon::ErrorCode**
- NONE : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::CustomAuthenticationType ,
ExitGames::LoadBalancing::DirectMode ,
ExitGames::LoadBalancing::DisconnectCause

- o -

- OBJECT : **ExitGames::Common::TypeCode**
- OCULUS : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType
- OFF : **ExitGames::Common::DebugLevel**
- OFFLINE : **ExitGames::Chat::UserStatus**
- ONLINE : **ExitGames::Chat::UserStatus**
- OPERATION_NOT_ALLOWED_IN_CURRENT_STATE :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause

- p -

- PHOTON_COMMAND : **ExitGames::Common::TypeCode**
- PLAYING : **ExitGames::Chat::UserStatus**
- PLAYSTATION : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType

- q -

- QUEUE_INCOMING_RELIABLE_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_INCOMING_UNRELIABLE_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_OUTGOING_ACKS_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_OUTGOING_RELIABLE_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_OUTGOING_UNRELIABLE_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_SENT_WARNING : **ExitGames::Photon::StatusCode**

- r -

- RANDOM_MATCHING :
ExitGames::LoadBalancing::MatchmakingMode
- reallocateArray() : **ExitGames::Common::MemoryManagement**

- s -

- SEND_ERROR : **ExitGames::Photon::StatusCode**
- SERIAL_MATCHING :
ExitGames::LoadBalancing::MatchmakingMode
- setAllocator() : **ExitGames::Common::MemoryManagement**
- setAllocatorToDefault() :
ExitGames::Common::MemoryManagement
- setMaxAllocSize() : **ExitGames::Common::MemoryManagement**
- setMaxSizeForAllocatorUsage() :
ExitGames::Common::MemoryManagement
- SHORT : **ExitGames::Common::TypeCode**
- SQL_LOBBY : **ExitGames::LoadBalancing::LobbyType**
- STEAM : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType

- **STRING** : **ExitGames::Common::TypeCode**
- **SUCCESS** : **ExitGames::Photon::ErrorCode**

- t -

- **TCP** : **ExitGames::Photon::ConnectionProtocol**
- **TIMEOUT_DISCONNECT** : **ExitGames::Chat::DisconnectCause** ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode

- u -

- **UDP** : **ExitGames::Photon::ConnectionProtocol**
- **Uninitialized** : **ExitGames::Chat::ClientState**
- **UNKNOWN** : **ExitGames::Common::TypeCode**

- w -

- **WARNINGS** : **ExitGames::Common::DebugLevel**
- **WS** : **ExitGames::Photon::ConnectionProtocol**
- **WSS** : **ExitGames::Photon::ConnectionProtocol**

- x -

- **XBOX** : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- `allocate()` : **ExitGames::Common::MemoryManagement**
- `allocateArray()` : **ExitGames::Common::MemoryManagement**
- `deallocate()` : **ExitGames::Common::MemoryManagement**
- `deallocateArray()` : **ExitGames::Common::MemoryManagement**
- `getIsSecure()` : **ExitGames::Photon::ConnectionProtocol**
- `getIsTCP()` : **ExitGames::Photon::ConnectionProtocol**
- `getIsUDP()` : **ExitGames::Photon::ConnectionProtocol**
- `getIsWebSocket()` : **ExitGames::Photon::ConnectionProtocol**
- `reallocateArray()` : **ExitGames::Common::MemoryManagement**
- `setAllocator()` : **ExitGames::Common::MemoryManagement**
- `setAllocatorToDefault()` :
ExitGames::Common::MemoryManagement
- `setMaxAllocSize()` : **ExitGames::Common::MemoryManagement**
- `setMaxSizeForAllocatorUsage()` :
ExitGames::Common::MemoryManagement

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

- a -

- ALL : **ExitGames::Common::DebugLevel**
- ALL_TO_ALL : **ExitGames::LoadBalancing::DirectMode**
- ARRAY : **ExitGames::Common::TypeCode**
- ASYNC_RANDOM_LOBBY :
ExitGames::LoadBalancing::LobbyType
- Authenticated : **ExitGames::Chat::ClientState**
- Authenticating : **ExitGames::Chat::ClientState**
- AWAY : **ExitGames::Chat::UserStatus**

- b -

- BOOLEAN : **ExitGames::Common::TypeCode**
- BYTE : **ExitGames::Common::TypeCode**
- BYTEARRAY : **ExitGames::Common::TypeCode**

- c -

- CONNECT : **ExitGames::Photon::StatusCode**
- CONNECTED : **ExitGames::Photon::PeerState**
- ConnectedToFrontEnd : **ExitGames::Chat::ClientState**
- ConnectedToNameServer : **ExitGames::Chat::ClientState**
- CONNECTING : **ExitGames::Photon::PeerState**
- ConnectingToFrontEnd : **ExitGames::Chat::ClientState**
- ConnectingToNameServer : **ExitGames::Chat::ClientState**
- CUSTOM : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::Common::TypeCode ,
ExitGames::LoadBalancing::CustomAuthenticationType
- CUSTOM_AUTHENTICATION_FAILED :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause

- d -

- DEFAULT : **ExitGames::LoadBalancing::LobbyType**
- DICTIONARY : **ExitGames::Common::TypeCode**
- DISCONNECT : **ExitGames::Photon::StatusCode**
- DISCONNECT_BY_SERVER :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- DISCONNECT_BY_SERVER_LOGIC :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- DISCONNECT_BY_SERVER_USER_LIMIT :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode
- Disconnected : **ExitGames::Chat::ClientState**
- DISCONNECTED : **ExitGames::Photon::PeerState**
- Disconnecting : **ExitGames::Chat::ClientState**
- DISCONNECTING : **ExitGames::Photon::PeerState**
- DisconnectingFromNameServer : **ExitGames::Chat::ClientState**
- DND : **ExitGames::Chat::UserStatus**
- DOUBLE : **ExitGames::Common::TypeCode**

- e -

- EBADCLASS : **ExitGames::Photon::ErrorCode**
- EBADPARAM : **ExitGames::Photon::ErrorCode**
- EFAILED : **ExitGames::Photon::ErrorCode**
- EG_NULL : **ExitGames::Common::TypeCode**
- EITEMBUSY : **ExitGames::Photon::ErrorCode**
- ENCRYPTION_ESTABLISHED : **ExitGames::Photon::StatusCode**
- ENCRYPTION_FAILED_TO_ESTABLISH :
ExitGames::Photon::StatusCode
- ENOMEMORY : **ExitGames::Photon::ErrorCode**
- ERRORS : **ExitGames::Common::DebugLevel**
- EXCEPTION : **ExitGames::Chat::DisconnectCause** ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode

- EXCEPTION_ON_CONNECT :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause ,
ExitGames::Photon::StatusCode

- f -

- FACEBOOK : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType
- FILL_ROOM : **ExitGames::LoadBalancing::MatchmakingMode**
- FLOAT : **ExitGames::Common::TypeCode**

- h -

- HASHTABLE : **ExitGames::Common::TypeCode**

- i -

- INFO : **ExitGames::Common::DebugLevel**
- INITIALIZING_APPLICATION : **ExitGames::Photon::PeerState**
- INTEGER : **ExitGames::Common::TypeCode**
- INTERNAL_RECEIVE_EXCEPTION :
ExitGames::Photon::StatusCode
- INVALID_AUTHENTICATION :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause
- INVALID_REGION : **ExitGames::Chat::DisconnectCause** ,
ExitGames::LoadBalancing::DisconnectCause
- INVISIBLE : **ExitGames::Chat::UserStatus**

- l -

- LFG : **ExitGames::Chat::UserStatus**
- LONG : **ExitGames::Common::TypeCode**

- m -

- MASTER_TO_ALL : **ExitGames::LoadBalancing::DirectMode**
- MAX_CCÜ_REACHED : **ExitGames::Chat::DisconnectCause** ,

ExitGames::LoadBalancing::DisconnectCause

- n -

- NET_ENETNONET : **ExitGames::Photon::ErrorCode**
- NET_ERROR : **ExitGames::Photon::ErrorCode**
- NET_MSGSIZE : **ExitGames::Photon::ErrorCode**
- NET_SUCCESS : **ExitGames::Photon::ErrorCode**
- NONE : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::CustomAuthenticationType ,
ExitGames::LoadBalancing::DirectMode ,
ExitGames::LoadBalancing::DisconnectCause

- o -

- OBJECT : **ExitGames::Common::TypeCode**
- OCULUS : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType
- OFF : **ExitGames::Common::DebugLevel**
- OFFLINE : **ExitGames::Chat::UserStatus**
- ONLINE : **ExitGames::Chat::UserStatus**
- OPERATION_NOT_ALLOWED_IN_CURRENT_STATE :
ExitGames::Chat::DisconnectCause ,
ExitGames::LoadBalancing::DisconnectCause

- p -

- PHOTON_COMMAND : **ExitGames::Common::TypeCode**
- PLAYING : **ExitGames::Chat::UserStatus**
- PLAYSTATION : **ExitGames::Chat::CustomAuthenticationType** ,
ExitGames::LoadBalancing::CustomAuthenticationType

- q -

- QUEUE_INCOMING_RELIABLE_WARNING :
ExitGames::Photon::StatusCode
- QUEUE_INCOMING_UNRELIABLE_WARNING :
ExitGames::Photon::StatusCode

- QUEUE_OUTGOING_ACKS_WARNING : **ExitGames::Photon::StatusCode**
- QUEUE_OUTGOING_RELIABLE_WARNING : **ExitGames::Photon::StatusCode**
- QUEUE_OUTGOING_UNRELIABLE_WARNING : **ExitGames::Photon::StatusCode**
- QUEUE_SENT_WARNING : **ExitGames::Photon::StatusCode**

- r -

- RANDOM_MATCHING : **ExitGames::LoadBalancing::MatchmakingMode**

- s -

- SEND_ERROR : **ExitGames::Photon::StatusCode**
- SERIAL_MATCHING : **ExitGames::LoadBalancing::MatchmakingMode**
- SHORT : **ExitGames::Common::TypeCode**
- SQL_LOBBY : **ExitGames::LoadBalancing::LobbyType**
- STEAM : **ExitGames::Chat::CustomAuthenticationType** , **ExitGames::LoadBalancing::CustomAuthenticationType**
- STRING : **ExitGames::Common::TypeCode**
- SUCCESS : **ExitGames::Photon::ErrorCode**

- t -

- TCP : **ExitGames::Photon::ConnectionProtocol**
- TIMEOUT_DISCONNECT : **ExitGames::Chat::DisconnectCause** , **ExitGames::LoadBalancing::DisconnectCause** , **ExitGames::Photon::StatusCode**

- u -

- UDP : **ExitGames::Photon::ConnectionProtocol**
- Uninitialized : **ExitGames::Chat::ClientState**
- UNKNOWN : **ExitGames::Common::TypeCode**

- w -

- WARNINGS : [ExitGames::Common::DebugLevel](#)
- WS : [ExitGames::Photon::ConnectionProtocol](#)
- WSS : [ExitGames::Photon::ConnectionProtocol](#)

- X -

- XBOX : [ExitGames::Chat::CustomAuthenticationType](#) ,
[ExitGames::LoadBalancing::CustomAuthenticationType](#)

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Class List

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

[detail level 1 2 3 4]

▼  ExitGames

▼  Chat

 AuthenticationValues

 Channel

 Client

 Listener

 Peer

▼  Common

▼  MemoryManagement

 AllocatorInterface

 ANSIStrng

 Base

 BaseCharString

 BaseListener

 CustomType

 CustomTypeBase

 CustomTypeFactory

 DeSerializer

 Dictionary

 DictionaryBase

 EGTime

☞ Hashtable	
☞ JString	
☞ JVector	
☞ KeyObject	
☞ LogFormatOptions	
☞ Logger	
☞ Object	
☞ Serializer	
☞ ToString	
☞ UTF8String	
☞ ValueObject	
▼ ☞ Lite	
☞ LitePeer	
▼ ☞ LoadBalancing	
☞ AuthenticationValues	
☞ Client	
☞ FriendInfo	
☞ Listener	
☞ LobbyStatsRequest	
☞ LobbyStatsResponse	
☞ MutablePlayer	
☞ MutableRoom	
☞ Peer	
☞ Player	
☞ RaiseEventOptions	
☞ Room	
☞ RoomOptions	
☞ WebFlags	
▼ ☞ Photon	
▼ ☞ NetworkPort	

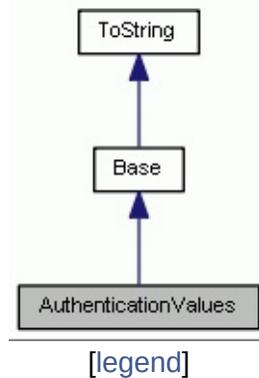
☰ Protocol	
☰ TCP	
☰ UDP	
☰ UDPAlternative	
☰ WS	
☰ WSS	
▼ ☰ Punchthrough	
☰ Puncher	
☰ PunchListener	
☰ RelayClient	
☰ eventData	
☰ OperationRequest	
☰ OperationResponse	
☰ PhotonListener	
☰ PhotonPeer	
☰ TrafficStats	
☰ TrafficStatsGameLevel	

© Exit Games ®, all rights reserved.

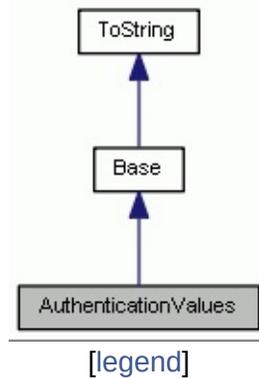
[Photon Documentation](#) | [Contact](#) | [Terms](#)

AuthenticationValues Class Reference

Inheritance diagram for AuthenticationValues:



Collaboration diagram for AuthenticationValues:



Public Member Functions

AuthenticationValues (void)

nByte **getType** (void) const

AuthenticationValues & **setType** (nByte type)

const **Common::JString** & **getParameters** (void) const

AuthenticationValues & **setParameters** (const **Common:**
¶meters)

AuthenticationValues & **setParametersWithUsernameAr**
(const **Common::JString** &usern
const **Common::JString** &token)

const **Common::JVector**< nByte > & **getData** (void) const

AuthenticationValues & **setData** (const **Common::JVecto**
nByte > &data)

const **Common::JString** & **getSecret** (void) const

const **Common::JString** & **getUserID** (void) const

AuthenticationValues & **setUserID** (const **Common::JStr**
&userID)

virtual **Common::JString** & **toString** (**Common::JString** &ret
withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString typeToString** (void) const

JString toString (bool withTypes=false) c

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Container for user authentication in **Photon**.

Remarks

On **Photon**, user authentication is optional but can be useful in many cases. If you want to use `Client::opFindFriends()`, a unique ID per user is very practical.

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

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

If you don't set a user ID through `setUserID()` for the **AuthenticationValues** instance that you pass to `Client::connect()`, then **Photon** generates a unique user ID (which fulfills the requirements of a GUID) for you, which can be retrieved through `Client::getUserID()`, once the **Client** instance has notified `Listener::connectReturn()` about having successfully finished the connection procedure. Once you have set a user ID, the **Client** instance caches it until you either override it or until the end of the lifetime of the **Client** instance.

To be able to rejoin a room and to be recognized there as the previous user it is critical to continue to use the same user ID.

Therefore you should store the user ID in permanent storage and set it to that same stored value whenever you want to connect as that user, even if you let **Photon** initially generate that ID. Otherwise **Photon** would generate a new user ID for you whenever you construct a new **Client** instance (i.e. when the user restarts your app).

Constructor & Destructor Documentation

§ AuthenticationValues()

AuthenticationValues (void)

Constructor.

Member Function Documentation

§ getType()

```
nByte getType ( void ) const
```

Returns

the type of the "Custom Authentication" service that will be used.

See also

[setType\(\)](#)

§ setType()

AuthenticationValues & setType (nByte type)

Sets the type of the "Custom Authentication" service that will be used. The initial value before the first call to this function is **CustomAuthenticationType::NONE**.

Note

Any custom authentication type aside from **CustomAuthenticationType::NONE** requires you to set up an authentication service of matching type for your appID at <https://www.photonengine.com/dashboard>

Parameters

type needs to match one of the values in **CustomAuthenticationType**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getType(), **CustomAuthenticationType**

§ getParameters()

```
const JString & getParameters ( void ) const
```

Returns

the HTTP GET parameters that will be forwarded to the authentication service.

See also

[setParameters\(\)](#), [setParametersWithUsernameAndToken\(\)](#),
[getData\(\)](#), [setData\(\)](#)

§ setParameters()

AuthenticationValues &
setParameters (const **Common::JString** & parameters)

Sets the HTTP GET parameters that will be forwarded to the authentication service to the provided parameters.

The provided parameter string must contain any (HTTP GET) parameters that are expected by the used authentication service.

Remarks

Standard HTTP GET parameters are used here and passed on to the authentication service that's defined for the provided authentication type in the **Photon** Cloud Dashboard.

Parameters

parameters needs to be a valid HTTP GET string (i.e. param1=value1¶m2=value2¶m3=value3)

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getParameters\(\)](#), [setParametersWithUsernameAndToken\(\)](#), [getData\(\)](#), [setData\(\)](#)

§ setParametersWithUsernameAndToken()

AuthenticationValues &

```
setParametersWithUsernameAndToken ( const Common::JString & username,
                                     const Common::JString & token
                                   )
```

Sets the HTTP GET parameters that will be forwarded to the authentication service to the provided username and token.

Calling this function is equivalent to `setParameters(Common::JString(L"username=") + username + "&token" + token)`.

Parameters

username the username of the user that should be authenticated

token the authentication token needed by the authentication service to verify the user

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getParameters\(\)](#), [setParameters\(\)](#), [getData\(\)](#), [setData\(\)](#)

§ getData()

```
const JVector< nByte > & getData ( void ) const
```

Returns

the HTTP POST data that will be forwarded to the authentication service.

See also

[getParameters\(\)](#), [setParameters\(\)](#),
[setParametersWithUsernameAndToken\(\)](#), [setData\(\)](#)

§ setData()

```
AuthenticationValues &  
setData ( const Common::JVector< nByte > & data )
```

Sets the HTTP POST data, that will be forwarded to the authentication service, to the provided data.

The provided data needs to match what is expected by the used authentication service.

Remarks

The provided data is passed on to the authentication service that's defined for the provided authentication type in the **Photon** Cloud Dashboard.

Parameters

data the data to be used in the body of the POST request.

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getParameters\(\)](#), [setParameters\(\)](#),
[setParametersWithUsernameAndToken\(\)](#), [getData\(\)](#)

§ getSecret()

```
const JString & getSecret ( void ) const
```

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

Remarks

This is publicly read-accessible only for debugging purposes. For normal operations it is entirely unnecessary for the app code to ever access this value.

Returns

the cached secret

§ getUserID()

```
const JString & getUserID ( void ) const
```

Returns

the unique user ID

See also

[setUserID\(\)](#)

§ `setUserID()`

AuthenticationValues &
`setUserID`

(const **Common::JString** & `userID`)

Sets the unique user ID.

Parameters

userID a string that needs to be unique per user among all users of your app

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getUserID\(\)](#)

§ toString()

```
JString & toString ( Common::JString & retStr,  
                  bool                    withTypes = false  
                  ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

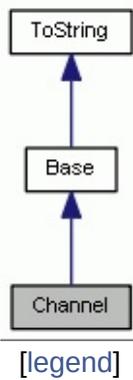
Returns

a JString representation of the instance and its contents for debugging purposes.

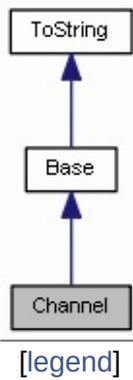
Implements **ToString**.

Channel Class Reference

Inheritance diagram for Channel:



Collaboration diagram for Channel:



Public Member Functions

	void	clearMessages (void)
const Common::JString &		getName (void) const
	unsigned int	getMessageCount (void) const
const Common::JVector < Common::JString > &		getSenders (void) const
const Common::JVector < Common::Object > &		getMessages (void) const
	bool	getIsPrivate (void) const
virtual Common::JString &		toString (Common::JString &retStr, bool withTypes=false) const
▶ Public Member Functions inherited from Base		
	virtual	~Base (void)
▶ Public Member Functions inherited from ToString		
	virtual	~ToString (void)
	virtual JString	typeToString (void) const
	JString	toString (bool

```
withTypes=false)  
const
```

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Represents channel or private chat (channel with 2 users)

Member Function Documentation

§ toString()

```
JString & toString ( Common::JString & retStr,  
                  bool                    withTypes = false  
                  )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++

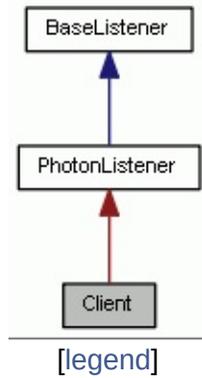
Client API 4.1.12.2

ExitGames > Chat > Client >

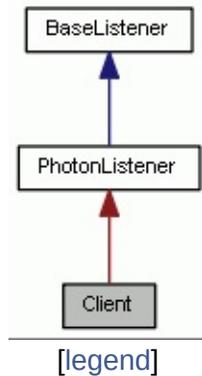
Client Class Reference

[Public Member Functions](#) |
[Static Public Member Functions](#) |
[List of all members](#)

Inheritance diagram for Client:



Collaboration diagram for Client:



Public Member Functions

	Client (Listener &listener, cc &applicationID, const CommonByte connectionProtocol=Photon::
virtual	~Client (void)
virtual bool	connect (const AuthenticationValues = AuthenticationValues , const Common::JString &nameServerAddress=M_NA
virtual void	disconnect (void)
virtual void	service (bool dispatchIncomingCommands)
virtual void	serviceBasic (void)
virtual bool	sendOutgoingCommands (
virtual bool	sendAcksOnly (void)
virtual bool	dispatchIncomingCommands
virtual void	fetchServerTimestamp (void)
virtual void	resetTrafficStats (void)
virtual void	resetTrafficStatsMaximumC
virtual Common::JString	vitalStatsToString (bool all)
virtual bool	opSubscribe (const Common::JString &channel, int messagesFromHistory=0)

virtual bool **opUnsubscribe** (const **Common::JString** &channelName)

template<typename Ftype >

bool **opPublishMessage** (const **C** &channelName, const Ftype &message)

template<typename Ftype >

bool **opPublishMessage** (const **C** &channelName, const Ftype &message, Common::Helpers::ArrayLength len)

template<typename Ftype >

bool **opPublishMessage** (const **C** &channelName, const Ftype &message, *pArrSizes)

template<typename Ftype >

bool **opSendPrivateMessage** (const **C** &userName, const Ftype &message)

template<typename Ftype >

bool **opSendPrivateMessage** (const **C** &userName, const Ftype pMessage, Common::Helpers::ArrayLength len, bool encrypt=false)

template<typename Ftype >

bool **opSendPrivateMessage** (const **C** &userName, const Ftype pMessage, *pArrSizes, bool encrypt=false)

virtual bool **opSetOnlineStatus** (int status)

template<typename Ftype >

bool **opSetOnlineStatus** (int status)

template<typename Ftype >

bool **opSetOnlineStatus** (int statu
typename Common::Helpers:
>::type arrSize)

template<typename Ftype >

bool **opSetOnlineStatus** (int statu
const short *pArrSizes)

virtual bool **opAddFriends** (const **Comm**
Common::JString > &userID)

virtual bool **opRemoveFriends** (const **C**
Common::JString > &userID)

int **getServerTimeOffset** (void)

int **getServerTime** (void) const

int **getBytesOut** (void) const

int **getBytesIn** (void) const

int **getByteCountCurrentDispa**

int **getByteCountLastOperatio**

int **getSentCountAllowance** (vc

void **setSentCountAllowance** (in

int **getTimePingInterval** (void) c

void **setTimePingInterval** (int tim

int **getRoundTripTime** (void) co

int **getRoundTripTimeVariance**

	int	getTimestampOfLastSocket	(void) const
	int	getDebugOutputLevel	(void) const
	bool	setDebugOutputLevel	(int d)
const Common::LogFormatOptions &		getLogFormatOptions	(void) const
	void	setLogFormatOptions	(const Common::LogFormatOptions)
	int	getIncomingReliableComm	
	short	getPeerID	(void) const
	int	getDisconnectTimeout	(void) const
	void	setDisconnectTimeout	(int c)
	int	getQueuedIncomingComm	
	int	getQueuedOutgoingComm	
	bool	getIsPayloadEncryptionAv	
	int	getResentReliableComman	
	int	getLimitOfUnreliableComm	
	void	setLimitOfUnreliableComm	
	bool	getCRCEnabled	(void) const
	void	setCRCEnabled	(bool crcEn)
	int	getPacketLossByCRC	(void)

```

bool getTrafficStatsEnabled (void) const
void setTrafficStatsEnabled (bool enabled)
int getTrafficStatsElapsedMs (void) const
const Photon::TrafficStats & getTrafficStatsIncoming (void) const
const Photon::TrafficStats & getTrafficStatsOutgoing (void) const
const Photon::TrafficStatsGameLevel & getTrafficStatsGameLevel (void) const
nByte getQuickResendAttempts (void) const
void setQuickResendAttempts (nByte attempts)
nByte getChannelCountUserChannel (void) const
const Common::JString & getUserID (void) const
int getState (void) const
int getDisconnectedCause (void) const
const Common::JString & getRegion (void) const
void setRegion (const Common::JString &region)
const Common::JVector< Channel * > & getPublicChannels (void) const
const Common::JVector< Channel * > & getPrivateChannels (void) const
const Channel * getPublicChannel (const Common::JString &channelName) const
const Channel * getPrivateChannel (const Common::JString &channelName) const

```

const

Static Public Member Functions

static short **getPeerCount** (void)

Detailed Description

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

This class must be instantiated with a **Chat::Listener** instance to get the callbacks and with application id that is setup as Photon Chat application. Integrate it into your game loop by calling **service()** regularly. Call **connect()** with an Name Server address. Note: Connect covers multiple messages between this client and the servers. A short workflow will connect you to a Chat server. Each **Chat::Client** resembles a user in chat. Before you send messages in any public channel, that channel must be subscribed. Private channels represent private chats and created automatically on private message sent or received. **getPublicChannels()** returns list of subscribed channels, containing messages and senders. **getPrivateChannels()** contains all incoming and sent private messages.

Constructor & Destructor Documentation

§ Client()

```
Client ( Listener & listener,  
        const Common::JString & applicationID,  
        const Common::JString & appVersion,  
        nByte connectionProtocol = Photon::Conne  
        )
```

Constructor.

Parameters

listener	pointer to the application's implementation of t callback interface.
applicationID	Photon Chat application id
appVersion	Photon Chat application version
connectionProtocol	connection protocol

§ ~Client()

`~Client (void)`

virtual

Destructor.

Member Function Documentation

§ connect()

```
bool  
connect ( const AuthenticationValues & authenticationValues = Authent  
          const Common::JString &     nameServerAddress = M_NAMES  
          )
```

Initiates a connection to the Photon name server. After a successful connection, the client automatically connects to a chat front end server and goes to Connected state. After that the client can subscribe to channels and send and receive messages.

Parameters

authenticationValues a user's authentication values used during connection to Photon. Authentication with Photon.

nameServerAddress used to specify a name server address different from the default Photon Cloud name server.

§ disconnect()

```
void disconnect ( void )
```

virtual

Disconnects from servers.

§ service()

```
void service ( bool dispatchIncomingCommands = true )
```

virtual

This function executes the PhotonPeer internal processes. Call this regularly!

This function is meant to be called frequently, like once per game loop. It handles the internal calls for keeping the PhotonPeer communication alive, and will take care of sending all local outgoing acknowledgements and messages, as well as dispatching incoming messages to the application and firing the corresponding callbacks. Internally **service()** calls the following functions:

1. **serviceBasic()**
2. **dispatchIncomingCommands()** (called withing a loop until all incoming commands have been dispatched.)
3. **sendOutgoingCommands()** (called withing a loop until everything queued for sending has been sent.)

service() is provided for convenience. If you need to tweak the performance, you can ignore **service()** and call its three subfunctions directly with individual time intervals, to gain more control over the internal communication process. For instance, calling **sendOutgoingCommands()** more rarely will result in less packets to be generated, as more commands will be accumulated into a single packet. See **sendOutgoingCommands()** for more information on efficiency.

For situations where you want to keep the connection alive, but can't process incoming messages (e.g. when loading a level), you can temporarily pass false for **dispatchIncomingCommands** to skip the calls to **dispatchIncomingCommands()**. Incoming commands will be stored in the incoming queue until they are dispatched again.

Parameters

dispatchIncomingCommands true = **dispatchIncomingCommands()**

will be called; false =
dispatchIncomingCommands()
won't be called, default is true

§ serviceBasic()

```
void serviceBasic ( void )
```

virtual

This function takes care of exchanging data with the system's network layer.

You only need to call this function in case you choose not to use `service()`, but call the subfunctions of `service()` directly. Please see the documentation of `service()` for more information.

`serviceBasic()` is called from within `service()`. If you decide not to use `service()`, then `serviceBasic()` needs to be called frequently, like once per game loop.

See also
`service()`

§ sendOutgoingCommands()

```
bool sendOutgoingCommands ( void )
```

virtual

This function initiates the transmission of outgoing commands.

Any Photon function that generates messages will store these messages as a "command" in an outgoing queue for later transmission. Commands can either be explicitly created operations generated for example by `opCustom()` or internally generated messages like acknowledgements for reliable messages from other players. **sendOutgoingCommands()** will initiate the data transmission by passing the outgoing commands to the system's sockets for immediate transmission.

In case of UDP **sendOutgoingCommands()** will also split the commands into multiple packets if needed and/of aggregate multiple commands together into one packet, if possible. Because of the latter calling `sendOutgoingCommands()` more rarely will result in less overhead, as there will be fewer packets for the clients to be sent and processed. The underlying platform can also limit the frequency in which outgoing packets can be sent and received. The downside of lower sending frequencies is a higher latency, until messages are exchanged and acknowledged, which may lead to a jerky gameplay.

To help you keeping track of the incoming and outgoing queues at development time and adjust your sending frequency, there will be a warning message sent to your `debugReturn` callback if a queue has exceeded the warning threshold.

Note

While **service()** is calling **serviceBasic()** implicitly, you will have to regularly call it yourself explicitly, when you use **sendOutgoingCommands()** and **dispatchIncomingCommands()** directly instead.

Usually you don't have to call **sendOutgoingCommands()** this explicitly, as this is done within **service()**.

See also
[service\(\)](#)

§ sendAcksOnly()

```
bool sendAcksOnly ( void )
```

virtual

Sends only ACKs (UDP) or Ping (TCP) instead of queued outgoing commands. Useful to pause sending actual data.

Note

While `service()` is calling `serviceBasic()` implicitly, you will have to regularly call it yourself explicitly, when you use `sendAcksOnly()` and `dispatchIncomingCommands()` instead.

§ dispatchIncomingCommands()

```
bool dispatchIncomingCommands ( void )
```

virtual

Checks for incoming commands waiting in the queue, and dispatches a single command to the application.

Dispatching means, that if the command is an operation response or an event, the appropriate callback function will be called).

dispatchIncomingCommands() will also take care of generating and queuing acknowledgments for incoming reliable commands. Please note that this function will only dispatch one command per all. If you want to dispatch every single command which is waiting in the queue, call `dispatchIncomingCommands()` within a while loop, until its return code is false.

Note

While **service()** is calling **serviceBasic()** implicitly, you will have to regularly call it yourself explicitly, when you use **sendOutgoingCommands()** and **dispatchIncomingCommands()** directly instead.

Returns

true if it has successfully dispatched a command, false otherwise (for example, when there has not been any command left in the queue, waiting for dispatching).

See also

service()

§ fetchServerTimestamp()

```
void fetchServerTimestamp ( void )
```

virtual

This will fetch the server's timestamp and update the approximation for **getServerTime()** and **getServerTimeOffset()**.

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 roundtriptime is low and the timestamp + roundtriptime / 2 is close to the original value.

§ resetTrafficStats()

```
void resetTrafficStats ( void )
```

virtual

Creates new instances of TrafficStats and starts a new timer for those.

§ resetTrafficStatsMaximumCounters()

```
void resetTrafficStatsMaximumCounters ( void )
```

virtual

Resets traffic stats values that can be maxed out.

§ vitalStatsToString()

Common::JString vitalStatsToString (bool **all**) const

virtual

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.

Parameters

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

Returns

stats as a string.

§ opSubscribe()

```
bool  
opSubscribe ( const Common::JVector< Common::JString > & channels  
             int messagesFromHistory  
             )
```

Sends a request to subscribe the client to the specified channels, optionally newer than a specific ID.

Parameters

channels list of channels to subscribe to.
messagesFromHistory 0: no history. 1 and higher: number of messages from history.

Returns

true if request sent

§ opUnsubscribe()

```
bool  
opUnsubscribe ( const Common::JVector< Common::JString > & cha
```

Unsubscribes the client from a list of channels.

The client will remove these channels from the PublicChannels dictionary immediately, if it could send the operation.

Parameters

channels list of channels to unsubscribe from.

Returns

true if request sent and channels removed

§ opPublishMessage() [1/3]

```
template< typename Ftype  
> bool opPublishMessage ( const Common::JString & channelName,  
                           const Ftype & message  
                           )
```

Sends a message to the specified public channel.

Parameters

channelName channel name
message message to send

Returns

false in case of an error, true otherwise

§ opPublishMessage() [2/3]

```
template<
typename Ftype >
bool
opPublishMessage ( const Common::JString &
                  const Ftype
                  typename Common::Helpers::ArrayLengthType< Fty
                  )
```

This is an overloaded member function, provided for convenience. It differs only in what argument(s) it accepts.

Parameters

channelName channel name
pMessageArray message to send
arrSize the number of elements in pParameterArray

§ opPublishMessage() [3/3]

```
template< typename
Ftype > bool
opPublishMessage      ( const Common::JString & channelName,
                       const Ftype           pMessageArray,
                       const short *        pArrSizes
                       )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- channelName** channel name
- pMessageArray** message to send
- pArrSizes** an array holding the number of elements for each dimension of pParameterArray

§ opSendMessage() [1/2]

```
template< typename
Ftype > bool
opSendMessage ( const Common::JString &
                const Ftype
                typename Common::Helpers::ArrayLengthType
                bool
                )
```

This is an overloaded member function, provided for convenience. It differs in what argument(s) it accepts.

Parameters

userName	user name
pMessageArray	message to send
arrSize	the number of elements in pParameterArray
encrypt	true to send the message encrypted, false (default)

§ opSendMessage() [2/2]

```
template< typename
Ftype > bool
opSendMessage ( const Common::JString & userName,
                const Ftype           pMessageArray,
                const short *         pArrSizes,
                bool                   encrypt = false
              )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

userName	user name
pMessageArray	message to send
pArrSizes	an array holding the number of elements for each dimension of pParameterArray
encrypt	true to send the message encrypted, false (default) to send it unencrypted

§ opSetOnlineStatus() [1/3]

```
bool opSetOnlineStatus ( int status )
```

virtual

Sets the user's status (pre-defined or custom) and a status message.

The predefined status values can be found in namespace **UserStatus**. States **UserStatus::INVISIBLE** and **UserStatus::OFFLINE** will make you offline for everyone and send no message.

Parameters

status predefined states are in namespace **UserStatus**. Other values can be used at will

Returns

false in case of an error, true otherwise

§ opSetOnlineStatus() [2/3]

```
template<
typename Ftype >
bool
opSetOnlineStatus ( int
                    const Ftype
                    typename Common::Helpers::ArrayLengthType< Fty
                    )
```

This is an overloaded member function, provided for convenience. It differs only in what argument(s) it accepts.

Parameters

- status** predefined states are in namespace **UserStatus**. (at will)
- pMessageArray** optional status message
- arrSize** the number of elements in pParameterArray

§ opSetOnlineStatus() [3/3]

```
template< typename Ftype > bool  
opSetOnlineStatus          ( int          status,  
                             const Ftype  pMessageArray,  
                             const short * pArrSizes  
                             )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- status** predefined states are in namespace **UserStatus**. Other values can be used at will
- pMessageArray** optional status message
- pArrSizes** an array holding the number of elements for each dimension of pParameterArray

§ opAddFriends()

bool

```
opAddFriends ( const Common::JVector< Common::JString > & userIDs
```

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

Parameters

userIDs list of friend user names

Returns

true if the command has been sent

§ opRemoveFriends()

```
bool  
opRemoveFriends ( const Common::JVector< Common::JString > & u
```

Removes users from the list on the **Chat** Server which will send you stat those.

Parameters

userIDs list of friend user names

Returns

true if the command has been sent

§ getServerTimeOffset()

```
int getServerTimeOffset ( void ) const
```

Returns

the difference between the local uptime and the Photon Server's system time in ms.

In real-time games it's often useful to relate game events to a global common timeline, that's valid for all players and independent from derivations throughout the clients' system times. The Photon Server's System Time can serve as this reference time. The `serverTimeOffset` represents the difference between the client's local system time and the Photon server's system time.

`ServerTime = serverTimeOffset + GETTIMEMS()`

The `serverTimeOffset` is fetched shortly after connect by **Photon**. Use `GETTIMEMS()` to get your local time in ms. You can let Photon refetch the offset by calling **`fetchServerTimestamp()`**. The `ServerTimeOffset` will be 0 until shortly after initial connect.

§ `getServerTime()`

```
int getServerTime ( void ) const
```

Returns

the Photon Server's system time ins ms.

see [getServerTimeOffset\(\)](#)

§ `getBytesOut()`

```
int getBytesOut ( void ) const
```

Returns

the total number of outgoing bytes transmitted by this PhotonPeer object.

See also

[getBytesIn\(\)](#)

§ `getBytesIn()`

```
int getBytesIn ( void ) const
```

Returns

the total number of incoming bytes received by this PhotonPeer object.

See also

[getBytesOut\(\)](#)

§ `getByteCountCurrentDispatch()`

```
int getByteCountCurrentDispatch ( void ) const
```

Returns

the size of the dispatched event or operation-result in bytes. This value is set before `onEvent()` or `onOperationResponse()` is called (within **`dispatchIncomingCommands()`**). Get this value directly in `onEvent()` or `onOperationResponse()`.

§ getByteCountLastOperation()

```
int getByteCountLastOperation ( void ) const
```

Returns

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. Get this value immediately after calling an operation.

§ `getSentCountAllowance()`

```
int getSentCountAllowance ( void ) const
```

Returns

the number of resend retries before a peer is considered lost/disconnected.

This is udp specific and will always return 0 for other protocols.

See also

[setSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ setSentCountAllowance()

```
void setSentCountAllowance ( int sentCountAllowance )
```

Sets the number of re-send retries before a peer is considered lost/disconnected.

This is udp specific and will do nothing at all for other protocols.

Parameters

sentCountAllowance the new number of re/-send retries before a peer is considered lost/disconnected.

See also

[getSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ getTimePingInterval()

```
int getTimePingInterval ( void ) const
```

Returns

the time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[setTimePingInterval\(\)](#)

§ setTimePingInterval()

```
void setTimePingInterval ( int timePingInterval )
```

Sets the time threshold in milliseconds since the last reliable command, before a ping will be sent.

Parameters

timePingInterval time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[getTimePingInterval\(\)](#)

§ getRoundTripTime()

```
int getRoundTripTime ( void ) const
```

Returns

the time in milliseconds until a reliable command is acknowledged by the server.

This is, what is commonly called a ping time or just a ping.

See also

[getRoundTripTimeVariance\(\)](#)

§ getRoundTripTimeVariance()

```
int getRoundTripTimeVariance ( void ) const
```

Returns

the variance of the roundtrip time in milliseconds. Gives a hint about how much the net latency is varying.

See also

[getRoundTripTime\(\)](#)

§ getTimestampOfLastSocketReceive()

```
int getTimestampOfLastSocketReceive ( void ) const
```

Returns

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.

§ `getDebugOutputLevel()`

```
int getDebugOutputLevel ( void ) const
```

Returns the current level of debug information that's passed on to [BaseListener::debugReturn\(\)](#).

Returns

one of the values in `DebugLevel`

See also

[setDebugOutputLevel\(\)](#)

§ `setDebugOutputLevel()`

```
bool setDebugOutputLevel ( int debugLevel )
```

Sets the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Parameters

debugLevel one of the values in `DebugLevel`

Returns

true if the new debug level has been set correctly, false otherwise.

See also

`getDebugOutputLevel()`

§ getLogFormatOptions()

```
const LogFormatOptions & getLogFormatOptions ( void ) const
```

Returns

the LogFormatOptions that are used by this instance.

See also

setFormatOptions()

§ setLogFormatOptions()

```
void  
setLogFormatOptions ( const Common::LogFormatOptions & formatO
```

Sets the log format options to the supplied value.

Parameters

formatOptions the new value to which the log format options will be

See also

getFormatOptions()

§ getIncomingReliableCommandsCount()

```
int getIncomingReliableCommandsCount ( void ) const
```

Returns

the total number of reliable commands currently waiting in the incoming queues of all channels or -1 if not connected.

§ getPeerID()

```
short getPeerID ( void ) const
```

Returns

this peer's ID as assigned by the server. Will be -1, if not connected.

§ getDisconnectTimeout()

```
int getDisconnectTimeout ( void ) const
```

Returns

the maximum time interval in milliseconds for doing resend retries before a peer is considered lost/disconnected.

See also

[setDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ setDisconnectTimeout()

```
void setDisconnectTimeout ( int disconnectTimeout )
```

Sets the maximum time in milliseconds for making re-send retries before a peer is considered lost/disconnected.

Parameters

disconnectTimeout resend max time in ms before a peer is considered lost/disconnected

See also

[getDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ getQueuedIncomingCommands()

```
int getQueuedIncomingCommands ( void ) const
```

Returns

the number of queued incoming commands in all channels or -1 if not connected

§ getQueuedOutgoingCommands()

```
int getQueuedOutgoingCommands ( void ) const
```

Returns

the number of queued outgoing commands in all channels or -1 if not connected

§ `getIsPayloadEncryptionAvailable()`

```
bool getIsPayloadEncryptionAvailable ( void ) const
```

Returns

this peer's encryption availability status. True if either payload encryption is available or if the connection protocol is UDP and UDP encryption is available or if the connection protocol is already secure on its own, false otherwise.

See also

[getIsPayloadEncryptionAvailable\(\)](#), `establishEncryption()`, `initUserDataEncryption()`, `initUDPEncryption()`

§ getResentReliableCommands()

```
int getResentReliableCommands ( void ) const
```

Returns

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

§ getLimitOfUnreliableCommands()

```
int getLimitOfUnreliableCommands ( void ) const
```

Returns

the limit for the queue of received unreliable commands.

See also

[setLimitOfUnreliableCommands\(\)](#)

§ setLimitOfUnreliableCommands()

```
void setLimitOfUnreliableCommands ( int value )
```

Sets the limit for the queue of received unreliable commands. This works only in UDP. This limit is applied when you call `dispatchIncomingCommands`. If this client (already) received more than this limit, 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. 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 commands up to specified limit. This value interacts with `dispatchIncomingCommands()`: If that is called less often, more commands get skipped.

See also

[getLimitOfUnreliableCommands\(\)](#)

§ getCRCEnabled()

```
bool getCRCEnabled ( void ) const
```

Returns

true if CRC enabled

See also

[setCRCEnabled](#)

§ setCRCEnabled()

```
void setCRCEnabled ( bool crcEnabled )
```

Enables or disables CRC. While not connected, this controls if the next connection(s) should use a per-package CRC checksum. If the client is in another state than 'connected', then this function has no effect except for logging an error.

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.

Note

This only has effect for UDP connections.

This does not have any effect for connections that use UDP datagram encryption (which always use a built-in checksum).

See also

[getCRCEnabled](#)

§ `getPacketLossByCRC()`

```
int getPacketLossByCRC ( void ) const
```

Returns

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

See also

[`setCRCEnabled`](#)

§ getTrafficStatsEnabled()

```
bool getTrafficStatsEnabled ( void ) const
```

Returns

true if traffic statistics of a peer are enabled. Default trafficStatsEnabled: false (disabled).

§ setTrafficStatsEnabled()

```
void setTrafficStatsEnabled ( bool trafficStatsEnabled )
```

Enables or disables the traffic statistics of a peer. Default trafficStatsEnabled: false (disabled).

§ getTrafficStatsElapsedMs()

```
int getTrafficStatsElapsedMs ( void ) const
```

Returns

the count of milliseconds the stats are enabled for tracking.

§ getTrafficStatsIncoming()

```
const Photon::TrafficStats & getTrafficStatsIncoming ( void ) const
```

Returns

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.

§ getTrafficStatsOutgoing()

```
const Photon::TrafficStats & getTrafficStatsOutgoing ( void ) const
```

Returns

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.

§ getTrafficStatsGameLevel()

```
const Photon::TrafficStatsGameLevel &  
getTrafficStatsGameLevel (void ) const
```

Returns

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).

§ getQuickResendAttempts()

```
nByte getQuickResendAttempts ( void ) const
```

Returns

the number of resend attempts for a reliable command that are done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

§ setQuickResendAttempts()

```
void setQuickResendAttempts ( nByte quickResendAttempts )
```

Returns

the number of resend attempts for a reliable command that are done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

§ getChannelCountUserChannels()

```
nByte getChannelCountUserChannels ( void ) const
```

The IDs from 0 to **getChannelCountUserChannels()**-1 can be passed as channelID to operations that offer this parameter.

Returns

the number of different channels that are available for sending operations on.

§ getPeerCount()

```
short getPeerCount ( void )
```

static

Returns

the count of peers, which have been initialized since the start of the application. Interesting mainly for debugging purposes.

§ getUserID()

```
const JString & getUserID ( void ) const
```

Returns the unique user id.

Returns
the user id

§ getState()

```
int getState ( void ) const
```

Returns client state

§ getDisconnectedCause()

```
int getDisconnectedCause ( void ) const
```

Returns cause of last disconnect event.

Returns

disconnect cause constant from [Chat::DisconnectCause](#).

See also

[Chat::DisconnectCause](#)

§ getRegion()

```
const JString & getRegion ( void ) const
```

Returns chat (Name Server) region.

§ setRegion()

```
void setRegion ( const Common::JString & region )
```

Sets chat (Name Server) region. Set it before **connect()** call.

Parameters

region region

§ getPublicChannels()

```
const JVector< Channel * > & getPublicChannels ( void ) const
```

Returns list of subscribed public channels.

Returns

list of subscribed channels

§ getPrivateChannels()

```
const JVector< Channel * > & getPrivateChannels ( void ) const
```

Returns list of private chats that client currently has.

Returns

list of private chats

§ getPublicChannel()

```
const Channel *  
getPublicChannel ( const Common::JString & channelName ) const
```

Search subscribed public channels by channel name.

Parameters

channelName channel name to search

Returns

found channel or NULL otherwise

§ getPrivateChannel()

```
const Channel *  
getPrivateChannel ( const Common::JString & userName ) const
```

Search private chat by user name.

Parameters

userName user name to search

Returns

found chat or NULL otherwise

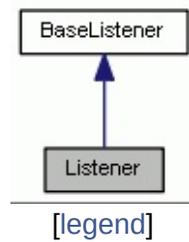
© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

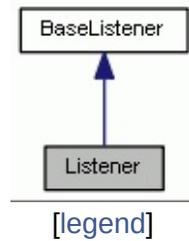
Listener Class

Reference **abstract**

Inheritance diagram for Listener:



Collaboration diagram for Listener:



Public Member Functions

virtual void **debugReturn** (int debugLevel, const **Common::JString** &string)=0

virtual void **onStateChange** (int state)=0

virtual void **connectionErrorReturn** (int errorCode)=0

virtual void **clientErrorReturn** (int errorCode)=0

virtual void **warningReturn** (int warningCode)=0

virtual void **serverErrorReturn** (int errorCode)=0

virtual void **connectReturn** (int errorCode, const **Common::JString** &errorString)=0

virtual void **disconnectReturn** (void)=0

virtual void **subscribeReturn** (const **Common::JVector**<
Common::JString > &channels, const
Common::JVector< bool > &results)=0

virtual void **unsubscribeReturn** (const **Common::JVector**<
Common::JString > &channels)=0

virtual void **onStatusUpdate** (const **Common::JString** &user, int
status, bool gotMessage, const **Common::Object**
&message)=0

virtual void **onGetMessages** (const **Common::JString**
&channelName, const **Common::JVector**<
Common::JString > &senders, const **Common::JVector**<
Common::Object > &messages)=0

virtual void **onPrivateMessage** (const **Common::JString** &sender,

```
const Common::Object &message, const  
Common::JString &channelName)=0
```

Detailed Description

Callback interface for **Chat** client side. Contains callback methods to notify your app about updates. Must be provided to new **Chat::Client** in constructor

Member Function Documentation

§ debugReturn()

```
virtual void  
debugReturn      ( int                debugLevel,  
                  const Common::JString & string  
                  )
```

pure virtual

This is the callback function for debug-messages.

Parameters

debugLevel one of the values in DebugLevel

string the formatted debug string

See also

BaseListener

Implements **BaseListener**.

§ onStateChange()

```
virtual void onStateChange ( int state )
```

pure virtual

Notifies app that client state changed.

Parameters

state new client state

See also

ClientState::ClientState

§ connectReturn()

```
virtual void  
connectReturn    ( int                errorCode,  
                  const Common::JString & errorString  
                  )
```

pure virtual

Client is connected now.

§ disconnectReturn()

```
virtual void disconnectReturn ( void )
```

pure virtual

Disconnection happened.

§ subscribeReturn()

```
virtual void  
subscribeReturn ( const Common::JVector< Common::JString > & ch  
                  const Common::JVector< bool > & res  
                  )
```

The result of the subscribe operation. Returns per channel name if the client is subscribed.

Parameters

channels channel names

results per channel result: true if subscribed

§ unsubscribeReturn()

```
virtual void  
unsubscribeReturn ( const Common::JVector< Common::JString > &
```

Result of unsubscribe operation. Returns per channel name if the channel

Parameters

channels channel names that are no longer subscribed

§ onStatusUpdate()

```
virtual void  
onStatusUpdate ( const Common::JString & user,  
                 int status,  
                 bool gotMessage,  
                 const Common::Object & message  
               ) pure virtual
```

The new status of another user (you get updates for users that are in your friends list).

Parameters

- user** name of the user
- status** new status of that user
- gotMessage** true if the status contains a message you should cache locally. False: This status update does not include a message (keep any you have).
- message** message that user set

§ onGetMessages()

```
virtual void  
onGetMessages ( const Common::JString & channelName, ch  
                const Common::JVector< Common::JString > & senders, se  
                const Common::JVector< Common::Object > & messages, me  
                )
```

Notifies the app that the client got new messages from the server. Number of messages in 'messages'. Sender with number '0' corresponds to number '0', sender with number '1' corresponds to message with number

Parameters

- channelName** channel from where messages came
- senders** list of users who sent messages
- messages** list of messages it self

§ onPrivateMessage()

```
virtual void  
onPrivateMessage ( const Common::JString & sender,  
                  const Common::Object & message,  
                  const Common::JString & channelName  
                  )
```

pure virtu:

Notifies the app about a private message

Parameters

- sender** user who sent this message
- message** the message itself
- channelName** the channel name for private messages (messages that you sent yourself get added to a channel per target username)

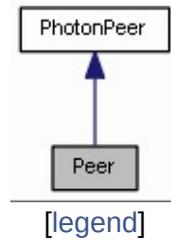
Photon C++ Client API 4.1.12.2

ExitGames > Chat > Peer >

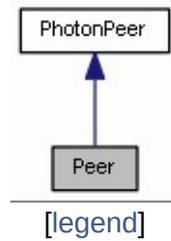
[Public Member Functions](#) | [List of all members](#)

Peer Class Reference

Inheritance diagram for Peer:



Collaboration diagram for Peer:



Public Member Functions

	Peer (Photon::PhotonListene connectionProtocol=Photon::Cc
virtual bool	opAuthenticateOnNameServe &appId, const Common::JStri Common::JString ®ion, co &authenticationValues= Authen
virtual bool	opAuthenticateOnFrontEnd (&secret)
virtual bool	opSubscribe (const Common: Common::JString > &channel
virtual bool	opUnsubscribe (const Comm Common::JString > &channel
template<typename Ftype >	bool opPublishMessage (const Co &channelName, const Ftype &r
template<typename Ftype >	bool opPublishMessage (const Co &channelName, const Ftype pM Common::Helpers::ArrayLength
template<typename Ftype >	bool opPublishMessage (const Co &channelName, const Ftype pM *pArrSizes)
template<typename Ftype >	bool opSendPrivateMessage (cons &userName, const Ftype &mes

template<typename Ftype >

bool **opSendPrivateMessage** (const &userName, const Ftype pMes:
Common::Helpers::ArrayLength
bool encrypt=false)

template<typename Ftype >

bool **opSendPrivateMessage** (const &userName, const Ftype pMes:
*pArrSizes, bool encrypt=false)

virtual bool **opSetOnlineStatus** (int status)

template<typename Ftype >

bool **opSetOnlineStatus** (int status,

template<typename Ftype >

bool **opSetOnlineStatus** (int status,
typename Common::Helpers::A
>::type arrSize)

template<typename Ftype >

bool **opSetOnlineStatus** (int status,
const short *pArrSizes)

virtual bool **opAddFriends** (const **Commo**
Common::JString > &userIDs,

virtual bool **opRemoveFriends** (const **Con**
Common::JString > &userIDs,

► Public Member Functions inherited from **PhotonPeer**

PhotonPeer (**PhotonListener**
connectionProtocol=Connector

virtual **~PhotonPeer** (void)

virtual bool **connect** (const **Common::JString** &appId=**Common::JString**)

template<typename Ftype >

bool **connect** (const **Common::JString** &appId, const **Common::JString**)

template<typename Ftype >

bool **connect** (const **Common::JString** &appId, const **Common::JString**, const typename **Common::Helpers::Array::type** arrSize)

template<typename Ftype >

bool **connect** (const **Common::JString** &appId, const **Common::JString**, const short *pArrSizes)

virtual void **disconnect** (void)

virtual void **service** (bool **dispatchIncoming**)

virtual void **serviceBasic** (void)

virtual bool **opCustom** (const **OperationRequest**, bool sendReliable, nByte char)

virtual bool **sendOutgoingCommands** (void)

virtual bool **sendAcksOnly** (void)

virtual bool **dispatchIncomingCommands** (void)

virtual bool **establishEncryption** (void)

virtual void **fetchServerTimestamp** (void)

virtual void **resetTrafficStats** (void)

virtual void **resetTrafficStatsMaximumCo**

virtual **Common::JString** **vitalStatsToString** (bool all) co

virtual void **pingServer** (const **Common::J**
int pingAttempts)

virtual void **initUserDataEncryption** (cons
> &secret)

virtual void **initUDPEncryption** (const **Cor**
&encryptSecret, const **Commo**
&HMACSecret)

PhotonListener * **getListener** (void)

int **getServerTimeOffset** (void) co

int **getServerTime** (void) const

int **getBytesOut** (void) const

int **getBytesIn** (void) const

int **getByteCountCurrentDispatc**

int **getByteCountLastOperation** (

int **getPeerState** (void) const

int **getSentCountAllowance** (void

void **setSentCountAllowance** (int s

int **getTimePingInterval** (void) cor

	void	setTimePingInterval	(int timeP
	int	getRoundTripTime	(void) cons
	int	getRoundTripTimeVariance	(\
	int	getTimestampOfLastSocketR	
	int	getDebugOutputLevel	(void) c
	bool	setDebugOutputLevel	(int dek
const Common::LogFormatOptions &		getLogFormatOptions	(void) c
	void	setLogFormatOptions	(const Common::LogFormatOptions
	int	getIncomingReliableCommar	
	short	getPeerID	(void) const
	int	getDisconnectTimeout	(void)
	void	setDisconnectTimeout	(int dis
	int	getQueuedIncomingComman	
	int	getQueuedOutgoingComman	
Common::JString		getServerAddress	(void) cons
	bool	getIsPayloadEncryptionAvail.	
	bool	getIsEncryptionAvailable	(voi

	int	getResentReliableCommands
	int	getLimitOfUnreliableCommar
	void	setLimitOfUnreliableCommar
	bool	getCRCEnabled (void) const
	void	setCRCEnabled (bool crcEnab
	int	getPacketLossByCRC (void) c
	bool	getTrafficStatsEnabled (void)
	void	setTrafficStatsEnabled (bool t
	int	getTrafficStatsElapsedMs (vo
	const TrafficStats &	getTrafficStatsIncoming (void
	const TrafficStats &	getTrafficStatsOutgoing (void
	const TrafficStatsGameLevel &	getTrafficStatsGameLevel (vo
	nByte	getQuickResendAttempts (vo
	void	setQuickResendAttempts (nE
	nByte	getConnectionProtocol (void)
	void	setConnectionProtocol (nByte
	nByte	getChannelCountUserChann

Additional Inherited Members

▶ **Static Public Member Functions inherited from PhotonPeer**

static short **getPeerCount** (void)

static unsigned int **getMaxAppIDLength** (void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

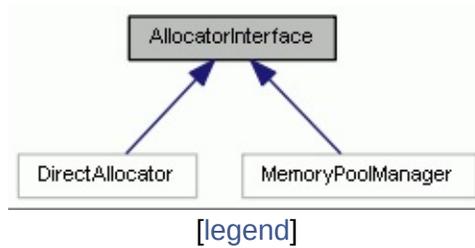
[ExitGames](#)[Common](#)[MemoryManagement](#)[AllocatorInterface](#)

AllocatorInterface

Class Reference **abstract**

[Public Member Functions |](#)[Static Public Member Functions |](#)[List of all members](#)

Inheritance diagram for AllocatorInterface:



Public Member Functions

virtual **~AllocatorInterface** (void)

virtual void **setMaxAllocSize** (size_t maxAllocSize)=0

virtual void * **alloc** (size_t size)=0

virtual void **dealloc** (void *p)=0

virtual void * **resize** (void *p, size_t size)=0

Static Public Member Functions

```
static AllocatorInterface * get (void)
```

Detailed Description

Custom Allocators to be used with Photons Memory Management need to inherit and implement this interface. The allocator that is used by Photon can be set through **setAllocator()**.

Constructor & Destructor Documentation

§ ~AllocatorInterface()

```
virtual ~AllocatorInterface ( void )
```

virtual

Destructor.

Member Function Documentation

§ setMaxAllocSize()

```
virtual void setMaxAllocSize ( size_t maxAllocSize )
```

pure virtual

This function gets called by **MemoryManagement::setMaxAllocSize()** and an implementation is required to behave as explained in the documentation of that function.

§ alloc()

```
virtual void* alloc ( size_t size )
```

pure virtual

This function gets called by **EG_MALLOC** and an implementation is required to behave as explained in the documentation of that macro.

§ dealloc()

```
virtual void dealloc ( void * p )
```

pure virtual

This function gets called by **EG_FREE** and an implementation is required to behave as explained in the documentation of that macro.

§ `resize()`

```
virtual void* resize ( void * p,  
                      size_t size  
                      )
```

pure virtual

This function gets called by **EG_REALLOC** and an implementation is required to behave as explained in the documentation of that macro.

§ get()

```
static AllocatorInterface* get ( void )
```

This function gets called by Photon exactly once in the lifetime of the app right before the very first allocation by **Photon** is made. The Allocator that is returned by this function will be used for all allocations by Photon until you set a different allocator through **setAllocator()**.

Calling **setAllocator()** right in the first line of main() is already too late to guarantee that every single allocation by Photon will use your custom allocator because global and file-level static variables and constants (referred to here simply as 'globals') will be created before the program execution enters main. If those globals are not POD-types, then they might allocate memory upon initialization and in case of classes that are provided by one of the Photon libs, such allocations will happen through Photon's memory management. Hence static allocations need to already use an allocator before the program enters main.

The way to set an allocator that is used for allocations by such globals, is to replace the default implementation of this function by your own implementation. This works in the same way like replacing the platform's default implementation of the global new and delete operators with your own implementations: Photon provides a default implementation of this function that gets used when you do not provide your own implementation, but when you do provide your own implementation, then the linker silently drops Photon's weak-linked default implementation and replaces Photon's call to it by a call to your implementation.

Usage example:

```
class Allocator : public
    ExitGames::Common::MemoryManagement::AllocatorInt
{
public:
    Allocator(void)
        : mCountAllocs(0)
        , mCountFrees(0)
    {
```

```

    }

    virtual ~Allocator(void)
    {
    }

    virtual void setMaxAllocSize(size_t maxAllocSize)
    {
    }

    virtual void* alloc(size_t size)
    {
    return malloc(size);
    }

    virtual void dealloc(void* p)
    {
        free(p);
    }

    virtual void* resize(void* p, size_t size)
    {
    return realloc(p, size);
    }
private:
    static void* operator new(size_t);
    static void* operator new[](size_t);
};

namespace ExitGames
{
    namespace Common
    {
        namespace MemoryManagement
        {
            AllocatorInterface*
            AllocatorInterface::get(void)

```



```

    ExitGames::Common::MemoryManagement::AllocatorInt
{
public:
    ReferenceCountedAllocator(void)
        : mRefCount(0)
    {
        retain();
    }

    ReferenceCountedAllocator*
    ReferenceCountedAllocator::retain(void)
    {
        std::lock_guard<std::mutex> lock(mMutex);
        ++mRefCount;
    return this;
    }

    void ReferenceCountedAllocator::release(void)
    {
        std::unique_lock<std::mutex> lock(mMutex);
    if(!--mRefCount)
        {
            lock.unlock();
        delete this;
        }
    }

    virtual void setMaxAllocSize(size_t maxAllocSize)
    {
    }

    virtual void* alloc(size_t size)
    {
    if(!size)
    return NULL;
        retain();
    return malloc(size);
    }
}

```

```

    }

    virtual void dealloc(void* p)
    {
    if(!p)
    return;
        free(p);
        release();
    }

    virtual void* resize(void* p, size_t size)
    {
    return realloc(p, size);
    }
private:
    virtual ~ReferenceCountedAllocator(void)
    {
    }

    long long mRefCount;
    std::mutex mMutex;
};

```

```

void foo(void)
{
    ReferenceCountedAllocator* pAllocator = new
    ReferenceCountedAllocator;
    ExitGames::Common::MemoryManagement::setAllocator(*p
    tor);
    ExitGames::Common::JString string = L"samplestring";
    ExitGames::Common::MemoryManagement::setAllocatorToD
    ();
    pAllocator->release();
    // some more code
} // only at this point, when the local JString varia
    'string' gets out of scope and hence destructed,
    hands its memory back to the allocator and pAlloc
    reference count reaches 0 so that it gets deletec

```

Note that ReferenceCountedAllocator makes its destructor private to ensure it only ever gets called by release(). A side-effect of this is that one can't call it in `AllocatorInterface::get()` (at least without leaking it). So there are use cases for both approaches of both example custom allocators: use the approach of class Allocator for a custom allocator that should be returned by `AllocatorInterface::get()`, and use the approach of ReferenceCountedAllocator for a custom allocator that should be able to have a limited lifetime.

Furthermore note that your custom allocator must be thread-safe (which is not the case for ReferenceCountedAllocator, if it would not protect mFree with a lock), as **Photon** might access it from multiple threads at once.

Finally if for some reason you don't want any allocations on the heap to happen while global and file level static variables are getting constructed, remember that it is completely up to you where the memory that you provide to Photon is from and how it's managed and that you can provide different allocators at different times. Hence the allocator that you return by `AllocatorInterface::get()` could look like this:

```
class Allocator : public
ExitGames::Common::MemoryManagement::AllocatorInterface { public
Allocator(void) : mCountBytes(0) { }
```

```
virtual ~Allocator(void) { }
```

```
virtual void setMaxAllocSize(size_t maxAllocSize) { }
```

```
virtual void* alloc(size_t size) { static const size_t MEM_SIZE = 16*1024;
byte memory[MEM_SIZE]; mCountBytes += size; if(mCountBytes > MEM_SIZE)
assert(false); return memory+mCountBytes-size; }
```

```
virtual void dealloc(void* p) { }
```

```
virtual void* resize(void* p, size_t size) { assert(false); return NULL; }
static void* operator new(size_t); static void* operator new[](size_t);
```

```
unsigned long long mCountBytes; }; This variant simply allocates the memory
a static byte-array and does not reuse any memory that is returned to it (
perfectly fine for memory that gets allocated in the constructor and deallocated
the destructor of a variable which has the same lifetime as the executable
```

Note that you need to make sure that the array on which the memory is allocated is big enough to cover all requests that occur until you set a different allocator.

the required amount might change when changes in your code happen or you update to a new Photon version, the `assert()` in `alloc` is important to make it hard to track down crashes in unrelated code.

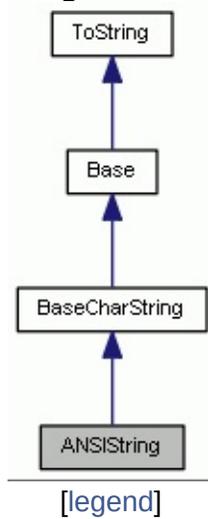
Be aware that this primitive variant that does not reuse any memory only makes sense when you set a different allocator through `setAllocator()` as early as possible because the longer you wait the bigger the static array will need to be to serve all requests without running out of memory.

© Exit Games ®, all rights reserved.

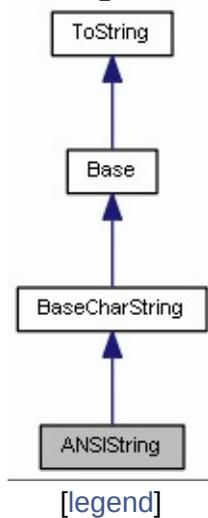
[Photon Documentation](#) | [Contact](#) | [Terms](#)

ANSIStrng Class Reference

Inheritance diagram for ANSIStrng:



Collaboration diagram for ANSIStrng:



Public Member Functions

ANSIString (void)

ANSIString (const **ANSIString** &str)

ANSIString (const **JString** &str)

ANSIString (const char *str)

ANSIString (const EG_CHAR *str)

~ANSIString (void)

ANSIString & **operator=** (const **ANSIString** &Rhs)

ANSIString & **operator=** (const **JString** &Rhs)

ANSIString & **operator=** (const char *Rhs)

ANSIString & **operator=** (const EG_CHAR *Rhs)

operator const char * (void) const

operator JString (void) const

JString **JStringRepresentation** (void) const

unsigned int **size** (void) const

► Public Member Functions inherited from **BaseCharString**

BaseCharString ()

virtual **~BaseCharString** (void)

const char * **cstr** (void) const

unsigned int **length** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual ~**Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual ~**ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **ANSIString** class is a container class for char* strings, encoded with the current locale.

This is the current locale implementation of **BaseCharString**. Please look at the doc of the abstract base class for more information.

Constructor & Destructor Documentation

§ ANSIStrIng() [1/5]

ANSIStrIng (void)

Constructor: Creates an empty **ANSIStrIng**.

§ ANSIStrng() [2/5]

ANSIStrng (const **ANSIStrng** & **str**)

Copy-Constructor: Creates a new **ANSIStrng** from a deep copy of the argument string.

Parameters

str The **ANSIStrng** string to copy.

§ ANSIStrng() [3/5]

ANSIStrng (const **JString** & *wstr*)

Copy-Constructor: Creates a new **ANSIStrng** from a deep copy of the argument string.

Parameters

wstr The **JString** string to copy.

§ ANSIString() [4/5]

ANSIString (const char * **str**)

Copy-Constructor: Creates a new **ANSIString** from a deep copy of the argument string.

Parameters

str The ANSI string to copy.

§ ANSIString() [5/5]

ANSIString (const EG_CHAR * **wstr**)

Copy-Constructor: Creates a new **ANSIString** from a deep copy of the argument string.

Parameters

wstr The Unicode String string to copy.

§ ~ANSIString()

~ANSIString (void)

Destructor.

Member Function Documentation

§ operator=() [1/4]

```
ANSIString & operator= ( const ANSIString & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [2/4]

```
ANSIString & operator= ( const JString & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [3/4]

```
ANSIString & operator= ( const char * Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [4/4]

```
ANSIString & operator= ( const EG_CHAR * Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator const char *()

```
operator const char * ( void ) const
```

virtual

operator const char*.

Copies a pointer to the content of its right operand into its left operand.

This overwrites old data in the left operand.

Implements **BaseCharString**.

§ operator JString()

operator **JString** (void) const

virtual

operator **JString**.

Copies a **JString** representation of its right operand into its left operand.

This overwrites old data in the left operand.

Implements **BaseCharString**.

§ JStringRepresentation()

JString JStringRepresentation (void) const

virtual

Returns

a **JString** representation of the string.

Implements **BaseCharString**.

§ size()

unsigned int size (void) const

virtual

The default implementation of this function will just return `length()`, but for multibyte strings like `UTF8String` the return values of `length()` and `size()` can differ.

Returns

the size of the string in bytes

Implements `BaseCharString`.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

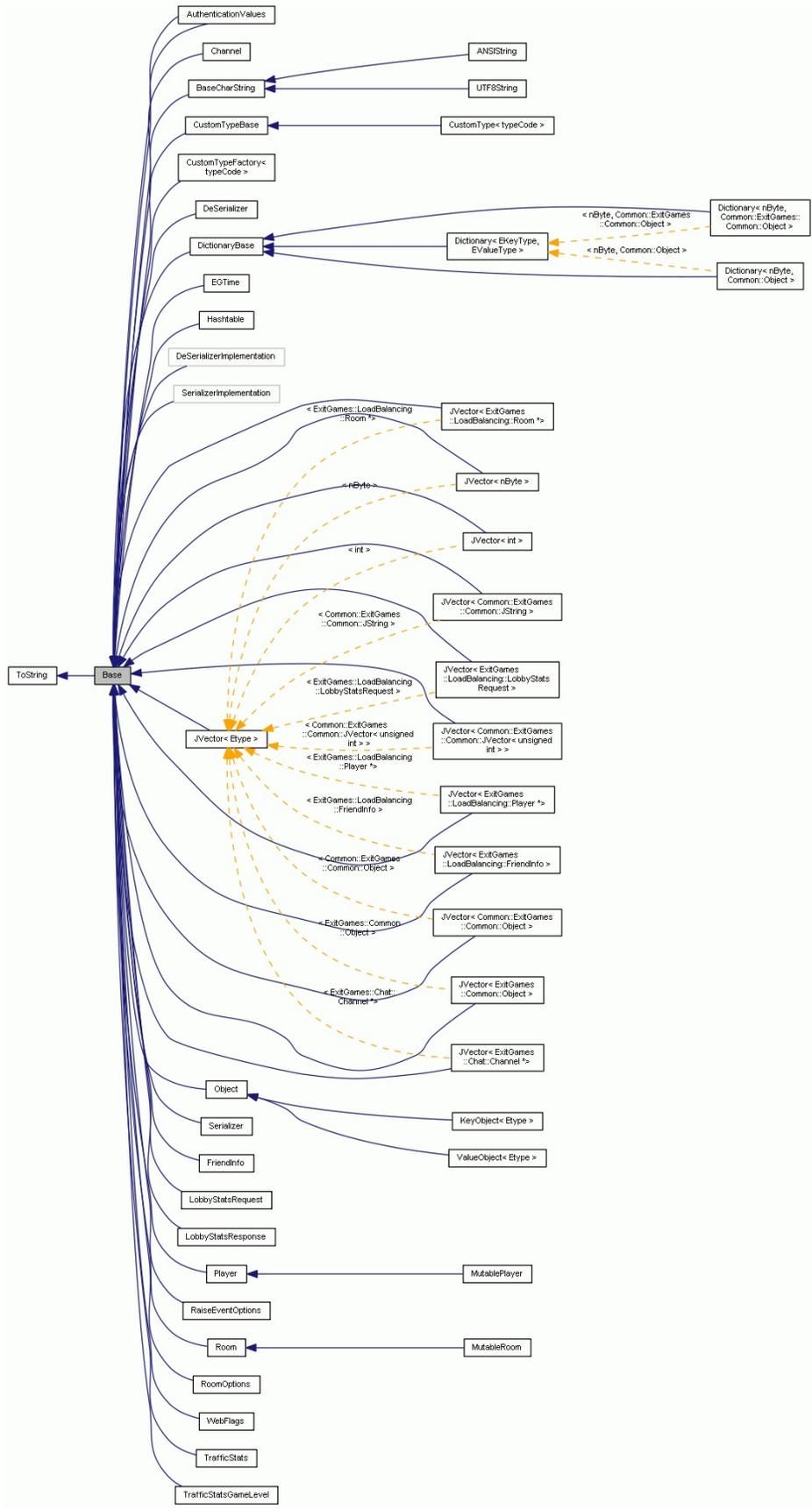
Client API 4.1.12.2

[ExitGames](#)[Common](#)[Base](#)

Base Class Reference

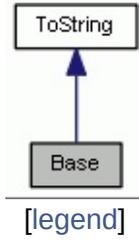
[Public Member Functions](#) |[Static Public Member Functions](#) |[List of all members](#)

Inheritance diagram for Base:



[legend]

Collaboration diagram for Base:



[legend]

Public Member Functions

virtual ~**Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual ~**ToString** (void)

virtual **JString** **typeToString** (void) const

virtual **JString** & **toString** (**JString** &retStr, bool withTypes=false)
const =0

JString **toString** (bool withTypes=false) const

Static Public Member Functions

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This is the base-class of all Utility-classes except of **JString**.

This class provides a common callback interface for transmitting debug messages from all utility classes to your application. Please refer to **setListener()** for more information.

See also

BaseListener , **setListener()**

Constructor & Destructor Documentation

§ ~Base()

~Base (void)

virtual

Destructor.

Member Function Documentation

§ `setListener()`

```
void setListener ( const BaseListener * baseListener )
```

static

Registers a listener for receiving debug information from the Exitgames Utility classes.

Please refer to **BaseListener** for more information and a code example.

Parameters

baseListener The listener, in which you want to receive the the events. Has to be a pointer to a class derived from **BaseListener**.

See also

BaseListener

§ `getDebugOutputLevel()`

```
int getDebugOutputLevel ( void )
```

static

Returns the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Returns

one of the values in `DebugLevel`

See also

`setDebugOutputLevel()`

§ `setDebugOutputLevel()`

```
bool setDebugOutputLevel ( int debugLevel )
```

static

Sets the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Parameters

`debugLevel` one of the values in `DebugLevel`

Returns

true if the new debug level has been set correctly, false otherwise.

See also

`getDebugOutputLevel()`

§ getLogFormatOptions()

```
const LogFormatOptions & getLogFormatOptions ( void )
```

static

Returns

the **LogFormatOptions** that are used by this instance.

See also

setFormatOptions()

§ setLogFormatOptions()

```
void  
setLogFormatOptions ( const LogFormatOptions & formatOptions )
```

Sets the log format options to the supplied value.

Parameters

formatOptions the new value to which the log format options will be set

See also

`getFormatOptions()`

Photon C++

Client API 4.1.12.2

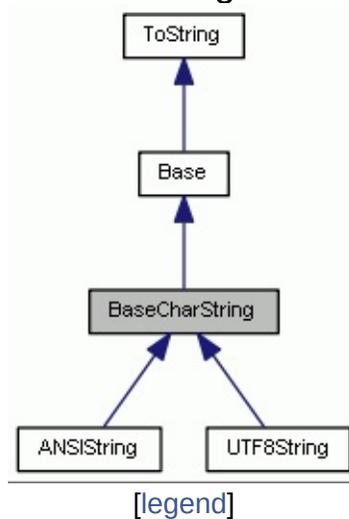
ExitGames > Common > BaseCharString

[Public Member Functions](#) | [List of all members](#)

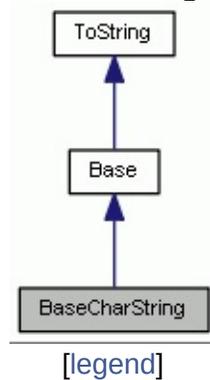
BaseCharString Class

Reference abstract

Inheritance diagram for BaseCharString:



Collaboration diagram for BaseCharString:



Public Member Functions

BaseCharString ()

virtual **~BaseCharString** (void)

virtual **operator const char *** (void) const =0

virtual **operator JString** (void) const =0

const char * **cstr** (void) const

virtual **JString JStringRepresentation** (void) const =0

unsigned int **length** (void) const

virtual unsigned int **size** (void) const =0

JString & toString (**JString** &retStr, bool withTypes=false)
const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **Tostring**

virtual **~ToString** (void)

virtual **JString typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **BaseCharString** class is the abstract base class for container classes, holding char* strings.

You are encouraged to add additional subclasses for encodings, for which there are no subclasses provided out of the box.

Subclasses of this class act as convenience classes for conversions between instances of class **JString** and char*'s.

The encoding of the char*'s is defined by the subclass. There should be one subclass for every supported encoding.

Subclasses of this class should only be used to hold or pass strings and for conversions between string encodings. Please use class **JString** for common string operations and modifications.

Constructor & Destructor Documentation

§ BaseCharString()

BaseCharString ()

Constructor.

§ ~BaseCharString()

`~BaseCharString (void)`

virtual

Destructor.

Member Function Documentation

§ operator const char *()

operator const char * (void) const

pure virtual

operator const char*.

Copies a pointer to the content of its right operand into its left operand.

This overwrites old data in the left operand.

Implemented in **ANSIString**, and **UTF8String**.

§ operator JString()

operator **JString** (void) const

pure virtual

operator **JString**.

Copies a **JString** representation of its right operand into its left operand.

This overwrites old data in the left operand.

Implemented in **ANSIString**, and **UTF8String**.

§ cstr()

```
const char * cstr ( void ) const
```

Remarks

The data, to which the pointer points to, is only valid as long as the instance is valid.

Returns

a pointer to a char array representation of the string.

§ JStringRepresentation()

JStringRepresentation (void) const

pure virtual

Returns

a **JString** representation of the string.

Implemented in **ANSIString**, and **UTF8String**.

§ length()

```
unsigned int length ( void ) const
```

Returns

the length of the string in characters

§ size()

unsigned int size (void) const

pure virtual

The default implementation of this function will just return `length()`, but for multibyte strings like `UTF8String` the return values of `length()` and `size()` can differ.

Returns

the size of the string in bytes

Implemented in `ANSIString`, and `UTF8String`.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.

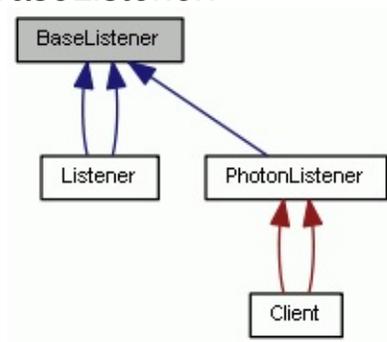
Photon C++ Client API 4.1.12.2

ExitGames > Common > BaseListener >

[Public Member Functions](#) | [List of all members](#)

BaseListener Class Reference **abstract**

Inheritance diagram for BaseListener:



[legend]

Public Member Functions

virtual void **debugReturn** (int debugLevel, const **JString** &string)=0

Detailed Description

This class defines the listener interface for the debug callback mechanism.

See also

[Base](#), [Base::setListener\(\)](#)

Member Function Documentation

§ debugReturn()

```
debugReturn ( int          debugLevel,  
              const JString & string  
            ) pure virtual
```

This is the callback function for debug-messages.

Parameters

debugLevel one of the values in [DebugLevel](#)
string the formatted debug string

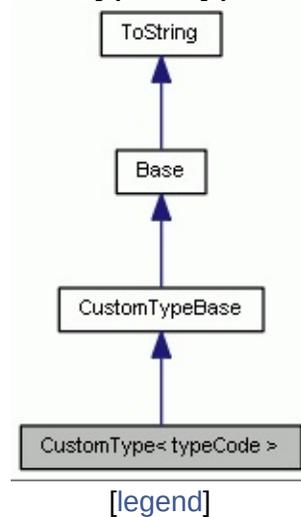
See also

[BaseListener](#)

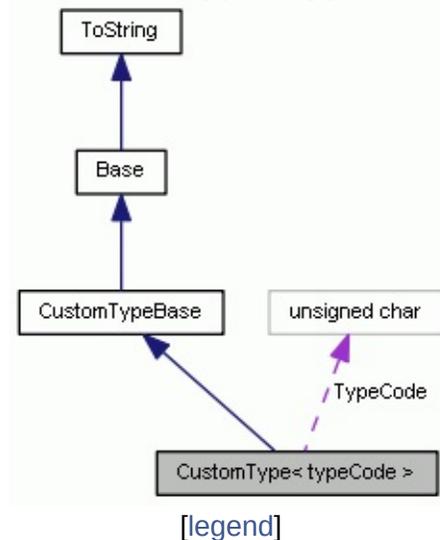
Implemented in [Listener](#), and [Listener](#).

CustomType< typeCode > Class Template Reference

Inheritance diagram for CustomType< typeCode > :



Collaboration diagram for CustomType< typeCode > :



Static Public Member Functions

static void **constructClass** (const **CustomTypeFactory**< typeCode > &factory)

static void **deconstructClass** (void)

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener** *baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const **LogFormatOptions** &options)

Static Public Attributes

static const nByte **TypeCode**

Additional Inherited Members

▶ Public Member Functions inherited from **CustomTypeBase**

virtual void **cleanup** (void)=0

virtual bool **compare** (const **CustomTypeBase** &other) const =0

virtual void **duplicate** (**CustomTypeBase** *pRetVal) const =0

virtual void **deserialize** (const nByte *pData, short length)=0

virtual short **serialize** (nByte *pRetVal) const =0

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **Tostring**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

virtual **JString** & **toString** (**JString** &retStr, bool withTypes=false)
const =0

JString **toString** (bool withTypes=false) const

Detailed Description

```
template<nByte typeCode>  
class ExitGames::Common::CustomType< typeCode >
```

The **CustomType** class provides you with an interface, to add support for additional data-types.

We only support a certain subset of **Datatypes** out of the box. If you need support for further datatypes, then you can easily add this support yourself by subclassing this class template and providing suitable implementations for the pure virtual functions, which are inherited from **CustomTypeBase**, in your subclass. You should only subclass every typecode once. typeCode 0 should be considered as reserved. So your first custom type would inherit from CustomType<1>, the second one from CustomType<2> and so on. Subclassing the same typecode multiple times will lead into undefined behavior as the typecode will determine the class as instance of which serialized data should be interpreted.

Remarks

When you are subclassing a specialization of **CustomType**, then you will also have to subclass the according specialization of **CustomTypeFactory** (the one for the same typecode).

See also

CustomTypeBase, **CustomTypeFactory**

Member Function Documentation

§ constructClass()

```
void  
constructClass ( const CustomTypeFactory< typeCode > & factory )
```

This static function initializes the class and has to be called once before any instance of a concrete subclass gets created. It registers the typecode and sets the factory-class to a copy of the passed parameter.

See also

[deconstructClass\(\)](#)

Parameters

factory an instance of the factory class, which will be used to create instances of this class

§ deconstructClass()

```
void deconstructClass ( void )
```

static

This static function cleans up the class and has to be called once after the last instance of a concrete subclass has been deallocated. It will then deallocate the shared instance of the according **CustomTypeFactory** subclass.

See also

constructClass()

Member Data Documentation

§ TypeCode

TypeCode

static

Check this public constant to find out the typecode of a custom type at runtime. This should normally not be of any interest.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

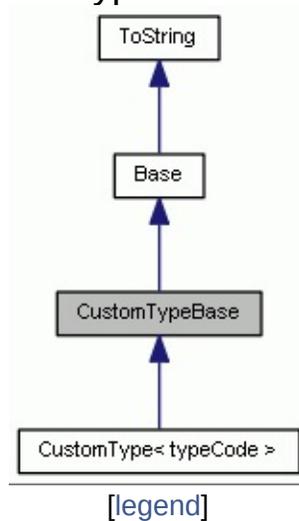
ExitGames > Common > CustomTypeBase >

[Public Member Functions](#) | [List of all members](#)

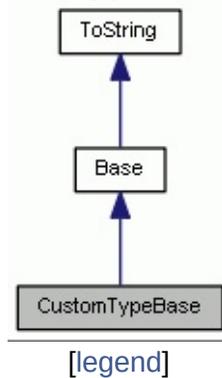
CustomTypeBase

Class Reference abstract

Inheritance diagram for CustomTypeBase:



Collaboration diagram for CustomTypeBase:



Public Member Functions

virtual void **cleanup** (void)=0

virtual bool **compare** (const **CustomTypeBase** &other) const =0

virtual void **duplicate** (**CustomTypeBase** *pRetVal) const =0

virtual void **deserialize** (const nByte *pData, short length)=0

virtual short **serialize** (nByte *pRetVal) const =0

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

virtual **JString** & **toString** (**JString** &retStr, bool withTypes=false)
const =0

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This is the abstract base class for the **CustomType** template and declares the interface, which you will have to implement, when subclassing **CustomType**.

For example implementations of these functions please refer to class `SampleCustomType` in `demo_typeSupport`.

See also

CustomType, **CustomTypeFactory**

Member Function Documentation

§ cleanup()

cleanup (void)

pure virtual

This function gets called, when the instance gets destroyed. This is the right place to do all the stuff, that you would normally do in the destructor. In the destructor you should instead just call this function, as instances of this class will be created and destroyed not only by constructors and destructors, but also by factory functions in situations, in which the class and therefor the constructor and destructor of the object instance to create/destroy are unknown.

§ compare()

```
compare ( const CustomTypeBase & other ) const
```

```
pure virtual
```

This function should be implemented to behave like an operator== would behave for the class, for which this function gets implemented.

For example for a wrapperclass around an integer it could just be implemented like this:

```
bool Foo::compare(const CustomTypeBase& other)
    const
{
    return typeid(*this) == typeid(other) && mInt ==
        ((Foo&)other).mInt;
}
```

Parameters

other the object to compare the instance with

Returns

true, if both objects are equal, false otherwise

§ duplicate()

```
duplicate ( CustomTypeBase * pRetVal ) const
```

pure virtual

This function shall save a copy of the instance, on which it has been called on, in its return value.

Parameters

pRetVal the object, to store a copy of the instance in - has to be of the instance type or a subclass of it, otherwise the behavior will be undefined

§ deserialize()

```
deserialize ( const nByte * pData,  
             short      length  
            )
```

pure virtual

This function initializes the instance, on which it has been called on, by deserializing the passed nByte-array, which has to be created by a call to **serialize()** on an instance of the same class before.

Previous data, stored in the instance, gets overwritten.

Parameters

pData a nByte-array, holding the deserialized payload of an object, which class has to be the same like the one of the instance, on which the function gets called

length the length of pData in elements

§ serialize()

```
serialize ( nByte * pRetVal ) const
```

pure virtual

This function serializes the payload of the instance on which it has been called, into the passed nByte-array and returns the length of that array. It is legal to pass a NULL-pointer and in that case this function still calculates the length of the data, which would have been stored in a non-NULL-pointer, but does not store any data. The behavior for providing a too small array is undefined.

Parameters

pRetVal the nByte-array to store the serialized payload of the instance in. Has to be of at least the needed length

Returns

the length of the data, that has actually been stored in the passed array

Photon C++

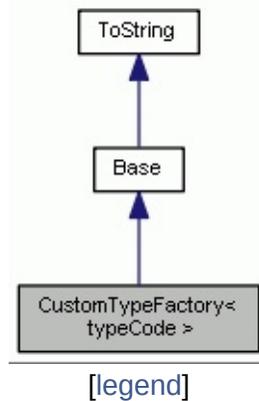
Client API 4.1.12.2

ExitGames > Common > CustomTypeFactory

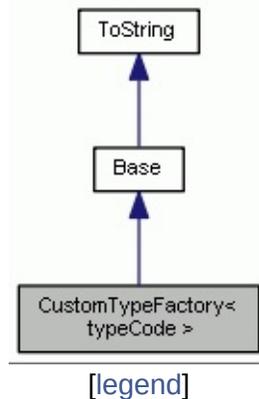
[Public Member Functions](#) | [List of all members](#)

CustomTypeFactory< typeCode > Class Template Reference abstract

Inheritance diagram for CustomTypeFactory< typeCode >:



Collaboration diagram for CustomTypeFactory< typeCode >:



Public Member Functions

virtual **~CustomTypeFactory**
(void)

virtual **CustomTypeFactory**< typeCode > * **copyFactory** (void) const
=0

virtual void **destroyFactory** (void)=0

virtual **CustomType**< typeCode > * **create** (short amount)
const =0

virtual **CustomType**< typeCode > * **copy** (const **CustomType**<
typeCode > *pToCopy,
short amount) const =0

virtual void **destroy** (const
CustomType< typeCode >
*pToDestroy) const =0

virtual unsigned int **sizeOf** (void) const =0

virtual **JString** & **toString** (**JString** &retStr,
bool withTypes=false)
const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString toString (bool
withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

```
template<nByte typeCode>  
class ExitGames::Common::CustomTypeFactory<  
typeCode >
```

This is the factory interface class template for the [CustomType](#) interface class template and offers an interface to create, copy and delete instances of [CustomType](#) subclasses, without the caller needing to know the names of the subclasses.

For every specialization of the [CustomType](#) template, that you subclass, you have to subclass the according specialization (meaning the one for the same typecode) of this class. Please refer to class `SampleCustomTypeFactory` in `demo_typeSupport` for an example implementation.

Remarks

You normally won't have to call functions from this class yourself, but the library does this for you.

See also

[CustomType](#), [CustomTypeBase](#)

Constructor & Destructor Documentation

§ ~CustomTypeFactory()

`~CustomTypeFactory (void)`

virtual

Destructor.

Member Function Documentation

§ copyFactory()

```
copyFactory ( void ) const
```

```
pure virtual
```

This function shall return a pointer to a freshly allocated copy of the instance, on which it has been called.

Returns

a pointer to a copy of the instance

§ destroyFactory()

destroyFactory (void)

pure virtual

This function shall deallocate the instance, on which it has been called on.

§ create()

```
create ( short amount ) const
```

```
pure virtual
```

This function shall allocate an array of the class, for which the template parameter specialization has been registered.

Parameters

amount the amount of elements to allocate

Returns

a pointer to the created array of CustomTypes

§ copy()

```
copy ( const CustomType< typeCode > * pToCopy,  
       short amount  
       ) const pure virtual
```

This function shall return a pointer to a freshly allocated copy of the passed array.

Parameters

pToCopy a pointer to the original array, which should be copied
amount the amount of elements of the array, pointed to by pToCopy

Returns

the created copy of the array

§ destroy()

```
destroy ( const CustomType< typeCode > * pToDestroy ) const pure virtu
```

This function shall deallocate the array, to which the passed pointer points.

Parameters

pToDestroy a pointer to an array, which has previously been allocated with **create()** or **copy()**

§ sizeof()

```
sizeof ( void ) const
```

pure virtual

Returns

the size of a single instance as determined by calling the sizeof()-operator, for the class, which is fabricated by this specialization of the factory

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++

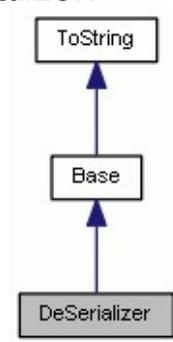
Client API 4.1.12.2

ExitGames > Common > DeSerializer

[Public Member Functions](#) | [List of all members](#)

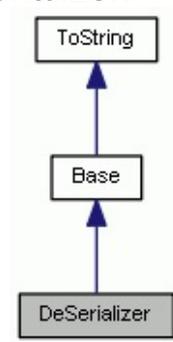
DeSerializer Class Reference

Inheritance diagram for DeSerializer:



[legend]

Collaboration diagram for DeSerializer:



[legend]

Public Member Functions

DeSerializer (const nByte *data, int size)

bool **pop** (**Object** &object)

JString & **toString** (**JString** &retStr, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

With a **DeSerializer** instance you can retrieve the original data that has been serialized into a byte-array by a **Serializer** instance, by the **Photon** Server or by other **Photon** Client platforms.

Constructor & Destructor Documentation

§ DeSerializer()

```
DeSerializer ( const nByte * data,  
              int           size  
              )
```

Constructor: Creates a new instance that contains the passed data as payload.

Parameters

data a byte array, that has been retrieved by a call to **Serializer::getData()**, an unchanged copy of such a byte array or a byte array that is otherwise guaranteed to 100% conform to the format that's used by **Serializer** (for example data, that has been serialized by a compatible version of the **Photon** Server or of other **Photon** Client platforms), otherwise the behavior of this class is undefined.

size the size in bytes of data

Member Function Documentation

§ pop()

```
bool pop ( Object & object )
```

This function will deserialize all data in the DeSerializer-instance, that has been serialized via a single call to **Serializer::push()**. If the **DeSerializer** instance has been created by passing a byte array that has been created by a **Serializer** instance on which multiple push() calls have taken place, then an equivalent amount of calls to this function will be valid.

Remarks

Any potentially existing old payload of parameter object will get overridden in a successful call. In case that there is nothing more to deserialize parameter object will remain unchanged. In case that a call to this function fails due to corrupt data (read: the byte array passed to the **DeSerializer** instance on construction has neither been retrieved by a call to **Serializer::getData()** nor been an unchanged copy of such data) the content of parameter object is undefined.

Parameters

object an Object-instance, in which the deserialized data will be stored

Returns

true on success, false when all data has already been deserialized in previous calls or when the data is corrupt and can't be deserialized

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )          const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.



Photon C++

Client API 4.1.12.2

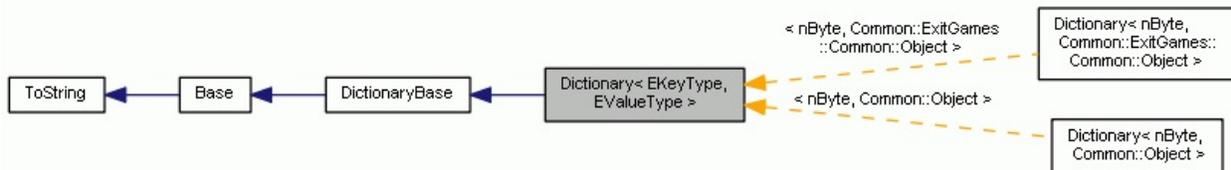
ExitGames > Common > Dictionary >

[Classes](#) | [Public Member Functions](#) |

[List of all members](#)

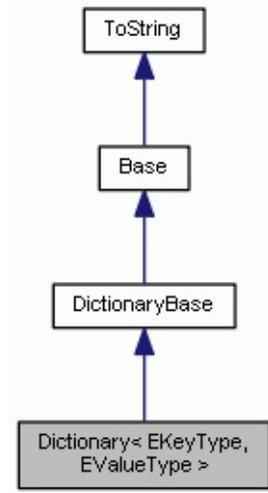
Dictionary< EKeyType, EValueType > Class Template Reference

Inheritance diagram for Dictionary< EKeyType, EValueType >:



[legend]

Collaboration diagram for Dictionary< EKeyType, EValueType >:



[legend]

Public Member Functions

Dictionary (void)

~Dictionary (void)

Dictionary (const **Dictionary**< EKeyType, EValueType > &toCopy)

Dictionary & **operator=** (const **Dictionary**< EKeyType, EValueType > &toCopy)

bool **operator==** (const **Dictionary**< EKeyType, EValueType > &toCompare) const

bool **operator!=** (const **Dictionary**< EKeyType, EValueType > &toCompare) const

const EValueType & **operator[]** (unsigned int index) const

EValueType & **operator[]** (unsigned int index)

const nByte * **getKeyTypes** (void) const

const nByte * **getValueTypes** (void) const

const unsigned int * **getValueDimensions** (void) const

void **put** (const **Dictionary**< EKeyType, EValueType > &src)

void **put** (const EKeyType &key, const EValueType &val)

void **put** (const EKeyType &key)

void **put** (const EKeyType &key, const EValueType pVal, typename Common::Helpers::ArrayLengthType< EValueType >::type size)

void **put** (const EKeyType &key, const EValueType pVal, const short *sizes)

const EValueType * **getValue** (const EKeyType &key) const

JVector< EKeyType > **getKeys** (void) const

void **remove** (const EKeyType &key)

bool **contains** (const EKeyType &key) const

JString **typeToString** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false) const

► Public Member Functions inherited from **DictionaryBase**

virtual **~DictionaryBase** (void)

DictionaryBase (const **DictionaryBase** &toCopy)

DictionaryBase & **operator=** (const **DictionaryBase** &toCopy)

bool **operator==** (const **DictionaryBase** &toCompare) const

bool **operator!=** (const **DictionaryBase** &toCompare) const

template<typename FKeyType >

void **remove** (const FKeyType &key)

```
template<typename FKeyType >
```

```
bool contains (const FKeyType &key) const
```

```
void removeAllElements (void)
```

```
JString toString (void) const
```

```
JString & toString (JString &retStr, bool withTypes=false)  
const
```

```
const Hashtable & getHashtable (void) const
```

```
unsigned int getSize (void) const
```

```
template<typename FKeyType >
```

```
const short * getValueSizes (const FKeyType &key) const
```

```
template<typename FKeyType , typename FValueType >
```

```
const FValueType * getValue (const FKeyType &key, const  
FValueType *) const
```

```
template<typename FKeyType >
```

```
const Object * getValue (const FKeyType &key, const Object  
*) const
```

```
template<typename FKeyType >
```

```
JVector< FKeyType > getKeys (const FKeyType *) const
```

```
JVector< Object > getKeys (const Object *) const
```

▶ **Public Member Functions inherited from Base**

```
virtual ~Base (void)
```

▶ **Public Member Functions inherited from ToString**

```
virtual ~ToString (void)
```

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

```
template<typename EKeyType, typename EValueType>  
class ExitGames::Common::Dictionary< EKeyType,  
EValueType >
```

The **Dictionary** class template together with the **Hashtable** class is one of the two main container classes for objects to be transmitted over **Photon** when using the C++ Client.

This class implements the well-known concept of a container structure storing an arbitrary number of key/value-pairs.

In contrast to a **Hashtable**, the types of both the keys and also the values in a **Dictionary** have to be the same for all entries. This takes flexibility, but it also improves type safety and means, that the type infos only have to be stored twice for the whole **Dictionary** (once for the key and once for the value), while in a **Hashtable** they have to be stored twice per entry. Therefore with Dictionaries transferring the same amount of key-value pairs will cause less traffic than with Hashtables.

Please have a look at the **Table of Datatypes** for a list of types, that are supported as keys and as values.

Please refer to the documentation for **put()** and **getValue()** to see how to store and access data in a **Dictionary**.

See also

put(), **getValue()**, **KeyObject**, **ValueObject**, **Hashtable**, **DictionaryBase**

Constructor & Destructor Documentation

§ Dictionary() [1/2]

Dictionary (void)

Constructor: Creates an empty instance.

§ ~Dictionary()

~Dictionary (void)

Destructor.

§ Dictionary() [2/2]

Dictionary (const **Dictionary**< EKeyType, EValueType > & **toCopy**)

Copy-Constructor: Creates a deep copy of the argument.

Parameters

toCopy The object to copy.

Member Function Documentation

§ operator=()

```
Dictionary<  
EKeyType,  
EValueType  
> &  
operator= ( const Dictionary< EKeyType, EValueType > & toCopy )
```

operator=. Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

§ operator==()

bool

operator== (const **Dictionary**< EKeyType, EValueType > & toCompare

operator==.

Returns

true, if both operands are equal, false otherwise.

Two instances are considered equal if they each hold the same number of entries and, for a given key, the corresponding values equal each other.

Two values are considered equal to each other, if instances of class **Obj** that are holding them as payloads, equal each other.

See also

Object::operator==()

§ operator!=()

```
bool  
operator!= ( const Dictionary< EKeyType, EValueType > & toCompare )
```

operator!=.

Returns

false, if **operator==()** would return true, true otherwise.

§ operator[]() [1/2]

```
const EValueType & operator[] ( unsigned int index ) const
```

operator[].

Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ operator[]() [2/2]

EValueType & operator[] (unsigned int *index*)

operator[].

Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ getKeyTypes()

```
const nByte * getKeyTypes ( void ) const
```

virtual

Returns

an array, holding the type code for the key type of the **Dictionary** and type codes for the key types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element in the array returned by **getValueTypes()** at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getKeyTypes()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the values **TypeCode::INTEGER** at index 0 and **TypeCode::SHORT** at index 1.

The codes returned by this function match the ones, that are stored in member variable "typename" of class template **Helpers::ConfirmAllowedKey**'s specializations. Only the types, for which specializations of that template exist, are valid **Dictionary** keys.

Reimplemented from **DictionaryBase**.

§ getValueTypes()

```
const nByte * getValueTypes ( void ) const
```

virtual

Returns

an array, holding the type code for the value type of the **Dictionary** and type codes for the value types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getValueTypes()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the values **TypeCode::DICTIONARY** at index 0 and **TypeCode::FLOAT** at index 1.

The codes returned by this function match the ones, that are stored in member variable "typename" of class template Helpers::ConfirmAllowed's specializations. Only the types, for which specializations of that template exist, are valid **Dictionary** values.

Reimplemented from **DictionaryBase**.

§ getValueDimensions()

```
const unsigned int * getValueDimensions ( void ) const
```

virtual

Returns

an array, holding the amount of array dimensions for the value type of the **Dictionary** and for the value types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element in the array returned by **getValueTypes()** at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getValueDimensions()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the value 1 (for 1D array) at index 0 and 2 (for 2D) at index 1. If a value type is no array, then this functions return value will contain 0 at the corresponding index.

Reimplemented from **DictionaryBase**.

§ put() [1/5]

```
void put ( const Dictionary< EKeyType, EValueType > & src )
```

Adds all pairs of a key and a corresponding value from the passed instance to the instance, on which it is called on. If a key is already existing, then its old value will be replaced with the new one.

Parameters

src instance, from which to add the content

Returns

nothing.

§ put() [2/5]

```
void put ( const EKeyType & key,  
          const EValueType & val  
          )
```

Adds a pair of a key and a corresponding value to the instance.

If the key is already existing, then it's old value will be replaced with the new one. Please have a look at the [table of datatypes](#) for a list of supported types for keys and values

Parameters

key the key to add

val the value to add

Returns

nothing.

§ put() [3/5]

```
void put ( const EKeyType & key )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload adds an empty object as value for the provided key.

§ put() [4/5]

```
void  
put ( const EKeyType &  
      const EValueType  
      typename Common::Helpers::ArrayLengthType< EValueType >::type  
      )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts singledimensional arrays and NULL-pointers pass parameter pVal. NULL pointers are only legal input, if size is 0

Parameters

- key** the key to add
- pVal** the value array to add
- size** the size of the value array

§ put() [5/5]

```
void put ( const EKeyType & key,  
          const EValueType pVal,  
          const short * sizes  
        )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts multidimensional arrays and NULL-pointers passed for parameter pVal. The array that is passed for parameter pVal has to be a pointer of the correct abstraction level, meaning a normal pointer for a singledimensional array, a doublepointer for a twodimensional array, a triplepointer for a threedimensional array and so on. For pVal NULL pointers are only legal input, if sizes[0] is 0. For sizes NULL is no valid input.

Parameters

key the key to add

pVal the value array to add

sizes the sizes for every dimension of the value array - the length of this array has to match the dimensions of pVal

§ getValue()

```
const EValueType * getValue ( const EKeyType & key ) const
```

Returns the corresponding value for a specified key.

Parameters

key Reference to the key to return the corresponding value for.

Returns

a pointer to the corresponding value if the **Hashtable** contains the specified key, NULL otherwise.

See also
[put\(\)](#)

§ getKeys()

```
JVector< EKeyType > getKeys ( void ) const
```

Returns

a **JVector** holding all keys contained in the **Hashtable**.

§ remove()

```
void remove ( const EKeyType & key )
```

Deletes the specified key and the corresponding value, if found in the **Hashtable**.

Parameters

key Pointer to the key of the key/value-pair to remove.

Returns

nothing.

See also

removeAllElements()

§ contains()

```
bool contains ( const EKeyType & key ) const
```

Checks, whether the **Hashtable** contains a certain key.

Parameters

key Pointer to the key to look up.

Returns

true if the specified key was found, false otherwise.

§ typeToString()

JString typeToString (void) const

virtual

Remarks

This function is intended for debugging purposes. For runtime type checking you should use RTTI's typeid() instead. Demangling and cutting off of namespaces will only happen on platforms, which offer a system functionality for demangling.

Returns

a string representation of the class name of the polymorphically correct runtime class of the instance, on which it is called on, after this class name has been demangled and eventual namespaces have been removed.

Reimplemented from **ToString**.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.



Photon C++

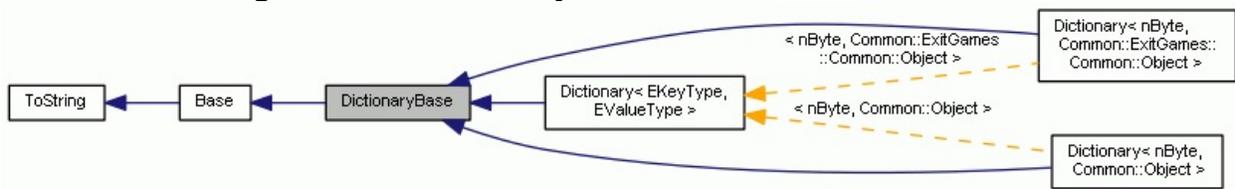
Client API 4.1.12.2

ExitGames > Common > DictionaryBase >

[Classes](#) | [Public Member Functions](#) |
[List of all members](#)

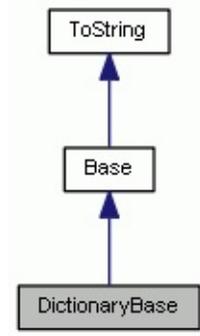
DictionaryBase Class Reference

Inheritance diagram for DictionaryBase:



[legend]

Collaboration diagram for DictionaryBase:



[legend]

Public Member Functions

virtual **~DictionaryBase** (void)

DictionaryBase (const **DictionaryBase** &toCopy)

DictionaryBase & **operator=** (const **DictionaryBase** &toCopy)

bool **operator==** (const **DictionaryBase** &toCompare) const

bool **operator!=** (const **DictionaryBase** &toCompare) const

template<typename FKeyType >

void **remove** (const FKeyType &key)

template<typename FKeyType >

bool **contains** (const FKeyType &key) const

void **removeAllElements** (void)

JString **typeToString** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false) const

const **Hashtable** & **getHashtable** (void) const

unsigned int **getSize** (void) const

virtual const nByte * **getKeyTypes** (void) const

virtual const nByte * **getValueTypes** (void) const

```
template<typename FKeyType >
    const short * getValueSizes (const FKeyType &key)
    const
```

```
virtual const unsigned int * getValueDimensions (void) const
```

```
template<typename FKeyType , typename FValueType >
    const FValueType * getValue (const FKeyType &key, const
    FValueType *) const
```

```
template<typename FKeyType >
    const Object * getValue (const FKeyType &key, const
    Object *) const
```

```
template<typename FKeyType >
    JVector< FKeyType > getKeys (const FKeyType *) const
```

```
JVector< Object > getKeys (const Object *) const
```

► **Public Member Functions inherited from Base**

```
virtual ~Base (void)
```

► **Public Member Functions inherited from ToString**

```
virtual ~ToString (void)
```

```
JString toString (bool withTypes=false) const
```

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **DictionaryBase** class is the base class for the **Dictionary** class template and intended to be used instead of **Dictionary** in cases, when the key type and/or value type of a **Dictionary** instance can't be known at compile time, but only at runtime.

Whenever possible you should use the class template **Dictionary** instead of **DictionaryBase** to enable compile time type safety and optimizations that need compile time type identification. However, when for example receiving unknown data over the network at runtime, the type of that data can't be non at compile time. In those cases **DictionaryBase** instances are used.

DictionaryBase instances only offer read only API: They can't be modified with the exception of replacing the complete instance with the content of another one. No single entries can be added, removed, or changed. Use the **Dictionary** sub class template for modifiable **Dictionary** instances.

Please have a look at the **Table of Datatypes** for a list of types, that are supported as keys and as values.

Please refer to the documentation for `put()` and `getValue()` to see how to store and access data in a **Dictionary**.

See also
`getValue()`, **Dictionary**

Constructor & Destructor Documentation

§ ~DictionaryBase()

~DictionaryBase (void)

virtual

Destructor.

§ DictionaryBase()

```
DictionaryBase ( const DictionaryBase & toCopy )
```

Copy-Constructor: Creates a deep copy of the argument.

Parameters

toCopy The object to copy.

Member Function Documentation

§ operator=()

```
DictionaryBase & operator= ( const DictionaryBase & toCopy )
```

operator=. Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

§ operator==()

```
bool operator== ( const DictionaryBase & toCompare ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

Two instances are considered equal if they each hold the same number of entries and, for a given key, the corresponding values equal each other.

Two values are considered equal to each other, if instances of class **Object**, that are holding them as payloads, equal each other.

See also

Object::operator==()

§ operator!=()

```
bool operator!= ( const DictionaryBase & toCompare ) const
```

operator!=.

Returns

false, if `operator==()` would return true, true otherwise.

§ remove()

```
void remove ( const FKeyType & key )
```

Deletes the specified key and the corresponding value, if found in the **Hashtable**.

Parameters

key Pointer to the key of the key/value-pair to remove.

Returns

nothing.

See also

removeAllElements()

§ contains()

```
bool contains ( const FKeyType & key ) const
```

Checks, whether the **Hashtable** contains a certain key.

Parameters

key Pointer to the key to look up.

Returns

true if the specified key was found, false otherwise.

§ removeAllElements()

```
void removeAllElements ( void )
```

Clears the **Hashtable**, which means deleting all its content.

Returns

nothing.

See also

[remove\(\)](#)

§ typeToString()

JString typeToString (void) const

virtual

Remarks

This function is intended for debugging purposes. For runtime type checking you should use RTTI's typeid() instead. Demangling and cutting off of namespaces will only happen on platforms, which offer a system functionality for demangling.

Returns

a string representation of the class name of the polymorphically correct runtime class of the instance, on which it is called on, after this class name has been demangled and eventual namespaces have been removed.

Reimplemented from **ToString**.

Reimplemented in **Dictionary< nByte, Common::ExitGames::Common::Object >**, and **Dictionary< nByte, Common::Object >**.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Reimplemented in **Dictionary< nByte, Common::ExitGames::Common::Object >**, and **Dictionary< nByte, Common::Object >**.

§ getHashtable()

```
const Hashtable & getHashtable ( void ) const
```

Returns

a readonly reference to a **Hashtable** representation of the **Dictionary** instance. The returned reference refers to the original payload data of the **Dictionary**, so its payload will change, if the payload of the according **Dictionary** instance changes.

§ getSize()

```
unsigned int getSize ( void ) const
```

Returns

the number of key/value pairs stored in the **Hashtable**.

§ getKeyTypes()

```
const nByte * getKeyTypes ( void ) const
```

virtual

Returns

an array, holding the type code for the key type of the **Dictionary** and type codes for the key types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element in the array returned by **getValueTypes()** at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getKeyTypes()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the values **TypeCode::INTEGER** at index 0 and **TypeCode::SHORT** at index 1.

The codes returned by this function match the ones, that are stored in member variable "typename" of class template **Helpers::ConfirmAllowedKey**'s specializations. Only the types, for which specializations of that template exist, are valid **Dictionary** keys.

Reimplemented in **Dictionary**< **EKeyType**, **EValueType** >, **Dictionary**< **nByte**, **Common::ExitGames::Common::Object** >, and **Dictionary**< **nByte**, **Common::Object** >.

§ getValueTypes()

```
const nByte * getValueTypes ( void ) const
```

virtual

Returns

an array, holding the type code for the value type of the **Dictionary** and type codes for the value types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getValueTypes()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the values **TypeCode::DICTIONARY** at index 0 and **TypeCode::FLOAT** at index 1.

The codes returned by this function match the ones, that are stored in member variable "typename" of class template Helpers::ConfirmAllowed's specializations. Only the types, for which specializations of that template exist, are valid **Dictionary** values.

Reimplemented in **Dictionary**< EKeyType, EValueType >, **Dictionary**< nByte, Common::ExitGames::Common::Object >, and **Dictionary**< nByte, Common::Object >.

§ getValueSizes()

```
const short * getValueSizes ( const FKeyType & key ) const
```

Returns

Object::getSizes() of the value, that corresponds to the passed key.

Parameters

key Reference to the key to return the corresponding value sizes for

§ getValueDimensions()

```
const unsigned int * getValueDimensions ( void ) const
```

virtual

Returns

an array, holding the amount of array dimensions for the value type of the **Dictionary** and for the value types of potential nested Dictionaries.

Only index 0 of the returned array is guaranteed to be valid. The existence of elements at other indices depends on the value of the element in the array returned by **getValueTypes()** at the previous index in the following way: Only when **getValueTypes()[i] == TypeCode::DICTIONARY**, then **getValueDimensions()[i+1]** will be valid.

Type information for nested Dictionaries will be stored like in the following example: **Dictionary**<int, Dictionary<short, float**>*> This is a **Dictionary**, with the key type being int and the value type being a 1D array of type Dictionary<short, float**>, so that all values are Dictionaries, which keys are shorts and which values are 2D arrays of float. This function's return value in this example will hold the value 1 (for 1D array) at index 0 and 2 (for 2D) at index 1. If a value type is no array, then this functions return value will contain 0 at the corresponding index.

Reimplemented in **Dictionary**< EKeyType, EValueType >, **Dictionary**< nByte, Common::ExitGames::Common::Object >, and **Dictionary**< nByte, Common::Object >.

§ `getValue()` [1/2]

```
const FValueType * getValue ( const FKeyType & key,  
                             const FValueType *  
                             ) const
```

Returns the corresponding value for a specified key.

Parameters

key Reference to the key to return the corresponding value for.

Returns

a pointer to the corresponding value if the **Hashtable** contains the specified key, NULL otherwise.

See also

`put()`

§ getValue() [2/2]

```
const Object * getValue ( const FKeyType & key,  
                        const Object *  
                        ) const
```

Returns the corresponding value for a specified key.

Parameters

key Reference to the key to return the corresponding value for.

Returns

a pointer to the corresponding value if the **Hashtable** contains the specified key, NULL otherwise.

See also

put()

§ getKeys() [1/2]

```
JVector< FKeyType > getKeys ( const FKeyType * ) const
```

Returns

a **JVector** holding all keys contained in the **Hashtable**.

§ `getKeys()` [2/2]

```
JVector< Object > getKeys ( const Object * ) const
```

Returns

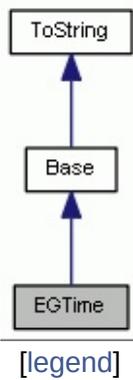
a **JVector** holding all keys contained in the **Hashtable**.

© Exit Games ®, all rights reserved.

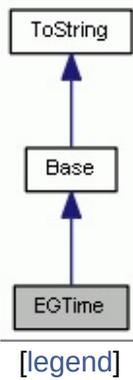
[Photon Documentation](#) | [Contact](#) | [Terms](#)

EGTime Class Reference

Inheritance diagram for EGTime:



Collaboration diagram for EGTime:



Public Member Functions

EGTime (int time)

~EGTime (void)

EGTime (const **EGTime** &toCopy)

EGTime & **operator=** (const **EGTime** &toCopy)

EGTime & **operator=** (const int &time)

const **EGTime** & **operator+=** (const **EGTime** &time)

const **EGTime** & **operator-=** (const **EGTime** &time)

EGTime **operator+** (const **EGTime** &time)

EGTime **operator-** (const **EGTime** &time)

bool **operator<** (const **EGTime** &time) const

bool **operator>** (const **EGTime** &time) const

bool **operator<=** (const **EGTime** &time) const

bool **operator>=** (const **EGTime** &time) const

bool **operator==** (const **EGTime** &time) const

bool **operator!=** (const **EGTime** &time) const

bool **overflowed** (const **EGTime** &time) const

JString & **toString** (**JString** &retStr, bool withTypes=false)
const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **EGTime** class is a container class for millisecond timestamps, which accounts for overflows when comparing two instances against each other.

The intended usage of this class is to compare 32 bit integer millisecond timestamps, which only differ in relatively small amounts of ms (a few seconds up to at max a few hours) from each other. 32bit timestamps have the advantage over 64 bit ones, that they need less bytes to store their information, which is of critical value in some situations. However 32 bit milliseconds timestamps overflow every about 49 days. Arithmetical calculations don't react well to those overflows for unsigned integers, but they continue to work fine for signed integers. However when comparing two timestamps, one from shortly before an overflow, one from shortly after, even signed integers won't work: the timestamp INT_MIN is one millisecond LATER than INT_MAX, but when comparing these two as integers, INT_MIN is smaller than INT_MAX. **EGTime** approaches this issue by introducing an overflow threshold of 24 hours. If time a is bigger than time, but not bigger than time b + 24 hours, than and only than, **EGTime** will also consider it as bigger. This way code like `if(timestamp1 < timestamp2)` will also work, when between these two timestamps an overflow has happened. The downside is, that this class won't work when comparing 2 timestamps, that differ by more than 24 hours.

Constructor & Destructor Documentation

§ `EGTime()` [1/2]

`EGTime` (int `time`)

Constructor: Creates an `EGTime` instance.

Parameters

`time` the time in milliseconds to initialize the instance with

§ ~EGTime()

~EGTime (void)

Destructor.

§ `EGTime()` [2/2]

```
EGTime ( const EGTime & toCopy )
```

Copy-Constructor: Creates a new **EGTime** instance from a deep copy of the argument instance.

Parameters

toCopy the **EGTime** instance to make a copy from

Member Function Documentation

§ operator=() [1/2]

```
EGTime & operator= ( const EGTime & toCopy )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [2/2]

EGTime & operator= (const int & time)

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator+=()

```
const EGTime & operator+= ( const EGTime & time )
```

operator+=.

Adds the right time to the left time.

§ operator-=()

```
const EGTime & operator-= ( const EGTime & time )
```

operator-=.

Subtracts the right time from the left time.

§ operator+()

```
EGTime operator+ ( const EGTime & time )
```

operator+.

Adds the right time to the left time and returns the result as a new **EGTime** instance.

§ operator-()

EGTime operator- (const **EGTime** & time)

operator-=.

Subtracts the right time from the left time and returns the result as a new **EGTime** instance.

§ operator<()

```
bool operator< ( const EGTime & time ) const
```

operator<.

Remarks

An **EGTime** instance is considered smaller than another one, if its payload is either smaller or more than 24 hours bigger than the other ones payload.

Returns

true, if the left operand is smaller than the right operand, false otherwise.

§ operator>()

```
bool operator> ( const EGTime & time ) const
```

operator>.

Remarks

An **EGTime** instance is considered bigger than another one, if its payload is either bigger or more than 24 hours smaller than the other ones payload.

Returns

true, if the left operand is bigger than the right operand, false otherwise.

§ operator<=()

```
bool operator<= ( const EGTime & time ) const
```

operator<=.

Remarks

An **EGTime** instance is considered smaller than another one, if its payload is either smaller or more than 24 hours bigger than the other ones payload.

Returns

true, if the left operand is smaller than or equal to the right operand, false otherwise.

§ operator>=()

```
bool operator>= ( const EGTime & time ) const
```

operator>=.

Remarks

An **EGTime** instance is considered bigger than another one, if its payload is either bigger or more than 24 hours smaller than the other ones payload.

Returns

true, if the left operand is bigger than or equal to the right operand, false otherwise.

§ operator==()

```
bool operator==( const EGTime & time ) const
```

operator==.

Returns

true, if both instances have equal values, false otherwise.

§ operator!==()

```
bool operator!=( const EGTime & time ) const
```

operator==.

Returns

false, if both instances have equal values, true otherwise.

§ overflowed()

```
bool overflowed ( const EGTime & time ) const
```

Returns

true, if the values of both instances differ by more than 24 hours,
false otherwise.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

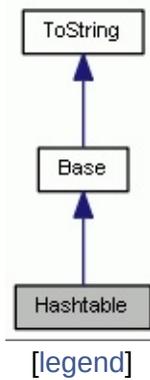
Returns

a **JString** representation of the instance and its contents for debugging purposes.

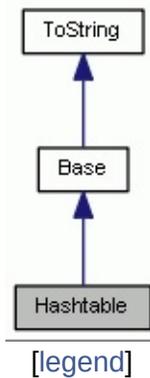
Implements **ToString**.

Hashtable Class Reference

Inheritance diagram for Hashtable:



Collaboration diagram for Hashtable:



Public Member Functions

Hashtable (void)

~Hashtable (void)

Hashtable (const **Hashtable** &toCopy)

Hashtable & **operator=** (const **Hashtable** &toCopy)

bool **operator==** (const **Hashtable** &toCompare) const

bool **operator!=** (const **Hashtable** &toCompare) const

const **Object** & **operator[]** (unsigned int index) const

Object & **operator[]** (unsigned int index)

void **put** (const **Hashtable** &src)

template<typename FKeyType , typename FValueType >

void **put** (const FKeyType &key, const FValueType &val)

template<typename FKeyType >

void **put** (const FKeyType &key)

template<typename FKeyType , typename FValueType >

void **put** (const FKeyType &key, const FValueType pVal, typename Common::Helpers::ArrayLengthType< FValueType >::type size)

template<typename FKeyType , typename FValueType >

void **put** (const FKeyType &key, const

FValueType pVal, const short *sizes)

template<typename FKeyType >

const **Object** * **getValue** (const FKeyType &key) const

unsigned int **getSize** (void) const

const **JVector**< **Object** > & **getKeys** (void) const

template<typename FKeyType >

void **remove** (const FKeyType &key)

template<typename FKeyType >

bool **contains** (const FKeyType &key) const

void **removeAllElements** (void)

JString & **toString** (**JString** &retStr, bool
withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **Hashtable** class together with the **Dictionary** class template is one of the two main container classes for objects to be transmitted over **Photon** when using the C++ Client.

This class implements the well-known concept of a container structure storing an arbitrary number of key/value-pairs.

In contrast to a **Dictionary**, the types of both the keys and also the values in a **Hashtable** can differ for every entry. This adds flexibility, but it also reduces type safety and means, that the type infos have to be stored twice (once for the key and once for the value) per entry in a **Hashtable**, while in a **Dictionary** it only has to be stored twice for the whole **Dictionary**, no matter how many entries are in there. Therefore with Dictionaries transferring the same amount of key-value pairs will cause less traffic than with Hashtables.

Please have a look at the **Table of Datatypes** for a list of types, that are supported as keys and as values.

Please refer to the documentation for **put()** and **getValue()** to see how to store and access data in a **Hashtable**.

See also

put(), **getValue()**, **KeyObject**, **ValueObject**, **Dictionary**

Constructor & Destructor Documentation

§ Hashtable() [1/2]

Hashtable (void)

Constructor: Creates an empty instance.

§ ~Hashtable()

~**Hashtable** (void)

Destructor.

§ Hashtable() [2/2]

```
Hashtable ( const Hashtable & toCopy )
```

Copy-Constructor: Creates a deep copy of the argument.

Parameters

toCopy The object to copy.

Member Function Documentation

§ operator=()

```
Hashtable & operator= ( const Hashtable & toCopy )
```

operator=. Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

§ operator==()

```
bool operator== ( const Hashtable & toCompare ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

Two instances are considered equal if they each hold the same number of entries and, for a given key, the corresponding values equal each other.

Two values are considered equal to each other, if instances of class **Object**, that are holding them as payloads, equal each other.

See also

Object::operator==()

§ operator!=()

```
bool operator!= ( const Hashtable & toCompare ) const
```

operator!=.

Returns

false, if **operator==()** would return true, true otherwise.

§ operator[]() [1/2]

```
const Object & operator[] ( unsigned int index ) const
```

operator[].

Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ operator[]() [2/2]

Object & operator[] (unsigned int *index*)

operator[].

Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ put() [1/5]

```
void put ( const Hashtable & src )
```

Adds all pairs of a key and a corresponding value from the passed instance to the instance, on which it is called on. If a key is already existing, then its old value will be replaced with the new one.

Parameters

src instance, from which to add the content

Returns

nothing.

§ put() [2/5]

```
void put ( const FKeyType & key,  
          const FValueType & val  
          )
```

Adds a pair of a key and a corresponding value to the instance.

If the key is already existing, then it's old value will be replaced with the new one. Please have a look at the [table of datatypes](#) for a list of supported types for keys and values

Parameters

key the key to add

val the value to add

Returns

nothing.

§ put() [3/5]

```
void put ( const FKeyType & key )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload adds an empty object as value for the provided key.

§ put() [4/5]

```
void  
put ( const FKeyType &  
      const FValueType  
      typename Common::Helpers::ArrayLengthType< FValueType >::type  
      )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts singledimensional arrays and NULL-pointers pass parameter pVal. NULL pointers are only legal input, if size is 0

Parameters

- key** the key to add
- pVal** the value array to add
- size** the size of the value array

§ put() [5/5]

```
void put ( const FKeyType & key,  
          const FValueType pVal,  
          const short * sizes  
        )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts multidimensional arrays and NULL-pointers passed for parameter pVal. The array that is passed for parameter pVal has to be a pointer of the correct abstraction level, meaning a normal pointer for a singledimensional array, a doublepointer for a twodimensional array, a triplepointer for a threedimensional array and so on. For pVal NULL pointers are only legal input, if sizes[0] is 0. For sizes NULL is no valid input.

Parameters

key the key to add

pVal the value array to add

sizes the sizes for every dimension of the value array - the length of this array has to match the dimensions of pVal

§ getValue()

```
const Object * getValue ( const FKeyType & key ) const
```

Returns the corresponding value for a specified key.

Parameters

key Reference to the key to return the corresponding value for.

Returns

a pointer to the corresponding value if the **Hashtable** contains the specified key, NULL otherwise.

See also
[put\(\)](#)

§ getSize()

```
unsigned int getSize ( void ) const
```

Returns

the number of key/value pairs stored in the **Hashtable**.

§ getKeys()

```
const JVector< Object > & getKeys ( void ) const
```

Returns

a **JVector** holding all keys contained in the **Hashtable**.

§ remove()

```
void remove ( const FKeyType & key )
```

Deletes the specified key and the corresponding value, if found in the **Hashtable**.

Parameters

key Pointer to the key of the key/value-pair to remove.

Returns

nothing.

See also

removeAllElements()

§ contains()

```
bool contains ( const FKeyType & key ) const
```

Checks, whether the **Hashtable** contains a certain key.

Parameters

key Pointer to the key to look up.

Returns

true if the specified key was found, false otherwise.

§ removeAllElements()

```
void removeAllElements ( void )
```

Clears the **Hashtable**, which means deleting all its content.

Returns

nothing.

See also

[remove\(\)](#)

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

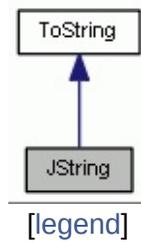
Returns

a **JString** representation of the instance and its contents for debugging purposes.

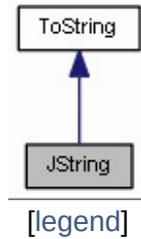
Implements **ToString**.

JString Class Reference

Inheritance diagram for JString:



Collaboration diagram for JString:



Public Member Functions

JString (unsigned int buflen=0)

JString (const char *Value)

JString (const EG_CHAR *Value)

JString (const **JString** &Value)

JString (const **UTF8String** &Value)

JString (const **ANSIString** &Value)

~JString (void)

JString & **operator=** (const **JString** &Rhs)

JString & **operator=** (const char *Rhs)

JString & **operator=** (const EG_CHAR *Rhs)

JString & **operator=** (const **UTF8String** &Rhs)

JString & **operator=** (const **ANSIString** &Rhs)

JString & **operator=** (char Rhs)

JString & **operator=** (signed char Rhs)

JString & **operator=** (unsigned char Rhs)

JString & **operator=** (EG_CHAR Rhs)

JString & **operator=** (short aNum)

JString & **operator=** (unsigned short aNum)

JString & **operator=** (int aNum)

JString & **operator=** (unsigned int aNum)

JString & **operator=** (long aNum)

JString & **operator=** (unsigned long aNum)

JString & **operator=** (long long aNum)

JString & **operator=** (unsigned long long aNum)

JString & **operator=** (float aNum)

JString & **operator=** (double aNum)

JString & **operator=** (long double aNum)

JString & **operator=** (bool aBool)

operator const EG_CHAR * (void) const

JString & **operator+=** (const **JString** &Rhs)

template<typename Etype >

JString & **operator+=** (const Etype &Rhs)

bool **operator==** (const **JString** &Rhs) const

bool **operator!=** (const **JString** &Rhs) const

bool **operator<** (const **JString** &Rhs) const

bool **operator>** (const **JString** &Rhs) const

bool **operator<=** (const **JString** &Rhs) const

bool **operator>=** (const **JString** &Rhs) const

EG_CHAR **operator[]** (unsigned int Index) const

EG_CHAR & **operator[]** (unsigned int Index)

unsigned int **capacity** (void) const

EG_CHAR **charAt** (unsigned int index) const

int **compareTo** (const **JString** &anotherString) const

const **JString** & **concat** (const **JString** &str)

const EG_CHAR * **cstr** (void) const

JString **deleteChars** (unsigned int start, unsigned int length) const

bool **endsWith** (const **JString** &suffix) const

void **ensureCapacity** (unsigned int minCapacity)

bool **equals** (const **JString** &anotherString) const

bool **equalsIgnoreCase** (const **JString** &anotherString) const

int **indexOf** (char ch) const

int **indexOf** (char ch, unsigned int fromIndex) const

int **indexOf** (EG_CHAR ch) const

int **indexOf** (EG_CHAR ch, unsigned int fromIndex)

const

int **indexOf** (const **JString** &str) const

int **indexOf** (const **JString** &str, unsigned int fromIndex) const

int **lastIndexOf** (char ch) const

int **lastIndexOf** (char ch, unsigned int fromIndex) const

int **lastIndexOf** (EG_CHAR ch) const

int **lastIndexOf** (EG_CHAR ch, unsigned int fromIndex) const

int **lastIndexOf** (const **JString** &str) const

int **lastIndexOf** (const **JString** &str, unsigned int fromIndex) const

unsigned int **length** (void) const

JString **replace** (char oldChar, char newChar) const

JString **replace** (EG_CHAR oldChar, EG_CHAR newChar) const

JString **replace** (const **JString** &match, const **JString** &replacement) const

bool **startsWith** (const **JString** &prefix) const

bool **startsWith** (const **JString** &prefix, unsigned int offset) const

JString **substring** (unsigned int beginIndex) const

JString **substring** (unsigned int beginIndex, unsigned int endIndex) const

JString **toLowerCase** (void) const

JString **toUpperCase** (void) const

int **toInt** (void) const

JString **trim** (void)

UTF8String **UTF8Representation** (void) const

ANSIString **ANSIRepresentation** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false)
const

► **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Related Functions

(Note that these are not member functions.)

```
template<typename _Elem , typename _Traits >  
std::basic_ostream< _Elem, _Traits > & operator<<  
 (::std::basic_ostream< _Elem,  
_Traits > &stream, const  
JString &string)
```

```
template<typename Etype >  
  
JString operator+ (const JString  
&Lsh, const Etype &Rsh)
```

```
template<typename Etype >  
  
JString operator+ (const Etype &Lsh,  
const JString &Rsh)
```

```
template<typename Etype >  
  
bool operator== (const JString  
&Lsh, const Etype &Rsh)
```

```
template<typename Etype >  
  
bool operator== (const Etype &Lsh,  
const JString &Rsh)
```

```
template<typename Etype >  
  
bool operator!= (const JString  
&Lsh, const Etype &Rsh)
```

```
template<typename Etype >  
  
bool operator!= (const Etype &Lsh,  
const JString &Rsh)
```

```
template<typename Etype >  
  
bool operator< (const JString
```

&Lsh, const Etype &Rsh)

template<typename Etype >

bool **operator<** (const Etype &Lsh,
const **JString** &Rsh)

template<typename Etype >

bool **operator>** (const **JString**
&Lsh, const Etype &Rsh)

template<typename Etype >

bool **operator>** (const Etype &Lsh,
const **JString** &Rsh)

template<typename Etype >

bool **operator<=** (const **JString**
&Lsh, const Etype &Rsh)

template<typename Etype >

bool **operator<=** (const Etype &Lsh,
const **JString** &Rsh)

template<typename Etype >

bool **operator>=** (const **JString**
&Lsh, const Etype &Rsh)

template<typename Etype >

bool **operator>=** (const Etype &Lsh,
const **JString** &Rsh)

JString **operator+** (const **JString**
&Lsh, const **JString** &Rsh)

Detailed Description

The **JString** class is a representation of Text strings, based on the String class from Sun Java.

This class is used to avoid dealing with char pointers/arrays directly, while staying independent from the String class in the Standard Template Library of C++, as some compilers do not implement the STL.

Constructor & Destructor Documentation

§ JString() [1/6]

JString (unsigned int `bufferlen = 0`)

explicit

Constructor: Creates an empty **JString**.

Remarks

By default no memory is allocated for the internal buffer. You can however pass the number of characters to allocate memory for. If that number is too big, then you will waste memory, but with a reasonable `bufferlen` you can avoid expensive later reallocations, when appending to the string.

Parameters

bufferlen optional, let the string allocate memory for x characters

§ JString() [2/6]

JString (const char * **Value**)

Copy-Constructor: Creates a new **JString** from a deep copy of the argument string.

Parameters

Value The UTF8 string to copy.

§ JString() [3/6]

JString (const EG_CHAR * **Value**)

Copy-Constructor: Creates a new **JString** from a deep copy of the argument string.

Parameters

Value The UTF16 string to copy.

§ JString() [4/6]

JString (const **JString** & **Value**)

Copy-Constructor: Creates a new **JString** from a deep copy of the argument string.

Parameters

Value The **JString** to copy.

§ JString() [5/6]

JString (const **UTF8String** & **Value**)

Copy-Constructor: Creates a new **JString** from a deep copy of the argument string.

Parameters

Value The **UTF8String** to copy.

§ JString() [6/6]

JString (const **ANSIString** & **Value**)

Copy-Constructor: Creates a new **JString** from a deep copy of the argument string.

Parameters

Value :The **ANSIString** to copy.

§ ~JString()

~JString (void)

Destructor.

Member Function Documentation

§ operator=() [1/21]

JString & operator= (const **JString** & **Rhs**)

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [2/21]

JString & operator= (const char * **Rhs**)

operator=.

Makes a deep copy of its right operand (which is assumed to be encoded as UTF8) into its left operand.

This overwrites old data in the left operand.

§ operator=() [3/21]

```
JString & operator= ( const EG_CHAR * Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [4/21]

```
JString & operator= ( const UTF8String & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [5/21]

```
JString & operator= ( const ANSIString & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [6/21]

JString & operator= (char aChar)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [7/21]

JString & operator= (signed char aChar)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [8/21]

JString & operator= (unsigned char aChar)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [9/21]

JString & operator= (EG_CHAR aWideChar)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [10/21]

JString & operator= (short aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [11/21]

JString & operator= (unsigned short aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [12/21]

JString & operator= (int aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [13/21]

JString & operator= (unsigned int aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [14/21]

JString & operator= (long aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [15/21]

JString & operator= (unsigned long aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [16/21]

```
JString & operator= ( long long aNum )
```

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [17/21]

JString & operator= (unsigned long long aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [18/21]

JString & operator= (float aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [19/21]

JString & operator= (double aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [20/21]

JString & operator= (long double aNum)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator=() [21/21]

JString & operator= (bool aBool)

operator=.

saves a string representation of its right operand in its left operand.

This overwrites old data in the left operand.

§ operator const EG_CHAR *()

```
operator const EG_CHAR * ( void ) const
```

operator const EG_CHAR*.

Copies a pointer to the content of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator+=() [1/2]

JString & operator+= (const **JString** & **Rhs**)

operator+=.

Attaches its right operand to its left operand.

§ operator+=() [2/2]

JString & operator+= (const Etype & Rhs)

operator+=.

Attaches its right operand to its left operand.

§ operator==()

```
bool operator==( const JString & Rhs ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

§ operator!==()

```
bool operator!=( const JString & Rhs ) const
```

operator!==()

Returns

false, if both operands are equal, true otherwise.

§ operator<()

```
bool operator< ( const JString & Rhs ) const
```

operator<. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is less than right operand, false otherwise.

§ operator>()

```
bool operator> ( const JString & Rhs ) const
```

operator>. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is greater than right operand, false otherwise.

§ operator<=()

```
bool operator<= ( const JString & Rhs ) const
```

operator<=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is less than or equal to the right operand,
false otherwise.

§ operator>=()

```
bool operator>= ( const JString & Rhs ) const
```

operator>=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is greater than or equal to the right operand, false otherwise.

§ operator[]() [1/2]

```
EG_CHAR operator[] ( unsigned int index ) const
```

operator[]. Accesses the character of the string at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ operator[]() [2/2]

EG_CHAR & operator[] (unsigned int *index*)

operator[]. Accesses the character of the string at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes

§ capacity()

```
unsigned int capacity ( void ) const
```

Returns the current capacity of the **JString**.

Returns

the current capacity in characters.

§ charAt()

```
EG_CHAR charAt ( unsigned int index ) const
```

Returns the character of the **JString** at the passed index. This does not check for valid indexes and shows undefined behavior for invalid indexes!

Parameters

index the index of the element, that should be returned. Must not be bigger than the current size of the string!

Returns

the character at the passed index.

§ compareTo()

```
int compareTo ( const JString & anotherString ) const
```

Checks for lexicographical differences between the **JString**, the function is called for, and the passed **JString**.

Parameters

anotherString the string to compare to

Returns

0, if both strings are equal, < 0, if the string, this function is called for, is smaller than the passed string, > 0 otherwise.

§ concat()

```
const JString & concat ( const JString & str )
```

Attaches the passed string to the string, the function is called for.

Parameters

str the string to attach

Returns

the string, the function was called for, after the parameter string was attached to it.

§ cstr()

```
const EG_CHAR * cstr ( void ) const
```

Returns a pointer to an EG_CHAR array representation of the **JString**. The data, the pointer points to, is valid, as long as the **JString** instance is valid.

Returns

a pointer to an EG_CHAR array representation of the string.

§ deleteChars()

```
JString deleteChars ( unsigned int start,  
                      unsigned int length  
                      )                const
```

Deletes a substring inside a returned copy of the string. This does not affect the original string.

Parameters

start start of the substring

length length of the substring

Returns

a copy of the string, after deleting the specified substring from the copy, or an empty string for invalid parameters

§ endsWith()

```
bool endsWith ( const JString & suffix ) const
```

Checks, if the **JString**, this function is called for, ends with the passed string.

Parameters

suffix the string to check for, if the other one ends with it

Returns

true, if the string, the function is called for, ends with the passed string, false otherwise.

§ ensureCapacity()

```
void ensureCapacity ( unsigned int minCapacity )
```

Resizes the **JString** to the passed capacity, if its old capacity has been smaller. Most likely the whole **JString** has to be copied into new memory, so this is an expensive operation for huge JStrings. Call this function first, before you use **concat()**-function and/or +=-operators a lot of times on this JString-instance, to avoid multiple expensive resizes through appending.

Parameters

minCapacity the new capacity for the **JString** in number of characters.

Returns

nothing.

§ equals()

```
bool equals ( const JString & anotherString ) const
```

Checks, if the **JString**, this function is called for, is equal to the passed string. This function is case-sensitive.

Parameters

anotherString the string to check for, if it is equal to other one

Returns

true, if both strings are equal to each other, false otherwise.

§ equalsIgnoreCase()

```
bool equalsIgnoreCase ( const JString & anotherString ) const
```

Checks, if the **JString**, this function is called for, is, equal to the passed string. This function is not case-sensitive.

Parameters

anotherString the string to check for, if it is equal to other one

Returns

true, if both strings are equal to each other, false otherwise.

§ indexOf() [1/6]

```
int indexOf ( char ch ) const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the first character of the string and goes forward, until the end of the string is reached.

Parameters

ch the character to search for

Returns

the index of the first occurrence of the parameter or -1 if it could not be found at all

§ indexOf() [2/6]

```
int indexOf ( char      ch,  
             unsigned int fromIndex  
            )          const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the passed index and goes forward, until the end of the string is reached.

Parameters

ch the character to search for
fromIndex the index, to begin the search from

Returns

the index of the first occurrence of the first parameter or -1 if it could not be found at all

§ indexOf() [3/6]

```
int indexOf ( EG_CHAR ch ) const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the first character of the string and goes forward, until the end of the string is reached.

Parameters

ch the character to search for

Returns

the index of the first occurrence of the parameter or -1 if it could not be found at all

§ indexOf() [4/6]

```
int indexOf ( EG_CHAR  ch,  
             unsigned int fromIndex  
             )          const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the passed index and goes forward, until the end of the string is reached.

Parameters

ch the character to search for
fromIndex the index, to begin the search from

Returns

the index of the first occurrence of the first parameter or -1 if it could not be found at all

§ indexOf() [5/6]

```
int indexOf ( const JString & str ) const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the first character of the string and goes forward, until the end of the string is reached.

Parameters

str the string to search for

Returns

the index of the first occurrence of the parameter or -1 if it could not be found at all

§ indexOf() [6/6]

```
int indexOf ( const JString & str,  
             unsigned int  fromIndex  
             )              const
```

Returns the index of the first occurrence of the parameter in the string, the function is called for. Searching begins at the passed index and goes forward, until the end of the string is reached.

Parameters

str the string to search for
fromIndex the index, to begin the search from

Returns

the index of the first occurrence of the first parameter or -1 if it could not be found at all

§ lastIndexOf() [1/6]

```
int lastIndexOf ( char ch ) const
```

Returns the index of the last occurrence of the parameter in the string, the function is called on. Searching begins at the last character of the string and goes forward, until the start of the string is reached.

Parameters

ch the character to search for

Returns

the index of the last occurrence of the parameter or -1 if it could not be found at all

§ lastIndexOf() [2/6]

```
int lastIndexOf ( char      ch,  
                 unsigned int fromIndex  
                 )          const
```

Returns the index of the last occurrence of the parameter in the string, the function is called on. Searching begins at the passed index and goes backward, until the start of the string is reached.

Parameters

ch the character to search for
fromIndex the index, to begin the search from

Returns

the index of the last occurrence of the first parameter or -1 if it could not be found at all

§ lastIndexOf() [3/6]

```
int lastIndexOf ( EG_CHAR ch ) const
```

Returns the index of the last occurrence of the parameter in the string, the function is called on. Searching begins at the last character of the string and goes forward, until the start of the string is reached.

Parameters

ch the character to search for

Returns

the index of the last occurrence of the parameter or -1 if it could not be found at all

§ lastIndexOf() [4/6]

```
int lastIndexOf ( EG_CHAR  ch,  
                 unsigned int fromIndex  
                 )          const
```

Returns the index of the last occurrence of the parameter in the string, the function is called on. Searching begins at the passed index and goes backward, until the start of the string is reached.

Parameters

ch the character to search for
fromIndex the index, to begin the search from

Returns

the index of the last occurrence of the first parameter or -1 if it could not be found at all

§ lastIndexOf() [5/6]

```
int lastIndexOf ( const JString & str ) const
```

Returns the index of the last occurrence of the parameter in the string, the function is called for. Searching begins at the last character of the string and goes forward, until the start of the string is reached.

Parameters

str the string to search for

Returns

the index of the last occurrence of the parameter or -1 if it could not be found at all

§ lastIndexOf() [6/6]

```
int lastIndexOf ( const JString & str,  
                 unsigned int   fromIndex  
                 )               const
```

Returns the index of the last occurrence of the parameter in the string, the function is called for. Searching begins at the passed index and goes backward, until the start of the string is reached.

Parameters

str the string to search for
fromIndex the index, to begin the search from

Returns

the index of the last occurrence of the first parameter or -1 if it could not be found at all

§ length()

```
unsigned int length ( void ) const
```

Returns

the length of the string in characters

§ replace() [1/3]

```
JString replace ( char oldChar,  
                  char newChar  
                  )      const
```

Searches the string for all occurrences of parameter 1 and replaces them with parameter 2. The result of the replacements is returned as a new instance, while the original string stays unchanged.

Parameters

oldChar the character to search for

newChar the character to replace the other one with

Returns

a copy of the string, the function was called for, in which all occurrences of parameter 1 have been replaced with parameter 2.

§ replace() [2/3]

```
JString replace ( EG_CHAR oldChar,  
                  EG_CHAR newChar  
                  )          const
```

Searches the string for all occurrences of parameter 1 and replaces them with parameter 2. The result of the replacements is returned as a new instance, while the original string stays unchanged.

Parameters

oldChar the character to search for

newChar the character to replace oldChar with

Returns

a copy of the string, the function was called for, in which all occurrences of parameter 1 have been replaced with parameter 2.

§ replace() [3/3]

```
JString replace ( const JString & match,  
                  const JString & replacement  
                  ) const
```

Searches the string for all occurrences of parameter 1 and replaces them with parameter 2. The result of the replacements is returned as a new instance, while the original string stays unchanged.

Parameters

match the substring to search for
replacement the string to replace match with

Returns

a copy of the string, the function was called for, in which all occurrences of parameter 1 have been replaced with parameter 2.

§ startsWith() [1/2]

```
bool startsWith ( const JString & prefix ) const
```

Checks, if the string, begins with the passed prefix

Parameters

prefix the prefix to search for

Returns

true, if the string begins with the prefix, false otherwise

§ startsWith() [2/2]

```
bool startsWith ( const JString & prefix,  
                 unsigned int   offset  
                 )               const
```

Checks, if the substring of the string, starting at the passed index, begins with the passed prefix

Parameters

prefix the prefix to search for

offset start of the substring to check for

Returns

true, if the substring begins with the prefix, false otherwise

§ substring() [1/2]

```
JString substring ( unsigned int beginIndex ) const
```

Returns a substring of the string, beginning at the passed index

Parameters

beginIndex start of the substring to return

Returns

a substring, beginning at the passed index

§ substring() [2/2]

```
JString substring ( unsigned int beginIndex,  
                    unsigned int endIndex  
                    )                const
```

Returns a substring of the string, beginning at the first passed index and ending at the second one

Parameters

beginIndex index of the first character of the substring to return

endIndex index of the last character of the substring to return + 1

Remarks

This function will treat the second index as first one and vice versa, if the first is bigger than the second one.

Returns

a substring, beginning at the first passed index and ending at the second one

§ toLowerCase()

JString toLowerCase (void) const

Copies the string and changes all upper case characters in the copy to lower case. This does not affect the original string.

Returns

a lowercase copy of the string

§ toUpperCase()

```
JString toUpperCase ( void ) const
```

Copies the string and changes all lower case characters in the copy to upper case. This does not affect the original string.

Returns

an uppercase copy of the string

§ toInt()

```
int toInt ( void ) const
```

Converts the string into an integer. Conversion ends at the first character, that can not be interpreted as a number.

Returns

the integer value produced by interpreting the string as a number or 0 if it could not be interpreted

§ trim()

JString trim (void)

Removes all whitespaces at the start and end of the string. e.g.: L" Hello World! " -> "Hello World!"

Returns

the string without any whitespaces at its start or end.

§ UTF8Representation()

```
UTF8String UTF8Representation ( void ) const
```

Converts the string to UTF8 and returns the converted string. Use this, if you need to pass the **JString** to an API, which does not support wide strings. This is a non-lossy conversion.

§ ANSIREpresentation()

```
ANSIString ANSIREpresentation ( void ) const
```

Converts the string to ANSI, using the current locale, and returns the converted string. Use this, if you need to pass the **JString** to an API, which does not support Unicode. Attention: This is a lossy conversion, if any characters in the string are not supported by the current locale (which is most likely for characters not common in western languages)!

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Friends And Related Function Documentation

§ operator<<()

```
std::basic_ostream<
_Elem, _Traits > &
operator<<      ( ::std::basic_ostream< _Elem, _Traits > & stream,
                 const JString &                          string
                 )
```

operator<<.

Used to print the **JString** to a std::wostream.

§ operator+() [1/3]

```
JString operator+ ( const JString & Lsh,  
                  const Etype & Rsh  
                  )
```

related

operator+.

Adds its right operand to its left operand and returns the result as a new **JString**.

§ operator+() [2/3]

```
JString operator+ ( const Etype & Lsh,  
                  const JString & Rsh  
                  )
```

related

operator+.

Adds its right operand to its left operand and returns the result as a new **JString**.

§ operator==() [1/2]

```
bool operator==( const JString & Lsh,  
                 const Etype & Rsh  
                 )
```

related

operator==.

Returns

true, if both operands are equal, false otherwise.

§ operator==() [2/2]

```
bool operator==( const Etype & Lsh,  
                 const JString & Rsh  
                 )
```

related

operator==.

Returns

true, if both operands are equal, false otherwise.

§ operator!==() [1/2]

```
bool operator!=( const JString & Lsh,  
                const Etype & Rsh  
                )
```

related

operator!==().

Returns

false, if both operands are equal, true otherwise.

§ operator!==() [2/2]

```
bool operator!=( const Etype & Lsh,  
                const JString & Rsh  
                )
```

related

operator!==().

Returns

false, if both operands are equal, true otherwise.

§ operator<() [1/2]

```
bool operator< ( const JString & Lsh,  
                const Etype &   Rsh  
                )
```

related

operator<. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is less than right operand, false otherwise.

§ operator<() [2/2]

```
bool operator< ( const Etype & Lsh,  
                const JString & Rsh  
                )
```

related

operator<. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is less than right operand, false otherwise.

§ operator>() [1/2]

```
bool operator> ( const JString & Lsh,  
                const Etype &   Rsh  
                )
```

related

operator>. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is greater than right operand, false otherwise.

§ operator>() [2/2]

```
bool operator> ( const Etype & Lsh,  
               const JString & Rsh  
               )
```

related

operator>. The return value indicates the lexicographic relation between the operands.

Returns

true, if left operand is greater than right operand, false otherwise.

§ operator<=() [1/2]

```
bool operator<= ( const JString & Lsh,  
                 const Etype &   Rsh  
                 )
```

related

operator<=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is less than or equal to the right operand,
false otherwise.

§ operator<=() [2/2]

```
bool operator<= ( const Etype & Lsh,  
                 const JString & Rsh  
                 )
```

related

operator<=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is less than or equal to the right operand,
false otherwise.

§ operator>=() [1/2]

```
bool operator>= ( const JString & Lsh,  
                const Etype & Rsh  
                )
```

related

operator>=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is greater than or equal to the right operand, false otherwise.

§ operator>=() [2/2]

```
bool operator>= ( const Etype & Lsh,  
                const JString & Rsh  
                )
```

related

operator>=. The return value indicates the lexicographic relation between the operands.

Returns

true, if the left operand is greater than or equal to the right operand, false otherwise.

§ operator+() [3/3]

```
JString operator+ ( const JString & Lsh,  
                  const JString & Rsh  
                  )
```

related

operator+.

Adds its right operand to its left operand and returns the result as a new **JString**.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

ExitGames

Common

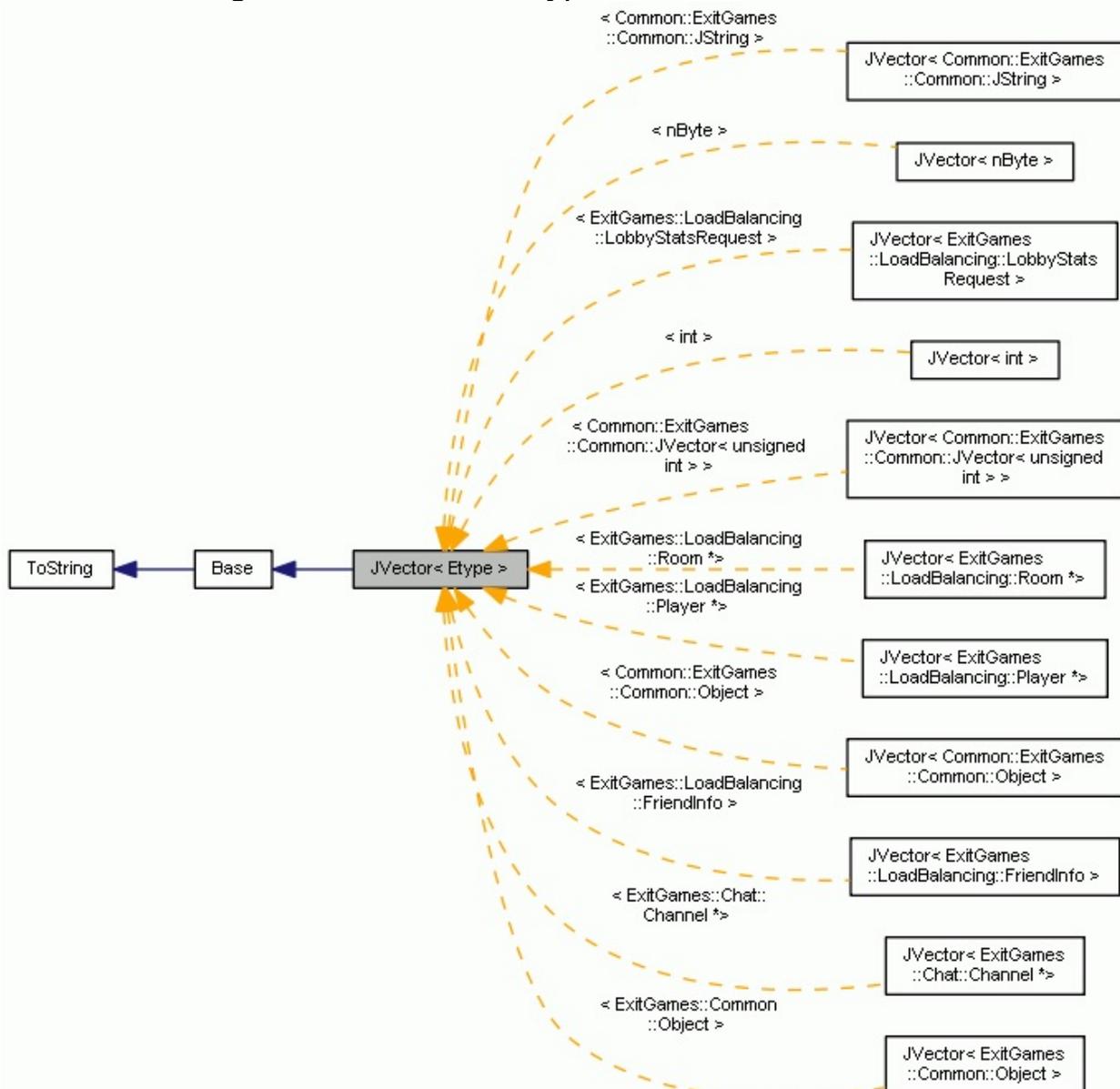
JVector

[Classes](#) | [Public Member Functions](#) |

[List of all members](#)

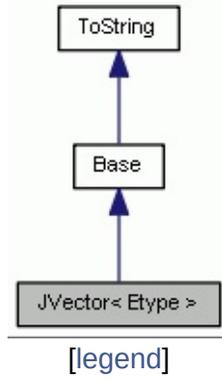
JVector< Etype > Class Template Reference

Inheritance diagram for JVector< Etype >:



[legend]

Collaboration diagram for JVector< Etype >:



Public Member Functions

JVector (unsigned int initialCapacity=0, unsigned int capacityIncrement=1)

JVector (const Etype *carray, unsigned int elementCount, unsigned int initialCapacity=0, unsigned int capacityIncrement=1)

virtual **~JVector** (void)

JVector (const **JVector**< Etype > &rhv)

JVector & **operator=** (const **JVector**< Etype > &rhv)

bool **operator==** (const **JVector**< Etype > &toCompare)
const

bool **operator!=** (const **JVector**< Etype > &toCompare)
const

const Etype & **operator[]** (unsigned int index) const

Etype & **operator[]** (unsigned int index)

unsigned int **getCapacity** (void) const

bool **contains** (const Etype &elem) const

const Etype & **getFirstElement** (void) const

int **getIndexOf** (const Etype &elem) const

bool **getIsEmpty** (void) const

const Etype & **getLastElement** (void) const

int **getLastIndexOf** (const Etype &elem) const

unsigned int **getSize** (void) const

const Etype * **getCArray** (void) const

void **copyInto** (Etype *array) const

void **addElement** (const Etype &obj)

void **addElements** (const **JVector**< Etype > &vector)

void **addElements** (const Etype *carray, unsigned int
elementCount)

void **ensureCapacity** (unsigned int minCapacity)

void **removeAllElements** (void)

bool **removeElement** (const Etype &obj)

void **trimToSize** (void)

const Etype & **getElementAt** (unsigned int index) const

void **insertElementAt** (const Etype &obj, unsigned int index)

void **removeElementAt** (unsigned int index)

void **setElementAt** (const Etype &obj, unsigned int index)

JString & **toString** (**JString** &retStr, bool withTypes=false) const

► **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

```
template<typename Etype>  
class ExitGames::Common::JVector< Etype >
```

This is a C++ implementation of the *Vector* Container class from Sun Java.

This class is based on the Java Vector class and as such contains all the public member functions of its Java equivalent. Unlike Java, typecasts are not necessary since C++ allows template instantiation of types at compile time. In addition to the Java public member functions, some operators were also added in order to take advantage of the operator overloading feature available in C++.

Constructor & Destructor Documentation

§ JVector() [1/3]

```
JVector ( unsigned int initialCapacity = 0,  
          unsigned int capacityIncrement = 1  
          )
```

Constructor.

Creates an empty **JVector** of elements of the type of the template parameter.

Parameters

- initialCapacity** the amount of elements, the **JVector** can take without need for resize. If you choose this too small, the **JVector** needs expensive resizes later (it's most likely, that the complete memory has to be copied to a new location on resize), if you choose it too big, you will waste much memory. The default is 40.
- capacityIncrement** Every time, when one adds an element to the Vector and it has no capacity left anymore, it's capacity will grow with this amount of elements on automatic resize. If you pass a too small value here, expensive resize will be needed more often, if you choose a too big one, possibly memory is wasted. The default is 10.

§ JVector() [2/3]

```
JVector ( const Etype * carray,  
          unsigned int  elementCount,  
          unsigned int  initialCapacity = 0,  
          unsigned int  capacityIncrement = 1  
          )
```

Constructor.

Creates a **JVector**, initialized with the passed carray the template parameter.

Parameters

- | | |
|--------------------------|---|
| carray | all elements of this array up to elementCount will get copied into the constructed instance |
| elementCount | shall not be greater than the actual element count of carray or undefined behavior will occur |
| initialCapacity | the amount of elements, the JVector can take without need for resize. Defaults to the value that gets passed for elementCount. If you choose this too small, the JVector needs expensive resizes later (it's most likely, that the complete memory has to be copied to a new location on resize), if you choose it too big, you will waste much memory. |
| capacityIncrement | Every time, when one adds an element to the Vector and it has no capacity left anymore, it's capacity will grow with this amount of elements on automatic resize. If you pass a too small value here, expensive resize will be needed more often, if you choose a too big one, possibly memory is wasted. The default is 10. |

§ ~JVector()

`~JVector (void)`

virtual

Destructor.

§ JVector() [3/3]

```
JVector ( const JVector< Etype > & toCopy )
```

Copy-Constructor.

Creates an object out of a deep copy of its parameter.

The parameter has to be of the same template overload as the object, you want to create.

Parameters

toCopy The object to copy.

Member Function Documentation

§ operator=()

```
JVector< Etype > & operator= ( const JVector< Etype > & toCopy )
```

operator=.

Makes a deep copy of its right operand into its left operand. Both operands have to be of the same template overload.

This overwrites old data in the left operand.

§ operator==()

```
bool operator==( const JVector< Etype > & toCompare ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise. Two instances are treated as equal, if they both contain the the same amount of elements and every element of one instance equals the other instance's element at the same index. If the element type is a pointer type, then the pointers are checked for equality, not the values, to which they point to.

§ operator!=()

```
bool operator!= ( const JVector< Etype > & toCompare ) const
```

operator!=.

Returns

false, if **operator==()** would return true, true otherwise.

§ operator[]() [1/2]

```
const Etype & operator[] ( unsigned int index ) const
```

operator[]. Wraps the function **getElementAt()**, so you have the same syntax like for arrays.

§ operator[]() [2/2]

Etype & operator[] (unsigned int *index*)

operator[]. Wraps the function **getElementAt()**, so you have the same syntax like for arrays.

§ getCapacity()

```
unsigned int getCapacity ( void ) const
```

Returns the current capacity of the **JVector**.

Returns

the current capacity.

§ contains()

```
bool contains ( const Etype & elem ) const
```

Checks, if the **JVector** contains the passed data as an element.

Parameters

elem a reference to the data, you want to check. Needs to be either a primitive type or an object of a class with an overloaded == operator.

Returns

true, if the element was found, false otherwise.

§ getFirstElement()

```
const Etype & getFirstElement ( void ) const
```

Returns the first element of the **JVector**. Shows undefined behavior for empty vectors.

Returns

the first element.

§ getIndexOf()

```
int getIndexOf ( const Etype & elem ) const
```

Searches the **JVector** from the first element in forward direction for the passed element and returns the first index, where it was found.

Parameters

elem the element, to search for.

Returns

the index of the first found of the passed element or -1, if the element could not be found at all.

§ getIsEmpty()

```
bool getIsEmpty ( void ) const
```

Checks, if the **JVector** is empty.

Returns

true, if the **JVector** is empty, or false, if it contains at least one element.

§ getLastElement()

```
const Etype & getLastElement ( void ) const
```

Returns the last element of the **JVector**. Shows undefined behavior for empty vectors.

Returns

the last element.

§ getLastIndexOf()

```
int getLastIndexOf ( const Etype & elem ) const
```

Searches the **JVector** from the last element in backward direction for the passed element and returns the first index, where it was found.

Parameters

elem the element, to search for.

Returns

the index of the first found of the passed element or -1, if the element could not be found at all.

§ getSize()

```
unsigned int getSize ( void ) const
```

Returns the size of the **JVector**.

Returns
the size.

§ `getCArray()`

```
const Etype * getCArray ( void ) const
```

Remarks

For a deep-copy `copyInto()` should be used. Use `getSize()` to find out the element count of the returned array.

Returns

a read-only pointer copy of the `Etype*`, that is internally used to store the elements.

§ copyInto()

```
void copyInto ( Etype * array ) const
```

Copies all elements of the **JVector** into the passed array. The caller has to make sure, that the array is big enough to take all elements of the vector, otherwise calling this function produces a buffer overflow.

Parameters

array an array of variables of the type of the template overload.

Returns

nothing.

§ addElement()

```
void addElement ( const Etype & elem )
```

Adds an element to the **JVector**. This automatically resizes the JVectors capacity to it's old size + the capacityIncrement, that you passed, when creating the vector (if you passed no value for capacityIncrement, then it was set to it's default value (see constructor doc)), if the size of the **JVector** has already reached it's capacity. When resizing occurs, then most likely the whole vector has to be copied to new memory. So this can be an expensive operation for huge vectors.

Note

When this function needs to increase the capacity, then all references/pointers to elements, that have been acquired before this function has been called, become invalid!

Parameters

elem the element to add.

Returns

nothing.

§ addElements() [1/2]

```
void addElements ( const JVector< Etype > & vector )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Calls the above function with `vector.getCArray()` and `vector.getSize()` as parameters

Parameters

vector the vector from which to copy the elements

§ addElements() [2/2]

```
void addElements ( const Etype * carray,  
                  unsigned int  elementCount  
                  )
```

Adds the first 'elementCount' elements of the provided array to the **JVector**. This automatically resizes the JVectors capacity to it's old size + 'elementCount', if the new size of the **JVector** is bigger than it's old capacity. When resizing occurs, then most likely the whole vector has to be copied to new memory. So this can be an expensive operation for huge vectors.

Note

When this function needs to increase the capacity, then all references/pointers to elements, that have been acquired before this function has been called, become invalid!

Parameters

carray the elements to add.
elementCount the number of elements to add - must not be greater than the size of carray.

Returns

nothing.

§ ensureCapacity()

```
void ensureCapacity ( unsigned int minCapacity )
```

Resizes the **JVector** to the passed capacity, if it's old capacity has been smaller. If resizing is needed, then the whole **JVector** has to be copied into new memory, so in that case this is an expensive operation for huge JVectors. Call this function, before you add a lot of elements to the vector, to avoid multiple expensive resizes through adding.

Note

When this function needs to increase the capacity, then all references/pointers to elements, that have been acquired before this function has been called, will get invalid!

Parameters

minCapacity the new capacity for the **JVector**.

Returns

nothing.

§ removeAllElements()

```
void removeAllElements ( void )
```

Clears the **JVector**.

Returns
nothing.

§ removeElement()

```
bool removeElement ( const Etype & obj )
```

Removes the passed element from the **JVector**.

Parameters

obj the element, to remove.

Returns

true, if the element has been removed, false, if it could not be found.

§ trimToSize()

```
void trimToSize ( void )
```

Trims the capacity of the **JVector** to the size, it currently uses. Call this function for a **JVector** with huge unused capacity, if you do not want to add further elements to it and if you are short on memory. This function copies the whole vector to new memory, so it is expensive for huge vectors. If you only add one element to the **JVector** later, it's copied again.

Note

Trimming a **JVector** instance (that isn't already optimally trimmed) will make all references/pointers to elements, that have been acquired before this function has been called, invalid!

§ getElementAt()

```
const Etype & getElementAt ( unsigned int index ) const
```

Returns the element of the **JVector** at the passed index. This does not check for valid indexes and shows undefined behavior for invalid indexes!

Parameters

index the index of the element, that should be returned. Must not be bigger than the current size of the vector!

Returns

the element at the passed index.

§ insertElementAt()

```
void insertElementAt ( const Etype & obj,  
                      unsigned int  index  
                      )
```

Inserts parameter one into the **JVector** at the index, passed as parameter two. Because all elements above or at the passed index have to be moved one position up, it is expensive, to insert an element at an low index into a huge **JVector**.

Parameters

obj the element, to insert.

index the position in the **JVector**, the element is inserted at.

Returns

nothing.

§ removeElementAt()

```
void removeElementAt ( unsigned int index )
```

Removes the element at the passed index from the **JVector**. Shows undefined behavior for invalid indexes.

Parameters

index the index of the element to remove.

Returns

nothing.

§ setElementAt()

```
void setElementAt ( const Etype & obj,  
                  unsigned int  index  
                  )
```

Sets the element at the passed index of the **JVector** to the passed new value. Shows undefined behavior for invalid indexes.

Parameters

obj the new value.

index the index of the element, which is set to the new value.

Returns

nothing.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

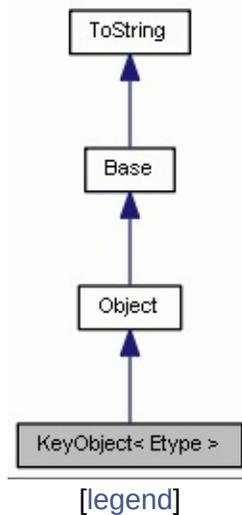
Returns

a **JString** representation of the instance and its contents for debugging purposes.

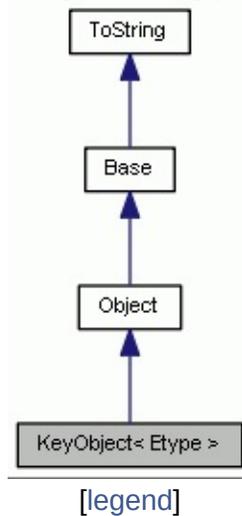
Implements **ToString**.

KeyObject< Etype > Class Template Reference

Inheritance diagram for KeyObject< Etype >:



Collaboration diagram for KeyObject< Etype >:



Public Member Functions

KeyObject (const **KeyObject**< Etype > &toCopy)

KeyObject (const **Object** &obj)

KeyObject (const **Object** *obj)

KeyObject (const typename
Helpers::ConfirmAllowedKey< Etype
>::type &data)

virtual **~KeyObject** (void)

virtual **KeyObject**< Etype > & **operator=** (const **KeyObject**< Etype > &toCopy)

virtual **KeyObject**< Etype > & **operator=** (const **Object** &toCopy)

Etype **getDataCopy** (void) const

Etype * **getDataAddress** (void) const

▶ Public Member Functions inherited from **Object**

Object (void)

virtual **~Object** (void)

Object (const **Object** &toCopy)

bool **operator==** (const **Object** &toCompare)
const

bool **operator!=** (const **Object** &toCompare)
const

nByte **getType** (void) const

nByte **getCustomType** (void) const

const short * **getSizes** (void) const

unsigned int **getDimensions** (void) const

JString & **toString** (**JString** &retStr, bool
withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

```
template<typename Etype>  
class ExitGames::Common::KeyObject< Etype >
```

Container class template for objects to be stored as keys in a **Hashtable** or **Dictionary**.

Remarks

In most cases the library will do the work of storing a key in a **KeyObject** for you, so for example you don't have to explicitly create an instance of this class, when storing a key-value pair in a **Dictionary** or **Hashtable** instance. However there are some situations, where you will receive instances of class **Object** or want to create them (for example **Hashtable::getKeys()** will return a **JVector<Object>**) and in that case casting those instances into **KeyObject**-instances can be a convenient way of assuring a type-safe access to their payloads.

Constructor & Destructor Documentation

§ KeyObject() [1/4]

```
KeyObject ( const KeyObject< Etype > & toCopy )
```

Copy-Constructor.

Creates an object out of a deep copy of its parameter.

The parameter has to be of the same template overload as the object, you want to create.

Parameters

toCopy The object to copy.

§ KeyObject() [2/4]

```
KeyObject ( const Object & obj )
```

Constructor.

Creates an object out of a deep copy of the passed **Object&**.

If the type of the content of the passed object does not match the template overload of the object to create, an empty object is created instead of a copy of the passed object, which leads to **getDataCopy()** and **getDataAddress()** return 0.

Parameters

obj The **Object&** to copy.

§ KeyObject() [3/4]

```
KeyObject ( const Object * obj )
```

Constructor.

Creates an object out of a deep copy of the passed Object*.

If the type of the content of the passed object does not match the template overload of the object to create, an empty object is created instead of a copy of the passed object, which leads to **getDataCopy()** and **getDataAddress()** return 0.

Parameters

obj The Object* to copy.

§ KeyObject() [4/4]

KeyObject (const typename Helpers::ConfirmAllowedKey< Etype >::typ

Constructor.

Creates an object out of a deep copy of the passed Etype.

Parameters

data The value to copy. Has to be of a supported type.

§ ~KeyObject()

`~KeyObject (void)`

virtual

Destructor.

Member Function Documentation

§ operator=() [1/2]

```
KeyObject< Etype > &  
operator= ( const KeyObject< Etype > & toCopy ) virtual
```

operator= : Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

§ operator=() [2/2]

```
KeyObject< Etype > & operator= ( const Object & toCopy )
```

virtual

operator= : Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

If the type of the content of the right operand does not match the template overload of the left operand, then the left operand stays unchanged.

Reimplemented from **Object**.

§ getDataCopy()

```
Etype getDataCopy ( void ) const
```

Returns a deep copy of the content of the object. If you only need access to the content, while the object still exists, you can use [getDataAddress\(\)](#) instead to avoid the deep copy. That is especially interesting for large content, of course.

If successful, the template overloads for array types of this function allocate the data for the copy, so you have to free (for arrays of primitive types) or delete (for arrays of class objects) it, as soon, as you do not need the array anymore. All non-array copies free there memory automatically, as soon as they leave their scope, same as the single indices of the array, as soon, as the array is freed.

In case of an error this function returns 0 for primitive return types and empty objects for classes.

Returns

a deep copy of the content of the object if successful, 0 or an empty object otherwise.

§ getDataAddress()

```
Etype * getDataAddress ( void ) const
```

Returns the address of the original content of the object. If you need access to the data above lifetime of the object, call [getDataCopy\(\)](#).

The return type is a pointer to the data, so it is a double-pointer, of course, for template overloads, which data already is a pointer.

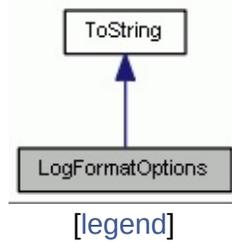
In case of an error, this function returns 0.

Returns

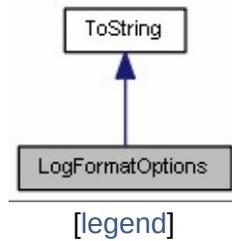
the address of the original content of the object, if successful, 0 otherwise.

LogFormatOptions Class Reference

Inheritance diagram for LogFormatOptions:



Collaboration diagram for LogFormatOptions:



Public Member Functions

bool **getAddDateTime** (void) const

LogFormatOptions & **setAddDateTime** (bool addTime)

bool **getAddLevel** (void) const

LogFormatOptions & **setAddLevel** (bool addLevel)

bool **getAddFile** (void) const

LogFormatOptions & **setAddFile** (bool addFile)

bool **getAddFunction** (void) const

LogFormatOptions & **setAddFunction** (bool addFunction)

unsigned int **getMaxNumberOfNamespaces** (void)
const

LogFormatOptions & **setMaxNumberOfNamespaces** (unsigned
int maxNumberOfNamespaces)

bool **getAddLine** (void) const

LogFormatOptions & **setAddLine** (bool addLine)

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool
withTypes=false) const

► **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString toString (bool withTypes=false) const

Detailed Description

Used to customize the formatting of the logging output that is generated by the **Logger** class. Each **Logger** instance holds its own **LogFormatOptions** instance that can be set through `Logger::setLogFormatOptions()`.

See also

`Logger::getFormatOptions()`, **`Logger::setFormatOptions()`**

Member Function Documentation

§ getAddDateTime()

```
bool getAddDateTime ( void ) const
```

Returns

'true', if log lines are prefixed with the date and time of the **EGLOG()** call, 'false' otherwise.

See also

setAddDateTime()

§ setAddDateTime()

LogFormatOptions & setAddDateTime (bool **addTime**)

Sets the value of the AddDateTime flag. The default value of this flag is 'true'.

Parameters

addTime 'true' instructs the **Logger** instance to prefix log lines with the date and time of the **EGLOG()** call, 'false' prevents it from doing so.

See also

getAddDateTime()

§ getAddLevel()

```
bool getAddLevel ( void ) const
```

Returns

'true', if log lines are prefixed with the **DebugLevel** of the **EGLOG()** call, 'false' otherwise.

See also

setAddLevel()

§ setAddLevel()

LogFormatOptions & setAddLevel (bool **addLevel**)

Sets the value of the AddLevel flag. The default value of this flag is 'true'.

Parameters

addLevel 'true' instructs the **Logger** instance to prefix log lines with the **DebugLevel** of the **EGLOG()** call, 'false' prevents it from doing so.

See also

getAddLevel()

§ getAddFile()

```
bool getAddFile ( void ) const
```

Returns

'true', if log lines are prefixed with the source file of the **EGLOG()** call, 'false' otherwise.

See also

[setAddFile\(\)](#)

§ setAddFile()

LogFormatOptions & setAddFile (bool **addFile**)

Sets the value of the AddFile flag. The default value of this flag is 'true'.

Parameters

addFile 'true' instructs the **Logger** instance to prefix log lines with the source file of the **EGLOG()** call, 'false' prevents it from doing so.

See also

[getAddFile\(\)](#)

§ `getAddFunction()`

```
bool getAddFunction ( void ) const
```

Returns

'true', if log lines are prefixed with the name of the function that did the **EGLOG()** call, 'false' otherwise.

See also

[setAddFunction\(\)](#)

§ setAddFunction()

LogFormatOptions & setAddFunction (bool **addFunction**)

Sets the value of the AddFunction flag. The default value of this flag is 'true'.

Parameters

addFunction 'true' instructs the **Logger** instance to prefix log lines with the name of the function that did the **EGLOG()** call, 'false' prevents it from doing so.

See also

getAddFunction()

§ getMaxNumberOfNamespaces()

```
unsigned int getMaxNumberOfNamespaces ( void ) const
```

Returns

the maximum number of namespaces that are included in the name of the function that did the **EGLOG()** call.

See also

[setMaxNumberOfNamespaces\(\)](#)

§ setMaxNumberOfNamespaces()

LogFormatOptions &
setMaxNumberOfNamespaces (unsigned int `maxNumberOfNamespaces`)

Sets the maximum number of namespaces that are included in the name function that did the **EGLOG()** call. The default value of this option is `UINT_MAX`, which means that all namespaces should be printed.

Note

The name of the class counts as a namespace in this context and in namespaces take precedence about outer namespaces.

Example:

The fully qualified name of the class member function includes 3 namespaces and the class name and is

`CompanyName::ProductName::ProjectName::ClassName::functionName()`

- If the `maxNumberOfNamespaces` is 0, then the printed function name will be `functionName()`.
- If the `maxNumberOfNamespaces` is 1, then the printed function name will be `ClassName::functionName()`.
- If the `maxNumberOfNamespaces` is 2, then the printed function name will be `ProjectName::ClassName::functionName()`.
- If the `maxNumberOfNamespaces` is 3, then the printed function name will be `ProductName::ProjectName::ClassName::functionName()`.
- If the `maxNumberOfNamespaces` is 4, then the printed function name will be `CompanyName::ProductName::ProjectName::ClassName::functionName()`.
- If the `maxNumberOfNamespaces` is 5, then the printed function name will still be `CompanyName::ProductName::ProjectName::ClassName::functionName()`.

Usually the values that make the most sense are:

- `UINT_MAX` (the fully qualified function name including all namespaces)
- 0 (just the function name itself)

- 1 (for class member functions this means the function name and the name, but no namespaces, while for free functions it means the function name and the name of the most inner namespace, but not the name any other namespaces)

Remarks

This option is only relevant when the values returned by [getAddFunction\(\)](#) is set to 'true' (which is the default).

Parameters

maxNumberOfNamespaces the maximum number of namespaces

See also

[getAddFunction\(\)](#), [setAddFunction\(\)](#), [getMaxNumberOfNamespaces\(\)](#)

§ `getAddLine()`

```
bool getAddLine ( void ) const
```

Returns

'true', if log lines are prefixed with the line of the **EGLOG()** call,
'false' otherwise.

See also

`setAddLine()`

§ setAddLine()

LogFormatOptions & setAddLine (bool **addLine**)

Sets the value of the AddLine flag. The default value of this flag is 'true'.

Parameters

addLine 'true' instructs the **Logger** instance to prefix log lines with the line of the **EGLOG()** call, 'false' prevents it from doing so.

See also

getAddLine()

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                    withTypes = false  
                    ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

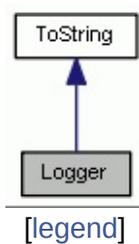
Returns

a **JString** representation of the instance and its contents for debugging purposes.

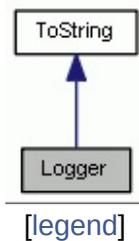
Implements **ToString**.

Logger Class Reference

Inheritance diagram for Logger:



Collaboration diagram for Logger:



Public Member Functions

Logger (int debugLevel=**DebugLevel::WARNINGS**)

void **log** (int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool printBrackets, unsigned int line, const EG_CHAR *dbgMsg,...) const

void **vlog** (int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool printBrackets, unsigned int line, const EG_CHAR *dbgMsg, va_list args) const

int **getDebugOutputLevel** (void) const

bool **setDebugOutputLevel** (int debugLevel)

void **setListener** (const **BaseListener** &listener)

const **LogFormatOptions** & **getFormatOptions** (void) const

void **setFormatOptions** (const **LogFormatOptions** &formatOptions)

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool withTypes=false) const

► Public Member Functions inherited from **Tostring**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Detailed Description

A **Logger** instance works as debugging API to send formatted strings to its current listener instance via the **EGLOG()**-macro.

Constructor & Destructor Documentation

§ Logger()

```
Logger (int debugLevel = DebugLevel1::WARNINGS)
```

Constructor: Creates a new **Logger** instance and sets the initial debug level.

Parameters

debugLevel the minimum debug level that a message must have to actually get logged by this **Logger** instance (the default is **DebugLevel::WARNINGS**)

Member Function Documentation

§ log()

```
void log ( int          debugLevel,  
          const EG_CHAR * file,  
          const EG_CHAR * function,  
          bool          printBrackets,  
          unsigned int  line,  
          const EG_CHAR * dbgMsg,  
          ...  
          )            const
```

Not intended for direct use - you should use the **EGLOG()** macro instead!

§ vlog()

```
void vlog ( int          debugLevel,  
           const EG_CHAR * file,  
           const EG_CHAR * function,  
           bool         printBrackets,  
           unsigned int  line,  
           const EG_CHAR * dbgMsg,  
           va_list       args  
         )              const
```

Not intended for direct use - you should use the **EGLOG()** macro instead!

§ `getDebugOutputLevel()`

```
int getDebugOutputLevel ( void ) const
```

Returns the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Returns

one of the values in `DebugLevel`

See also

`setDebugOutputLevel()`

§ `setDebugOutputLevel()`

```
bool setDebugOutputLevel ( int debugLevel )
```

Sets the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Parameters

`debugLevel` one of the values in `DebugLevel`

Returns

true if the new debug level has been set correctly, false otherwise.

See also

`getDebugOutputLevel()`

§ setListener()

```
void setListener ( const BaseListener & listener )
```

Sets parameter "listener" as receiver for all debug output information, which gets logged by the **Logger** instance, on which this function has been called.

Parameters

listener a reference to an instance of a subclass of **BaseListener**

Returns

true if the new debug level has been set correctly, false otherwise.

See also

getDebugOutputLevel()

§ getFormatOptions()

```
const LogFormatOptions & getFormatOptions ( void ) const
```

Returns

the **LogFormatOptions** that are used by this instance.

See also

setFormatOptions()

§ setFormatOptions()

```
void setFormatOptions ( const LogFormatOptions & formatOptions )
```

Sets the log format options to the supplied value.

Parameters

formatOptions the new value to which the log format options will be set

See also

[getFormatOptions\(\)](#)

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                               withTypes = false  
                    )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

- retStr** reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string
- withTypes** set to true, to include type information in the generated string

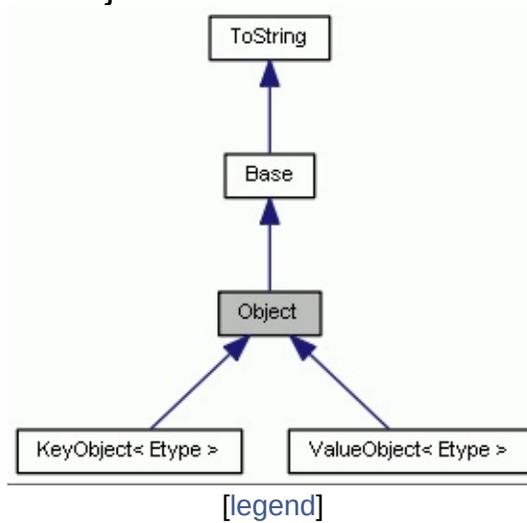
Returns

a **JString** representation of the instance and its contents for debugging purposes.

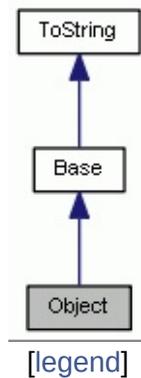
Implements **ToString**.

Object Class Reference

Inheritance diagram for Object:



Collaboration diagram for Object:



Public Member Functions

Object (void)

virtual **~Object** (void)

Object (const **Object** &toCopy)

virtual **Object** & **operator=** (const **Object** &toCopy)

bool **operator==** (const **Object** &toCompare) const

bool **operator!=** (const **Object** &toCompare) const

nByte **getType** (void) const

nByte **getCustomType** (void) const

const short * **getSizes** (void) const

unsigned int **getDimensions** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Container class designed to hold all types of objects that are supported by the library.

Object is the common base for the template classes **KeyObject** and **ValueObject**, that provide a more convenient interface for handling Objects.

Remarks

We do recommend to use **KeyObject** and **ValueObject** instead whenever possible, as they provide a more type-safe and more convenient interface for dealing with Objects. However in situations where an array or a container class holding multiple Objects of different types is absolutely needed, using the **Object** interface can be the only option.

See also

KeyObject, **ValueObject**

Constructor & Destructor Documentation

§ Object() [1/2]

Object (void)

Constructor: Creates an empty **Object**. You have to set the content with operator= before you can use the object.

§ ~Object()

`~Object (void)`

virtual

Destructor.

§ Object() [2/2]

Object (const **Object** & toCopy)

Copy-Constructor: Creates an **Object** containing a deep copy of the argument passed.

Parameters

toCopy The object to copy.

Member Function Documentation

§ operator=()

Object & operator= (const **Object** & toCopy)

virtual

operator= : Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

Reimplemented in **ValueObject< Etype >**, and **KeyObject< Etype >**.

§ operator==()

```
bool operator== ( const Object & toCompare ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

Two instances are considered equal, if all of the following is true:

- their types as returned by **getType()** match
- their payloads' dimension-counts as returned by **getDimensions()** match
- their custom types as returned by **getCustomType()** match
- their payload sizes as returned by **getSizes()** match on every dimension
- every element in every dimension of an instance's payload equals the according element in the other instance's payload (non-array payloads are handled as the first element in a 1D arrays with an element count of 1)

§ operator!==()

```
bool operator!=( const Object & toCompare ) const
```

operator!==()

Returns

false, if **operator==()** would return true, true otherwise.

§ getType()

```
nByte getType ( void ) const
```

Returns the type of the object.

The return value should be one of the constants representing the serialize-able data types supported by Neutron/Photon. Please refer to EG_Object for a complete list.

Returns

the type of the object.

§ getCustomType()

```
nByte getCustomType ( void ) const
```

Returns the type of the object.

This will return the custom type, if **getType()** returns EG_CUSTOM. If **getType()** returns something else than EG_CUSTOM, then the custom type is not in use for that object instance and this will return 0.

Returns

the custom type of a object.

§ `getSizes()`

```
const short * getSizes ( void ) const
```

Returns an array holding the amounts of elements of the instance's payload for each dimension of the payload. The amount of elements in the returned array of sizes will equal the return value of `getDimensions()`, but it will always be at least 1, even when `getDimensions()` returns 0. So, if the payload of the instance is not an array, then this function will return an array with 1 element, if the payload is a 1D array, then it will return an array with 1 element, for a 2D array payload it will return an array with 2 elements, for a 3D array payload an array with 3 elements and so on.

Returns

the sizes of all dimensions of the array contained in the **Object**.

§ getDimensions()

```
unsigned int getDimensions ( void ) const
```

Returns the amount of dimensions for objects holding multi-dimensional array data, 1 for single-dimensional arrays and 0 for non-array data.

Returns

the amount of dimensions for the data.

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++

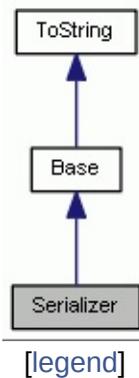
Client API 4.1.12.2

ExitGames > Common > Serializer >

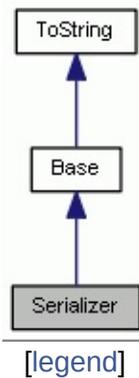
[Public Member Functions](#) | [List of all members](#)

Serializer Class Reference

Inheritance diagram for Serializer:



Collaboration diagram for Serializer:



Public Member Functions

const nByte * **getData** (void) const

int **getSize** (void) const

template<typename T >

bool **push** (const T &data)

template<typename T >

bool **push** (const T pData, typename
Helpers::ArrayLengthType< T >::type arraySize)

template<typename T >

bool **push** (const T pData, const short *arraySizes)

JString & **toString** (**JString** &retStr, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual ~**Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual ~**ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

The **Serializer** class serializes everything, that gets added to it, into a byte-array.

You can add data of **supported types** to a **Serializer** instance by calling **push()** on it, passing that data. A call to **push()** will automatically serialize the data before adding it to the instance. The content of a **Serializer** instance can be retrieved in form of a byte array by calling **getData()**.

If you do multiple calls to **push()** on the same instance, then the array retrieved by **getData()** will contain serialized representations of the passed data for all of them. The order in which these representations are stored will match the order of the calls.

The serialized data can be used to construct a **DeSerializer** instance from it, which provides the interface for retrieving the original datatypes from the byte array.

Remarks

The byte array that is returned by **getData()** has been serialized into a format, that can be deserialized by the **Photon** Server and other **Photon** Client platforms.

Member Function Documentation

§ getData()

```
const nByte * getData ( void ) const
```

Remarks

The size of the array, that's returned by this function, can be retrieved by calling **getSize()** on the the same instance without adding new data to the instance between the calls to this function and to **getSize()**.

Returns

the payload in form of a byte array

§ getSize()

```
int getSize ( void ) const
```

Returns

the size in bytes of the payload

§ push() [1/3]

```
bool push ( const T & data )
```

Adds a serialized representation of parameter data to the Serializer-instance on which it is called.

Template Parameters

T type of parameter data - has to be of one of the **supported datatypes**

Parameters

data data to serialize

Returns

true if successful, false in case of an error

§ push() [2/3]

```
bool push ( const T                                     pData,  
            typename Helpers::ArrayLengthType< T >::type arraySize  
            )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts singledimensional arrays and NULL-pointers passed for parameter pData.

Parameters

pData array of data to serialize
arraySize the size of the value array

§ push() [3/3]

```
bool push ( const T      pData,  
            const short * arraySizes  
            )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts multidimensional arrays and NULL-pointers passed for parameter val. The array, passed for parameter pData has to be a pointer of the correct abstraction level, meaning a normal pointer for a singledimensional array, a doublepointer for a twodimensional array, a triplepointer for a threedimensional array and so on.

Parameters

pData array of data to serialize

arraySizes the sizes for every dimension of the array

§ toString()

```
JString & toString ( JString & retStr,  
                    bool      withTypes = false  
                    )      const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implements **ToString**.



Photon C++

Client API 4.1.12.2

[ExitGames](#)

[Common](#)

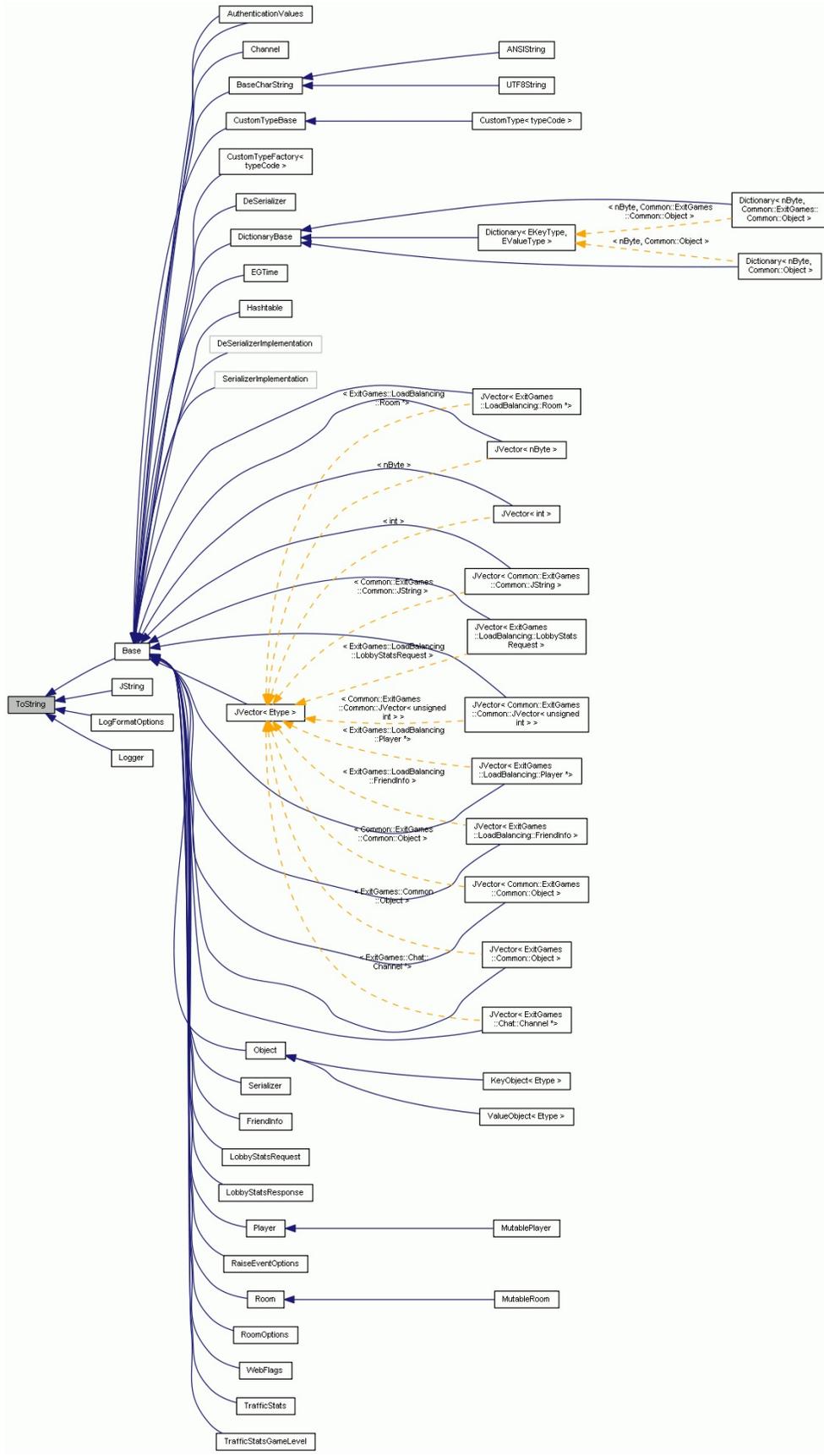
[ToString](#)

[Public Member Functions](#) | [List of all members](#)

ToString Class

Reference abstract

Inheritance diagram for ToString:



[legend]

Public Member Functions

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

virtual **JString** & **toString** (**JString** &retStr, bool withTypes=false)
const =0

JString **toString** (bool withTypes=false) const

Detailed Description

This class provides an interface for printing the payload of an instance of any subclass to a string.

Every subclass of this class will provide an implementation for **toString()** in its public interface and will therefor be printable. The implementations for container classes will include the output-strings of their elements into their own output string.

Constructor & Destructor Documentation

§ ~ToString()

`~ToString (void)`

virtual

Destructor.

Member Function Documentation

§ toString()

JString toString (void) const

virtual

Remarks

This function is intended for debugging purposes. For runtime type checking you should use RTTI's typeid() instead. Demangling and cutting off of namespaces will only happen on platforms, which offer a system functionality for demangling.

Returns

a string representation of the class name of the polymorphically correct runtime class of the instance, on which it is called on, after this class name has been demangled and eventual namespaces have been removed.

Reimplemented in **Dictionary< EKeyType, EValueType >**, **Dictionary< nByte, Common::ExitGames::Common::Object >**, **Dictionary< nByte, Common::Object >**, and **DictionaryBase**.

§ toString() [1/2]

```
toString ( JString & retStr,  
          bool          withTypes = false  
          )             const
```

pure virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

- retStr** reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string
- withTypes** set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

Implemented in **JString**, **JVector< Etype >**, **JVector< Common::ExitGames::Common::JString >**, **JVector< nByte >**, **JVector< ExitGames::LoadBalancing::LobbyStatsRequest >**, **JVector< int >**, **JVector< Common::ExitGames::Common::JVector< unsigned int > >**, **JVector< ExitGames::LoadBalancing::Room *>**, **JVector< ExitGames::LoadBalancing::Player *>**, **JVector< Common::ExitGames::Common::Object >**, **JVector< ExitGames::LoadBalancing::FriendInfo >**, **JVector< ExitGames::Chat::Channel *>**, **JVector< ExitGames::Common::Object >**, **RoomOptions**, **Dictionary<**

**EKeyType, EValueType >, Dictionary< nByte,
Common::ExitGames::Common::Object >, Dictionary< nByte,
Common::Object >, Hashtable, TrafficStatsGameLevel,
RaiseEventOptions, Object, EGTime, TrafficStats,
AuthenticationValues, AuthenticationValues, Player,
LogFormatOptions, Room, Channel, Logger, WebFlags,
DictionaryBase, BaseCharString, CustomTypeFactory< typeCode
>, Serializer, FriendInfo, LobbyStatsRequest,
LobbyStatsResponse, and DeSerializer.**

§ toString() [2/2]

```
JString toString ( bool withTypes = false ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

withTypes set to true, to include type information in the generated string

Returns

a **JString** representation of the instance and its contents for debugging purposes.

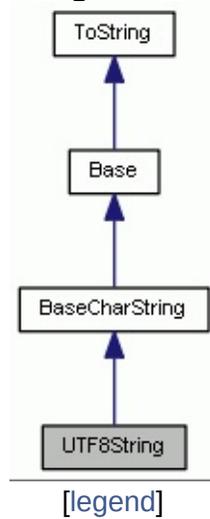
See also

JString

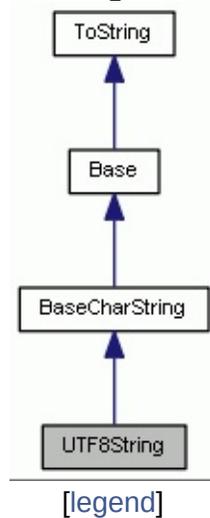
UTF8String Class Reference

[Classes](#) | [Public Member Functions](#) |
[Static Public Member Functions](#) |
[List of all members](#)

Inheritance diagram for UTF8String:



Collaboration diagram for UTF8String:



Public Member Functions

UTF8String (void)

UTF8String (const **UTF8String** &str)

UTF8String (const **JString** &str)

UTF8String (const char *str)

UTF8String (const EG_CHAR *str)

~UTF8String (void)

UTF8String & **operator=** (const **UTF8String** &Rhs)

UTF8String & **operator=** (const **JString** &Rhs)

UTF8String & **operator=** (const char *Rhs)

UTF8String & **operator=** (const EG_CHAR *Rhs)

operator const char * (void) const

operator JString (void) const

JString **JStringRepresentation** (void) const

unsigned int **size** (void) const

► **Public Member Functions inherited from BaseCharString**

BaseCharString ()

virtual **~BaseCharString** (void)

const char * **cstr** (void) const

unsigned int **length** (void) const

JString & **toString** (**JString** &retStr, bool withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual ~**Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual ~**ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Static Public Member Functions

static unsigned int **size** (const **JString** &str)

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener** *baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const **LogFormatOptions** &options)

Detailed Description

The **UTF8String** class is a container class for char* strings, encoded with UTF8.

This is the UTF8 implementation of **BaseCharString**. Please look at the doc of the abstract base class for more information.

Constructor & Destructor Documentation

§ UTF8String() [1/5]

UTF8String (void)

Constructor: Creates an empty **UTF8String**.

§ UTF8String() [2/5]

```
UTF8String ( const UTF8String & wstr )
```

Copy-Constructor: Creates a new **UTF8String** from a deep copy of the argument string.

Parameters

wstr The **UTF8String** to copy.

§ UTF8String() [3/5]

UTF8String (const **JString** & **wstr**)

Copy-Constructor: Creates a new **UTF8String** from a deep copy of the argument string.

Parameters

wstr The **JString** to copy.

§ UTF8String() [4/5]

UTF8String (const char * **str**)

Copy-Constructor: Creates a new **UTF8String** from a deep copy of the argument string.

Parameters

str The UTF8 string to copy.

§ UTF8String() [5/5]

```
UTF8String ( const EG_CHAR * wstr )
```

Copy-Constructor: Creates a new **UTF8String** from a deep copy of the argument string.

Parameters

wstr The Unicode string to copy.

§ ~UTF8String()

~UTF8String (void)

Destructor.

Member Function Documentation

§ operator=() [1/4]

```
UTF8String & operator= ( const UTF8String & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [2/4]

```
UTF8String & operator= ( const JString & Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [3/4]

```
UTF8String & operator= ( const char * Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator=() [4/4]

```
UTF8String & operator= ( const EG_CHAR * Rhs )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator const char *()

```
operator const char * ( void ) const
```

virtual

operator const char*.

Copies a pointer to the content of its right operand into its left operand.

This overwrites old data in the left operand.

Implements **BaseCharString**.

§ operator JString()

operator **JString** (void) const

virtual

operator **JString**.

Copies a **JString** representation of its right operand into its left operand.

This overwrites old data in the left operand.

Implements **BaseCharString**.

§ JStringRepresentation()

JString JStringRepresentation (void) const

virtual

Returns

a **JString** representation of the string.

Implements **BaseCharString**.

§ `size()` [1/2]

`unsigned int size (void) const`

virtual

The default implementation of this function will just return `length()`, but for multibyte strings like `UTF8String` the return values of `length()` and `size()` can differ.

Returns

the size of the string in bytes

Implements `BaseCharString`.

§ size() [2/2]

unsigned int size (const **JString** & str)

static

Parameters

str a **JString** instance

Returns

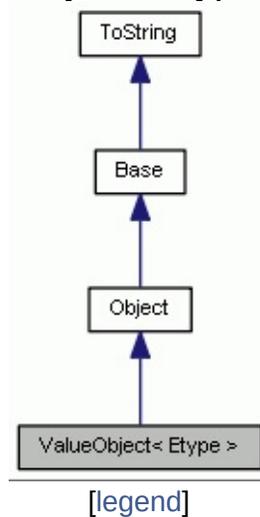
the size in bytes that an UTF8 representation of the provided **JString** instance would have

© Exit Games ®, all rights reserved.

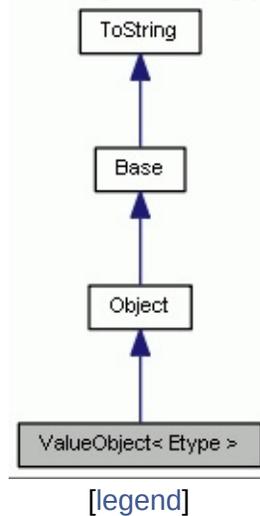
[Photon Documentation](#) | [Contact](#) | [Terms](#)

ValueObject< Etype > Class Template Reference

Inheritance diagram for ValueObject< Etype >:



Collaboration diagram for ValueObject< Etype >:



Public Member Functions

ValueObject (const **ValueObject**< Etype > &toCopy)

ValueObject (const **O** &obj)

ValueObject (const **O** *obj)

ValueObject (const typename
Helpers::ConfirmAllow
Etype >::type &data)

ValueObject (const typename
Helpers::ConfirmAllow
Etype >::type pData,
typename
Helpers::ArrayLength1
Etype >::type size)

ValueObject (const typename
Helpers::ConfirmAllow
Etype >::type pData, c
short *sizes)

virtual **~ValueObject** (void)

virtual **ValueObject**< Etype > & **operator=** (const **ValueObject**< Etype > &toCopy)

virtual **ValueObject**< Etype > & **operator=** (const **Obj**

&toCopy)

const Helpers::ArrayLengthType< Etype >::type * **getSizes** (void) const

Etype **getDataCopy** (void) c

Etype * **getDataAddress** (void) const

► Public Member Functions inherited from **Object**

Object (void)

virtual **~Object** (void)

Object (const **Object** &toCopy)

bool **operator==** (const **Object** &toCompare) const

bool **operator!=** (const **Object** &toCompare) const

nByte **getType** (void) const

nByte **getCustomType** (void) const

const short * **getSizes** (void) const

unsigned int **getDimensions** (void) const

JString & **toString** (**JString** &result, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToStdString**

virtual **~ToStdString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool
withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

```
template<typename Etype>  
class ExitGames::Common::ValueObject< Etype >
```

Container class template for objects to be stored as values in a **Hashtable** or **Dictionary**.

Remarks

In most cases the library will do the work of storing a value in a **ValueObject** for you, so for example you don't have to explicitly create an instance of this class, when storing a key-value pair in a **Dictionary** or **Hashtable** instance. However there are some situations, where you will receive instances of class **Object** or want to create them (for example **Hashtable::getValue()** will return an **Object**) and in that case casting those instances into **ValueObject**-instances can be a convenient way of assuring a type-safe access to their payloads.

Constructor & Destructor Documentation

§ ValueObject() [1/6]

```
ValueObject ( const ValueObject< Etype > & toCopy )
```

Copy-Constructor.

Creates an object out of a deep copy of its parameter.

The parameter has to be of the same template overload as the object, you want to create.

Parameters

toCopy The object to copy.

§ ValueObject() [2/6]

```
ValueObject ( const Object & obj )
```

Constructor.

Creates an object out of a deep copy of the passed **Object**&.

If the type of the content of the passed object does not match the template overload of the object to create, an empty object is created instead of a copy of the passed object, which leads to **getDataCopy()** and **getDataAddress()** returning 0.

Parameters

obj The **Object**& to copy.

§ ValueObject() [3/6]

```
ValueObject ( const Object * obj )
```

Constructor.

Creates an object out of a deep copy of the passed Object*.

If the type of the content of the passed object does not match the template overload of the object to create, an empty object is created instead of a copy of the passed object, which leads to **getDataCopy()** and **getDataAddress()** return 0.

Parameters

obj The Object* to copy.

§ ValueObject() [4/6]

ValueObject (const typename Helpers::ConfirmAllowed< Etype >::type &

Constructor.

Creates an object out of a deep copy of the passed single-value Etype.

Parameters

data The value to copy. Has to be of a supported type.

§ ValueObject() [5/6]

```
ValueObject ( const typename Helpers::ConfirmAllowed< Etype >::type  
               typename Helpers::ArrayLengthType< Etype >::type  
               )
```

Constructor.

Creates an object out of a deep copy of the passed single-dimensional E array.

Parameters

pData The array to copy.

size The element count of data.

§ ValueObject() [6/6]

```
ValueObject ( const typename Helpers::ConfirmAllowed< Etype >::type  
               const short *  
               )
```

Constructor.

Creates an object out of a deep copy of the passed multi-dimensional Et array.

Parameters

pData The array to copy.

sizes The array of element counts for the different dimensions of d

§ ~ValueObject()

~ValueObject (void)

virtual

Destructor.

Member Function Documentation

§ operator=() [1/2]

```
ValueObject< Etype >  
& operator= ( const ValueObject< Etype > & toCopy ) virtual
```

operator= : Makes a deep copy of its right operand into its left operand.
This overwrites old data in the left operand.

§ operator=() [2/2]

```
ValueObject< Etype > & operator= ( const Object & toCopy )
```

virtual

operator= : Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

If the type of the content of the right operand does not match the template overload of the left operand, then the left operand stays unchanged.

Reimplemented from **Object**.

§ getDataCopy()

```
Etype getDataCopy ( void ) const
```

Returns a deep copy of the content of the object. If you only need access to the content, while the object still exists, you can use [getDataAddress\(\)](#) instead to avoid the deep copy. That is especially interesting for large content, of course.

If successful, the template overloads for array types of this function allocate the data for the copy by calling [allocateArray<Etype>\(\)](#), so you have to call [deallocateArray\(\)](#) on it, as soon, as you do not need the array anymore. All non-array copies free their memory automatically, as soon as they leave their scope, same as the single indices of the array, as soon, as the array is freed.

In case of an error this function returns 0 for primitive return types and for arrays and an empty object for classes.

Returns

a deep copy of the content of the object if successful, 0 or an empty object otherwise.

§ getDataAddress()

```
Etype * getDataAddress ( void ) const
```

Returns the address of the original content of the object. If you need access to the data beyond the lifetime of the object, call **getDataCopy()** instead of this function.

The return type is a pointer to the data, so it is a double-pointer for template overloads, for which the data itself already is a pointer.

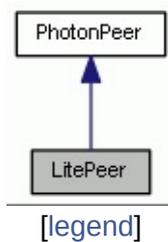
In case of an error, this function returns NULL.

Returns

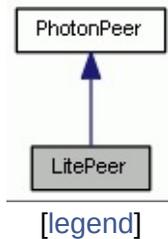
the address of the original content of the object, if successful, NULL otherwise.

LitePeer Class Reference

Inheritance diagram for LitePeer:



Collaboration diagram for LitePeer:



Public Member Functions

LitePeer (**Photon::PhotonList**
connectionProtocol=Photon::Cc

virtual **~LitePeer** (void)

template<typename Ftype >

bool **opRaiseEvent** (bool reliable, F
eventCode, nByte channelID=0
eventCaching=EventCache::DC
*targetPlayers=NULL, short nur
receiverGroup=ReceiverGroup:
interestGroup=0)

template<typename Ftype >

bool **opRaiseEvent** (bool reliable, F
typename Common::Helpers::A
>::type arrSize, nByte eventCoc
eventCaching=EventCache::DC
*targetPlayers=NULL, short nur
receiverGroup=ReceiverGroup:
interestGroup=0)

template<typename Ftype >

bool **opRaiseEvent** (bool reliable, F
short *pArrSizes, nByte eventC
nByte eventCaching=EventCac
int *targetPlayers=NULL, short
receiverGroup=ReceiverGroup:
interestGroup=0)

virtual bool **opJoin** (const **Common::JStri**
Common::Hashtable
&gameProperties=**Common::H**
Common::Hashtable
&actorProperties=**Common::H**
broadcastActorProperties=false

virtual bool **opLeave** (void)

virtual bool **opChangeGroups** (const **Common** *pGroupsToRemove, const **Common** *pGroupsToAdd)

virtual bool **opSetPropertiesOfActor** (int actorID, **Common::Hashtable** &properties, bool broadcast, int channelID=0)

virtual bool **opSetPropertiesOfGame** (**Common** &properties, bool broadcast, int channelID)

virtual bool **opGetProperties** (nByte channelID)

virtual bool **opGetPropertiesOfActor** (int actorID, **Common** *properties, short numProperties, **Common** *actorNrList=NULL, short numActors)

virtual bool **opGetPropertiesOfActor** (int actorID, **Common** numProperties, const int *actorNrList, short numActors=0, nByte channelID)

virtual bool **opGetPropertiesOfGame** (**Common** *properties, short numProperties, int channelID)

virtual bool **opGetPropertiesOfGame** (**Common** numProperties, nByte channelID)

► **Public Member Functions inherited from PhotonPeer**

PhotonPeer (**PhotonListener** listener, **ConnectionProtocol** connectionProtocol=Connector)

virtual **~PhotonPeer** (void)

virtual bool **connect** (const **Common::JST**

Common::JString &appID=Co

template<typename Ftype >

bool **connect** (const **Common::JString**
Common::JString &appID, coi

template<typename Ftype >

bool **connect** (const **Common::JString**
Common::JString &appID, coi
typename Common::Helpers::A
>::type arrSize)

template<typename Ftype >

bool **connect** (const **Common::JString**
Common::JString &appID, coi
const short *pArrSizes)

virtual void **disconnect** (void)

virtual void **service** (bool **dispatchIncomir**

virtual void **serviceBasic** (void)

virtual bool **opCustom** (const **OperationR**
bool sendReliable, nByte charr

virtual bool **sendOutgoingCommands** (vo

virtual bool **sendAcksOnly** (void)

virtual bool **dispatchIncomingCommands**

virtual bool **establishEncryption** (void)

virtual void **fetchServerTimestamp** (void)

virtual void **resetTrafficStats** (void)

virtual void **resetTrafficStatsMaximumCo**

virtual **Common::JString** **vitalStatsToString** (bool all) co

virtual void **pingServer** (const **Common::J**
int pingAttempts)

virtual void **initUserDataEncryption** (cons
> &secret)

virtual void **initUDPEncryption** (const **Cor**
&encryptSecret, const **Commo**
&HMACSecret)

PhotonListener * **getListener** (void)

int **getServerTimeOffset** (void) co

int **getServerTime** (void) const

int **getBytesOut** (void) const

int **getBytesIn** (void) const

int **getByteCountCurrentDispatc**

int **getByteCountLastOperation** (

int **getPeerState** (void) const

int **getSentCountAllowance** (void

void **setSentCountAllowance** (int s

int **getTimePingInterval** (void) cor

	void	setTimePingInterval	(int timeP
	int	getRoundTripTime	(void) cons
	int	getRoundTripTimeVariance	(v
	int	getTimestampOfLastSocketR	
	int	getDebugOutputLevel	(void) c
	bool	setDebugOutputLevel	(int deb
const Common::LogFormatOptions &		getLogFormatOptions	(void) c
	void	setLogFormatOptions	(const Common::LogFormatOptions
	int	getIncomingReliableCommar	
	short	getPeerID	(void) const
	int	getDisconnectTimeout	(void)
	void	setDisconnectTimeout	(int dis
	int	getQueuedIncomingComman	
	int	getQueuedOutgoingComman	
Common::JString		getServerAddress	(void) cons
	bool	getIsPayloadEncryptionAvail.	
	bool	getIsEncryptionAvailable	(voi
	int	getResentReliableCommands	

int **getLimitOfUnreliableCommar**

void **setLimitOfUnreliableCommar**

bool **getCRCEnabled** (void) const

void **setCRCEnabled** (bool crcEnab

int **getPacketLossByCRC** (void) c

bool **getTrafficStatsEnabled** (void)

void **setTrafficStatsEnabled** (bool t

int **getTrafficStatsElapsedMs** (vo

const **TrafficStats** & **getTrafficStatsIncoming** (void

const **TrafficStats** & **getTrafficStatsOutgoing** (void

const **TrafficStatsGameLevel** & **getTrafficStatsGameLevel** (vo

nByte **getQuickResendAttempts** (vo

void **setQuickResendAttempts** (nE

nByte **getConnectionProtocol** (void)

void **setConnectionProtocol** (nByte

nByte **getChannelCountUserChann**

Additional Inherited Members

▶ **Static Public Member Functions inherited from PhotonPeer**

static short **getPeerCount** (void)

static unsigned int **getMaxAppIDLength** (void)

Detailed Description

A **LitePeer** is an extended PhotonPeer and implements the operations offered by the **Lite** Application of the **Photon** Server SDK.

This class is used by many of our demos 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.

Constructor & Destructor Documentation

§ LitePeer()

```
LitePeer ( Photon::PhotonListener & listener,  
          nByte                               connectionProtocol = Photon::Co  
        )
```

Constructor

Parameters

listener Pointer to the application's implementation of `Photon::PhotonListener` interface. Has to be valid for at least the lifetime of the `LitePeer` instance, which is created by this constructor.

connectionProtocol Protocol to use to connect to `Photon`. One of `Photon::ConnectionProtocol`.

See also

`PhotonListener`

`Photon::ConnectionProtocol`

§ ~LitePeer()

~LitePeer (void)

virtual

Destructor.

Member Function Documentation

§ opRaiseEvent() [1/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool      reliable,
               Ftype     parameters,
               nByte     eventCode,
               nByte     channelId = 0,
               nByte     eventCaching = EventCache::DO_NOT_CACHE,
               const int * targetPlayers = NULL,
               short     numTargetPlayers = 0,
               nByte     receiverGroup = ReceiverGroup::OTHERS,
               nByte     interestGroup = 0
             )
```

Sends in-game data to all other players in the game, who will receive it in their [PhotonListener::onEvent\(\)](#) callback.

The eventCode should be used to define the event's type and content respectively. The payload has to be one of the datatypes that are listed as supported for values at [serializable datatypes](#). Receiving clients can access it with key `EventKey::DATA`.

This function provides the option to raise events reliable or unreliable. While both result in ordered events, the latter ones might be lost, causing gaps in the resulting event sequence. On the other hand, they cause less overhead and are optimal for data that is replaced soon.

Sending is not done immediately, but in intervals of [PhotonPeer::service\(\)](#) calls.

It is recommended to keep the payload as simple as possible, as the data is typically sent multiple times per second. This easily adds up to a huge amount of data otherwise.

As soon as the **Photon** Server acknowledged the reception of the **opRaiseEvent()** operation, the local application will be notified by a call to the **PhotonListener::onOperationResponse()** callback with the parameter `opCode` being set to `OperationCode::RAISE_EV`.

```
Hashtable ev;  
ev.put(POS_X, player.getPositionX());  
mPeer->opRaiseEvent(true, ev, eventCode);
```

Returns

true, if successful, false otherwise

See also

[PhotonListener::onEvent\(\)](#),
[PhotonListener::onOperationResponse\(\)](#), [Table of Datatypes](#)

Parameters

reliable	true = operation will be sent reliably; false = no resend in case of packet loss - will be ignored, when not using UDP as protocol
parameters	the payload of the event to raise - has to be provided in the form of one of the supported data types, specified at Table of Datatypes
eventCode	number for arbitrary classification of the type of event (like '1' for position updates, '2' for chat messages, and so on).
channelID	the logical channel, default is 0. See Fragmentation and Channels for more information.
eventCaching	has to be one of the constants specified in EventCache , default is <code>EventCache::DO_NOT_CACHE</code>
targetPlayers	the actorNrs of the clients, which should receive the event, set to NULL, to send the event to all actors in the room
numTargetPlayers	the number of actorNrs passed (array size)
receiverGroup	has to be one of the constants specified in ReceiverGroup , default is <code>ReceiverGroup::OTHERS</code>

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 is 0.

§ opRaiseEvent() [2/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool
                Ftype
                typename Common::Helpers::ArrayLengthType< Ftype >
                nByte
                nByte
                nByte
                const int *
                short
                nByte
                nByte
                )
```

This is an overloaded member function, provided for convenience. It differs in the number of argument(s) it accepts.

Parameters

reliable	true = operation will be sent reliably; false = no reliability when not using UDP as protocol
pParameterArray	the payload array of the event to raise - has to be one of the supported data types, specified at Table of supported data types
arrSize	the number of elements in pParameterArray
eventCode	number for arbitrary classification of the type of messages, and so on).
channelID	the logical channel, default is 0. See Fragmentation
eventCaching	has to be one of the constants specified in EventCache : DO_NOT_CACHE
targetPlayers	the actorNrs of the clients, which should receive the event
numTargetPlayers	the number of actorNrs passed (array size)

receiverGroup
interestGroup

has to be one of the constants specified in **Reci**
defines to which interest group the event is sent
groups. Group 0 is always sent to all, default is

§ opRaiseEvent() [3/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool          reliable,
               Ftype        pParameterArray,
               const short * pArrSizes,
               nByte        eventCode,
               nByte        channelID = 0,
               nByte        eventCaching = EventCache::DO_NOT_CACHE,
               const int *  targetPlayers = NULL,
               short        numTargetPlayers = 0,
               nByte        receiverGroup = ReceiverGroup::OTHERS,
               nByte        interestGroup = 0
             )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

reliable	true = operation will be sent reliably; false = no resend in case of packet loss - will be ignored, when not using UDP as protocol
pParameterArray	the payload array of the event to raise - has to be provided in the form of an array of one of the supported data types, specified at Table of Datatypes
pArrSizes	an array holding the number of elements for each dimension of pParameterArray
eventCode	number for arbitrary classification of the type of event (like '1' for position updates, '2' for chat messages, and so on).
channelID	the logical channel, default is 0. See Fragmentation and Channels for more

	information.
eventCaching	has to be one of the constants specified in EventCache , default is <code>EventCache::DO_NOT_CACHE</code>
targetPlayers	the actorNrs of the clients, which should receive the event, set to NULL, to send the event to all actors in the room
numTargetPlayers	the number of actorNrs passed (array size)
receiverGroup	has to be one of the constants specified in ReceiverGroup , default is <code>ReceiverGroup::OTHERS</code>
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 is 0.

§ opJoin()

```
bool
opJoin ( const Common::JString &   gameld,
         const Common::Hashtable & gameProperties = Common::Hashtable(),
         const Common::Hashtable & actorProperties = Common::Hashtable(),
         bool                      broadcastActorProperties = false
       )
```

This function joins the room with the given name on the **Photon** Server.

This operation will join an existing room by name or create one if the name has not been used yet.

Rooms (or games) are simply identified by name. **Lite** assumes that users will want to get into a room - no matter if it existed before or not, so it might be a good idea to create a room if it does not exist. If you want to make sure a room is created (new, empty), the client should 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 create a distinct one.

Each actor (a.k.a. player) in a room will get events that are raised for the player (if he is contained in the receiver list).

To distinguish the actors, each gets a consecutive actornumber. This is used in the return callback for operation Join. Number 1 is the lowest actornumber in the room and the client with that actornumber created the room.

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

Joining a room will result in a call to **PhotonListener::onOperationResponse**, the opCode being set to OPC_RT_JOIN. Joining a room will also trigger EV_RT_JOIN for all players in the room, to inform them about the new player.

Parameters

gameId	any ID string to identify the game
gameProperties	optional, set of game properties, by cor only used if game is new/created
actorProperties	optional, set of actor properties
broadcastActorProperties	true to broadcast actor proprties in join/ to not broadcast them, default is false

Returns

true, if successful, false otherwise

See also

[PhotonListener::onEvent\(\)](#), [PhotonListener::onOperationResponseLeave\(\)](#)

§ opLeave()

```
bool opLeave ( void )
```

virtual

Leaves a room, which has been previously joined with **opJoin()**.

Leaving a room will result in a call to **PhotonListener::onOperationResponse()** with the opCode being set to OPC_RT_LEAVE. This operation also triggers an event EV_RT_LEAVE for the remaining players in the room. This event includes the number of the player who left in key EV_RT_KEY_ACTORNR.

Returns

true, if successful, false otherwise

See also

PhotonListener::onEvent(),
PhotonListener::onOperationResponse(), **opJoin()**

§ opChangeGroups()

```
bool  
opChangeGroups ( const Common::JVector< nByte > * pGroupsToRemove  
                 const Common::JVector< nByte > * pGroupsToAdd  
                 )
```

Operation to handle this client's interest groups (for events inside rooms)

Note the difference between passing NULL and **&JVector<nByte>()**: NULL will not add/remove any groups. **&JVector<nByte>()** will add/remove all (existing). First, removing groups is executed. This way, you could leave all groups only the ones provided.

Parameters

pGroupsToRemove Groups to remove from interest. NULL will not remove any. **&JVector<nByte>()** will remove all.

pGroupsToAdd Groups to add to interest. NULL will not add any. **&JVector<nByte>()** will add all current.

Returns

true, if successful, false otherwise

§ opSetPropertiesOfActor()

```
bool  
opSetPropertiesOfActor ( int actorNr,  
                        const Common::Hashtable & properties,  
                        bool broadcast,  
                        nByte channelID = 0  
                        )
```

Adds or updates properties for the player, to whom the passed actorNr. belongs to

Parameters

- actorNr** the actorNr of the player for whom properties are being provided
- properties** the properties to add or update for this player. See **Photon Properties** for more information
- broadcast** passing true will send the event EV_SETPROPERTIES all other players in the game
- channelID** the channelIndex, see **Fragmentation and Channels**. Default is 0

Returns

true, if successful, false otherwise

See also

Photon Properties, **opGetPropertiesOfActor()**

§ opSetPropertiesOfGame()

```
bool  
opSetPropertiesOfGame ( const Common::Hashtable & properties,  
                        bool broadcast,  
                        nByte channelID = 0  
                      )
```

Adds or updates properties for the currently joined room.

Parameters

properties the properties to add or update for this room. See [Photon Properties](#) for more information

broadcast passing true will send the event EV_SETPROPERTIES other players in the game

channelID the channelIndex, see [Fragmentation and Channels](#). Default is 0

Returns

true, if successful, false otherwise

See also

[Photon Properties](#), [opGetPropertiesOfGame\(\)](#)

§ opGetProperties()

```
bool opGetProperties ( nByte channelID = 0 )
```

virtual

Creates a request to get all properties of the currently joined room and all players, which are inside it at the moment, when the server processes this operation. See [Photon Properties](#)

Parameters

channelID the channel index. See [Fragmentation and Channels](#)

Returns

true, if successful, false otherwise

See also

[Photon Properties](#)

§ opGetPropertiesOfActor() [1/2]

```
bool  
opGetPropertiesOfActor ( const Common::JString * properties,  
                        short numProperties,  
                        const int * actorNrList = NULL,  
                        short numActors = 0,  
                        nByte channelID = 0  
                        )
```

Creates a request to get the selected properties of the players with the specified actor numbers.

See [Photon Properties](#)

Parameters

- properties** an array of the key strings to the requested properties, pass NULL to get all properties for the requested actors
- numProperties** the number of the key strings passed (array size)
- actorNrList** the list of actorNrs of the players for whom to request properties, pass NULL to get the requested properties for all actors
- numActors** the number of actorNrs passed (array size)
- channelID** the channel index. See [Fragmentation and Channels](#)

Returns

true, if successful, false otherwise

See also

[Photon Properties](#), [opSetPropertiesOfActor\(\)](#)

§ opGetPropertiesOfActor() [2/2]

```
bool opGetPropertiesOfActor ( const nByte * properties,  
                             short          numProperties,  
                             const int *   actorNrList = NULL,  
                             short         numActors = 0,  
                             nByte        channelID = 0  
                             )
```

virtual

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- properties** an array of the byte keys to the requested properties, pass NULL to get all properties for the requested actors
- numProperties** the number of the key strings passed (array size)
- actorNrList** the list of actorNrs of the players for whom to request properties, pass NULL to get the requested properties for all actors
- numActors** the number of actorNrs passed (array size)
- channelID** the channel index. See [Fragmentation and Channels](#)

§ opGetPropertiesOfGame() [1/2]

```
bool  
opGetPropertiesOfGame ( const Common::JString * properties,  
                        short numProperties,  
                        nByte channelId = 0  
                      )
```

Creates a request to get the selected properties of the currently joined room.

See [Photon Properties](#)

Parameters

properties an array of the key strings of the properties to request, pass NULL to get all properties

numProperties the number of the key strings passed (array size)

channelID the channel index. See [Fragmentation and Channels](#)

Returns

true, if successful, false otherwise

See also

[Photon Properties](#), [opSetPropertiesOfGame\(\)](#)

§ opGetPropertiesOfGame() [2/2]

```
bool opGetPropertiesOfGame ( const nByte * properties,  
                             short        numProperties,  
                             nByte       channelId = 0  
                             )
```

virtual

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- properties** an array of the byte keys to the requested properties, pass NULL to get all properties
- numProperties** the number of the key bytes passed (array size)
- channelID** the channel index. See [Fragmentation and Channels](#)

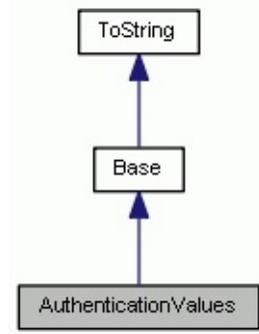
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > AuthenticationValues

[Public Member Functions](#) | [List of all members](#)

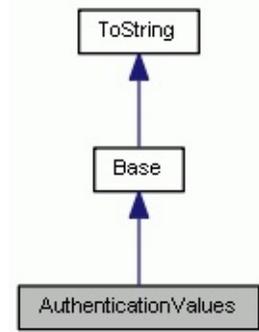
AuthenticationValues Class Reference

Inheritance diagram for AuthenticationValues:



[legend]

Collaboration diagram for AuthenticationValues:



[legend]

Public Member Functions

AuthenticationValues (void)

nByte **getType** (void) const

AuthenticationValues & **setType** (nByte type)

const **Common::JString** & **getParameters** (void) const

AuthenticationValues & **setParameters** (const **Common:**
¶meters)

AuthenticationValues & **setParametersWithUsernameAr**
(const **Common::JString** &usern
const **Common::JString** &token)

const **Common::JVector**< nByte > & **getData** (void) const

AuthenticationValues & **setData** (const **Common::JVecto**
nByte > &data)

const **Common::JString** & **getSecret** (void) const

const **Common::JString** & **getUserID** (void) const

AuthenticationValues & **setUserID** (const **Common::JStr**
&userID)

virtual **Common::JString** & **toString** (**Common::JString** &ret
withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString typeToString** (void) const

JString toString (bool withTypes=false) c

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Container for user authentication in **Photon**.

Remarks

On **Photon**, user authentication is optional but can be useful in many cases. If you want to use **Client::opFindFriends()**, a unique ID per user is very practical.

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

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

If you don't set a user ID through **setUserID()** for the **AuthenticationValues** instance that you pass to **Client::connect()**, then **Photon** generates a unique user ID (which fulfills the requirements of a GUID) for you, which can be retrieved through **Client::getUserID()**, once the **Client** instance has notified **Listener::connectReturn()** about having successfully finished the connection procedure. Once you have set a user ID, the **Client** instance caches it until you either override it or until the end of the lifetime of the **Client** instance.

To be able to rejoin a room and to be recognized there as the previous user it is critical to continue to use the same user ID.

Therefore you should store the user ID in permanent storage and set it to that same stored value whenever you want to connect as that user, even if you let **Photon** initially generate that ID. Otherwise **Photon** would generate a new user ID for you whenever you construct a new **Client** instance (i.e. when the user restarts your app).

Constructor & Destructor Documentation

§ AuthenticationValues()

AuthenticationValues (void)

Constructor.

Member Function Documentation

§ getType()

```
nByte getType ( void ) const
```

Returns

the type of the "Custom Authentication" service that will be used.

See also

[setType\(\)](#)

§ setType()

AuthenticationValues & setType (nByte type)

Sets the type of the "Custom Authentication" service that will be used. The initial value before the first call to this function is **CustomAuthenticationType::NONE**.

Note

Any custom authentication type aside from **CustomAuthenticationType::NONE** requires you to set up an authentication service of matching type for your appID at <https://www.photonengine.com/dashboard>

Parameters

type needs to match one of the values in **CustomAuthenticationType**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getType(), **CustomAuthenticationType**

§ getParameters()

```
const JString & getParameters ( void ) const
```

Returns

the HTTP GET parameters that will be forwarded to the authentication service.

See also

[setParameters\(\)](#), [setParametersWithUsernameAndToken\(\)](#),
[getData\(\)](#), [setData\(\)](#)

§ setParameters()

AuthenticationValues &
setParameters (const **Common::JString** & parameters)

Sets the HTTP GET parameters that will be forwarded to the authentication service to the provided parameters.

The provided parameter string must contain any (HTTP GET) parameters that are expected by the used authentication service.

Remarks

Standard HTTP GET parameters are used here and passed on to the authentication service that's defined for the provided authentication type in the **Photon** Cloud Dashboard.

Parameters

parameters needs to be a valid HTTP GET string (i.e. param1=value1¶m2=value2¶m3=value3)

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getParameters(), **setParametersWithUsernameAndToken()**, **getData()**, **setData()**

§ setParametersWithUsernameAndToken()

AuthenticationValues &

```
setParametersWithUsernameAndToken ( const Common::JString & username,
                                     const Common::JString & token
                                   )
```

Sets the HTTP GET parameters that will be forwarded to the authentication service to the provided username and token.

Calling this function is equivalent to `setParameters(Common::JString(L"username=") + username + "&token" + token)`.

Parameters

username the username of the user that should be authenticated

token the authentication token needed by the authentication service to verify the user

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getParameters\(\)](#), [setParameters\(\)](#), [getData\(\)](#), [setData\(\)](#)

§ getData()

```
const JVector< nByte > & getData ( void ) const
```

Returns

the HTTP POST data that will be forwarded to the authentication service.

See also

[getParameters\(\)](#), [setParameters\(\)](#),
[setParametersWithUsernameAndToken\(\)](#), [setData\(\)](#)

§ setData()

```
AuthenticationValues &  
setData ( const Common::JVector< nByte > & data )
```

Sets the HTTP POST data, that will be forwarded to the authentication service, to the provided data.

The provided data needs to match what is expected by the used authentication service.

Remarks

The provided data is passed on to the authentication service that's defined for the provided authentication type in the **Photon** Cloud Dashboard.

Parameters

data the data to be used in the body of the POST request.

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getParameters\(\)](#), [setParameters\(\)](#),
[setParametersWithUsernameAndToken\(\)](#), [getData\(\)](#)

§ getSecret()

```
const JString & getSecret ( void ) const
```

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

Remarks

This is publicly read-accessible only for debugging purposes. For normal operations it is entirely unnecessary for the app code to ever access this value.

Returns

the cached secret

§ getUserID()

```
const JString & getUserID ( void ) const
```

Returns

the unique user ID

See also

[setUserID\(\)](#)

§ `setUserID()`

AuthenticationValues &
`setUserID`

(const **Common::JString** & `userID`)

Sets the unique user ID.

Parameters

userID a string that needs to be unique per user among all users of your app

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getUserID\(\)](#)

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                               withTypes = false  
                    )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++

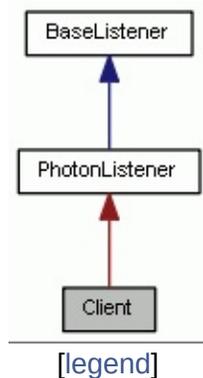
Client API 4.1.12.2

ExitGames > LoadBalancing > Client >

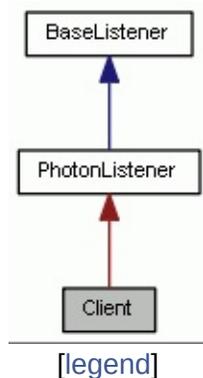
Client Class Reference

[Public Member Functions](#) |
[Static Public Member Functions](#) |
[List of all members](#)

Inheritance diagram for Client:



Collaboration diagram for Client:



Public Member Functions

		Client (LoadBalance Common::JString & Common::JString & connectionProtocol= bool autoLobbyStats regionSelectionMode bool useAlternativeP
	virtual	~Client (void)
	virtual bool	connect (const Auth &authenticationValue Common::JString & Common::JString & nByte serverType=S
	virtual void	disconnect (void)
	virtual void	service (bool dispat
	virtual void	serviceBasic (void)
	virtual bool	opCustom (const P &operationRequest, bool encrypt=false)
	virtual bool	sendOutgoingCom
	virtual bool	sendAcksOnly (voi
	virtual bool	dispatchIncomingC
	virtual void	fetchServerTimesta
	virtual void	resetTrafficStats (v

virtual void **resetTrafficStatsMa**

virtual **Common::JString** **vitalStatsToString** (

virtual bool **opJoinLobby** (const
&lobbyName=**Comm**
lobbyType=**LobbyTy**

virtual bool **opLeaveLobby** (voi

virtual bool **opCreateRoom** (con
RoomOptions &opt
Common::JVector<
&expectedUsers=**Cc**
(
)

virtual bool **opJoinOrCreateRo**
&gameID, const **Ro**
int cacheSliceIndex=
Common::JString >
Common::JString >

virtual bool **opJoinRoom** (const
rejoin=false, int cach
Common::JVector<
&expectedUsers=**Cc**
(
)

virtual bool **opJoinRandomRo**
&customRoomPrope
maxPlayers=0, nByt
matchmakingMode=
const **Common::JSi**
&lobbyName=**Comm**
lobbyType=**LobbyTy**
Common::JString &
const **Common::JV**
&expectedUsers=**Cc**

)

virtual bool **opLeaveRoom** (bool
sendAuthCookie=fal

template<typename Ftype >

bool **opRaiseEvent** (bool
nByte eventCode, cc
&options=**RaiseEve**

template<typename Ftype >

bool **opRaiseEvent** (bool
typename Common:
>::type arrSize, nByt
RaiseEventOptions

template<typename Ftype >

bool **opRaiseEvent** (bool
const short *pArrSize
RaiseEventOptions

virtual bool **opFindFriends** (con
short numFriendsToI

virtual bool **opLobbyStats** (con:
LoadBalancing::Lo
&lobbiesToQuery=**C**
LoadBalancing::Lo

virtual bool **opChangeGroups** (
*pGroupsToRemove
*pGroupsToAdd)

virtual bool **opCustomAuthenti**
AuthenticationValu

virtual bool **opWebRpc** (const C

template<typename Ftype >

bool **opWebRpc** (const C
Ftype ¶meters,

template<typename Ftype >

bool **opWebRpc** (const C
Ftype pParameterAr
Common::Helpers::A
bool sendAuthCooki

template<typename Ftype >

bool **opWebRpc** (const C
Ftype pParameterAr
sendAuthCookie=fal

virtual bool **selectRegion** (const

virtual bool **reconnectAndRejoi**

template<typename Ftype >

bool **sendDirect** (const F
fallbackRelay=false)

template<typename Ftype >

bool **sendDirect** (const F
Common::Helpers::A
int targetPlayer, bool

template<typename Ftype >

bool **sendDirect** (const F
*pArrSizes, int target

template<typename Ftype >

int **sendDirect** (const F
Common::JVector<
&targetPlayers=**Con**
fallbackRelay=false)

template<typename Ftype >

int **sendDirect** (const F
Common::Helpers::A
const **Common::JV**
&targetPlayers=**Con**
fallbackRelay=false)

template<typename Ftype >

int **sendDirect** (const F
*pArrSizes, const **Cc**
&targetPlayers=**Con**
fallbackRelay=false)

int **getServerTimeOffs**

int **getServerTime** (void)

int **getBytesOut** (void)

int **getBytesIn** (void) cc

int **getByteCountCurre**

int **getByteCountLastC**

int **getSentCountAllow**

void **setSentCountAllow**

int **getTimePingInterva**

void **setTimePingInterva**

int **getRoundTripTime**

int **getRoundTripTime**

int **getTimestampOfLa**

	int	getDebugOutputLe
	bool	setDebugOutputLe
const Common::LogFormatOptions &		getLogFormatOptic
	void	setLogFormatOptic Common::LogForm
	int	getIncomingReliab
	short	getPeerID (void) cor
	int	getDisconnectTime
	void	setDisconnectTime
	int	getQueuedIncomin
	int	getQueuedOutgoin
	bool	getIsPayloadEncry
	bool	getIsEncryptionAv
	int	getResentReliableC
	int	getLimitOfUnreliab
	void	setLimitOfUnreliab
	bool	getCRCEnabled (vc
	void	setCRCEnabled (bc
	int	getPacketLossByC

	bool	<code>getTrafficStatsEnal</code>
	void	<code>setTrafficStatsEnak</code>
	int	<code>getTrafficStatsElap</code>
const <code>Photon::TrafficStats</code> &		<code>getTrafficStatsInco</code>
const <code>Photon::TrafficStats</code> &		<code>getTrafficStatsOutg</code>
const <code>Photon::TrafficStatsGameLevel</code> &		<code>getTrafficStatsGam</code>
	nByte	<code>getQuickResendAt</code>
	void	<code>setQuickResendAt</code>
	nByte	<code>getChannelCountU</code>
	int	<code>getState</code> (void) cons
const <code>Common::JString</code> &		<code>getMasterserverAd</code>
	int	<code>getCountPlayersInq</code>
	int	<code>getCountGamesRu</code>
	int	<code>getCountPlayersOr</code>
	<code>MutableRoom</code> &	<code>getCurrentlyJoined</code>
const <code>Common::JVector< Room * ></code> &		<code>getRoomList</code> (void)
const <code>Common::JVector< Common::JString ></code> &		<code>getRoomNameList</code>
	bool	<code>getIsInRoom</code> (void)

bool **getIsInGameRoom**

bool **getIsInLobby** (void)

bool **getAutoJoinLobby**

void **setAutoJoinLobby**

MutablePlayer & **getLocalPlayer** (void)

const **Common::JVector**< **FriendInfo** > & **getFriendList** (void)

int **getFriendListAge** (void)

int **getDisconnectedCa**

const **Common::JString** & **getUserID** (void) const

const **Common::JString** & **getRegionWithBest**

Static Public Member Functions

static short **getPeerCount** (void)

Detailed Description

This class implements the Photon LoadBalancing work flow by using a **Peer**. It keeps a state and automatically executes transitions between the Master and Game Servers.

This class (and the **Player**, **MutablePlayer**, **Room** and **MutableRoom** classes) might be extended to implement your own logic.

However this is not necessary. You can also just put your game specific network logic into a class that uses this class as is, which is the recommended approach.

Override **MutableRoom:createPlayer()** when subclassing **Player**, **getMutablePlayerFactory() + MutablePlayerFactory::create() + MutablePlayerFactory::destroy()** when subclassing **MutablePlayer**, **createRoom()** when subclassing **Room** and **getMutableRoomFactory() + MutableRoomFactory::create() + MutableRoomFactory::destroy()** when subclassing **MutableRoom**.

Remarks

Extension notes: An extension of this class should override the functions that are inherited from **Photon::PhotonListener**, as they are called when the state changes. Call the base implementation first, then pick the operation response, event or state that you want to react to and put it in a switch-case.

We try to provide demos to each platform where this API can be used, so lookout for those.

Constructor & Destructor Documentation

§ Client()

```
Client ( LoadBalancing::Listener & listener,  
        const Common::JString & applicationID,  
        const Common::JString & appVersion,  
        nByte connectionProtocol = Photon::Conn  
        bool autoLobbyStats = false,  
        nByte regionSelectionMode = RegionSele  
        bool useAlternativePorts = false  
    )
```

Constructor.

Parameters

listener	Reference to the application's implementatic callback interface. Has to be valid for at least one Client instance, which is created by this constructor.
applicationID	A unique ID of your application. Must match the ID specified in your dashboard for Photon Cloud. This parameter is ignored by Photon Server.
appVersion	Only clients that use the exact same appVersion can connect to each other. You can use different values to separate clients from each other that should not be able to connect to each other or to even see each other, i.e. incompatible game or public, closed-beta, QA, staging and development. This parameter gets ignored by Photon Server.
connectionProtocol	The protocol to use to connect to the Photon Server. Must be one of the constants specified in ConnectionProtocol.
autoLobbyStats	Pass true, if you want to automatically receive lobby stats, false otherwise. Call opLobbyStats() to receive lobby stats update.
regionSelectionMode	Determines how the Photon Cloud Region to connect to should be selected. Must match or be different from the one specified in RegionSelectionMode. This parameter is ignored when connecting to Photon Server.

useAlternativePorts Determines if the the standard or the alterna be used. This parameter currently is only rel ConnectionProtocol::UDP is passed for para connectionProtocol and gets ignored otherw might block connections that use one port-ra connections that use the other, so when con fails, then you may want to try with the other

See also

[Listener](#), [ConnectionProtocol](#), [RegionSelectionMode](#), [NetworkPort](#)

§ ~Client()

`~Client (void)`

virtual

Destructor.

Member Function Documentation

§ connect()

```
bool
connect ( const AuthenticationValues & authenticationValues = Authent
          const Common::JString &      username = L"",
          const Common::JString &      serverAddress = M_NAMESERVER,
          nByte                          serverType = ServerType::NAME
        )
```

This function starts establishing a connection to a Photon server. The server address and port arrive in `Listener::connectReturn()`.

The connection is successfully established when the Photon client receives a response from the server. The connect-attempt fails when a network error occurs or the server is not responding. A call to this function starts an asynchronous operation. The result of the operation gets returned through the `Listener::connectReturn()` callback function. If it returns false, then the connect-attempt has already failed locally. If it returns true, then the connection was successful, when `Listener::connectReturn()` got called with `errorCode` set to 0.

Parameters

- | | |
|-----------------------------|--|
| authenticationValues | An instance of class AuthenticationValues |
| username | The users display name as shown to other users. Do not confuse with the users unique ID for identification purposes, which is part of the AuthenticationValues |
| serverAddress | A null terminated string containing the IP address and optionally the port number to connect to, in IPv4 or IPv6 format, examples: "192.168.0.1:5055", "udp.gameserver.com", "udp.gameserver.com:5055", "[2002:C0A8:1::]:5055". Note that IPv6 addresses use square brackets to indicate where the address and port begins. If no port is given, then the default port will be used. |
| serverType | One of the values in <code>ServerType</code> . Must match the server type. |

Photon server that is reachable at the given . Should be `ServerType::NAME_SERVER` for `ServerType::MASTER_SERVER` for self-hosted instances. You should **NOT** directly pass the master server with Photon Cloud, but always use the master server.

Returns

true, if it could successfully start establishing a connection (the result of the callback function in this case) or false, if an error occurred and the connection was not established (no callback function will be called then).

See also

[disconnect\(\)](#), `NetworkPort`

§ disconnect()

```
void disconnect ( void )
```

virtual

This function generates a disconnection request that will be sent to the Photon server. The servers response will arrive in `Listener::disconnectReturn()`.

If the disconnection is completed successfully, then the `Listener::disconnectReturn()` callback will be called.

Remarks

If a game room is joined, when this function gets called, then the local player leaves that room as if `opLeaveRoom()` has been called with parameter 'willComeBack' set to 'true'. Please see there for further information about leaving rooms. However no call to `Listener::leaveRoomReturn()` will happen when leaving a game room is triggered through a call to `disconnect()`.

See also

`connect()`, `opLeaveRoom()`

§ service()

```
void service ( bool dispatchIncomingCommands = true )
```

virtual

This function executes the PhotonPeer internal processes. Call this regularly!

This function is meant to be called frequently, like once per game loop. It handles the internal calls for keeping the PhotonPeer communication alive, and will take care of sending all local outgoing acknowledgements and messages, as well as dispatching incoming messages to the application and firing the corresponding callbacks. Internally **service()** calls the following functions:

1. **serviceBasic()**
2. **dispatchIncomingCommands()** (called withing a loop until all incoming commands have been dispatched.)
3. **sendOutgoingCommands()** (called withing a loop until everything queued for sending has been sent.)

service() is provided for convenience. If you need to tweak the performance, you can ignore **service()** and call its three subfunctions directly with individual time intervals, to gain more control over the internal communication process. For instance, calling **sendOutgoingCommands()** more rarely will result in less packets to be generated, as more commands will be accumulated into a single packet. See **sendOutgoingCommands()** for more information on efficiency.

For situations where you want to keep the connection alive, but can't process incoming messages (e.g. when loading a level), you can temporarily pass false for **dispatchIncomingCommands** to skip the calls to **dispatchIncomingCommands()**. Incoming commands will be stored in the incoming queue until they are dispatched again.

Parameters

dispatchIncomingCommands true = **dispatchIncomingCommands()**

will be called; false =
dispatchIncomingCommands()
won't be called, default is true

§ serviceBasic()

```
void serviceBasic ( void )
```

virtual

This function takes care of exchanging data with the system's network layer.

You only need to call this function in case you choose not to use `service()`, but call the subfunctions of `service()` directly. Please see the documentation of `service()` for more information.

`serviceBasic()` is called from within `service()`. If you decide not to use `service()`, then `serviceBasic()` needs to be called frequently, like once per game loop.

See also
`service()`

§ opCustom()

```
bool  
opCustom ( const Photon::OperationRequest & operationRequest,  
           bool sendReliable,  
           nByte channelId = 0,  
           bool encrypt = false  
           )
```

vir

Sends a custom operation to a custom Server, using reliable or unreliable Photon transmission.

Allows the client to send a custom operation to the Photon server (which has to be modified accordingly). The Server can be extended and modified for special purposes like server side collision detection or a consistent world.

You need to be connected (see [connect\(\)](#)) prior to calling [opCustom\(\)](#).

Parameters

- operationRequest** holds the payload of the operation
- sendReliable** = operation will be sent reliably; false = no reser in case of packet loss - will be ignored, when no using udp as protocol
- channelID** the logical channel, default is 0. See [Fragmentation and Channels](#) for more information.
- encrypt** true = encrypt message; false = no encryption

Returns

true, if successful, false otherwise

§ sendOutgoingCommands()

```
bool sendOutgoingCommands ( void )
```

virtual

This function initiates the transmission of outgoing commands.

Any Photon function that generates messages will store these messages as a "command" in an outgoing queue for later transmission. Commands can either be explicitly created operations generated for example by **opCustom()** or internally generated messages like acknowledgements for reliable messages from other players. **sendOutgoingCommands()** will initiate the data transmission by passing the outgoing commands to the system's sockets for immediate transmission.

In case of UDP **sendOutgoingCommands()** will also split the commands into multiple packets if needed and/of aggregate multiple commands together into one packet, if possible. Because of the latter calling **sendOutgoingCommands()** more rarely will result in less overhead, as there will be fewer packets for the clients to be sent and processed. The underlying platform can also limit the frequency in which outgoing packets can be sent and received. The downside of lower sending frequencies is a higher latency, until messages are exchanged and acknowledged, which may lead to a jerky gameplay.

To help you keeping track of the incoming and outgoing queues at development time and adjust your sending frequency, there will be a warning message sent to your **debugReturn** callback if a queue has exceeded the warning threshold.

Note

While **service()** is calling **serviceBasic()** implicitly, you will have to regularly call it yourself explicitly, when you use **sendOutgoingCommands()** and **dispatchIncomingCommands()** directly instead.

Usually you don't have to call **sendOutgoingCommands()** this explicitly, as this is done within **service()**.

See also
[service\(\)](#)

§ sendAcksOnly()

```
bool sendAcksOnly ( void )
```

virtual

Sends only ACKs (UDP) or Ping (TCP) instead of queued outgoing commands. Useful to pause sending actual data.

Note

While `service()` is calling `serviceBasic()` implicitly, you will have to regularly call it yourself explicitly, when you use `sendAcksOnly()` and `dispatchIncomingCommands()` instead.

§ `dispatchIncomingCommands()`

```
bool dispatchIncomingCommands ( void )
```

virtual

Checks for incoming commands waiting in the queue, and dispatches a single command to the application.

Dispatching means, that if the command is an operation response or an event, the appropriate callback function will be called).

`dispatchIncomingCommands()` will also take care of generating and queuing acknowledgments for incoming reliable commands. Please note that this function will only dispatch one command per all. If you want to dispatch every single command which is waiting in the queue, call `dispatchIncomingCommands()` within a while loop, until its return code is false.

Note

While **`service()`** is calling **`serviceBasic()`** implicitly, you will have to regularly call it yourself explicitly, when you use **`sendOutgoingCommands()`** and **`dispatchIncomingCommands()`** directly instead.

Returns

true if it has successfully dispatched a command, false otherwise (for example, when there has not been any command left in the queue, waiting for dispatching).

See also

`service()`

§ fetchServerTimestamp()

```
void fetchServerTimestamp ( void )
```

virtual

This will fetch the server's timestamp and update the approximation for **getServerTime()** and **getServerTimeOffset()**.

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 roundtriptime is low and the timestamp + roundtriptime / 2 is close to the original value.

§ resetTrafficStats()

```
void resetTrafficStats ( void )
```

virtual

Creates new instances of TrafficStats and starts a new timer for those.

§ resetTrafficStatsMaximumCounters()

```
void resetTrafficStatsMaximumCounters ( void )
```

virtual

Resets traffic stats values that can be maxed out.

§ vitalStatsToString()

Common::JString vitalStatsToString (bool **all**) const

virtual

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.

Parameters

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

Returns

stats as a string.

§ opJoinLobby()

```
bool  
opJoinLobby ( const Common::JString & lobbyName = Common::JString  
              nByte                               lobbyType = LobbyType::DEFAULT  
              )
```

Joins the specified lobby.

This function sends a request to the server to join the specified lobby. If it is successful, then `Listener::joinLobbyReturn()` gets called when the operation has successfully been finished. Please see [Matchmaking Guide](#) regarding the differences between the various lobby types.

Remarks

A **Client** instance can only be inside one room at a time. Therefore this operation will fail and return false, if the client is already inside another lobby or inside a game room. Leave the other room first, before calling this operation.

For the same reason entering a game room implicitly causes the client to leave the lobby, so if you want to return to the previously joined lobby after leaving that game room, you must explicitly join it again.

Note

If the auto-join lobby feature is enabled (which is the default! - it can be turned off by a call to `setAutoJoinLobby()`), then the client automatically joins the default lobby when successfully connecting to **Photon** and leaving a game room. Call `setAutoJoinLobby(false)` before calling `opJoinLobby()` to work properly.

Parameters

lobbyName the unique name of the lobby to join

lobbyType one of the values in **LobbyType**

Returns

true, if the request could successfully be queued for sending to the server; false otherwise.

See also

[opLeaveLobby\(\)](#), [setAutoJoinLobby\(\)](#), [getAutoJoinLobby\(\)](#),
[Listener::joinLobbyReturn\(\)](#)

§ opLeaveLobby()

```
bool opLeaveLobby ( void )
```

virtual

Leaves the currently joined lobby.

This function sends a request to the server to leave the currently joined lobby. If it returns true, then Listener::leaveLobbyReturn() gets called when the operation has successfully been finished.

Remarks

This operation will fail and return false if the client does not currently reside inside any lobby.

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

[opJoinLobby\(\)](#), Listener::leaveLobbyReturn()

§ opCreateRoom()

```
bool  
opCreateRoom ( const Common::JString &                               gan  
               const RoomOptions &                               opti  
               const Common::JVector< Common::JString > & exp  
               )
```

Creates and enters a new game room.

This function sends a request to the server to create the specified game room. `Listener::createRoomReturn()` gets called when the operation has been finished.

If you don't want to create a unique room name, pass `L""` as name and the server will generate a unique name for you. Room names are unique.

A room will be attached to the lobby that you have specified in the passed `RoomOptions`. If you are in no lobby, then the default lobby will be used.

Multiple lobbies can help to separate players by map or skill or game type (defined by name and type or as default).

Remarks

A **Client** instance can only be inside one room at a time. Therefore, it cannot be inside another game room. Any lobby the client currently resides in will be left. If a room with the specified name does already exist, then the operation will fail and be called with an error code.

Parameters

gameID The name to create a room with. Must be unique and not an empty string, then the server will assign a GUID as name.

options An instance of **RoomOptions**, that can be used to specify room options.

expectedUsers Sets a list of user IDs for which the server should reserve seats.

Returns

true, if the request could successfully be queued for sending to the server.

See also

[opJoinOrCreateRoom\(\)](#), [opJoinRoom\(\)](#), [opJoinRandomRoom\(\)](#),
[Listener::createRoomReturn\(\)](#)

§ opJoinOrCreateRoom()

```
bool  
opJoinOrCreateRoom ( const Common::JString &  
                    const RoomOptions &  
                    int  
                    const Common::JVector< Common::JString >  
                    )
```

Joins the specified game room or creates and enters a new game room

This function sends a request to the server to join the specified game room. `Listener::joinOrCreateRoomReturn()` gets called when the operation has

Unlike **opJoinRoom()**, this operation does not fail if the room does not exist before actually creating it: Any invited player (whoever is first) can call this

This operation does not allow you to re-join a game. To return to a room, use `opRejoinRoom()` previously.

Remarks

A **Client** instance can only be inside one room at a time. Therefore, if you try to join another game room, any lobby the client currently resides in will implicitly be closed. If the room is full or closed, then this operation will fail and `Listener::`

Parameters

- gameID** A unique identifier for the game room to join or create, and assign a GUID as name.
- options** An instance of **RoomOptions**, that can be used to specify options. If `options` is null, it will be ignored when the room already exists.
- cacheSliceIndex** Allows to request a specific cache slice - all event listeners will be notified after joining the room - see **Lite::EventManager**
- expectedUsers** Sets a list of user IDs for which the server should check if the room already exists, then this list will be merged with the existing list.

Returns

§ opJoinRoom()

```
bool  
opJoinRoom ( const Common::JString &          gamel  
             bool                               rejoin  
             int                                cache  
             const Common::JVector< Common::JString > & expect  
             )
```

Joins the specified game room.

This function sends a request to the server to join the specified game room. The `Listener::joinRoomReturn()` gets called when the operation has been finished.

This function is useful when you are using a lobby to list rooms and know the lobby (you specify the lobby name, region and app version), so it does not matter which lobby the room is in.

It's usually better to use **opJoinOrCreateRoom()** for invitations. Then it will create the room if it doesn't exist.

Remarks

A **Client** instance can only be inside one room at a time. Therefore, if the client is already inside another game room, any lobby the client currently resides in will be closed and the client will be moved to the new room.

If a room with the specified name does not exist or if the room is full, the `Listener::joinRoomReturn()` will get called with an error code.

Parameters

- gameID** A unique identifier for the game room to join.
- rejoin** Needs to be false if this is the initial join of this room.
- cacheSliceIndex** Allows to request a specific cache slice - all events published to the client after joining the room - see `CacheSliceIndex`.
- expectedUsers** Sets a list of user IDs for which the server should expect to see players. This list will be merged with any previous list.

Returns

true, if the request could successfully be queued for sending to the s

See also

[opCreateRoom\(\)](#), [opJoinOrCreateRoom\(\)](#), [opJoinRandomRoom](#)
Listener::joinRoomReturn()

§ opJoinRandomRoom()

```
bool
opJoinRandomRoom ( const Common::Hashtable &
                   nByte
                   nByte
                   const Common::JString &
                   nByte
                   const Common::JString &
                   const Common::JVector< Common::JString > &
                   )
```

Joins a random game room.

This function sends a request to the server to join a random game room. It is called when the operation has been finished.

Remarks

A **Client** instance can only be inside one room at a time. Therefore it cannot be inside another game room. Any lobby the client currently resides in will be closed. If no rooms are fitting or available (all full, closed or not visible), then the client will get called with an error code.

Parameters

customRoomProperties	Used as a filter for matchmaking. The server will use the specified filters. Note that only those filters that are specified in the lobby will be used for matchmaking, so filters not specified in the list of properties to show will be ignored.
maxPlayers	Must match the value of a room's maxPlayers.
matchmakingMode	Needs to be one of the values in MatchmakingMode.
lobbyName	The name of the lobby in which matchmaking will be considered for matchmaking.
lobbyType	The type of the lobby in which matchmaking will be considered. Note that a lobby with the same name can be used for matchmaking, as a lobby name only needs to be specified.

sqlLobbyFilter

Only used for **LobbyType::SQL_LOBBY** filtering against certain room properties.

expectedUsers

Sets a list of user IDs for which the server players. This list will be merged with any p

Returns

true, if the request could successfully be queued for sending to the s

See also

[opCreateRoom\(\)](#), [opJoinOrCreateRoom\(\)](#), [opJoinRandomRoom](#)
[Listener::joinRoomReturn\(\)](#), [Matchmaking and Lobby](#)

§ opLeaveRoom()

```
bool opLeaveRoom ( bool willComeBack = false,  
                  bool sendAuthCookie = false  
                  )
```

virtual

Leaves the currently joined game room.

This function sends a request to the server to leave the currently joined game room. If it returns true, then `Listener::leaveRoomReturn()` gets called when the operation has successfully been finished.

Remarks

This operation will fail and return false if the client does not currently reside inside any game room.

Parameters

willComeBack If this is set to 'true', then the player becomes inactive and the client could later rejoin the room as the very same player. 'false' means the player leaves the room for good. Note that the player only stays inactive for at maximum as many milliseconds as you have set the `playerTtl` to during room creation (see [RoomOptions::setPlayerTtl\(\)](#)). The default is 'false'.

sendAuthCookie Pass 'true' to set the `sendAuthCookie` web flag (please see [Webhooks v1.2](#) for further information). The default is 'false'.

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

[opCreateRoom\(\)](#), [opJoinOrCreateRoom\(\)](#), [opJoinRoom\(\)](#), [opJoinRandomRoom\(\)](#), [MutableRoom](#), [RoomOptions](#),

Listener::leaveRoomReturn()

§ opRaiseEvent() [1/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool                reliable,
               const Ftype &      parameters,
               nByte              eventCode,
               const RaiseEventOptions & options = RaiseEventOptions()
             )
```

Sends in-game data to other players in the game, who will receive it in the Listener::customEventAction() callback.

The eventCode should be used to define the event's type and content respectively. The payload has to be one of the datatypes that are listed as supported for values at [serializable datatypes](#).

This function provides the option to raise events reliably or unreliably. While both result in ordered events, the ones that got sent with the latter option might get lost, causing gaps in the resulting event sequence. On the other hand, they cause less overhead and are optimal for data that is replaced soon.

Note: the value of the reliability option only takes effect when the ConnectionProtocol passed to [Client\(\)](#) equals ConnectionProtocol::UDP (which is the default for most platforms) and the message is small enough to not get fragmented into several UDP packets (rule of thumb: you can safely assume that the message fits into a single UDP packet, when its payload size is below 1kb), otherwise the message gets sent reliably, even when the reliability option asks for sending it unreliably.

Sending is not done immediately, but in intervals of [service\(\)](#) calls.

It is recommended to keep the payload as simple as possible, especially events that get raised multiple times per second. This easily adds up to a huge amount of data otherwise.

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

Listener::customEventAction(), [Table of Datatypes](#)

Parameters

- reliable** true = the operation will be sent reliably; false = no resend in case of packet loss - will be ignored, when not using ConnectionProtocol::UDP
- parameters** the payload of the event to raise - has to be provided in the form of one of the supported data types, specified at [Table of Datatypes](#)
- eventCode** number for arbitrary classification of the type of the event (like '1' for position updates, '2' for chat messages, and on).
- options** see [RaiseEventOptions](#)

§ opRaiseEvent() [2/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool
                const Ftype
                typename Common::Helpers::ArrayLengthType< Ftype >
                nByte
                const RaiseEventOptions &
                )
```

This is an overloaded member function, provided for convenience. It differs in what argument(s) it accepts.

Parameters

reliable	true = operation will be sent reliably; false = no retransmission, ignored, when not using UDP as protocol
pParameterArray	the payload array of the event to raise - has to be of one of the supported data types, specified at Types
arrSize	the number of elements in pParameterArray
eventCode	number for arbitrary classification of the type of event (e.g. for chat messages, and so on).
options	see RaiseEventOptions

§ opRaiseEvent() [3/3]

```
template<
typename
Ftype > bool
opRaiseEvent ( bool                reliable,
               const Ftype        pParameterArray,
               const short *      pArrSizes,
               nByte              eventCode,
               const RaiseEventOptions & options = RaiseEventOptions
               )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- reliable** true = operation will be sent reliably; false = no resend in case of packet loss - will be ignored, when not using UDP as protocol
- pParameterArray** the payload array of the event to raise - has to be provided in the form of an array of one of the supported data types, specified at **Table of Datatypes**
- pArrSizes** an array holding the number of elements for each dimension of pParameterArray
- eventCode** number for arbitrary classification of the type of event (like '1' for position updates, '2' for chat messages, and so on).
- options** see **RaiseEventOptions**

§ opFindFriends()

```
bool  
opFindFriends ( const Common::JString * friendsToFind,  
               short numFriendsToFind  
               )
```

virtual

Requests the rooms and online states for the specified list of friends. All clients should set a unique UserID before connecting. The result can be accessed through **getFriendList()** after the corresponding call to Listener::onFindFriendsResponse() has been received.

This function can be called when the caller does not currently reside in a game room to find the rooms played by a selected list of users. The result can be accessed by a call to **getFriendList()** and is empty before the first response has arrived in Listener::onFindFriendsResponse(). **getFriendListAge()** can be used to retrieve the amount of milliseconds that have passed since the value that is returned by **getFriendList()** has been updated for the last time.

Users identify themselves by passing their UserIDs to **AuthenticationValues::setUserID()**.

The list of userIDs must be fetched from some other source (not provided by **Photon**).

Remarks

This operation will fail and return false if the client does currently reside inside a game room or if the result for a previous call to this function has not arrived yet.

Parameters

friendsToFind An array of unique userIDs.
numFriendsToFind The element count of friendsToFind.

Returns

true, if the request could successfully be queued for sending to the

server, false otherwise.

See also

[getFriendList\(\)](#), [getFriendListAge\(\)](#),
[Listener::onFindFriendsResponse\(\)](#)

§ opLobbyStats()

bool

opLobbyStats (const **Common::JVector**< **LoadBalancing::LobbyStatsRequest**)

Sends the specified list of **LobbyStatsRequest** objects to the server. The Listener::onLobbyStatsResponse().

This function can be called when the caller does not currently reside in a lobby.

Remarks

This operation will fail and return false if the client does currently reside in a lobby.

Note

Pass 'true' for the 'autoLobbyStats' parameter of **Client()** to automatically do so, it makes little sense to also additionally call this function. Only use the 'autoLobbyStats' parameter of **Client()** to achieve fine-grain control over when to reduce traffic when you have lots of lobbies, but only rarely need to manually call this function.

Parameters

lobbiesToQuery A **Common::JVector** containing a **LobbyStatsRequest** object for each lobby to query.

Returns

true, if the request could successfully be queued for sending to the server; otherwise, false.

See also

Client(), Listener::onLobbyStatsResponse(), Listener::onLobbyStatsResponse()

§ opChangeGroups()

```
bool  
opChangeGroups ( const Common::JVector< nByte > * pGroupsToRemove  
                 const Common::JVector< nByte > * pGroupsToAdd  
                 )
```

Updates the clients interest groups (for events inside of game rooms).

This function can be called from inside of a game room to change the list of interest groups inside that room to which the local client is subscribed to. For each **opRaiseEvent()** call one can specify the interest groups to which that event should be sent in the **RaiseEventOptions**. When doing so, only clients that are subscribed to those interest groups will receive that event.

Note the difference between passing NULL and the address of an empty instance:

- NULL won't add/remove any groups.
- a JVector without any elements will add/remove all (existing) groups

First, removing groups is executed. This way, you could leave all groups only the ones provided.

Changes become active not immediately but when the server executes the next operation (approximately **getRoundTripTime()/2** milliseconds after the **opChangeGroups()** is sent it).

Remarks

This operation will fail and return false if the client does not currently exist inside a game room.

Parameters

pGroupsToRemove Groups to remove from interest. NULL will not remove any. An empty instance will remove all.

pGroupsToAdd Groups to add to interest. NULL will not add any. An empty instance will add all existing groups.

Returns

true, if the request could successfully be queued for sending to the s
false otherwise.

See also

[opRaiseEvent\(\)](#), [RaiseEventOptions::setInterestGroups\(\)](#), [Interestg](#)

§ opCustomAuthenticationSendNextStepData()

bool

opCustomAuthenticationSendNextStepData (const **AuthenticationValues**

Used in conjunction with Listener::onCustomAuthenticationIntermediateStep() for custom authentication.

While normally custom authentication is single-legged, occasionally a client can perform a multi-step authentication. This means that the client sends some authentication data when calling **connect()** and the server does not respond with a final result (e.g. a connect attempt due to an authentication error), but with some intermediate data. Your Listener::onCustomAuthenticationIntermediateStep() implementation should call opCustomAuthenticationSendNextStepData() to pass the next step data to this function to continue the authentication process after the **connect()** call.

Remarks

This operation will fail and return false if the client is not currently expecting authentication data. You are only expected to call opCustomAuthenticationSendNextStepData() after you have received a call to Listener::onCustomAuthenticationIntermediateStep() beforehand and are expecting authentication data. If you are not expecting authentication data, then the connection flow pauses until this call has been made. This is expected if the custom authentication that you have set up is single-legged (which is the most common) or if you have not set up any custom authentication at all, in which case all authentication attempts always fail in these scenarios.

Parameters

authenticationValues An instance of class **AuthenticationValues**

Returns

true, if the request could successfully be queued for sending to the server

See also

connect(), Listener::onCustomAuthenticationIntermediateStep(), **AuthenticationValues**

§ opWebRpc() [1/4]

```
bool opWebRpc ( const Common::JString & uriPath )
```

virtual

Makes **Photon** call your custom web-service by path/name with the given parameters (converted into JSON).

A WebRPC calls a custom, http-based function on a server that you provide. The uriPath is relative to a "base path" which is configured on the server side. The sent parameters get converted to Json. Vice versa, the response of the web-service will be converted back, when it gets sent back to the **Client**, where it arrives in Listener::webRpcReturn().

To use this feature, you have to setup your server:

For a **Photon** Cloud application visit the Dashboard and setup "WebHooks". The BaseUrl is used for WebRPCs as well.

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

Listener::webRpcReturn(), [Table of Datatypes](#), [Webhooks](#)

Parameters

uriPath the URL path to call, relative to the baseUrl configured on **Photon's** server-side

§ opWebRpc() [2/4]

```
template<
typename Ftype
> bool
opWebRpc      ( const Common::JString & uriPath,
                const Ftype &           parameters,
                bool                     sendAuthCookie = false
              )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- uriPath** the URL path to call, relative to the baseUrl configured on **Photon**'s server-side
- parameters** the parameters to send to the web-service method - has to be provided in the form of one of the supported data types, specified at **Table of Datatypes**
- sendAuthCookie** defines if the authentication cookie gets sent to a WebHook (if setup)

§ opWebRpc() [3/4]

```
template<
typename
Ftype >
bool
opWebRpc ( const Common::JString &
           const Ftype
           typename Common::Helpers::ArrayLengthType< Ftype >::ty
           bool
           )
```

This is an overloaded member function, provided for convenience. It differs only in what argument(s) it accepts.

Parameters

- uriPath** the URL path to call, relative to the baseUrl configuration side
- pParameterArray** the parameter array to send to the web-service in the form of a 1D array of one of the supported **Table of Datatypes**
- arrSize** the number of elements in pParameterArray
- sendAuthCookie** defines if the authentication cookie gets sent to a

§ opWebRpc() [4/4]

```
template<
typename Ftype
> bool
opWebRpc      ( const Common::JString & uriPath,
                const Ftype                pParameterArray,
                const short *              pArrSizes,
                bool                       sendAuthCookie = false
              )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- uriPath** the URL path to call, relative to the baseUrl configured on **Photon**'s server-side
- pParameterArray** the parameter array to send to the web-service method - has to be provided in the form of an array of one of the supported data types, specified at **Table of Datatypes**
- pArrSizes** an array holding the number of elements for each dimension of pParameterArray
- sendAuthCookie** defines if the authentication cookie gets sent to a WebHook (if setup)

§ selectRegion()

```
bool  
selectRegion      ( const Common::JString & selectedRegion ) virtual
```

Used in conjunction with `Listener::onAvailableRegions()` and `RegionSelectionMode::SELECT` to select a certain server region to connect to.

If you pass `RegionSelectionMode::SELECT` for parameter 'regionSelectionMode' to `Client()`, then the `Client` does not automatically choose a server region to connect to on its own during the connection flow, but upon retrieving the list of available regions and the list of server addresses that can be used to ping those regions it passes those lists to your implementation of `Listener::onAvailableRegions()` and pauses the connection flow. You then need to choose one of the available regions and select it by passing its name to this function to continue the connection flow.

The list of available regions for **Photon** Public Cloud is available at [Regions](#). However more regions might be added over time after you have released your application and the list of available regions might differ when your appID is associated with a dedicated Cloud or when you connect to a non-default name server address. Also a certain region might be temporarily unavailable for maintenance. Furthermore some regions might consist out of multiple different clusters, while others don't. Therefore you should always assure that the region name that you pass to this function actually matches one of the entries in the list of available regions. Also be prepared to select a fall back option in case that your preferred region is not available.

A typical list of available regions might look like this (more or less regions might be available and the order of the entries is undefined and might change without notice): "eu", "us", "usw", "cae", "asia", "jp", "au", "sa", "in", "kr"

When multiple clusters per region are set up for your appID for some regions, then the list might look like this: "eu/Default", "eu/Cluster2",

"us/Default", "us/Cluster2", "usw", "cae", "asia", "jp", "au", "sa", "in", "kr"

Examples for valid strings to pass for the 'eu' region for parameter 'selectedRegion' with the above example lists (adapt accordingly for other regions):

- "eu" - Valid when at least one cluster is available in region 'eu', selects the default cluster for that region.
- "eu/Default" - Only valid when a cluster with the exact name "Default" is available in region 'eu'.
- "eu/Cluster2" - Only valid when a cluster with the exact name "Cluster2" is available in region 'eu'.
- "eu/*" - Only valid when at least 2 clusters are setup in region 'eu' of which at least one is available. The server randomly selects one of the available clusters in the specified region. This string is not contained in the list of available regions and must be constructed by your code when it is valid and when you intend to select a random cluster.

In case of the server randomly selecting a cluster, parameter 'cluster' of `Listener::connectReturn()` contains the name of the cluster to which the client has connected. Otherwise that parameter is an empty string.

Remarks

This operation will fail and return false if 'regionSelectionMode' has not been set to `RegionSelectionMode::SELECT` upon construction of this class instance.

Parameters

selectedRegion Must be a valid region name that matches one of the entries in the list of available regions that got passed to `Listener::onAvailableRegions()`

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

[Client\(\)](#), [connect\(\)](#), `Listener::onAvailableRegions()`

§ reconnectAndRejoin()

```
bool reconnectAndRejoin ( void )
```

virtual

Reconnects the the server and rejoins the last previously joined room.

This function reconnects directly to the game server to which it has previously been connected to and sends a request to the server to join the last previously joined game room. If it returns true, then Listener::joinRoomReturn() gets called when the operation has been finished.

The usual requirements for a rejoin apply, meaning the room must still exist, the local player must have entered it before, but it must not have left it for good, but only have become inactive and the playerTTL for the local player in that room must not have run out yet, otherwise this operation will fail and Listener::joinRoomReturn() will get called with an error code.

Remarks

This function will fail and return false if no game room has been entered since the creation of the class instance or if the client is still/already in a connected state.

reconnectAndRejoin() is quicker than the combination of **connect()** and **opJoinRoom()**.

Returns

true, if the request could successfully be queued for sending to the server, false otherwise.

See also

connect(), **opJoinRoom()**, Listener::joinRoomReturn()

§ sendDirect() [1/6]

```
template< typename Ftype >  
bool sendDirect          ( const Ftype & parameters,  
                          int           targetPlayer,  
                          bool          fallbackRelay = false  
                          )
```

Sends in-game data to other players in the game, who will receive it in their `Listener::onDirectMessage()` callback. Data that gets sent with this function, gets sent over a direct peer to peer connection, when possible.

For the Photon clients to attempt to establish direct peer to peer connections to each other when entering a room you need set the the **DirectMode** Option either to **DirectMode::MASTER_TO_ALL** or to **DirectMode::ALL_TO_ALL** on the **RoomOptions** instance that you provide on room creation. Only when a direct connection to a certain client exists, data can be exchanged with it directly. Otherwise this function either falls back to sending it through the Photon game server with **opRaiseEvent()**, or doesn't send it at all, depending on the value of the 'fallbackRelay' parameter. Data transfer on a direct p2p connection always happens unreliably over UDP even when a different connection protocol has been chosen for connections to the **Photon** servers in the constructor of this class. However data transfer over the fall-back relay uses the protocol that has been selected for connections to the **Photon** server when calling the constructor.

It is recommended to keep the payload as simple as possible, as the data is typically sent multiple times per second. This easily adds up to a huge amount of data otherwise.

Note

A direct connection to a certain client is not guaranteed to exist, even when **RoomOptions::setDirectMode()** specifies that the Clients should attempt to establish it, as NAT punch-through does not have a 100% success rate. In the case that a direct message

is preferable, but a relayed one would be acceptable when no direct connection exists, the 'fallbackRelay' option comes into play. Furthermore if a client loses its connection to Photon while other clients can still reach the server, then that client most likely lost its internet connection and direct messages won't reach it anymore either.

Remarks

This function provides a rather low-level raw UDP socket like way to send data. If you need any higher level functionality like reliable data delivery, support for bigger messages, message caching, interest groups or webforwarding, then please use [opRaiseEvent\(\)](#) instead.

See also

[Listener::onDirectMessage\(\)](#), [opRaiseEvent\(\)](#), [DirectMode](#), [RoomOptions::getDirectMode\(\)](#), [RoomOptions::setDirectMode\(\)](#)

Parameters

- parameters** the data to send - has to be provided in the form of one of the supported data types, specified at [Table of Datatypes](#) - must be less than 1200 bytes
- targetPlayer** the player number of the intended receiver of the message - must be the number of another active player inside the same room as the sender
- fallbackRelay** true if the Photon game server that hosts the room should be used as a fallback relay (by an automatic call to [opRaiseEvent\(\)](#)) when no direct connection to the other client exists, false otherwise

Returns

true, if the request could successfully be sent (this does not guarantee that it will be received), false otherwise.

§ sendDirect() [2/6]

```
template<
typename
Ftype >
bool
sendDirect ( const Ftype
              typename Common::Helpers::ArrayLengthType< Ftype >::ty
              int
              bool
            )
```

This is an overloaded member function, provided for convenience. It differs from other overloaded functions only in what argument(s) it accepts.

Parameters

- pParameterArray** the data to send - has to be provided in the form of an array of the supported data types, specified at [Table of Data Types](#). The array must be less than 1200 bytes
- arrSize** the number of elements in pParameterArray
- targetPlayer** the player number of the intended receiver of the data. If the number of another active player inside the same room is provided, the data will be sent to that player.
- fallbackRelay** true if the Photon game server that hosts the room should use a fallback relay (by an automatic call to [opRaiseEvent](#)) if the intended receivers to which no direct connection exists, false otherwise.

Returns

true, if the request could successfully be sent (this does not guarantee that the data was received), false otherwise.

§ sendDirect() [3/6]

```
template< typename Ftype > bool
sendDirect          ( const Ftype  pParameterArray,
                    const short * pArrSizes,
                    int           targetPlayer,
                    bool          fallbackRelay = false
                    )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- pParameterArray** the the data to send - has to be provided in the form of an array of one of the supported data types, specified at [Table of Datatypes](#) - must be less than 1200 bytes
- pArrSizes** an array holding the number of elements for each dimension of pParameterArray
- targetPlayer** the player number of the intended receiver of the message - must be the number of another active player inside the same room as the sender
- fallbackRelay** true if the Photon game server that hosts the room should be used as a fallback relay (by an automatic call to [opRaiseEvent\(\)](#)) for all specified receivers to which no direct connection exists, false otherwise

Returns

true, if the request could successfully be sent (this does not guarantee that it will be received), false otherwise.

§ sendDirect() [4/6]

```
template<
typename
Ftype >
bool
sendDirect ( const Ftype &                parameters,
              const Common::JVector< int > & targetPlayers = Common::
              bool                          fallbackRelay = false
            )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- parameters** the data to send - has to be provided in the form of one of the supported data types, specified at [Table of Datatypes](#). The size must be less than 1200 bytes
- targetPlayers** the player numbers of the intended receivers of the request. They must be the numbers of other active players inside the room, not including the sender
- fallbackRelay** true if the Photon game server that hosts the room should be used as a fallback relay (by an automatic call to [opRaiseError](#) to the specified receivers to which no direct connection exists). Otherwise false

Returns

the number of target players, for which the request could be successful (but does not guarantee that it will be received).

§ sendDirect() [5/6]

```
template<
typename
Ftype >
bool
sendDirect ( const Ftype
              typename Common::Helpers::ArrayLengthType< Ftype >::ty
              const Common::JVector< int > &
              bool
            )
```

This is an overloaded member function, provided for convenience. It differs in the number of argument(s) it accepts.

Parameters

- pParameterArray** the data to send - has to be provided in the form of an array of the types, specified at [Table of Datatypes](#) - must be of the same type as the data to be sent
- arrSize** the number of elements in pParameterArray
- targetPlayers** the player numbers of the intended receivers of the data. If the number of target players is 0, the data will be sent to all other active players inside the same room as the sender.
- fallbackRelay** true if the Photon game server that hosts the room should automatically call to [opRaiseEvent\(\)](#) for all spectators if a connection exists, false otherwise

Returns

the number of target players, for which the request could successfully be received).

§ sendDirect() [6/6]

```
template<
typename
Ftype >
bool
sendDirect ( const Ftype                pParameterArray,
             const short *              pArrSizes,
             const Common::JVector< int > & targetPlayers = Common::JVector< int >(),
             bool                       fallbackRelay = false
           )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- pParameterArray** the data to send - has to be provided in the form of one of the supported data types, specified at [Table 1: Datatypes](#) - must be less than 1200 bytes
- pArrSizes** an array holding the number of elements for each pParameterArray
- targetPlayers** the player numbers of the intended receivers of the data - must be the numbers of other active players in the room as the sender
- fallbackRelay** true if the Photon game server that hosts the room should be used as a fallback relay (by an automatic call to [opRaiseEvent\(\)](#)) for all specified receivers to which a connection exists, false otherwise

Returns

the number of target players, for which the request could be successful (this does not guarantee that it will be received).

§ getServerTimeOffset()

```
int getServerTimeOffset ( void ) const
```

Returns

the difference between the local uptime and the Photon Server's system time in ms.

In real-time games it's often useful to relate game events to a global common timeline, that's valid for all players and independent from derivations throughout the clients' system times. The Photon Server's System Time can serve as this reference time. The `serverTimeOffset` represents the difference between the client's local system time and the Photon server's system time.

`ServerTime = serverTimeOffset + GETTIMEMS()`

The `serverTimeOffset` is fetched shortly after connect by **Photon**. Use `GETTIMEMS()` to get your local time in ms. You can let Photon refetch the offset by calling **`fetchServerTimestamp()`**. The `ServerTimeOffset` will be 0 until shortly after initial connect.

§ `getServerTime()`

```
int getServerTime ( void ) const
```

Returns

the Photon Server's system time in ms.

see [getServerTimeOffset\(\)](#)

§ `getBytesOut()`

```
int getBytesOut ( void ) const
```

Returns

the total number of outgoing bytes transmitted by this PhotonPeer object.

See also

[getBytesIn\(\)](#)

§ `getBytesIn()`

```
int getBytesIn ( void ) const
```

Returns

the total number of incoming bytes received by this PhotonPeer object.

See also

[getBytesOut\(\)](#)

§ `getByteCountCurrentDispatch()`

```
int getByteCountCurrentDispatch ( void ) const
```

Returns

the size of the dispatched event or operation-result in bytes. This value is set before `onEvent()` or `onOperationResponse()` is called (within **`dispatchIncomingCommands()`**). Get this value directly in `onEvent()` or `onOperationResponse()`.

§ getByteCountLastOperation()

```
int getByteCountLastOperation ( void ) const
```

Returns

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. Get this value immediately after calling an operation.

§ getSentCountAllowance()

```
int getSentCountAllowance ( void ) const
```

Returns

the number of resend retries before a peer is considered lost/disconnected.

This is udp specific and will always return 0 for other protocols.

See also

[setSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ setSentCountAllowance()

```
void setSentCountAllowance ( int sentCountAllowance )
```

Sets the number of re-send retries before a peer is considered lost/disconnected.

This is udp specific and will do nothing at all for other protocols.

Parameters

sentCountAllowance the new number of re/-send retries before a peer is considered lost/disconnected.

See also

[getSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ getTimePingInterval()

```
int getTimePingInterval ( void ) const
```

Returns

the time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[setTimePingInterval\(\)](#)

§ setTimePingInterval()

```
void setTimePingInterval ( int timePingInterval )
```

Sets the time threshold in milliseconds since the last reliable command, before a ping will be sent.

Parameters

timePingInterval time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[getTimePingInterval\(\)](#)

§ getRoundTripTime()

```
int getRoundTripTime ( void ) const
```

Returns

the time in milliseconds until a reliable command is acknowledged by the server.

This is, what is commonly called a ping time or just a ping.

See also

[getRoundTripTimeVariance\(\)](#)

§ `getRoundTripTimeVariance()`

```
int getRoundTripTimeVariance ( void ) const
```

Returns

the variance of the roundtrip time in milliseconds. Gives a hint about how much the net latency is varying.

See also

[getRoundTripTime\(\)](#)

§ getTimestampOfLastSocketReceive()

```
int getTimestampOfLastSocketReceive ( void ) const
```

Returns

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.

§ `getDebugOutputLevel()`

```
int getDebugOutputLevel ( void ) const
```

Returns the current level of debug information that's passed on to [BaseListener::debugReturn\(\)](#).

Returns

one of the values in `DebugLevel`

See also

[setDebugOutputLevel\(\)](#)

§ `setDebugOutputLevel()`

```
bool setDebugOutputLevel ( int debugLevel )
```

Sets the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Parameters

debugLevel one of the values in `DebugLevel`

Returns

true if the new debug level has been set correctly, false otherwise.

See also

`getDebugOutputLevel()`

§ getLogFormatOptions()

```
const LogFormatOptions & getLogFormatOptions ( void ) const
```

Returns

the LogFormatOptions that are used by this instance.

See also

setFormatOptions()

§ setLogFormatOptions()

```
void  
setLogFormatOptions ( const Common::LogFormatOptions & formatO
```

Sets the log format options to the supplied value.

Parameters

formatOptions the new value to which the log format options will be

See also

getFormatOptions()

§ getIncomingReliableCommandsCount()

```
int getIncomingReliableCommandsCount ( void ) const
```

Returns

the total number of reliable commands currently waiting in the incoming queues of all channels or -1 if not connected.

§ getPeerID()

```
short getPeerID ( void ) const
```

Returns

this peer's ID as assigned by the server. Will be -1, if not connected.

§ getDisconnectTimeout()

```
int getDisconnectTimeout ( void ) const
```

Returns

the maximum time interval in milliseconds for doing resend retries before a peer is considered lost/disconnected.

See also

[setDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ setDisconnectTimeout()

```
void setDisconnectTimeout ( int disconnectTimeout )
```

Sets the maximum time in milliseconds for making re-send retries before a peer is considered lost/disconnected.

Parameters

disconnectTimeout resend max time in ms before a peer is considered lost/disconnected

See also

[getDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ getQueuedIncomingCommands()

```
int getQueuedIncomingCommands ( void ) const
```

Returns

the number of queued incoming commands in all channels or -1 if not connected

§ getQueuedOutgoingCommands()

```
int getQueuedOutgoingCommands ( void ) const
```

Returns

the number of queued outgoing commands in all channels or -1 if not connected

§ `getIsPayloadEncryptionAvailable()`

```
bool getIsPayloadEncryptionAvailable ( void ) const
```

Returns

this peer's payload encryption availability status. True if payload encryption is available, false otherwise.

See also

[getIsEncryptionAvailable\(\)](#), `establishEncryption()`,
`initUserDataEncryption()`

§ getIsEncryptionAvailable()

```
bool getIsEncryptionAvailable ( void ) const
```

Returns

this peer's encryption availability status. True if either payload encryption is available or if the connection protocol is UDP and UDP encryption is available or if the connection protocol is already secure on its own, false otherwise.

See also

[getIsPayloadEncryptionAvailable\(\)](#), [establishEncryption\(\)](#), [initUserDataEncryption\(\)](#), [initUDPEncryption\(\)](#)

§ getResentReliableCommands()

```
int getResentReliableCommands ( void ) const
```

Returns

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

§ getLimitOfUnreliableCommands()

```
int getLimitOfUnreliableCommands ( void ) const
```

Returns

the limit for the queue of received unreliable commands.

See also

[setLimitOfUnreliableCommands\(\)](#)

§ setLimitOfUnreliableCommands()

```
void setLimitOfUnreliableCommands ( int value )
```

Sets the limit for the queue of received unreliable commands. This works only in UDP. This limit is applied when you call `dispatchIncomingCommands`. If this client (already) received more than this limit, 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. 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 commands up to specified limit. This value interacts with `dispatchIncomingCommands()`: If that is called less often, more commands get skipped.

See also

[getLimitOfUnreliableCommands\(\)](#)

§ getCRCEnabled()

```
bool getCRCEnabled ( void ) const
```

Returns

true if CRC enabled

See also

[setCRCEnabled](#)

§ setCRCEnabled()

```
void setCRCEnabled ( bool crcEnabled )
```

Enables or disables CRC. While not connected, this controls if the next connection(s) should use a per-package CRC checksum. If the client is in another state than 'connected', then this function has no effect except for logging an error.

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.

Note

This only has effect for UDP connections.

This does not have any effect for connections that use UDP datagram encryption (which always use a built-in checksum).

See also

[getCRCEnabled](#)

§ `getPacketLossByCRC()`

```
int getPacketLossByCRC ( void ) const
```

Returns

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

See also

[`setCRCEnabled`](#)

§ getTrafficStatsEnabled()

```
bool getTrafficStatsEnabled ( void ) const
```

Returns

true if traffic statistics of a peer are enabled. Default trafficStatsEnabled: false (disabled).

§ setTrafficStatsEnabled()

```
void setTrafficStatsEnabled ( bool trafficStatsEnabled )
```

Enables or disables the traffic statistics of a peer. Default trafficStatsEnabled: false (disabled).

§ getTrafficStatsElapsedMs()

```
int getTrafficStatsElapsedMs ( void ) const
```

Returns

the count of milliseconds the stats are enabled for tracking.

§ getTrafficStatsIncoming()

```
const Photon::TrafficStats & getTrafficStatsIncoming ( void ) const
```

Returns

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.

§ getTrafficStatsOutgoing()

```
const Photon::TrafficStats & getTrafficStatsOutgoing ( void ) const
```

Returns

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.

§ getTrafficStatsGameLevel()

```
const Photon::TrafficStatsGameLevel &  
getTrafficStatsGameLevel (void ) const
```

Returns

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).

§ getQuickResendAttempts()

```
nByte getQuickResendAttempts ( void ) const
```

Returns

the number of resend attempts for a reliable command that are done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

§ setQuickResendAttempts()

```
void setQuickResendAttempts ( nByte quickResendAttempts )
```

Sets the number of resend attempts for a reliable command can be done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

Remarks

The default value is 0. Any later resend attempt will then double the time before the next resend takes place. The max value is 4. Make sure to set `SentCountAllowance` to a slightly higher value, as more repeats will get done.

§ getChannelCountUserChannels()

```
nByte getChannelCountUserChannels ( void ) const
```

The IDs from 0 to **getChannelCountUserChannels()**-1 can be passed as channelID to operations that offer this parameter.

Returns

the number of different channels that are available for sending operations on.

§ getPeerCount()

```
short getPeerCount ( void )
```

static

Returns

the count of peers, which have been initialized since the start of the application. Interesting mainly for debugging purposes.

§ getState()

```
int getState ( void ) const
```

The Current state this **Client** instance is in. Be Careful: several states are "transitions" that lead to other states.

Note

This is publicly available purely for informational purposes (i.e. when debugging) and your logic should not rely on certain state changes, but should instead wait for the dedicated callbacks.

Returns

one of the values defined in **PeerStates**

§ getMasterserverAddress()

```
const JString & getMasterserverAddress ( void ) const
```

Returns

the address of the master server to which the client is connected when it is not inside a game room.

§ getCountPlayersIngame()

```
int getCountPlayersIngame ( void ) const
```

Returns

the count of players that are currently participating in games on game servers that are in the same cluster (game servers assigned to the same master server) as the local client. Each **Photon** Cloud region consists of at least one, but potentially multiple separate clusters.

Remarks

This value is only getting updated when the client is on the master server.

§ getCountGamesRunning()

```
int getCountGamesRunning ( void ) const
```

Returns

the count of rooms that are currently existing on game servers in the same cluster (game servers assigned to the same master server) as the one the local client is connected to. Each **Photon** Cloud region consists of at least one, but potentially multiple separate clusters.

Remarks

This value is only getting updated when the client is on the master server.

§ getCountPlayersOnline()

```
int getCountPlayersOnline ( void ) const
```

Returns

the total count of players that are currently connected to the same cluster (clients that are connected to the same master server or to a game server that is assigned to the same master server) as the local client. Each **Photon** Cloud region consists of at least one, but potentially multiple separate clusters.

Remarks

This value is only getting updated when the client is on the master server.

§ getCurrentlyJoinedRoom()

MutableRoom & getCurrentlyJoinedRoom (void)

Returns

a non-const reference to a **MutableRoom** instance that represents the currently joined room.

Remarks

The behavior when accessing the referenced instance after leaving the room in which that reference has been obtained and the behavior when calling this function without being inside a room is undefined.

Note

Attention: Do not assign the return value of this function to a **MutableRoom** variable, but only assign it to a **MutableRoom** reference or simply directly operate on the function return value, as assigning it to a variable means that accessing that variable lets you operate on a local copy and operations that change that copy don't affect the actual room.

§ getRoomList()

```
const JVector< Room * > & getRoomList ( void ) const
```

Returns

the list of all visible rooms.

Remarks

The value that is returned by this function is only updated inside a lobby of **LobbyType::DEFAULT**. Clients that are inside a lobby of a different **LobbyType**, or in no lobby at all, do not receive room list updates. The same **Client** instance can't be inside of multiple rooms at once. The term 'room' includes game rooms and lobbies. Therefor a **Client** instance is not able to receive room list updates while it resides inside of a game room.

To show up in the lobby the **IsVisible** flag of a room needs to be set to true (which is the default value). The **MaxPlayers** setting and the current amount of players inside a room do not influence the rooms visibility, nor does the **IsOpen** flag: If the maximum amount of players is already inside of the room or if the room is closed, then the room is still included in the room list, but attempts to join it will fail.

See also

getRoomNameList()

§ `getRoomNameList()`

```
const JVector< JString > & getRoomNameList ( void ) const
```

Returns

the list of the names of all visible rooms.

The entries in the returned `JVector` instance are guaranteed to be in the same order like the entries in the `JVector` instance that is returned by `getRoomList()`. The same remarks apply to this function as are mentioned for `getRoomList()`.

See also

`getRoomList()`

§ `getIsInRoom()`

```
bool getIsInRoom ( void ) const
```

Returns

true if this client instance currently resides within a room, false otherwise.

Remarks

The term 'room' includes game rooms and lobbies.

See also

[getIsInGameRoom\(\)](#), [getIsInLobby\(\)](#)

§ `getIsInGameRoom()`

```
bool getIsInGameRoom ( void ) const
```

Returns

true if this client instance currently resides within a game room,
false otherwise.

See also

[getIsInRoom\(\)](#), [getIsInLobby\(\)](#)

§ `getIsInLobby()`

```
bool getIsInLobby ( void ) const
```

Returns

true if this client instance currently resides within a lobby, false otherwise.

See also

[getIsInRoom\(\)](#), [getIsInGameRoom\(\)](#)

§ getAutoJoinLobby()

```
bool getAutoJoinLobby ( void ) const
```

Returns

the current value of the autoJoinLobby flag.

Remarks

The value of the autoJoinLobby flag determines if the client will automatically join the default lobby whenever it has successfully connected and whenever it leaves a game room.

See also

[setAutoJoinLobby\(\)](#)

§ setAutoJoinLobby()

```
void setAutoJoinLobby ( bool autoJoinLobby )
```

Sets the value of the autoJoinLobby flag.

Parameters

autoJoinLobby the new value to which the flag will be set

Remarks

The value of the autoJoinLobby flag determines if the client will automatically join the default lobby whenever it has successfully connected and whenever it leaves a game room.

See also

[getAutoJoinLobby\(\)](#)

§ getPlayer()

MutablePlayer & getPlayer (void)

Returns

a non-const reference to the **MutablePlayer** instance that is representing the local player.

§ getFriendList()

```
const JVector< FriendInfo > & getFriendList ( void ) const
```

Returns

the latest locally cached state of the friend list.

Remarks

You can request the latest state of the local clients friend list from the server by a call to **opFindFriends()**. **Listener::onFindFriendsResponse()** informs you when the servers response has arrived. The list that is returned by this function reflects the state that the server has sent in its latest response to an update request or in other words the most up to date state that is available locally at the time of the call.

See also

opFindFriends(), **Listener::onFindFriendsResponse()**, **FriendInfo**, **getFriendListAge()**

§ getFriendListAge()

```
int getFriendListAge ( void ) const
```

Returns

the time in ms that has passed since the last update has been applied to the list that is returned by [getFriendList\(\)](#) or 0 if either no friendlist is available yet or if a request for an update is in progress at the time of the call.

§ getDisconnectedCause()

```
int getDisconnectedCause ( void ) const
```

Summarizes (aggregates) the different causes for disconnects of a client. A disconnect can be caused by: errors in the network connection or some vital operation failing (which is considered "high level"). While operations always trigger a call to `OnOperationResponse`, connection related changes are treated in `OnStatusChanged`. The **DisconnectCause** is set in either case and summarizes the causes for any disconnect in a single state value which can be used to display (or debug) the cause for disconnection.

Returns

the disconnect cause.

§ getUserID()

```
const JString & getUserID ( void ) const
```

Returns

the unique user ID

See also

setUserID()

§ getRegionWithBestPing()

```
const JString & getRegionWithBestPing ( void ) const
```

Returns

the region code of the **Photon** Cloud region to which the client has the best ping.

Remarks

When you specify `RegionSelectionMode::BEST` on constructing the **Client** instance, then on first connect the **Client** will acquire a list of available regions and of their addresses and ping each of them multiple times. Afterwards it will connect to the region with the lowest average ping. After you got a call to `Listener::connectReturn()`, the region code of the region that the **Client** has chosen based on the ping results can get accessed by a call to this function. Later calls to `connect()` will use that cached region code to avoid re-doing the time-consuming ping-procedure and therefore to keep the time short that is needed for establishing a connection. For the same reason it is recommended that you acquire the result of the ping-procedure through this function and store it in local persistent storage, so that you can use it with `RegionSelectionMode::SELECT`. This way you can avoid the time-consuming pinging procedure even for the first connect after constructing the class, if you already have the region code for the region with the best ping stored locally from a connection on another **Client** instance (for example after your app has been shut down and restarted). However in this case you may want to provide an option to your users through which they can delete your locally stored region code and this way trigger a re-pinging on the next construction of a **Client** instance.

Note

This function will return an empty string, if no ping result is available (yet), which is the case when another `RegionSelectionMode` than `BEST` has been chosen or when you have not received the call to `Listener::connectReturn()` yet that corresponds to your first successfully established connection

since the construction of this class.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++ Client API 4.1.12.2

ExitGames

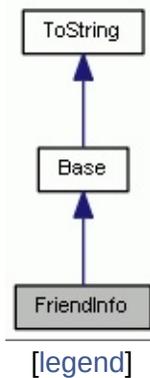
LoadBalancing

FriendInfo

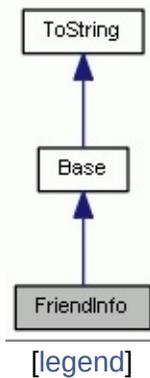
Public Member Functions | List of all members

FriendInfo Class Reference

Inheritance diagram for FriendInfo:



Collaboration diagram for FriendInfo:



Public Member Functions

Common::JString **getUserID** (void) const

bool **getIsOnline** (void) const

Common::JString **getRoom** (void) const

bool **getIsInRoom** (void) const

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Used to store the information about a friend's online status and in which room he/she is active.

See also

[Client::getFriendList\(\)](#), [Client::getFriendListAge\(\)](#),
[Client::opFindFriends\(\)](#), [Listener::onFindFriendsResponse\(\)](#)

Member Function Documentation

§ getUserID()

```
JString getUserID ( void ) const
```

Returns

the user ID of the friend

§ getIsOnline()

```
bool getIsOnline ( void ) const
```

Returns

true if the friend is online, false otherwise

§ getRoom()

```
JString getRoom ( void ) const
```

Returns

the name of the room in which the friend currently is active in, or an empty string, if it is not active inside any room at all.

§ getIsInRoom()

```
bool getIsInRoom ( void ) const
```

Returns

true if the friend is active inside a room, false otherwise.

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                    withTypes = false  
                    ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++ Client API 4.1.12.2

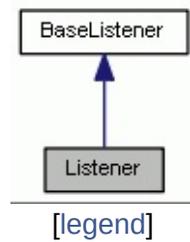
ExitGames > LoadBalancing > Listener

[Public Member Functions](#) | [List of all members](#)

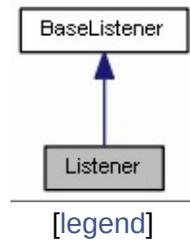
Listener Class

Reference **abstract**

Inheritance diagram for Listener:



Collaboration diagram for Listener:



Public Member Functions

virtual void **debugReturn** (int debugLevel, const **Common::JString** &string)=0

virtual void **connectionErrorReturn** (int errorCode)=0

virtual void **clientErrorReturn** (int errorCode)=0

virtual void **warningReturn** (int warningCode)=0

virtual void **serverErrorReturn** (int errorCode)=0

virtual void **joinRoomEventAction** (int playerNr, const **Common::JVector**< int > &playernrs, const **Player** &player)=0

virtual void **leaveRoomEventAction** (int playerNr, bool isInactive)=0

virtual void **customEventAction** (int playerNr, nByte eventCode, const **Common::Object** &eventContent)=0

virtual void **connectReturn** (int errorCode, const **Common::JString** &errorString, const **Common::JString** ®ion, const **Common::JString** &cluster)=0

virtual void **disconnectReturn** (void)=0

virtual void **createRoomReturn** (int localPlayerNr, const **Common::Hashtable** &roomProperties, const **Common::Hashtable** &playerProperties, int errorCode, const **Common::JString** &errorString)=0

virtual void **joinOrCreateRoomReturn** (int localPlayerNr, const **Common::Hashtable** &roomProperties, const **Common::Hashtable** &playerProperties, int errorCode, const **Common::JString** &errorString)=0

virtual void **joinRoomReturn** (int localPlayerNr, const **Common::Hashtable** &roomProperties, const **Common::Hashtable** &playerProperties, int errorCode, const **Common::JString** &errorString)=0

virtual void **joinRandomRoomReturn** (int localPlayerNr, const **Common::Hashtable** &roomProperties, const **Common::Hashtable** &playerProperties, int errorCode, const **Common::JString** &errorString)=0

virtual void **leaveRoomReturn** (int errorCode, const **Common::JString** &errorString)=0

virtual void **joinLobbyReturn** (void)=0

virtual void **leaveLobbyReturn** (void)=0

virtual void **onFindFriendsResponse** (void)

virtual void **onLobbyStatsResponse** (const **Common::JVector**<**LobbyStatsResponse** > &)

virtual void **webRpcReturn** (int, const **Common::JString** &, const **Common::JString** &, int, const **Common::Dictionary**<**Common::Object**, **Common::Object** > &)

virtual void **onRoomListUpdate** (void)

virtual void **onRoomPropertiesChange** (const **Common::Hashtable** &)

virtual void **onPlayerPropertiesChange** (int, const **Common::Hashtable** &)

virtual void **onAppStatsUpdate** (void)

virtual void **onLobbyStatsUpdate** (const **Common::JVector**<

LobbyStatsResponse > &)

virtual void **onCacheSliceChanged** (int)

virtual void **onMasterClientChanged** (int, int)

virtual void **onCustomAuthenticationIntermediateStep** (const **Common::Dictionary**< **Common::JString**, **Common::Object** > &)

virtual void **onAvailableRegions** (const **Common::JVector**< **Common::JString** > &, const **Common::JVector**< **Common::JString** > &)

virtual void **onSecretReceival** (const **Common::JString** &)

virtual void **onDirectMessage** (const **Common::Object** &, int, bool)

virtual void **onCustomOperationResponse** (const **Photon::OperationResponse** &operationResponse)

Member Function Documentation

§ debugReturn()

```
virtual void  
debugReturn      ( int                debugLevel,  
                  const Common::JString & string  
                  )
```

pure virtual

This is the callback function for debug-messages.

Parameters

debugLevel one of the values in DebugLevel

string the formatted debug string

See also

BaseListener

Implements **BaseListener**.

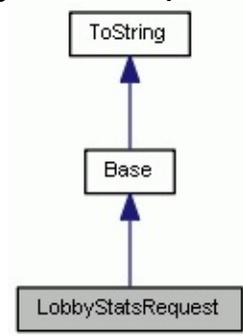
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > LobbyStatsRequest

[Public Member Functions](#) | [List of all members](#)

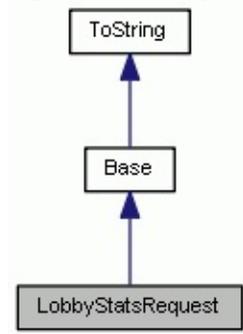
LobbyStatsRequest Class Reference

Inheritance diagram for LobbyStatsRequest:



[legend]

Collaboration diagram for LobbyStatsRequest:



[legend]

Public Member Functions

LobbyStatsRequest (const
Common::JString
&name=**Common::JString**(), nByte
type=**LobbyType::DEFAULT**)

const **Common::JString** & **getName** (void) const

nByte **getType** (void) const

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool
withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **Tostring**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Passed to **Client::opLobbyStats()**. Each instance of this class holds the name and the type of a lobby for which the caller of **Client::opLobbyStats()** wants to request statistics.

See also

Client::opLobbyStats(), **Listener::onLobbyStatsResponse()**,
Listener::onLobbyStatsUpdate(), **LobbyStatsResponse**

Constructor & Destructor Documentation

§ LobbyStatsRequest()

```
LobbyStatsRequest ( const Common::JString & name = Common::JStr  
                    nByte                               type = LobbyType::DE  
                    )
```

Constructor: Creates a new instance with the specified parameters.

Note

Lobby names are only unique per lobby type and multiple lobbies with the same name, but different type, can exist in parallel. Hence a lobby with the same name but with a different type is treated as a different lobby.

Parameters

name see setLobbyName() - optional, defaults to an empty JString instance.

type see setLobbyType() - optional, defaults to **LobbyType::DEFAULT**. Must be one of the values in **Lobby**

Member Function Documentation

§ getName()

```
const JString & getName ( void ) const
```

Returns

the lobby name

See also

[LobbyStatsRequest\(\)](#)

§ getType()

```
nByte getType ( void ) const
```

Returns

the lobby type

See also

[LobbyStatsRequest\(\)](#)

§ toString()

```
JString & toString ( Common::JString & retStr,  
                  bool withTypes = false  
                  ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

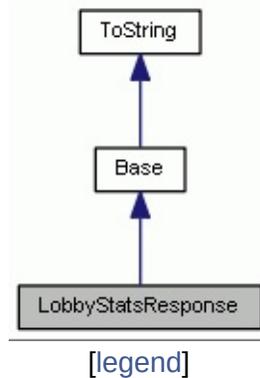
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > LobbyStatsResponse

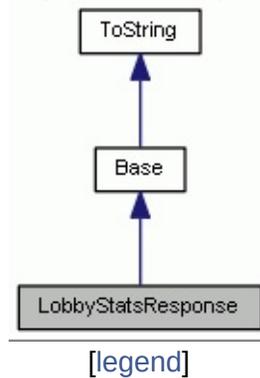
[Public Member Functions](#) | [List of all members](#)

LobbyStatsResponse Class Reference

Inheritance diagram for LobbyStatsResponse:



Collaboration diagram for LobbyStatsResponse:



Public Member Functions

const **Common::JString** & **getName** (void) const

nByte **getType** (void) const

int **getPeerCount** (void) const

int **getRoomCount** (void) const

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Passed to `Listener::onLobbyStatsResponse()`, `Listener::onLobbyStatsUpdate()`. Each instance of this class holds the name, the type and the statistics (peer count and room count) of one specific lobby. Each lobby can be uniquely identified by the combination of its name and type.

See also

[Client::opLobbyStats\(\)](#), `Listener::onLobbyStatsResponse()`, `Listener::onLobbyStatsUpdate()`, [LobbyStatsRequest](#)

Member Function Documentation

§ getName()

```
const JString & getName ( void ) const
```

Returns

the lobby name. Each lobby can be uniquely identified by the combination of its name and type.

§ getType()

```
nByte getType ( void ) const
```

Returns

the lobby type. Each lobby can be uniquely identified by the combination of its name and type.

§ getPeerCount()

```
int getPeerCount ( void ) const
```

Returns

the number of clients that currently reside in this specific lobby

§ getRoomCount()

```
int getRoomCount ( void ) const
```

Returns

the number of clients that currently exist and that belong to this specific lobby.

On room creation the creator of the room can specify the name and type of the lobby to which that room gets assigned in the **RoomOptions**.

§ toString()

```
JString & toString ( Common::JString & retStr,  
                  bool                    withTypes = false  
                  )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

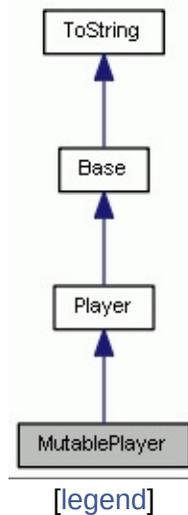
Returns

a JString representation of the instance and its contents for debugging purposes.

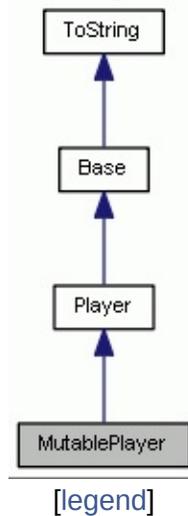
Implements **ToString**.

MutablePlayer Class Reference

Inheritance diagram for MutablePlayer:



Collaboration diagram for MutablePlayer:



Public Member Functions

MutablePlayer (const **MutablePlayer** &toCopy)

virtual **MutablePlayer** & **operator=** (const **Player** &toCopy)

virtual **MutablePlayer** & **operator=** (const **MutablePlayer** &toCopy)

void **setName** (const **Common::JString** &name, const **WebFlags** &webflags=**WebFlags**())

void **mergeCustomProperties** (const **Common::Hashtable** &customProperties, const **WebFlags** &webflags=**WebFlags**())

template<typename ktype , typename vtype >

void **addCustomProperty** (const ktype &key, const vtype &value, const **WebFlags** &webflags=**WebFlags**())

template<typename ktype , typename vtype >

void **addCustomProperty** (const ktype &key, const vtype pValueArray, typename **Common::Helpers::ArrayLengthType**<vtype >::type arrSize, const **WebFlags** &webflags=**WebFlags**())

template<typename ktype , typename vtype >

void **addCustomProperty** (const ktype &key, const vtype pValueArray, const short *pArrSizes, const **WebFlags** &webflags=**WebFlags**())

```
void addCustomProperties (const  
Common::Hashtable  
&customProperties, const WebFlags  
&webflags=WebFlags())
```

```
template<typename ktype >
```

```
void removeCustomProperty (const ktype  
&key, const WebFlags  
&webflags=WebFlags())
```

```
template<typename ktype >
```

```
void removeCustomProperties (const ktype  
*keys, unsigned int count, const  
WebFlags &webflags=WebFlags())
```

► Public Member Functions inherited from **Player**

```
virtual ~Player (void)
```

```
Player (const Player &toCopy)
```

```
int getNumber (void) const
```

```
const Common::JString & getName () const
```

```
const Common::JString & getUserID () const
```

```
const Common::Hashtable & getCustomProperties () const
```

```
bool getIsInactive (void) const
```

```
bool getIsMasterClient (void) const
```

```
bool operator== (const Player &player)  
const
```

```
virtual Common::JString & toString (Common::JString &retStr,  
bool withTypes=false) const
```

virtual **Common::JString toString** (bool withTypes, bool withCustomProperties) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Member Function Documentation

§ operator=()

MutablePlayer & operator= (const **Player** & toCopy)

virtual

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

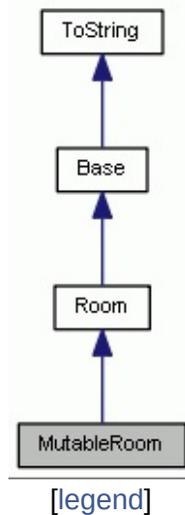
Reimplemented from **Player**.

© Exit Games ®, all rights reserved.

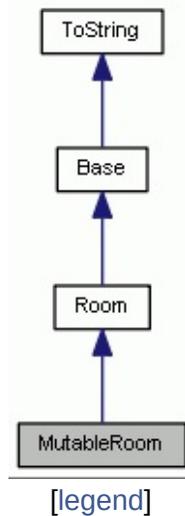
[Photon Documentation](#) | [Contact](#) | [Terms](#)

MutableRoom Class Reference

Inheritance diagram for MutableRoom:



Collaboration diagram for MutableRoom:



Public Member Functions

	MutableRoom (const
virtual MutableRoom &	operator= (const Rc
virtual MutableRoom &	operator= (const Mu
nByte	getPlayerCount (vo
void	setMaxPlayers (nBy &webflags= WebFlag
void	setIsOpen (bool isO &webflags= WebFlag
bool	getIsVisible (void) c
void	setIsVisible (bool is &webflags= WebFlag
const Common::JVector < Player * > &	getPlayers (void) cc
const Player *	getPlayerForNumb
int	getMasterClientID (
const Common::JVector < Common::JString > &	getPropsListedInLc
void	setPropsListedInLc Common::JString > Common::JVector < &expectedList= Com Common::JString > &webflags= WebFlag
int	getPlayerTtl (void) c

	int	getEmptyRoomTtl (
	bool	getSuppressRoomI
const Common::JVector < Common::JString > *		getPlugins (void) cc
	bool	getPublishUserID (
const Common::JVector < Common::JString > &		getExpectedUsers
	void	setExpectedUsers Common::JString > WebFlags &webflag
	void	mergeCustomProp Common::Hashtab Common::Hashtab &expectedCustomPi const WebFlags &w
template<typename ktype , typename vtype >	void	addCustomPropert &value, const Comn &expectedCustomPi const WebFlags &w
template<typename ktype , typename vtype >	void	addCustomPropert pValueArray, typena Common::Helpers::A arrSize, const Comr &expectedCustomPi const WebFlags &w
template<typename ktype , typename vtype >	void	addCustomPropert pValueArray, const s

Common::Hashtab
&expectedCustomPi
const **WebFlags** &w

void **addCustomPropert**
&customProperties,
&expectedCustomPi
const **WebFlags** &w

template<typename ktype >

void **removeCustomPro**
Common::Hashtab
&expectedCustomPi
const **WebFlags** &w

template<typename ktype >

void **removeCustomPro**
unsigned int count, c
&expectedCustomPi
const **WebFlags** &w

virtual **Common::JString** **toString** (bool withT,
withCustomPropertie
const

► Public Member Functions inherited from **Room**

virtual **~Room** (void)

Room (const **Room**

const **Common::JString** & **getName** (void) const

nByte **getMaxPlayers** (void)

bool **getIsOpen** (void) const

nByte **getDirectMode** (void)

const **Common::Hashtable** & **getCustomProperti**

bool **operator==** (const R

virtual **Common::JString** & **toString** (**Common:**
withTypes=false) cor

virtual **Common::JString** **toString** (bool withT,
const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void)

JString **toString** (bool withT,

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Member Function Documentation

§ operator=()

MutableRoom & operator= (const **Room** & toCopy)

virtual

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

Reimplemented from **Room**.

§ getPlayerCount()

```
nByte getPlayerCount ( void ) const
```

virtual

Returns

the count of players that are currently inside this room

Reimplemented from **Room**.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

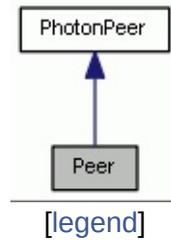
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > Peer >

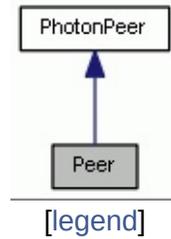
[Public Member Functions](#) | [List of all members](#)

Peer Class Reference

Inheritance diagram for Peer:



Collaboration diagram for Peer:



Public Member Functions

	Peer (Photon::PhotonListene connectionProtocol=Photon::C
virtual void	disconnect (void)
virtual bool	opJoinLobby (const Common &lobbyName= Common::JStri lobbyType= LobbyType::DEFA
virtual bool	opLeaveLobby (void)
virtual bool	opCreateRoom (const Comm RoomOptions &options= Room Common::Hashtable &customLocalPlayerProperties: const Common::JVector < Cor &expectedUsers= Common::JV ())
virtual bool	opJoinRoom (const Common RoomOptions &options= Room Common::Hashtable &customLocalPlayerProperties: bool createlfNotExists=false, bc cacheSliceIndex=0, const Com Common::JString > &expecte Common::JString >())
virtual bool	opJoinRandomRoom (const C &customRoomProperties= Com maxPlayers=0, nByte matchmakingMode= Matchmak const Common::JString &lobbyName= Common::JStri lobbyType= LobbyType::DEFA Common::JString &sqlLobbyF

```
const Common::JVector< Cor
&expectedUsers=Common::J
()
```

```
virtual bool opLeaveRoom (bool willCome
sendAuthCookie=false)
```

```
template<typename Ftype >
```

```
bool opRaiseEvent (bool reliable, c
nByte eventCode, const RaiseE
&options=RaiseEventOptions(
```

```
template<typename Ftype >
```

```
bool opRaiseEvent (bool reliable, c
typename Common::Helpers::A
>::type arrSize, nByte eventCo
RaiseEventOptions &options=
```

```
template<typename Ftype >
```

```
bool opRaiseEvent (bool reliable, c
const short *pArrSizes, nByte e
RaiseEventOptions &options=
```

```
virtual bool opAuthenticate (const Comm
Common::JString &appVersio
AuthenticationValues
&authenticationValues=Authen
lobbyStats=false, const Comm
&regionCode=Common::JStri
```

```
virtual bool opAuthenticateOnce (const C
const Common::JString &app'
connectionProtocol, nByte encr
AuthenticationValues
&authenticationValues=Authen
lobbyStats=false, const Comm
&regionCode=Common::JStri
```

```
virtual bool opFindFriends (const Comm
```

short numFriendsToFind)

virtual bool **opLobbyStats** (const **Common::LoadBalancing::LobbyStatsR** &lobbiesToQuery=**Common::LoadBalancing::LobbyStatsR**

virtual bool **opChangeGroups** (const **Common::LoadBalancing::LobbyStatsR** *pGroupsToRemove, const **Common::LoadBalancing::LobbyStatsR** *pGroupsToAdd)

virtual bool **opWebRpc** (const **Common::LoadBalancing::LobbyStatsR**

template<typename Ftype >

bool **opWebRpc** (const **Common::LoadBalancing::LobbyStatsR** ¶meters, bool sendAuthCookie)

template<typename Ftype >

bool **opWebRpc** (const **Common::LoadBalancing::LobbyStatsR** *pParameterArray, typename **Common::Helpers::ArrayLengthT** bool sendAuthCookie=false)

template<typename Ftype >

bool **opWebRpc** (const **Common::LoadBalancing::LobbyStatsR** *pParameterArray, const short *p sendAuthCookie=false)

virtual bool **opGetRegions** (bool encrypted &appId)

virtual bool **opSetPropertiesOfPlayer** (int **Common::Hashtable** &properties **Common::Hashtable** &expectedProperties=**Common::LoadBalancing::WebFlags** webFlags=**WebFlags**())

virtual bool **opSetPropertiesOfRoom** (const **Common::LoadBalancing::LobbyStatsR** &properties, const **Common::LoadBalancing::WebFlags**

&expectedProperties=**Common**
webFlags=**WebFlags**())

► **Public Member Functions inherited from PhotonPeer**

PhotonPeer (**PhotonListener**
connectionProtocol=**Connector**

virtual **~PhotonPeer** (void)

virtual bool **connect** (const **Common::JSt**
Common::JString &appId=**Co**

template<typename Ftype >

bool **connect** (const **Common::JSt**
Common::JString &appId, coi

template<typename Ftype >

bool **connect** (const **Common::JSt**
Common::JString &appId, coi
typename **Common::Helpers::A**
>::type arrSize)

template<typename Ftype >

bool **connect** (const **Common::JSt**
Common::JString &appId, coi
const short *pArrSizes)

virtual void **service** (bool **dispatchIncomir**

virtual void **serviceBasic** (void)

virtual bool **opCustom** (const **OperationR**
bool sendReliable, nByte charr

virtual bool **sendOutgoingCommands** (vo

virtual bool **sendAcksOnly** (void)

	virtual bool	dispatchIncomingCommands
	virtual bool	establishEncryption (void)
	virtual void	fetchServerTimestamp (void)
	virtual void	resetTrafficStats (void)
	virtual void	resetTrafficStatsMaximumCo
	virtual Common::JString	vitalStatsToString (bool all) co
	virtual void	pingServer (const Common::J int pingAttempts)
	virtual void	initUserDataEncryption (cons > &secret)
	virtual void	initUDPEncryption (const Cor &encryptSecret, const Commo &HMACSecret)
	PhotonListener *	getListener (void)
	int	getServerTimeOffset (void) co
	int	getServerTime (void) const
	int	getBytesOut (void) const
	int	getBytesIn (void) const
	int	getByteCountCurrentDispatc
	int	getByteCountLastOperation (
	int	getPeerState (void) const

	int	getSentCountAllowance	(void)
	void	setSentCountAllowance	(int s
	int	getTimePingInterval	(void) cor
	void	setTimePingInterval	(int timeP
	int	getRoundTripTime	(void) cons
	int	getRoundTripTimeVariance	(v
	int	getTimestampOfLastSocketR	
	int	getDebugOutputLevel	(void) c
	bool	setDebugOutputLevel	(int deb
const Common::LogFormatOptions &		getLogFormatOptions	(void) c
	void	setLogFormatOptions	(const Common::LogFormatOptions
	int	getIncomingReliableCommar	
	short	getPeerID	(void) const
	int	getDisconnectTimeout	(void)
	void	setDisconnectTimeout	(int dis
	int	getQueuedIncomingComman	
	int	getQueuedOutgoingComman	

Common::JString **getServerAddress** (void) const

bool **getIsPayloadEncryptionAvailable**

bool **getIsEncryptionAvailable** (void)

int **getResentReliableCommands**

int **getLimitOfUnreliableCommands**

void **setLimitOfUnreliableCommands**

bool **getCRCEnabled** (void) const

void **setCRCEnabled** (bool crcEnabled)

int **getPacketLossByCRC** (void) const

bool **getTrafficStatsEnabled** (void)

void **setTrafficStatsEnabled** (bool enabled)

int **getTrafficStatsElapsedMs** (void)

const **TrafficStats** & **getTrafficStatsIncoming** (void)

const **TrafficStats** & **getTrafficStatsOutgoing** (void)

const **TrafficStatsGameLevel** & **getTrafficStatsGameLevel** (void)

nByte **getQuickResendAttempts** (void)

void **setQuickResendAttempts** (nByte attempts)

nByte **getConnectionProtocol** (void)

void **setConnectionProtocol** (nByte

nByte **getChannelCountUserChann**

Additional Inherited Members

▶ **Static Public Member Functions inherited from PhotonPeer**

static short **getPeerCount** (void)

static unsigned int **getMaxAppIDLength** (void)

Member Function Documentation

§ disconnect()

```
void disconnect ( void )
```

virtual

Initiates the disconnection from the Photon server. The servers response will arrive in [PhotonListener::onStatusChanged\(\)](#).

This function generates a disconnection request that will be sent to the Photon server. If the disconnection is completed successfully, then the [PhotonListener::onStatusChanged\(\)](#) callback will be called, with a statusCode of `StatusCodes::DISCONNECT`.

Remarks

If a game room is joined, when this function gets called, then the local player leaves that room as if `opLeaveRoom()` has been called with parameter 'willComeBack' set to 'true'. Please see there for further information about leaving rooms. However no call to `Listener::leaveRoomReturn()` will happen when leaving a game room is triggered through a call to [disconnect\(\)](#).

See also

[connect\(\)](#)

Reimplemented from [PhotonPeer](#).

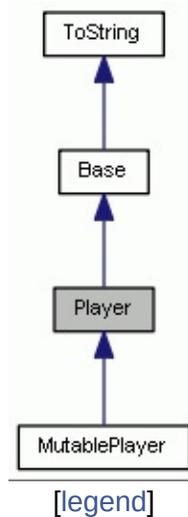
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > Player >

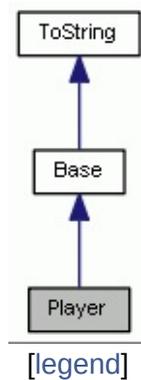
[Public Member Functions](#) | [List of all members](#)

Player Class Reference

Inheritance diagram for Player:



Collaboration diagram for Player:



Public Member Functions

virtual **~Player** (void)

Player (const **Player** &toCopy)

virtual **Player** & **operator=** (const **Player** &toCopy)

int **getNumber** (void) const

const **Common::JString** & **getName** () const

const **Common::JString** & **getUserID** () const

const **Common::Hashtable** & **getCustomProperties** () const

bool **getIsInactive** (void) const

bool **getIsMasterClient** (void) const

bool **operator==** (const **Player** &player)
const

virtual **Common::JString** & **toString** (**Common::JString** &retStr,
bool withTypes=false) const

virtual **Common::JString** **toString** (bool withTypes, bool
withCustomProperties) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **Tostring**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Each client inside a **MutableRoom** is represented by an instance of this class.

Player instances are only valid in the context of the `MutableRoom()` instance from which they have been retrieved.

See also

MutablePlayer, **MutableRoom**, `MutableRoom::getPlayers()`,
`MutableRoom::getPlayerForNumber()`

Constructor & Destructor Documentation

§ ~Player()

`~Player (void)`

virtual

Destructor.

§ Player()

```
Player ( const Player & toCopy )
```

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

```
Player & operator= ( const Player & toCopy )
```

virtual

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

Reimplemented in **MutablePlayer**.

§ `getNumber()`

```
int getNumber ( void ) const
```

Returns

the player number

The player number serves as a the unique identifier of a player inside the current room.

It is assigned per room and only valid in the context of that room. A player number is never re-used for another player inside the same room.

If a player leaves a room entirely (either explicitly through a call to **`Client::opLeaveRoom()`** without passing 'true' for parameter 'willComeBack' or implicitly because his `playerTtl` runs out (see **`RoomOptions::setPlayerTtl()`**)) and joins it again afterwards, then he is treated as an entirely new player and gets assigned a new player number.

If a player becomes inactive (either explicitly through a call to **`Client::opLeaveRoom()`** with passing 'true' for parameter 'willComeBack' or implicitly by by getting disconnected) and then rejoins the same room before his `playerTtl` runs out, then he is treated as the same player an keeps his previously assigned player number.

§ getName()

```
const JString & getName ( void ) const
```

Returns

the non-unique nickname of this player

A player might change his own name.

Such a change is synced automatically with the server and other clients in the same room.

§ getUserID()

```
const JString & getUserID ( void ) const
```

Returns

the unique user ID of this player

This value is only available when the room got created with `RoomOptions::setPublishUserId(true)`. Otherwise it will be an empty string.

Useful for `Client::opFindFriends()` and for blocking slots in a room for expected users (see `MutableRoom::getExpectedUsers()`).

See also

[AuthenticationValues](#)

§ getCustomProperties()

```
const Hashtable & getCustomProperties ( void ) const
```

Returns

the custom properties of this player

Read-only cache for the custom properties of a player. A client can change the custom properties of his local player instance through class **MutablePlayer**. The Custom Properties of remote players are automatically updated when they change.

§ getIsInactive()

```
bool getIsInactive ( void ) const
```

Returns

'true' if a player is inactive, 'false' otherwise.

Inactive players keep their spot in a room but otherwise behave as if offline (no matter what their actual connection status is).

The room needs a `PlayerTtl != 0` (see [RoomOptions::setPlayerTtl\(\)](#)) for players to be able to become inactive. If a player is inactive for longer than the `PlayerTtl`, then the server will remove this player from the room.

§ getIsMasterClient()

```
bool getIsMasterClient ( void ) const
```

Returns

'true' if this player is the Master **Client** of the current room, 'false' otherwise.

There is always exactly one master client. The creator of a room gets assigned the role of master client on room creation.

When the current master client leaves the room or becomes inactive and there is at least one active player inside the room, then the role of master client gets reassigned by the server to an active client. As soon as one client becomes active again in a room with only inactive clients, the role of master client will be assigned to this active client.

Whenever the role of master client gets assigned to a different client, all active clients inside the same room get informed about it by a call to `Listener::onMasterClientChanged()`.

You can use the master client when you want one client to be an authoritative instance.

See also

`MutableRoom::getMasterClientID()`,
`Listener::onMasterClientChanged()`,
`DirectMode::MASTER_TO_ALL`

§ operator==()

```
bool operator==( const Player & player ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

Two **Player** instances are considered equal, if **getNumber()** returns equal values for both of them.

§ toString() [1/2]

```
JString & toString ( Common::JString & retStr,  
                  bool                    withTypes = false  
                  )                          const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

§ toString() [2/2]

```
JString toString ( bool withTypes,  
                  bool withCustomProperties  
                  )      const
```

virtual

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- withTypes** set to true, to include type information in the generated string
- withCustomProperties** set to true, to include the custom properties in the generated string

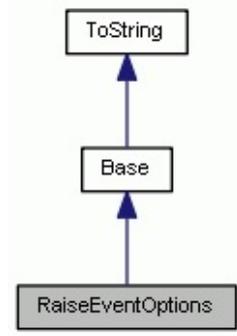
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > RaiseEventOptions

[Public Member Functions](#) | [List of all members](#)

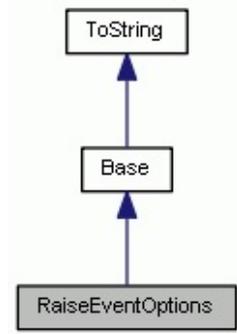
RaiseEventOptions Class Reference

Inheritance diagram for RaiseEventOptions:



[legend]

Collaboration diagram for RaiseEventOptions:



[legend]

Public Member Functions

RaiseEventOptions (nByte channelId=0, r
eventCaching=Lite::EventCache::DO_NOT
const int *targetPlayers=NULL, short
numTargetPlayers=0, nByte
receiverGroup=Lite::ReceiverGroup::OTHE
nByte interestGroup=0, const **WebFlags**
&webFlags=**WebFlags**(), int cacheSliceInd

~RaiseEventOptions (void)

RaiseEventOptions (const **RaiseEventOp**
&toCopy)

RaiseEventOptions & **operator=** (const **RaiseEventOptions** &toC

nByte **getChannelID** (void) const

RaiseEventOptions & **setChannelID** (nByte channelId)

nByte **getEventCaching** (void) const

RaiseEventOptions & **setEventCaching** (nByte eventCaching)

const int * **getTargetPlayers** (void) const

short **getNumTargetPlayers** (void) const

RaiseEventOptions & **setTargetPlayers** (const int *targetPlayers,
numTargetPlayers)

nByte **getReceiverGroup** (void) const

RaiseEventOptions & **setReceiverGroup** (nByte receiverGroup)

nByte **getInterestGroup** (void) const

RaiseEventOptions & **setInterestGroup** (nByte interestGroup)

const **WebFlags** & **getWebFlags** (void) const

RaiseEventOptions & **setWebFlags** (const **WebFlags** &webFlags)

int **getCacheSliceIndex** (void) const

RaiseEventOptions & **setCacheSliceIndex** (int cacheSliceIndex)

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This class aggregates the various optional parameters that can be passed to **Client::opRaiseEvent()**.

See also

Client::opRaiseEvent()

Constructor & Destructor Documentation

§ RaiseEventOptions() [1/2]

```
RaiseEventOptions ( nByte          channelId = 0,  
                    nByte          eventCaching = Lite::EventCa  
                    const int *    targetPlayers = NULL,  
                    short          numTargetPlayers = 0,  
                    nByte          receiverGroup = Lite::Receiv  
                    nByte          interestGroup = 0,  
                    const WebFlags & webFlags = webFlags(),  
                    int            cacheSliceIndex = 0  
                    )
```

Constructor: Creates a new instance with the specified parameters.

Parameters

channelID	see setChannelID() - optional, defaults to 0.
eventCaching	see setEventCaching() - optional, defaults to Lite::EventCache::DO_NOT_CACHE.
targetPlayers	see setTargetPlayers() - optional, defaults to N
numTargetPlayers	see setTargetPlayers() - optional, defaults to 0
receiverGroup	see setReceiverGroup() - optional, defaults to Lite::ReceiverGroup::OTHERS.
interestGroup	see setInterestGroup() - optional, defaults to 0
webFlags	see setWebFlags() - optional, defaults to a default WebFlags instance.
cacheSliceIndex	see setCacheSliceIndex() - optional, defaults to t

§ ~RaiseEventOptions()

`~RaiseEventOptions (void)`

Destructor.

§ RaiseEventOptions() [2/2]

RaiseEventOptions (const **RaiseEventOptions** & **toCopy**)

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

RaiseEventOptions &
operator=

(const **RaiseEventOptions** & toCopy)

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ getChannelID()

```
nByte getChannelID ( void ) const
```

Returns

the currently set channel ID

See also

[setChannelID\(\)](#)

§ setChannelID()

RaiseEventOptions & setChannelID (nByte **channelID**)

Sets the channel ID.

Please see **Fragmentation and Channels** for further information.

Parameters

channelID the ID of the channel on which to send the message.
Needs to be in the range from 0 to
Client::getChannelCountUserChannels()

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getChannelID()

§ `getEventCaching()`

```
nByte getEventCaching ( void ) const
```

Returns

the currently set event caching option

See also

[setEventCaching\(\)](#)

§ setEventCaching()

RaiseEventOptions & setEventCaching (nByte **eventCaching**)

Sets the event caching option.

This option defines if the server should simply send the event, put it in the cache, remove events that are like this one or if the Cache Slice should be modified. Leave this to the default value of DO_NOT_CACHE to not use the EventCache at all.

For a more in-depth description about event caching please see [Cached Events](#)

Remarks

When using one of the options SLICE_SET_INDEX, SLICE_PURGE_INDEX or SLICE_PURGE_UP_TO_INDEX, you also need to provide a value for the CacheSliceIndex by a call to [setCacheSliceIndex\(\)](#). All other options except [setChannelID\(\)](#) and also all other parameters of [Client::opRaiseEvent\(\)](#) get ignored in this case.

Note

The value that you set for this option gets ignored if any of the following statements is true:

[getReceiverGroup\(\)](#) == ReceiverGroup::MASTER_CLIENT

[getTargetPlayers\(\)](#) != NULL

[getInterestGroup\(\)](#) != 0

Parameters

eventCaching needs to be one of the values from [Lite::EventCache](#)

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getEventCaching\(\)](#), [Cached Events](#)

§ `getTargetPlayers()`

```
const int * getTargetPlayers ( void ) const
```

Returns

the currently set array of target players

See also

[getNumTargetPlayers\(\)](#), [setTargetPlayers\(\)](#)

§ `getNumTargetPlayers()`

```
short getNumTargetPlayers ( void ) const
```

Returns

the number of elements in the array that is returned by `getTargetPlayers()`

See also

`getTargetPlayers()`, `setTargetPlayers()`

§ setTargetPlayers()

RaiseEventOptions &
setTargetPlayers

```
( const int * targetPlayers,  
  short      numTargetPlayers  
)
```

Sets the target players.

Set this to the **Player** numbers of the clients, which should receive the event. The default value when not setting anything is NULL and equivalent to an array that consists of the player numbers of all clients inside the room except for the sending client itself. **Player** Numbers that do not correspond to any active player inside the room will get ignored by the server.

Note

If you set this option to anything else than NULL, then any value that might have been passed for **setEventCaching()** will be ignored.

The options **setTargetPlayers()**, **setInterestGroup()** and **setReceiverGroup()** provide alternative ways of specifying the receivers of an event and can not be combined with each other.

If **getTargetPlayers()** evaluates to !NULL, then the value for the target players gets used and the values for the other 2 options get ignored.

Otherwise, if **getInterestGroup()** evaluates to !0, then the value for the interest group gets used and the value for the receiver group gets ignored.

Else the value for the receiver group gets used.

Parameters

targetPlayers

either NULL (to reset the value of the option to the default) or an array of integer values that correspond to the player numbers of the intended receivers

numTargetPlayers the element count of the array that is passed for targetPlayers

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getTargetPlayers\(\)](#), [getNumTargetPlayers\(\)](#)

§ `getReceiverGroup()`

```
nByte getReceiverGroup ( void ) const
```

Returns

the currently set receiver group

See also

[setReceiverGroup\(\)](#)

§ setReceiverGroup()

RaiseEventOptions & setReceiverGroup (nByte receiverGroup)

Sets the receiver group.

Set this to one of the values from **Lite::ReceiverGroup**. The default value when not setting anything is `Lite::ReceiverGroup::OTHERS`.

Note

If you set this option to `Lite::ReceiverGroup::MASTER_CLIENT`, then any value that might have been passed for **setEventCaching()** will be ignored.

The options **setTargetPlayers()**, **setInterestGroup()** and **setReceiverGroup()** provide alternative ways of specifying the receivers of an event and can not be combined with each other.

If **getTargetPlayers()** evaluates to `!NULL`, then the value for the target players gets used and the values for the other 2 options get ignored.

Otherwise, if **getInterestGroup()** evaluates to `!0`, then the value for the interest group gets used and the value for the receiver group gets ignored.

Else the value for the receiver group gets used.

Parameters

receiverGroup needs to be one of the values from **Lite::ReceiverGroup**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getReceiverGroup()

§ `getInterestGroup()`

```
nByte getInterestGroup ( void ) const
```

Returns

the currently set interest group

See also

[setInterestGroup\(\)](#)

§ setInterestGroup()

RaiseEventOptions & setInterestGroup (nByte **interestGroup**)

Sets the interest group.

Set this to a value between 0 and 255. The default value when not setting anything is 0. More information about interest groups can be found at [Interest Groups](#).

Note

If you set this option to anything else than 0, then any value that might have been passed for **setEventCaching()** will be ignored.

The options **setTargetPlayers()**, **setInterestGroup()** and **setReceiverGroup()** provide alternative ways of specifying the receivers of an event and can not be combined with each other.

If **getTargetPlayers()** evaluates to !NULL, then the value for the target players gets used and the values for the other 2 options get ignored.

Otherwise, if **getInterestGroup()** evaluates to !0, then the value for the interest group gets used and the value for the receiver group gets ignored.

Else the value for the receiver group gets used.

Parameters

interestGroup the number of the interest group to which the event should be sent

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getInterestGroup(), [Interest Groups](#)

§ `getWebFlags()`

```
const WebFlags & getWebFlags ( void ) const
```

Returns

the currently set web flags options

See also

[`setWebFlags\(\)`](#)

§ setWebFlags()

RaiseEventOptions & setWebFlags (const **WebFlags** & webFlags)

Sets the web flags options.

For more information see class **WebFlags**.

Parameters

webFlags an instance of class **WebFlags**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getWebFlags(), **WebFlags**

§ `getCacheSliceIndex()`

```
int getCacheSliceIndex ( void ) const
```

Returns

the currently set cache slice index

See also

[setCacheSliceIndex\(\)](#)

§ setCacheSliceIndex()

RaiseEventOptions & setCacheSliceIndex (int **cacheSliceIndex**)

Sets the index of the cache slice that should be used in conjunction with the value that you have passed to **setEventCaching()**.

When you pass one of the options SLICE_SET_INDEX, SLICE_PURGE_INDEX or SLICE_PURGE_UP_TO_INDEX to **setEventCaching()**, then you also need to provide the cache slice index for that option to **setCacheSliceIndex()**.

Parameters

cacheSliceIndex the index of the cache slice to which the event should be added

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getCacheSliceIndex(), **setEventCaching()**, **Cached Events**

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                    withTypes = false  
                    )                    const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

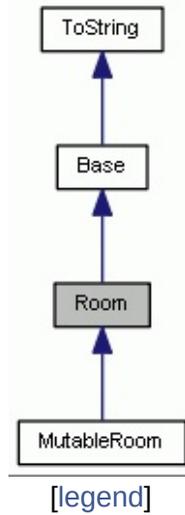
Returns

a JString representation of the instance and its contents for debugging purposes.

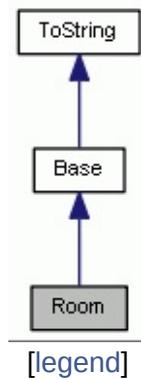
Implements **ToString**.

Room Class Reference

Inheritance diagram for Room:



Collaboration diagram for Room:



Public Member Functions

virtual **~Room** (void)

Room (const **Room** &toCopy)

virtual **Room** & **operator=** (const **Room** &toCopy)

const **Common::JString** & **getName** (void) const

virtual nByte **getPlayerCount** (void) const

nByte **getMaxPlayers** (void) const

bool **getIsOpen** (void) const

nByte **getDirectMode** (void) const

const **Common::Hashtable** & **getCustomProperties** (void) const

bool **operator==** (const **Room** &room) const

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool withTypes=false) const

virtual **Common::JString** **toString** (bool withTypes, bool withCustomProperties) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Each visible room inside the list of rooms in a lobby of type **LobbyType::DEFAULT** is represented by an instance of this class.

The information that is available through the various getters is regularly updated by the server as long as the client is inside the lobby. It is not updated and information will become outdated while the client is inside of a room.

See also

MutableRoom, **Client::getRoomList()**

Constructor & Destructor Documentation

§ ~Room()

~Room (void)

virtual

Destructor.

§ Room()

```
Room ( const Room & toCopy )
```

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

```
Room & operator= ( const Room & toCopy )
```

virtual

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

Reimplemented in **MutableRoom**.

§ getName()

```
const JString & getName ( void ) const
```

Returns

the name of the room

A rooms name is a unique identifier (per region and virtual appid) for a room/match.

It can be set set by the client on room creation as parameter of **Client::opCreateRoom()** or **Client::opJoinOrCreateRoom()**.

The name can't be changed once the room is created.

§ getPlayerCount()

```
nByte getPlayerCount ( void ) const
```

virtual

Returns

the count of players that are currently inside this room

Reimplemented in [MutableRoom](#).

§ getMaxPlayers()

```
nByte getMaxPlayers ( void ) const
```

Returns

the limit of players for this room. If a room is full (**getPlayerCount()** == **getMaxPlayers()**), joining this room will fail.

§ getIsOpen()

```
bool getIsOpen ( void ) const
```

Returns

'true' if the room can be joined, 'false' otherwise.

This does not affect listing in a lobby but joining a room will fail if it is not open.

If it is not open, then a room is excluded from random matchmaking.

Due to racing conditions, found matches might become closed even while you join them. Simply find another room in this scenario.

§ getDirectMode()

```
nByte getDirectMode ( void ) const
```

Returns

one of the values in **DirectMode**

This returns **DirectMode::NONE**, unless the client that created the room has set something else through **RoomOptions::setDirectMode()**

§ getCustomProperties()

```
const Hashtable & getCustomProperties ( void ) const
```

Returns

the custom properties of this room

Read-only cache for those custom properties of a room, which have been included in the list of properties to show in lobby (see **RoomOptions::setPropsListedInLobby()** and **MutableRoom::setPropsListedInLobby()**).

A client can change the custom properties of the currently joined room through class **MutableRoom**. The initial custom properties of a room can be set through class **RoomOptions**.

§ operator==()

```
bool operator==( const Room & room ) const
```

operator==.

Returns

true, if both operands are equal, false otherwise.

Two **Room** instances are considered equal, if **getName()** returns equal values for both of them.

§ toString() [1/2]

```
JString & toString ( Common::JString & retStr,  
                  bool                    withTypes = false  
                  ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

§ toString() [2/2]

```
JString toString ( bool withTypes,  
                  bool withCustomProperties  
                  )      const
```

virtual

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- withTypes** set to true, to include type information in the generated string
- withCustomProperties** set to true, to include the custom properties in the generated string

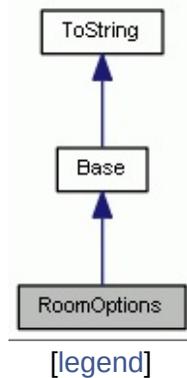
Photon C++ Client API 4.1.12.2

ExitGames > LoadBalancing > RoomOptions >

[Public Member Functions](#) | [List of all members](#)

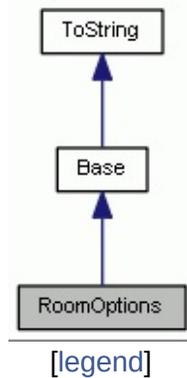
RoomOptions Class Reference

Inheritance diagram for RoomOptions:



[legend]

Collaboration diagram for RoomOptions:



[legend]

Public Member Functions

RoomOptions (bool isOpen=true, nByte lobbyName, const **Common::Hashtable** &customRoomProperties, const **Common::JV** &propsListedInLobby, **Common::JString** &lobbyName=**Common::JString**, lobbyType=**LobbyType**, playerTtl=0, int emptyTimeout, bool suppressRoomEvents, **Common::JVector** &plugins, bool directMode=**DirectMode**)

~RoomOptions (void)

RoomOptions (const **RoomOptions** &other)

RoomOptions & **operator=** (const **RoomOptions** &other)

bool **getIsVisible** (void) const

RoomOptions & **setIsVisible** (bool isVisible)

bool **getIsOpen** (void) const

RoomOptions & **setIsOpen** (bool isOpen)

nByte **getMaxPlayers** (void) const

RoomOptions & **setMaxPlayers** (nByte maxPlayers)

const **Common::Hashtable** & **getCustomRoomProperties** (void) const

	RoomOptions &	setCustomRoomPr Common::Hashtab
const Common::JVector < Common::JString > &		getPropsListedInLc
	RoomOptions &	setPropsListedInLc Common::JVector < &propsListedInLobby
const Common::JString &		getLobbyName (voi
	RoomOptions &	setLobbyName (co &lobbyName)
	nByte	getLobbyType (voic
	RoomOptions &	setLobbyType (nBy
	int	getPlayerTtl (void) c
	RoomOptions &	setPlayerTtl (int pla
	int	getEmptyRoomTtl (
	RoomOptions &	setEmptyRoomTtl (
	bool	getSuppressRooml
	RoomOptions &	setSuppressRooml suppressRoomEven
const Common::JVector < Common::JString > *		getPlugins (void) cc
	RoomOptions &	setPlugins (const C Common::JString >
	bool	getPublishUserID (

RoomOptions & **setPublishUserID** (l

nByte **getDirectMode** (void

RoomOptions & **setDirectMode** (nBy

virtual **Common::JString** & **toString** (**Common:**
withTypes=false) cor

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void)

JString **toString** (bool withT

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This class aggregates the various optional parameters that can be specified on room creation.

See also

[Client::opCreateRoom\(\)](#), [Client::opJoinOrCreateRoom\(\)](#)

Constructor & Destructor Documentation

§ RoomOptions() [1/2]

```
RoomOptions ( bool isVisi
              bool isOp
              nByte max
              const Common::Hashtable & cust
              const Common::JVector< Common::JString > & prop
              const Common::JString & lobb
              nByte lobb
              int play
              int emp
              bool sup
              const Common::JVector< Common::JString > * pPlu
              bool pub
              nByte dire
          )
```

Constructor: Creates a new instance with the specified parameters.

Parameters

isVisible	see setVisible() - optional, defaults to tr
isOpen	see setIsOpen() - optional, defaults to tru
maxPlayers	see setMaxPlayers() - optional, defaults :
customRoomProperties	see setCustomRoomProperties() - optic
propsListedInLobby	see setPropsListedInLobby() - optional,
lobbyName	see setLobbyName() - optional, defaults
lobbyType	see setLobbyType() - optional, defaults t
playerTtl	see setPlayerTtl() - optional, defaults to (
emptyRoomTtl	see setEmptyRoomTtl() - optional, defau
suppressRoomEvents	see setSuppressRoomEvents() - option
pPlugins	see setPlugins() - optional, defaults to N
publishUserID	see setPublishUserID() - optional, defau

directMode

see [setDirectMode\(\)](#) - optional, defaults 1

§ ~RoomOptions()

~RoomOptions (void)

Destructor.

§ RoomOptions() [2/2]

RoomOptions (const **RoomOptions** & **toCopy**)

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

```
RoomOptions & operator= ( const RoomOptions & toCopy )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ getIsVisible()

```
bool getIsVisible ( void ) const
```

Returns

the currently set value for the isVisible flag

See also

[setIsVisible\(\)](#)

§ setIsVisible()

RoomOptions & setIsVisible (bool **isVisible**)

Sets the initial state of the rooms visibility flag.

A room that is not visible is excluded from the room lists that are sent to the clients in lobbies. An invisible room can be joined by name but is excluded from random matchmaking.

Use this to "hide" a room and simulate "private rooms". Players can exchange a room name and create the room as invisible to avoid anyone else joining it.

Remarks

This function sets the initial value that is used for room creation. To change the value of the flag for an already existing room, see `MutableRoom::setIsVisible()`.

Parameters

isVisible the new value to which the flag will be set

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getIsVisible\(\)](#)

§ getIsOpen()

```
bool getIsOpen ( void ) const
```

Returns

the currently set value for the isOpen flag

See also

[setIsOpen\(\)](#)

§ setIsOpen()

RoomOptions & setIsOpen (bool **isOpen**)

Sets the initial state of the rooms `isOpen` flag.

If a room is closed, then no further player can join it until the room gets reopened again. A closed room can still be listed in the lobby (use **setIsVisible()** to control lobby-visibility).

Remarks

This function sets the initial value that is used for room creation. To change the value of the flag for an already existing room, see `MutableRoom::setIsOpen()`.

Parameters

isOpen the new value to which the flag will be set

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getIsOpen()

§ getMaxPlayers()

```
nByte getMaxPlayers ( void ) const
```

Returns

the currently set max players

See also

setMaxPlayers()

§ setMaxPlayers()

RoomOptions & setMaxPlayers (nByte **maxPlayers**)

Sets the initial value for the max players setting of the room.

This function sets the maximum number of players that can be inside the room at the same time, including inactive players. 0 means "no limit".

Remarks

This function sets the initial value that is used for room creation. To change the max players setting of an already existing room, see `MutableRoom::setMaxPlayers()`.

Parameters

maxPlayers the new maximum amount of players that can be inside the room at the same time

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getMaxPlayers\(\)](#)

§ getCustomRoomProperties()

```
const Hashtable & getCustomRoomProperties ( void ) const
```

Returns

the currently set custom room properties

See also

[setCustomRoomProperties\(\)](#)

§ setCustomRoomProperties()

RoomOptions &

setCustomRoomProperties (const **Common::Hashtable** & customRoom

Sets the initial custom properties of a room.

Custom room properties are any key-value pairs that you need to define setup. The shorter your key strings are, the better. Example: Map, Mode L"m" when used with L"Map"), TileSet (could be L"t").

Note

JString is the only supported type for custom property keys. For custom property values you can use any type that is listed in the [Table of Data Types](#).

Remarks

This function sets the initial custom properties that are used for room setup. To change the custom properties of an already existing room, see [MutableRoom::mergeCustomProperties\(\)](#), [MutableRoom::addCustomProperties\(\)](#), [MutableRoom::addCustomProperties\(\)](#), [MutableRoom::removeCustomProperty\(\)](#) and [MutableRoom::removeCustomProperties\(\)](#).

Parameters

customRoomProperties a Hashtable of custom property key-value pairs

Returns

a reference to the instance on which it was called to allow for chainin setter calls

See also

[getCustomRoomProperties\(\)](#), [setPropsListedInLobby\(\)](#)

§ getPropsListedInLobby()

```
const JVector< JString > & getPropsListedInLobby ( void ) const
```

Returns

the currently set list of properties to show in the lobby

See also

[setPropsListedInLobby\(\)](#)

§ setPropsListedInLobby()

RoomOptions &

setPropsListedInLobby (const **Common::JVector**< **Common::JString** >

Sets the initial list of custom properties of the room that should be shown

List the keys of all the custom room properties that should be available to lobby. Use with care. Unless a custom property is essential for matchmaking should not be sent to the lobby, which causes traffic and delays for client. No custom properties are sent to the lobby.

Note

Properties that are intended to be shown in the lobby should be as compact as possible. Literally every single byte counts here as this info needs to be sent to the lobby for every single visible room, so that with lots of users online a quickly adds up to a lot of data.

Remarks

This function sets the initial list of property keys. To change which properties are shown in the lobby for an already existing room see `MutableRoom::setPropsListedInLobby`

Parameters

propsListedInLobby the keys of the custom room properties that should be shown in the lobby

Returns

a reference to the instance on which it was called to allow for chaining

See also

[getPropsListedInLobby\(\)](#), [setCustomRoomProperties\(\)](#)

§ getLobbyName()

```
const JString & getLobbyName ( void ) const
```

Returns

the currently set lobby name

See also

[setLobbyName\(\)](#)

§ setLobbyName()

RoomOptions &
setLobbyName (const **Common::JString** & lobbyName)

Sets the name of the lobby to which the room gets added to.

Rooms can be assigned to different lobbies on room creation.

Client::opJoinRandomRoom() only uses those room for matchmaking that are assigned to the lobby in which it is told to be looking for rooms. A lobby can be joined by a call to **Client::opJoinLobby()** and inside lobbies of certain types clients can receive room lists that contain all visible rooms that are assigned to that lobby.

Remarks

If you don't set a lobby name or if you set it to an empty string, then any value that is passed for **setLobbyType()** gets ignored and the room gets added to the default lobby.

Lobbies are unique per lobbyName plus lobbyType, so multiple different lobbies may have the same name, as long as they are of a different type.

Parameters

lobbyName identifies for the lobby and needs to be unique within the scope of the lobbyType

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getLobbyName(), **setLobbyType()**, Matchmaking Guide

§ getLobbyType()

```
nByte getLobbyType ( void ) const
```

Returns

the currently set lobby type

See also

[setLobbyType\(\)](#)

§ setLobbyType()

RoomOptions & setLobbyType (nByte lobbyType)

Sets the type of the lobby to which the room gets added to. Must be one of the values in **LobbyType**

Please see [Matchmaking Guide](#) regarding the differences between the various lobby types.

Note

This option gets ignored and the room gets added to the default lobby, if you don't also set the lobby name to a non-empty string via a call to **setLobbyName()**.

Parameters

lobbyType one of the values in **LobbyType**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getLobbyType\(\)](#), [setLobbyName\(\)](#), **LobbyType**, [Matchmaking Guide](#)

§ getPlayerTtl()

```
int getPlayerTtl ( void ) const
```

Returns

the currently set player time to live in milliseconds

See also

[setPlayerTtl\(\)](#)

§ setPlayerTtl()

RoomOptions & setPlayerTtl (int **playerTtl**)

Sets the player time to live in milliseconds.

If a client disconnects or if it leaves a room with the 'willComeBack' flag set to true, its player becomes inactive first and only gets removed from the room after this timeout.

- -1 and INT_MAX set the inactivity time to 'infinite'.
- 0 (default) deactivates player inactivity.
- All other positive values set the inactivity time to their value in milliseconds.
- All other negative values get ignored and the behavior for them is as if the default value was used.

Note

A player is only able to rejoin a room in its existing player slot while it is still inactive. Once it has left for good it will be treated as a completely new player.

Parameters

playerTtl a value between -1 and INT_MAX

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getPlayerTtl\(\)](#), [Client::opLeaveRoom\(\)](#), [Client::disconnect\(\)](#)

§ getEmptyRoomTtl()

```
int getEmptyRoomTtl ( void ) const
```

Returns

the currently set empty room time to live in milliseconds

See also

[setEmptyRoomTtl\(\)](#)

§ setEmptyRoomTtl()

RoomOptions & setEmptyRoomTtl (int **emptyRoomTtl**)

Sets the room time to live in milliseconds.

The amount of time in milliseconds that Photon servers should wait before disposing an empty room. A room is considered empty when there is no active player joined to it. So the room disposal timer starts when the last active player leaves. When a player joins or rejoins the room, then the countdown is reset.

By default, the maximum value allowed is:

- 300000ms (5 minutes) on Photon Cloud
- 60000ms (1minute) on Photon Server
- 0 (default) means that an empty room gets instantly disposed.
- All positive values set the keep-alive time to their value in milliseconds.
- All negative values get ignored and the behavior for them is as if the default value was used.

Note

The disposal of a room means that the room gets removed from memory on the server side. Without accordingly configured Webhooks this also means that the room will be destroyed and all data related to it (like room and player properties, event caches, inactive players, etc.) gets deleted. When [Webhooks](#) for the used appID have been been setup for [room persistence](#), then disposed rooms get stored for later retrieval. Stored rooms get reconstructed in memory when a player joins or rejoins them

Parameters

emptyRoomTtl a value between 0 and INT_MAX

Returns

a reference to the instance on which it was called to allow for

chaining multiple setter calls

See also

[getEmptyRoomTtl\(\)](#), [Persistence Guide](#), [Webhooks FAQ](#)

§ `getSuppressRoomEvents()`

```
bool getSuppressRoomEvents ( void ) const
```

Returns

the currently set value for the `suppressRoomEvents` flag

See also

[`setSuppressRoomEvents\(\)`](#)

§ `setSuppressRoomEvents()`

RoomOptions &

`setSuppressRoomEvents`

(bool `suppressRoomEvents`)

Sets the value of the `suppressRoomEvents` flag which determines if the server should skip room events for joining and leaving players.

Setting this flag to true makes the client unaware of the other players in a room. That can save some traffic if you have some server logic that updates players, but it can also limit the client's usability.

Parameters

`suppressRoomEvents` the new value to which the flag will be set

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[`getSuppressRoomEvents\(\)`](#)

§ getPlugins()

```
const JVector< JString > * getPlugins ( void ) const
```

Returns

the currently set list of plugins

See also

[setPlugins\(\)](#)

§ setPlugins()

RoomOptions

& setPlugins (const **Common::JVector**< **Common::JString** > * pPlu

Informs the server of the expected plugin setup.

The operation will fail in case of a plugin mismatch returning `ErrorCode::PLUGIN_MISMATCH`. Setting an empty `JVector` means that the client expects no plugin to be setup. Note: for backwards compatibility `NULL` (the default value) omits any check.

Parameters

pPlugins the expected plugins

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getPlugins\(\)](#)

§ `getPublishUserID()`

```
bool getPublishUserID ( void ) const
```

Returns

the currently set value for the `publishUserID` flag

See also

[setPublishUserID\(\)](#)

§ setPublishUserID()

RoomOptions & setPublishUserID (bool **publishUserID**)

Defines if the UserIDs of players get "published" in the room. Useful for **Client::opFindFriends()**, if players want to play another game together.

When you set this to true, Photon will publish the UserIDs of the players in that room. In that case, you can use **Player::getUserID()**, to access any player's userID. This is useful for FindFriends and to set "expected users" to reserve slots in a room (see **Client::opCreateRoom()**, **Client::opJoinOrCreateRoom()** and **Client::opJoinRoom()**).

Parameters

publishUserID true, if userIDs should be published, false otherwise

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getPublishUserID()

§ getDirectMode()

```
nByte getDirectMode ( void ) const
```

Returns

the currently set value for the **DirectMode** flag

See also

[setDirectMode\(\)](#)

§ setDirectMode()

RoomOptions & setDirectMode (nByte **directMode**)

Sets the **DirectMode** that should be used for this room.

The value of this option determines if clients establish direct peer to peer connections with other clients that can then be used to send them direct peer to peer messages with **Client::sendDirect()**.

Parameters

directMode one of the values in **DirectMode**

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getDirectMode()

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                               withTypes = false  
                    )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

Photon C++ Client API 4.1.12.2

ExitGames

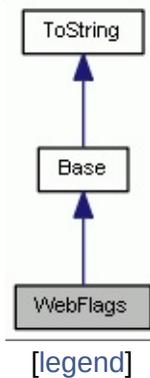
LoadBalancing

WebFlags

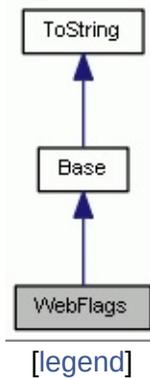
[Public Member Functions](#) | [List of all members](#)

WebFlags Class Reference

Inheritance diagram for WebFlags:



Collaboration diagram for WebFlags:



Public Member Functions

WebFlags (nByte webFlags=0)

nByte **getFlags** (void) const

WebFlags & **setFlags** (nByte webFlags)

bool **getHttpForward** (void) const

WebFlags & **setHttpForward** (bool httpWebForward)

bool **getSendAuthCookie** (void) const

WebFlags & **setSendAuthCookie** (bool
sendAuthCookie)

bool **getSendSync** (void) const

WebFlags & **setSendSync** (bool sendSync)

bool **getSendState** (void) const

WebFlags & **setSendState** (bool sendState)

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool
withTypes=false) const

▶ Public Member Functions inherited from **Base**

virtual **~Base** (void)

▶ Public Member Functions inherited from **ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString toString (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

Optional flags to be used with **RaiseEventOptions::setWebFlags()** and with various property setters of the **MutablePlayer** and the **MutableRoom** class to control the behavior of forwarded HTTP requests.

Please see [Webhooks v1.2](#) for further information.

See also

[Webhooks v1.2](#), [RaiseEventOptions::setWebFlags\(\)](#),
[MutablePlayer](#), [MutableRoom](#)

Constructor & Destructor Documentation

§ WebFlags()

WebFlags (nByte **webFlags** = 0)

Constructor: Creates a new instance with the specified parameters.

Parameters

webFlags see **setFlags()** - optional, defaults to 0.

Member Function Documentation

§ getFlags()

```
nByte getFlags ( void ) const
```

Returns

the currently set flags

See also

[setFlags\(\)](#)

§ setFlags()

WebFlags & setFlags (nByte **webFlags**)

Sets the values of all flags at once.

Internally all boolean flags are encoded as bits into a single byte variable. This byte can be retrieved with **getFlags()** and set with **setFlags()**.

Parameters

webFlags

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

getFlags()

§ getHttpForward()

```
bool getHttpForward ( void ) const
```

Returns

the currently set value of the httpWebForward flag

See also

[setHttpForward\(\)](#)

§ setHttpForward()

WebFlags & setHttpForward (bool **httpWebForward**)

Sets the value of the httpWebForward flag.

Parameters

httpWebForward true or false

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getHttpForward\(\)](#)

§ getSendAuthCookie()

```
bool getSendAuthCookie ( void ) const
```

Returns

the currently set value of the sendAuthCookie flag

See also

[setSendAuthCookie\(\)](#)

§ setSendAuthCookie()

WebFlags & setSendAuthCookie (bool **sendAuthCookie**)

Sets the value of the sendAuthCookie flag.

Parameters

sendAuthCookie true or false

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getSendAuthCookie\(\)](#)

§ `getSendSync()`

```
bool getSendSync ( void ) const
```

Returns

the currently set value of the `sendSync` flag

See also

[setSendSync\(\)](#)

§ setSendSync()

WebFlags & setSendSync (bool **sendSync**)

Sets the value of the sendSync flag.

Parameters

sendSync true or false

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getSendSync\(\)](#)

§ `getSendState()`

```
bool getSendState ( void ) const
```

Returns

the currently set value of the `sendState` flag

See also

[setSendState\(\)](#)

§ setSendState()

WebFlags & setSendState (bool **sendState**)

Sets the value of the sendState flag.

Parameters

sendState true or false

Returns

a reference to the instance on which it was called to allow for chaining multiple setter calls

See also

[getSendState\(\)](#)

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                    withTypes = false  
                    )                    const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

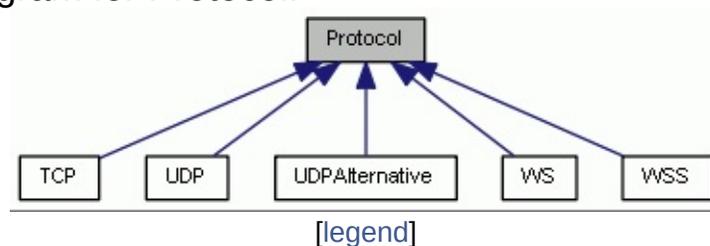
Photon C++ Client API 4.1.12.2

ExitGames > Photon > NetworkPort > Protocol

[Public Attributes](#) | [List of all members](#)

Protocol Struct Reference

Inheritance diagram for Protocol:



Collaboration diagram for Protocol:



Public Attributes

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

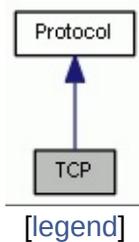
Client API 4.1.12.2

ExitGames > Photon > NetworkPort > TCP >

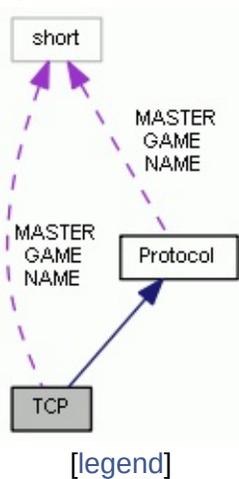
[Static Public Attributes](#) | [List of all members](#)

TCP Struct Reference

Inheritance diagram for TCP:



Collaboration diagram for TCP:



Static Public Attributes

static const unsigned short **MASTER**

static const unsigned short **GAME**

static const unsigned short **NAME**

Additional Inherited Members

▶ **Public Attributes inherited from Protocol**

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

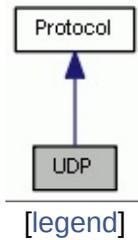
Client API 4.1.12.2

ExitGames > Photon > NetworkPort > UDP

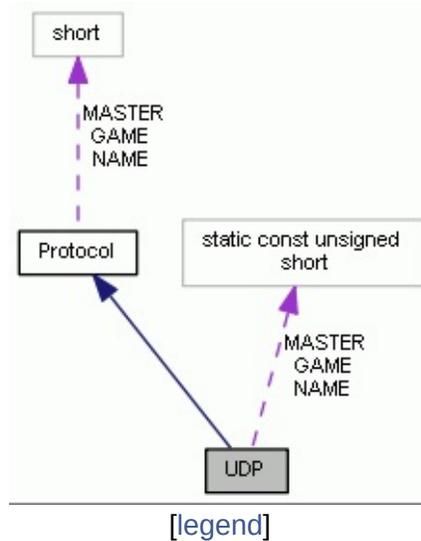
[Static Public Attributes](#) | [List of all members](#)

UDP Struct Reference

Inheritance diagram for UDP:



Collaboration diagram for UDP:



Static Public Attributes

static const unsigned short **MASTER**

static const unsigned short **GAME**

static const unsigned short **NAME**

Additional Inherited Members

▶ **Public Attributes inherited from Protocol**

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

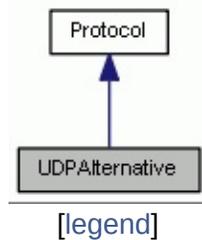
Client API 4.1.12.2

ExitGames > Photon > NetworkPort > UDPAlternative >

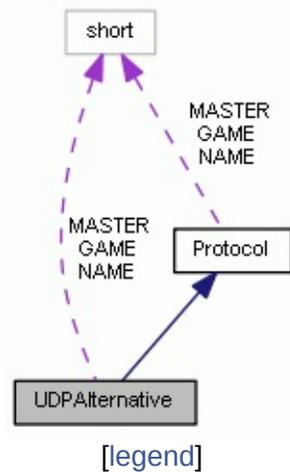
[Static Public Attributes](#) | [List of all members](#)

UDPAlternative Struct Reference

Inheritance diagram for UDPAlternative:



Collaboration diagram for UDPAlternative:



Static Public Attributes

static const unsigned short **NAME**

static const unsigned short **MASTER**

static const unsigned short **GAME**

Additional Inherited Members

▶ **Public Attributes inherited from Protocol**

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

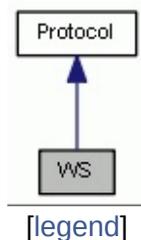
Client API 4.1.12.2

ExitGames > Photon > NetworkPort > WS

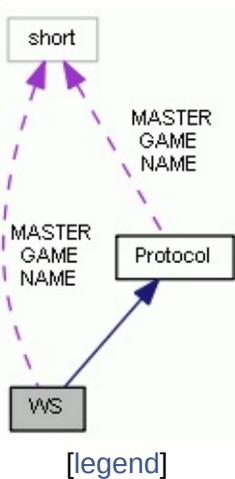
[Static Public Attributes](#) | [List of all members](#)

WS Struct Reference

Inheritance diagram for WS:



Collaboration diagram for WS:



Static Public Attributes

static const unsigned short **MASTER**

static const unsigned short **GAME**

static const unsigned short **NAME**

Additional Inherited Members

▶ **Public Attributes inherited from Protocol**

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

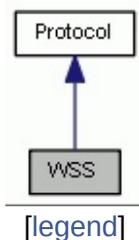
Client API 4.1.12.2

ExitGames > Photon > NetworkPort > WSS

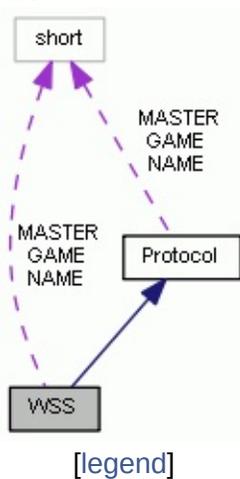
[Static Public Attributes](#) | [List of all members](#)

WSS Struct Reference

Inheritance diagram for WSS:



Collaboration diagram for WSS:



Static Public Attributes

static const unsigned short **MASTER**

static const unsigned short **GAME**

static const unsigned short **NAME**

Additional Inherited Members

▶ **Public Attributes inherited from Protocol**

const unsigned short **MASTER**

const unsigned short **GAME**

const unsigned short **NAME**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [Punchthrough](#) > [Puncher](#) >

[Public Member Functions](#) | [List of all members](#)

Puncher Class Reference

Public Member Functions

Puncher ([RelayClient](#) *pRelayClient, const [Common::Logger](#) &logger)

bool **init** ([PunchListener](#) *pPunchListener)

void **clear** (void)

bool **startPunch** (int remotelD)

bool **sendDirect** (const [Common::JVector](#)< nByte > &buffer, int targetID, bool fallbackRelay)

int **sendDirect** (const [Common::JVector](#)< nByte > &buffer, const [Common::JVector](#)< int > &targetIDs, bool fallbackRelay)

bool **processPackage** (const [Common::JVector](#)< nByte > &packet, bool relay, int relayRemotelD)

void **service** (void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

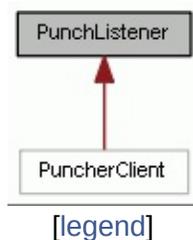
ExitGames > Photon > Punchthrough > PunchListener >

[Public Member Functions](#) | [List of all members](#)

PunchListener Class

Reference **abstract**

Inheritance diagram for PunchListener:



Public Member Functions

virtual void **onReceiveDirect** (const **Common::JVector**< nByte >
&inBuf, int remotelD, bool relay)=0

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

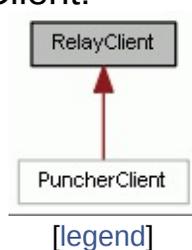
Photon C++ Client API 4.1.12.2

ExitGames > Photon > Punchthrough > RelayClient

[Public Member Functions](#) | [List of all members](#)

RelayClient Class Reference **abstract**

Inheritance diagram for RelayClient:



Public Member Functions

virtual int **getLocalID** (void)=0

virtual bool **sendRelay** (const **Common::JVector**< nByte > &buffer,
const **Common::JVector**< int > &targetIDs)=0

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [EventData](#) >

[Public Member Functions](#) | [List of all members](#)

EventData Class Reference

Public Member Functions

	~EventData
	EventData EventData
	EventData & operator= EventData
const Common::Object &	operator[] index) const
	Common::JString toString (b withParam withParam const
	Common::Object getParam (nByte para const
	nByte getCode (v
const Common::Dictionary< nByte, Common::Object > &	getParam const

Detailed Description

Contains all components of a **Photon** Event.

Constructor & Destructor Documentation

§ ~EventData()

~EventData (void)

Destructor.

§ `EventData()`

```
EventData ( const EventData & toCopy )
```

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

```
EventData & operator= ( const EventData & toCopy )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator[]()

```
const Object & operator[] ( unsigned int index ) const
```

operator[]. Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes.

§ toString()

```
JString toString ( bool withParameters = false,  
                  bool withParameterTypes = false  
                  )      const
```

Parameters

- withParameters** determines if the payload of the event should be included in the returned string
- withParameterTypes** determines if the type information should be included for the payload

Returns

a JString representation of the instance for debugging purposes.

§ getParameterForCode()

```
Object getParameterForCode ( nByte parameterCode ) const
```

Alternative access to the Parameters.

Parameters

parameterCode The key code of an event value

Returns

The parameters value, or an empty Object instance if the key does not exist in the parameters.

§ getCode()

```
nByte getCode ( void ) const
```

Returns

the event code that identifies the type of the event.

§ getParameters()

```
const Dictionary< nByte, Object > & getParameters ( void ) const
```

Returns

all parameters of the event.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [OperationRequest](#)

[Public Member Functions](#) | [List of all members](#)

OperationRequest **Class Reference**

Public Member Functions

	OperationRequest (nByte op: const OperationRequestParam ¶meters= OperationRequ
	~OperationRequest (void)
	OperationRequest (const Ope &toCopy)
OperationRequest &	operator= (const OperationRe
const Common::Object &	operator[] (unsigned int index)
Common::JString	toString (bool withParameters: withParameterTypes=false) cor
Common::Object	getParameterForCode (nByte const
nByte	getOperationCode (void) cons
const OperationRequestParameters &	getParameters (void) const
OperationRequestParameters &	getParameters (void)
void	setParameters (const OperationRequestParameters

Detailed Description

This is a container for an Operation request, which consists of a code and parameters.

Constructor & Destructor Documentation

§ `OperationRequest()` [1/2]

```
OperationRequest ( nByte operationCode,
                  const OperationRequestParameters & parameters
                  )
```

Constructor: Creates a new instance with the specified parameters.

Parameters

operationCode identifies the type of the operation.

parameters the payload of the operation.

§ ~OperationRequest()

~OperationRequest (void)

Destructor.

§ `OperationRequest()` [2/2]

`OperationRequest` (const `OperationRequest` & `toCopy`)

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

`toCopy` The instance to copy.

Member Function Documentation

§ operator=()

```
OperationRequest & operator= ( const OperationRequest & toCopy )
```

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator[]()

```
const Object & operator[] ( unsigned int index ) const
```

operator[]. Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes.

§ toString()

```
JString toString ( bool withParameters = false,  
                  bool withParameterTypes = false  
                  ) const
```

Parameters

- withParameters** determines if the payload of the event should be included in the returned string
- withParameterTypes** determines if the type information should be included for the payload

Returns

a JString representation of the instance for debugging purposes.

§ getParameterForCode()

```
Object getParameterForCode ( nByte parameterCode ) const
```

Alternative access to the Parameters.

Parameters

parameterCode The key code of an event value

Returns

The parameters value, or an empty Object instance if the key does not exist in the parameters.

§ getOperationCode()

```
nByte getOperationCode ( void ) const
```

Returns

the operation code that identifies the type of the operation request.

§ getParameters() [1/2]

```
const OperationRequestParameters & getParameters ( void ) const
```

Returns

a read only reference to all parameters of the operation request.

§ `getParameters()` [2/2]

`OperationRequestParameters` & `getParameters (void)`

Returns

a mutable reference to all parameters of the operation request.

§ setParameters()

```
void  
setParameters ( const OperationRequestParameters & parameters )
```

Parameters

parameters Sets the payload of the operation.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [OperationResponse](#)

[Public Member Functions](#) | [List of all members](#)

OperationResponse

Class Reference

Public Member Functions

	~Operation (void)
	OperationI (const OperationI &toCopy)
OperationResponse &	operator= OperationI &toCopy)
const Common::Object &	operator[] index) const
Common::JString	toString (b withDebugI bool withPa bool withParam const
Common::Object	getParam (nByte para const
	nByte getOperati const
	short getReturn const
const Common::JString &	getDebugM const

```
const Common::Dictionary< nByte, Common::Object > & getParam  
const
```

Detailed Description

Contains the servers response for an **OperationRequest** sent by this client.

Constructor & Destructor Documentation

§ ~OperationResponse()

`~OperationResponse (void)`

Destructor.

§ OperationResponse()

OperationResponse (const **OperationResponse** & **toCopy**)

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

Parameters

toCopy The instance to copy.

Member Function Documentation

§ operator=()

OperationResponse &
operator=

(const **OperationResponse** & toCopy)

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

§ operator[]()

```
const Object & operator[] ( unsigned int index ) const
```

operator[]. Accesses the value at the given index like in an array. This does not check for valid indexes and shows undefined behavior for invalid indexes.

§ toString()

```
JString toString ( bool withDebugMessage = false,  
                  bool withParameters = false,  
                  bool withParameterTypes = false  
                  )      const
```

Parameters

- withDebugMessage** determines if the debug message that the server may send in case of an error should be included in the returned string
- withParameters** determines if the payload of the event should be included in the returned string
- withParameterTypes** determines if the type information should be included for the payload

Returns

a JString representation of the instance for debugging purposes.

§ getParameterForCode()

```
Object getParameterForCode ( nByte parameterCode ) const
```

Alternative access to the Parameters.

Parameters

parameterCode The key code of an response value

Returns

The parameters value, or an empty Object instance if the key does not exist in the parameters.

§ getOperationCode()

```
nByte getOperationCode ( void ) const
```

Returns

the operation code that identifies the type of the operation.

§ getReturnCode()

```
short getReturnCode ( void ) const
```

Returns

the result code of the operation, 0 in case of success, an operation specific error code otherwise.

§ getDebugMessage()

```
const JString & getDebugMessage ( void ) const
```

Returns

extended debugging information in case that **getReturnCode()** returns !0, an empty string otherwise.

§ getParameters()

```
const Dictionary< nByte, Object > & getParameters ( void ) const
```

Returns

all parameters of the operation response.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

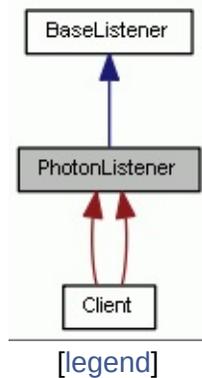
Client API 4.1.12.2

ExitGames > Photon > PhotonListener

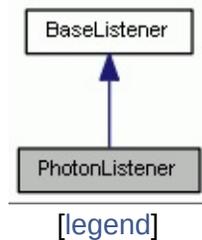
[Public Member Functions](#) | [List of all members](#)

PhotonListener Class Reference abstract

Inheritance diagram for PhotonListener:



Collaboration diagram for PhotonListener:



Public Member Functions

virtual **~PhotonListener** (void)

virtual void **onOperationResponse** (const **OperationResponse** &operationResponse)=0

virtual void **onStatusChanged** (int statusCode)=0

virtual void **onEvent** (const **EventData** &eventData)=0

virtual void **onPingResponse** (const **Common::JString** &address, unsigned int pingResult)

▶ **Public Member Functions inherited from BaseListener**

virtual void **debugReturn** (int debugLevel, const **JString** &string)=0

Constructor & Destructor Documentation

§ ~PhotonListener()

```
virtual ~PhotonListener ( void )
```

virtual

Destructor.

Member Function Documentation

§ onOperationResponse()

```
virtual void  
onOperationResponse ( const OperationResponse & operationRespon
```

This function gets called by the library as callback to operations in response to operations sent to the **Photon** Server providing the response values from the server.

This callback is used as general callback for all operations. The type of operation is identified by an operation code.

An operation's response is summarized by the return code: an int type (0 for OK or some error code defined by the application, which is defining the code itself). The opCode defines the type of operation called on **Photon** and its return values. They are provided as a Hashtable which contains the components of **Photon**, including keys for operation code and return code. Each operation has its opCode and returnCode but anything else can be defined serverside.

Parameters

operationResponse the **OperationResponse**

§ onStatusChanged()

```
virtual void onStatusChanged ( int statusCode )
```

pure virtual

onStatusChanged is used to denote errors or simply state-changes of the respective **PhotonPeer**.

State change callback

When this function is used to signalize a state-change, the **statusCode** will be one of these: **StatusCode::CONNECT** the connection to the **Photon** Server was established **StatusCode::DISCONNECT** the connection was closed (due to an API-call or a timeout)

Furthermore this function will be called by **Photon** to inform about connection errors and warnings. Check **StatusCode.h** for a list.

Parameters

statusCode see description

§ onEvent()

```
virtual void onEvent ( const EventData & eventData )
```

```
pure virtual
```

This is the event handler function for all Events transmitted by **PhotonPeer**.

Whenever a **Photon** event is sent and received, the receiving peer will be notified via this function. Please refer to **Sending and receiving data** for more information.

This way, an application can react on any event, based on its event code.

The following events are reported by default: EV_RT_JOIN
EV_RT_LEAVE

These events are predefined and will be triggered as soon as a player has joined or has left the room in which the local player is currently active in. To transmit in-room data, define your own events as needed for your application, and transmit them using `LitePeer::opRaiseEvent()`.

All events which are raised in reaction to some player's actions (like sending data) contain the actor number of the sending player in the "parameters" Hashtable.

If the received event has been raised by another player by calling `LitePeer::opRaiseEvent()`, the transmitted payload hashtable will be stored in the "parameters" hashtable of at key `EV_RT_KEY_DATA`. Please refer to the demos for sample code.

Parameters

eventData the **EventData**

See also

Sending and receiving data, `LitePeer::opRaiseEvent()`

§ onPingResponse()

```
virtual void  
onPingResponse      ( const Common::JString & address,  
                    unsigned int           pingResult  
                    )
```

virtual

This is the callback for **PhotonPeer::pingServer()**.

Each ping signal that has been sent through **PhotonPeer::pingServer()** results in a call to this function, providing the address to which the ping has been sent and the time in milliseconds that has passed between sending the ping and receiving the servers response.

Note

: This function is not available on platforms that do not support those parts of the stdlib that have been introduced with C++ 11.

: Also this function is not available on platforms that do not support multithreading.

Parameters

address the address, which has been pinged
pingResult the time in ms

See also

PhotonPeer::pingServer()

Photon C++

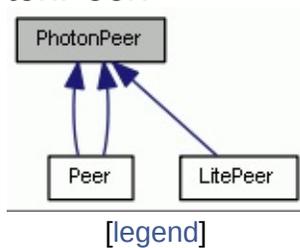
Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [PhotonPeer](#)

PhotonPeer Class Reference

[Public Member Functions](#) |
[Static Public Member Functions](#) |
[List of all members](#)

Inheritance diagram for PhotonPeer:



Public Member Functions

		PhotonPeer (PhotonListener connectionProtocol=Connector
	virtual	~PhotonPeer (void)
	virtual bool	connect (const Common::JString Common::JString &appId=Co
template<typename Ftype >	bool	connect (const Common::JString Common::JString &appId, coi &customData)
template<typename Ftype >	bool	connect (const Common::JString Common::JString &appId, coi pCustomdataArray, typename Common::Helpers::ArrayLength arrSize)
template<typename Ftype >	bool	connect (const Common::JString Common::JString &appId, coi pCustomdataArray, const short
	virtual void	disconnect (void)
	virtual void	service (bool dispatchIncomir
	virtual void	serviceBasic (void)
	virtual bool	opCustom (const OperationR &operationRequest, bool sendF channelID=0, bool encrypt=fals

virtual bool **sendOutgoingCommands** (void)

virtual bool **sendAcksOnly** (void)

virtual bool **dispatchIncomingCommands**

virtual bool **establishEncryption** (void)

virtual void **fetchServerTimestamp** (void)

virtual void **resetTrafficStats** (void)

virtual void **resetTrafficStatsMaximumCo**

virtual **Common::JString** **vitalStatsToString** (bool all) co

virtual void **pingServer** (const **Common::J**
unsigned int pingAttempts)

virtual void **initUserDataEncryption** (cons
Common::JVector< nByte > &

virtual void **initUDPEncryption** (const **Cor**
nByte > &encryptSecret, const
nByte > &HMACSecret)

PhotonListener * **getListener** (void)

int **getServerTimeOffset** (void) co

int **getServerTime** (void) const

int **getBytesOut** (void) const

int **getBytesIn** (void) const

int **getByteCountCurrentDispatc**

	int	getByteCountLastOperation	(void) const
	int	getPeerState	(void) const
	int	getSentCountAllowance	(void) const
	void	setSentCountAllowance	(int s)
	int	getTimePingInterval	(void) const
	void	setTimePingInterval	(int timeP)
	int	getRoundTripTime	(void) const
	int	getRoundTripTimeVariance	(void) const
	int	getTimestampOfLastSocketR	(void) const
	int	getDebugOutputLevel	(void) const
	bool	setDebugOutputLevel	(int deb)
const Common::LogFormatOptions &		getLogFormatOptions	(void) const
	void	setLogFormatOptions	(const Common::LogFormatOptions)
	int	getIncomingReliableCommar	(void) const
	short	getPeerID	(void) const
	int	getDisconnectTimeout	(void) const
	void	setDisconnectTimeout	(int dis)

	int	getQueuedIncomingComman
	int	getQueuedOutgoingComman
	Common::JString	getServerAddress (void) const
	bool	getIsPayloadEncryptionAvail:
	bool	getIsEncryptionAvailable (voi
	int	getResentReliableCommands
	int	getLimitOfUnreliableCommar
	void	setLimitOfUnreliableCommar
	bool	getCRCEnabled (void) const
	void	setCRCEnabled (bool crcEnab
	int	getPacketLossByCRC (void) c
	bool	getTrafficStatsEnabled (void)
	void	setTrafficStatsEnabled (bool t
	int	getTrafficStatsElapsedMs (vo
	const TrafficStats &	getTrafficStatsIncoming (void
	const TrafficStats &	getTrafficStatsOutgoing (void
	const TrafficStatsGameLevel &	getTrafficStatsGameLevel (vo
	nByte	getQuickResendAttempts (vo
	void	setQuickResendAttempts (nB

quickResendAttempts)

nByte **getConnectionProtocol** (void)

void **setConnectionProtocol** (nByte
connectionProtocol)

nByte **getChannelCountUserChann**

Static Public Member Functions

static short **getPeerCount** (void)

static unsigned int **getMaxAppIDLength** (void)

Detailed Description

The **PhotonPeer** class provides an API for reliable and unreliable realtime communication.

PhotonPeer uses the callback interface **PhotonListener** that needs to be implemented by your application, to receive results and events from the Photon Server.

Constructor & Destructor Documentation

§ PhotonPeer()

```
PhotonPeer ( PhotonListener & listener,  
            nByte                connectionProtocol = ConnectionProto  
            )
```

Constructor.

Parameters

listener

Reference to the application's implementation Listener callback interface. Has to be valid for lifetime of the **PhotonPeer** instance, which is this constructor.

connectionProtocol The protocol to use to connect to Photon. Must be one of the constants specified in **ConnectionProt**

See also

PhotonListener, **ConnectionProtocol**

§ ~PhotonPeer()

~PhotonPeer (void)

virtual

Destructor.

Member Function Documentation

§ connect() [1/4]

```
bool  
connect ( const Common::JString & ipAddr,  
          const Common::JString & appId = Common::JString()  
          )
```

virtual

This function starts establishing a connection to a Photon server. The servers response will arrive in **PhotonListener::onStatusChanged()**.

The connection is successfully established when the Photon client received a valid response from the server. The connect-attempt fails when a network error occurs or when server is not responding. A call to this function starts an asynchronous operation. The result of this operation gets returned through the **PhotonListener::onStatusChanged()** callback function.

Parameters

ipAddr A null terminated string containing the IP address or domain name and optionally the port number to connect to. IP addresses can be in IPv4 or IPv6 format, examples: "192.168.0.1", "192.168.0.1:5055", "udp.gameserver.com", "udp.gameserver.com:5055", "[2002:C0A8:1::]", "[2002:C0A8:1::]:5055". Note that IPv6 addresses must include square brackets to indicate where the address itself end and the port begins. If no port is given, then the default port for the chosen protocol and server type will be used.

appId the appId (default: an empty string)

Returns

true, if it could successfully start establishing a connection (the result will be passed in the callback function in this case) or false, if an error occurred and the connection could not be established (the callback function will not be called then).

See also

disconnect(), NetworkPort

§ connect() [2/4]

```
bool connect ( const Common::JString & ipAddr,  
              const Common::JString & appID,  
              const Ftype &          customData  
            )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- ipAddr** Null terminated string containing IP address or domain name and optionally a port of server to connect. Should be in usual format\
"address[:port]", for example\
"192.168.0.1:5055" or "udp.gameserver.com". If no port is given, port 5055 will be used by default.
- appID** the appID (default\
: an empty string)
- customData** custom data to send to the server when initializing the connection - has to be provided in the form of one of the supported data types, specified at [Table of Datatypes](#)

§ connect() [3/4]

```
bool  
connect ( const Common::JString &  
          const Common::JString &  
          const Ftype  
          typename Common::Helpers::ArrayLengthType< Ftype >::type  
        )
```

This is an overloaded member function, provided for convenience. It differs from other function only in what argument(s) it accepts.

This overload accepts singledimensional arrays and NULL-pointers pass pCustomDataArray. NULL pointers are only legal input, if arrSize is 0

Parameters

ipAddr	Null terminated string containing IP address or optionally a port of server to connect. Should be in the form of "address[:port]", for example: "192.168.0.1:5" or "udp.gameserver.com". If no port is given, port is assumed to be the default.
appId	the appId (default: an empty string)
pCustomDataArray	custom data to send to the server when initialized - has to be provided in the form of a 1D array of supported data types, specified at Table of Data Types
arrSize	the element count of the customData array

§ connect() [4/4]

```
bool connect ( const Common::JString & ipAddr,  
              const Common::JString & appId,  
              const Ftype                pCustomDataArray,  
              const short *              pArrSizes  
              )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This overload accepts multidimensional arrays and NULL-pointers passed for parameter pCustomDataArray. The array that is passed for parameter pCustomDataArray has to be a pointer of the correct abstraction level, meaning a normal pointer for a singledimensional array, a doublepointer for a twodimensional array, a triplepointer for a threedimensional array and so on. For pCustomDataArray NULL pointers are only legal input, if pArrSizes[0] is 0. For pArrSizes NULL is no valid input.

Parameters

ipAddr	Null terminated string containing IP address or domain name and optionally a port of server to connect. Should be in usual format: "address[:port]", for example: "192.168.0.1:5055" or "udp.gameserver.com". If no port is given, port 5055 will be used by default.
appId	the appId (default: an empty string)
pCustomDataArray	custom data to send to the server when initializing the connection - has to be provided in the form of an array of one of the supported data types, specified at Table of Datatypes
pArrSizes	the element counts for every dimension of the custom data array - the element count of

this array has to match the dimensions of
the custom data array

§ disconnect()

```
void disconnect ( void )
```

virtual

Initiates the disconnection from the Photon server. The servers response will arrive in **PhotonListener::onStatusChanged()**.

This function generates a disconnection request that will be sent to the Photon server. If the disconnection is completed successfully, then the **PhotonListener::onStatusChanged()** callback will be called, with a statusCode of **StatusCode::DISCONNECT**.

See also
[connect\(\)](#)

Reimplemented in **Peer**.

§ service()

```
void service ( bool dispatchIncomingCommands = true )
```

virtual

This function executes the **PhotonPeer** internal processes. Call this regularly!

This function is meant to be called frequently, like once per game loop. It handles the internal calls for keeping the **PhotonPeer** communication alive, and will take care of sending all local outgoing acknowledgements and messages, as well as dispatching incoming messages to the application and firing the corresponding callbacks. Internally **service()** calls the following functions:

1. **serviceBasic()**
2. **dispatchIncomingCommands()** (called withing a loop until all incoming commands have been dispatched.)
3. **sendOutgoingCommands()** (called withing a loop until everything queued for sending has been sent.)

service() is provided for convenience. If you need to tweak the performance, you can ignore **service()** and call its three subfunctions directly with individual time intervals, to gain more control over the internal communication process. For instance, calling **sendOutgoingCommands()** more rarely will result in less packets to be generated, as more commands will be accumulated into a single packet. See **sendOutgoingCommands()** for more information on efficiency.

For situations where you want to keep the connection alive, but can't process incoming messages (e.g. when loading a level), you can temporarily pass false for **dispatchIncomingCommands** to skip the calls to **dispatchIncomingCommands()**. Incoming commands will be stored in the incoming queue until they are dispatched again.

Parameters

dispatchIncomingCommands true =
dispatchIncomingCommands()

will be called; false =
dispatchIncomingCommands()
won't be called, default is true

§ serviceBasic()

```
void serviceBasic ( void )
```

virtual

This function takes care of exchanging data with the system's network layer.

You only need to call this function in case you choose not to use [service\(\)](#), but call the subfunctions of [service\(\)](#) directly. Please see the documentation of [service\(\)](#) for more information.

[serviceBasic\(\)](#) is called from within [service\(\)](#). If you decide not to use [service\(\)](#), then [serviceBasic\(\)](#) needs to be called frequently, like once per game loop.

See also
[service\(\)](#)

§ opCustom()

```
bool opCustom ( const OperationRequest & operationRequest,  
               bool sendReliable,  
               nByte channelId = 0,  
               bool encrypt = false  
               )
```

virtual

Sends a custom operation to a custom Server, using reliable or unreliable Photon transmission.

Allows the client to send a custom operation to the Photon server (which has to be modified accordingly). The Server can be extended and modified for special purposes like server side collision detection or a consistent world.

You need to be connected (see [connect\(\)](#)) prior to calling [opCustom\(\)](#).

Parameters

operationRequest holds the payload of the operation

sendReliable = operation will be sent reliably; false = no resend in case of packet loss - will be ignored, when not using udp as protocol

channelID the logical channel, default is 0. See [Fragmentation and Channels](#) for more information.

encrypt true = encrypt message; false = no encryption

Returns

true, if successful, false otherwise

§ sendOutgoingCommands()

```
bool sendOutgoingCommands ( void )
```

virtual

This function initiates the transmission of outgoing commands.

Any Photon function that generates messages will store these messages as a "command" in an outgoing queue for later transmission. Commands can either be explicitly created operations generated for example by **opCustom()** or internally generated messages like acknowledgements for reliable messages from other players. **sendOutgoingCommands()** will initiate the data transmission by passing the outgoing commands to the system's sockets for immediate transmission.

In case of UDP **sendOutgoingCommands()** will also split the commands into multiple packets if needed and/of aggregate multiple commands together into one packet, if possible. Because of the latter calling **sendOutgoingCommands()** more rarely will result in less overhead, as there will be fewer packets for the clients to be sent and processed. The underlying platform can also limit the frequency in which outgoing packets can be sent and received. The downside of lower sending frequencies is a higher latency, until messages are exchanged and acknowledged, which may lead to a jerky gameplay.

To help you keeping track of the incoming and outgoing queues at development time and adjust your sending frequency, there will be a warning message sent to your **debugReturn** callback if a queue has exceeded the warning threshold.

Note

While **service()** is calling **serviceBasic()** implicitly, you will have to regularly call it yourself explicitly, when you use **sendOutgoingCommands()** and **dispatchIncomingCommands()** directly instead.

Usually you don't have to call **sendOutgoingCommands()** this explicitly, as this is done within **service()**.

See also
[service\(\)](#)

§ sendAcksOnly()

```
bool sendAcksOnly ( void )
```

virtual

Sends only ACKs (UDP) or Ping (TCP) instead of queued outgoing commands. Useful to pause sending actual data.

Note

While `service()` is calling `serviceBasic()` implicitly, you will have to regularly call it yourself explicitly, when you use `sendAcksOnly()` and `dispatchIncomingCommands()` instead.

§ dispatchIncomingCommands()

```
bool dispatchIncomingCommands ( void )
```

virtual

Checks for incoming commands waiting in the queue, and dispatches a single command to the application.

Dispatching means, that if the command is an operation response or an event, the appropriate callback function will be called).

dispatchIncomingCommands() will also take care of generating and queuing acknowledgments for incoming reliable commands. Please note that this function will only dispatch one command per all. If you want to dispatch every single command which is waiting in the queue, call `dispatchIncomingCommands()` within a while loop, until its return code is false.

Note

While **service()** is calling **serviceBasic()** implicitly, you will have to regularly call it yourself explicitly, when you use **sendOutgoingCommands()** and **dispatchIncomingCommands()** directly instead.

Returns

true if it has successfully dispatched a command, false otherwise (for example, when there has not been any command left in the queue, waiting for dispatching).

See also

service()

§ establishEncryption()

```
bool establishEncryption ( void )
```

virtual

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

If **establishEncryption()** returns true, then Photon will inform you about the successful establishment or a failure by calling **PhotonListener::onStatusChanged()** with the statusCode being either **StatusCode::ENCRYPTION_ESTABLISHED** or **StatusCode::ENCRYPTION_FAILED_TO_ESTABLISH**

Returns

true if encryption has been successfully initiated, false otherwise.

See also

getIsEncryptionAvailable(),
getIsPayloadEncryptionAvailable(), **initUDPEncryption()**,
initUserDataEncryption()

§ fetchServerTimestamp()

```
void fetchServerTimestamp ( void )
```

virtual

This will fetch the server's timestamp and update the approximation for **getServerTime()** and **getServerTimeOffset()**.

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 roundtriptime is low and the timestamp + roundtriptime / 2 is close to the original value.

§ resetTrafficStats()

```
void resetTrafficStats ( void )
```

virtual

Creates new instances of **TrafficStats** and starts a new timer for those.

§ resetTrafficStatsMaximumCounters()

```
void resetTrafficStatsMaximumCounters ( void )
```

virtual

Resets traffic stats values that can be maxed out.

§ vitalStatsToString()

```
JString vitalStatsToString ( bool all ) const
```

virtual

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.

Parameters

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

Returns

stats as a string.

§ pingServer()

```
void pingServer ( const Common::JString & address,  
                 unsigned int           pingAttempts  
                 )
```

virtual

Sends a ping signal to the specified address.

Each call to this function results in a number of calls to **PhotonListener::onPingResponse()** that equals the value which has been passed for parameter pingAttempts.

This function can be used to ping multiple **Photon** servers and determine the one with the lowest latency.

As the latency of the same server may vary it can make sense to send multiple ping attempts. In that case the next attempt gets sent when either the servers response for the previous attempt has been received or when that previous attempt has timed out.

Multiple calls to this function do not get queued, but run in parallel.

A valid **Photon** server must run at the specified address.

Note

This function is not available on platforms that do not support those parts of the stdlib that have been introduced with C++ 11.

This function is not available on platforms that do not support multi-threading.

Parameters

address the address, which should be pinged
pingAttempts the amount of ping signals to send

See also

PhotonListener::onPingResponse()

§ `initUserDataEncryption()`

```
void  
initUserDataEncryption ( const Common::JVector< nByte > & secret )
```

Initializes `userData` encryption with the provided key.

Note

You must also provide the same key to the server to which you want connect. It needs to be an aes256 key and must not have been received through an unsecured connection.

Remarks

If you don't already have generated a key that you can access securely on both, the client and the server, you may want to consider to use **`establishEncryption()`** instead, which also initializes `userData` encryption, but does generate suitable keys on client and server side itself.

Parameters

`secret` an aes256 key

See also

`getIsEncryptionAvailable()`, **`getIsPayloadEncryptionAvailable()`**, **`establishEncryption()`**, **`initUDPEncryption()`**

§ initUDPEncryption()

```
void  
initUDPEncryption ( const Common::JVector< nByte > & encryptSecret  
                   const Common::JVector< nByte > & HMACSecret  
                   )
```

Initializes UDP packet Data encryption with the provided keys.

This function has no effect for non-UDP connections, but you may still call it while having an active connection that uses a different protocol. In that case, the keys will be stored in case that you switch the protocol at the time of a later re-connect. For XB1 UDP connections UDP packet encryption is a mandatory requirement by Microsoft. On other platforms you may also consider to use [establishEncryption\(\)](#) or [initUserDataEncryption\(\)](#), which provide alternative encryption implementations that do also work with other connection protocols.

Note

You must also provide the same keys to the server to which you want to connect. They need to be aes256 keys and must not have been received through an unsecured connection.

This function is only available on Windows Desktop, Windows Store, and Xbox 1.

Parameters

encryptSecret an aes256 key used for packet encryption

HMACSecret an aes256 key used for packet authentication

See also

[getIsEncryptionAvailable\(\)](#), [establishEncryption\(\)](#),
[initUserDataEncryption\(\)](#)

§ getListener()

```
PhotonListener * getListener ( void )
```

Returns

a pointer to the application's implementation of the Listener callback interface, as passed to the constructor of **PhotonPeer**.

§ getServerTimeOffset()

```
int getServerTimeOffset ( void ) const
```

Returns

the difference between the local uptime and the Photon Server's system time in ms.

In real-time games it's often useful to relate game events to a global common timeline, that's valid for all players and independent from derivations throughout the clients' system times. The Photon Server's System Time can serve as this reference time. The `serverTimeOffset` represents the difference between the client's local system time and the Photon server's system time.

`ServerTime = serverTimeOffset + GETTIMEMS()`

The `serverTimeOffset` is fetched shortly after connect by **Photon**. Use `GETTIMEMS()` to get your local time in ms. You can let Photon refetch the offset by calling **`fetchServerTimestamp()`**. The `ServerTimeOffset` will be 0 until shortly after initial connect.

§ `getServerTime()`

```
int getServerTime ( void ) const
```

Returns

the Photon Server's system time in ms.

see [getServerTimeOffset\(\)](#)

§ getBytesOut()

```
int getBytesOut ( void ) const
```

Returns

the total number of outgoing bytes transmitted by this **PhotonPeer** object.

See also

[getBytesIn\(\)](#)

§ `getBytesIn()`

```
int getBytesIn ( void ) const
```

Returns

the total number of incoming bytes received by this **PhotonPeer** object.

See also

[getBytesOut\(\)](#)

§ `getByteCountCurrentDispatch()`

```
int getByteCountCurrentDispatch ( void ) const
```

Returns

the size of the dispatched event or operation-result in bytes. This value is set before `onEvent()` or `onOperationResponse()` is called (within **`dispatchIncomingCommands()`**). Get this value directly in `onEvent()` or `onOperationResponse()`.

§ getByteCountLastOperation()

```
int getByteCountLastOperation ( void ) const
```

Returns

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. Get this value immediately after calling an operation.

§ getPeerState()

```
int getPeerState ( void ) const
```

Returns

the current state of the **PhotonPeer** object

The state of the **PhotonPeer** object is changed internally upon connection and disconnection, and will be one of the values of the **PeerState** enum.

See also

connect(), **disconnect()**

§ `getSentCountAllowance()`

```
int getSentCountAllowance ( void ) const
```

Returns

the number of resend retries before a peer is considered lost/disconnected.

This is udp specific and will always return 0 for other protocols.

See also

[setSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ setSentCountAllowance()

```
void setSentCountAllowance ( int sentCountAllowance )
```

Sets the number of re-send retries before a peer is considered lost/disconnected.

This is udp specific and will do nothing at all for other protocols.

Parameters

sentCountAllowance the new number of re/-send retries before a peer is considered lost/disconnected.

See also

[getSentCountAllowance\(\)](#) [getDisconnectTimeout\(\)](#)
[setDisconnectTimeout\(\)](#)

§ getTimePingInterval()

```
int getTimePingInterval ( void ) const
```

Returns

the time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[setTimePingInterval\(\)](#)

§ setTimePingInterval()

```
void setTimePingInterval ( int timePingInterval )
```

Sets the time threshold in milliseconds since the last reliable command, before a ping will be sent.

Parameters

timePingInterval time threshold in milliseconds since the last reliable command, before a ping will be sent.

See also

[getTimePingInterval\(\)](#)

§ getRoundTripTime()

```
int getRoundTripTime ( void ) const
```

Returns

the time in milliseconds until a reliable command is acknowledged by the server.

This is, what is commonly called a ping time or just a ping.

See also

[getRoundTripTimeVariance\(\)](#)

§ `getRoundTripTimeVariance()`

```
int getRoundTripTimeVariance ( void ) const
```

Returns

the variance of the roundtrip time in milliseconds. Gives a hint about how much the net latency is varying.

See also

[`getRoundTripTime\(\)`](#)

§ getTimestampOfLastSocketReceive()

```
int getTimestampOfLastSocketReceive ( void ) const
```

Returns

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.

§ `getDebugOutputLevel()`

```
int getDebugOutputLevel ( void ) const
```

Returns the current level of debug information that's passed on to [BaseListener::debugReturn\(\)](#).

Returns

one of the values in `DebugLevel`

See also

[setDebugOutputLevel\(\)](#)

§ `setDebugOutputLevel()`

```
bool setDebugOutputLevel ( int debugLevel )
```

Sets the current level of debug information that's passed on to `BaseListener::debugReturn()`.

Parameters

debugLevel one of the values in `DebugLevel`

Returns

true if the new debug level has been set correctly, false otherwise.

See also

`getDebugOutputLevel()`

§ getLogFormatOptions()

```
const LogFormatOptions & getLogFormatOptions ( void ) const
```

Returns

the LogFormatOptions that are used by this instance.

See also

setFormatOptions()

§ setLogFormatOptions()

```
void  
setLogFormatOptions ( const Common::LogFormatOptions & formatO
```

Sets the log format options to the supplied value.

Parameters

formatOptions the new value to which the log format options will be

See also

getFormatOptions()

§ getIncomingReliableCommandsCount()

```
int getIncomingReliableCommandsCount ( void ) const
```

Returns

the total number of reliable commands currently waiting in the incoming queues of all channels or -1 if not connected.

§ getPeerID()

```
short getPeerID ( void ) const
```

Returns

this peer's ID as assigned by the server. Will be -1, if not connected.

§ getDisconnectTimeout()

```
int getDisconnectTimeout ( void ) const
```

Returns

the maximum time interval in milliseconds for doing resend retries before a peer is considered lost/disconnected.

See also

[setDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ setDisconnectTimeout()

```
void setDisconnectTimeout ( int disconnectTimeout )
```

Sets the maximum time in milliseconds for making re-send retries before a peer is considered lost/disconnected.

Parameters

disconnectTimeout resend max time in ms before a peer is considered lost/disconnected

See also

[getDisconnectTimeout\(\)](#) [getSentCountAllowance\(\)](#)
[setSentCountAllowance\(\)](#)

§ getQueuedIncomingCommands()

```
int getQueuedIncomingCommands ( void ) const
```

Returns

the number of queued incoming commands in all channels or -1 if not connected

§ getQueuedOutgoingCommands()

```
int getQueuedOutgoingCommands ( void ) const
```

Returns

the number of queued outgoing commands in all channels or -1 if not connected

§ getAddress()

```
JString getAddress ( void ) const
```

Returns

the IP or url of the server, to which the peer is connected to

§ `getIsPayloadEncryptionAvailable()`

```
bool getIsPayloadEncryptionAvailable ( void ) const
```

Returns

this peer's payload encryption availability status. True if payload encryption is available, false otherwise.

See also

[getIsEncryptionAvailable\(\)](#), [establishEncryption\(\)](#),
[initUserDataEncryption\(\)](#)

§ getIsEncryptionAvailable()

```
bool getIsEncryptionAvailable ( void ) const
```

Returns

this peer's encryption availability status. True if either payload encryption is available or if the connection protocol is UDP and UDP encryption is available or if the connection protocol is already secure on its own, false otherwise.

See also

[getIsPayloadEncryptionAvailable\(\)](#), [establishEncryption\(\)](#), [initUserDataEncryption\(\)](#), [initUDPEncryption\(\)](#)

§ getResentReliableCommands()

```
int getResentReliableCommands ( void ) const
```

Returns

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

§ getLimitOfUnreliableCommands()

```
int getLimitOfUnreliableCommands ( void ) const
```

Returns

the limit for the queue of received unreliable commands.

See also

[setLimitOfUnreliableCommands\(\)](#)

§ setLimitOfUnreliableCommands()

```
void setLimitOfUnreliableCommands ( int value )
```

Sets the limit for the queue of received unreliable commands. This works only in UDP. This limit is applied when you call `dispatchIncomingCommands`. If this client (already) received more than this limit, 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. 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 commands up to specified limit. This value interacts with `dispatchIncomingCommands()`: If that is called less often, more commands get skipped.

See also

[getLimitOfUnreliableCommands\(\)](#)

§ getCRCEnabled()

```
bool getCRCEnabled ( void ) const
```

Returns

true if CRC enabled

See also

[setCRCEnabled](#)

§ setCRCEnabled()

```
void setCRCEnabled ( bool crcEnabled )
```

Enables or disables CRC. While not connected, this controls if the next connection(s) should use a per-package CRC checksum. If the client is in another state than 'connected', then this function has no effect except for logging an error.

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.

Note

This only has effect for UDP connections.

This does not have any effect for connections that use UDP datagram encryption (which always use a built-in checksum).

See also

[getCRCEnabled](#)

§ `getPacketLossByCRC()`

```
int getPacketLossByCRC ( void ) const
```

Returns

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

See also

[`setCRCEnabled`](#)

§ getTrafficStatsEnabled()

```
bool getTrafficStatsEnabled ( void ) const
```

Returns

true if traffic statistics of a peer are enabled. Default trafficStatsEnabled: false (disabled).

§ setTrafficStatsEnabled()

```
void setTrafficStatsEnabled ( bool trafficStatsEnabled )
```

Enables or disables the traffic statistics of a peer. Default trafficStatsEnabled: false (disabled).

§ getTrafficStatsElapsedMs()

```
int getTrafficStatsElapsedMs ( void ) const
```

Returns

the count of milliseconds the stats are enabled for tracking.

§ getTrafficStatsIncoming()

```
const TrafficStats & getTrafficStatsIncoming ( void ) const
```

Returns

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.

§ getTrafficStatsOutgoing()

```
const TrafficStats & getTrafficStatsOutgoing ( void ) const
```

Returns

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.

§ getTrafficStatsGameLevel()

```
const TrafficStatsGameLevel &  
getTrafficStatsGameLevel (void ) const
```

Returns

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).

§ getQuickResendAttempts()

```
nByte getQuickResendAttempts ( void ) const
```

Returns

the number of resend attempts for a reliable command that are done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

§ setQuickResendAttempts()

```
void setQuickResendAttempts ( nByte quickResendAttempts )
```

Sets the number of resend attempts for a reliable command can be done in quick succession (after $\text{RoundTripTime} + 4 * \text{RoundTripTimeVariance}$).

Remarks

The default value is 0. Any later resend attempt will then double the time before the next resend takes place. The max value is 4. Make sure to set `SentCountAllowance` to a slightly higher value, as more repeats will get done.

§ getConnectionProtocol()

```
nByte getConnectionProtocol ( void ) const
```

Returns

the currently set connection protocol.

Note

The value returned is not guaranteed to be the value used for the currently active connection, but only the value that has last been passed to **setConnectionProtocol()**. The reason therefor is that whatever you pass to **setConnectionProtocol()** won't take effect until you re-connect.

§ setConnectionProtocol()

```
void setConnectionProtocol ( nByte connectionProtocol )
```

Sets the connection protocol to be used with the next **connect()** call.

Note

This does not have any effect on the protocol that is used for an already active connection. So you need to re-connect after setting a different connection protocol for the changes to actually take effect.

§ getChannelCountUserChannels()

```
nByte getChannelCountUserChannels ( void ) const
```

The IDs from 0 to **getChannelCountUserChannels()**-1 can be passed as channelID to operations that offer this parameter.

Returns

the number of different channels that are available for sending operations on.

§ getPeerCount()

```
short getPeerCount ( void )
```

static

Returns

the count of peers, which have been initialized since the start of the application. Interesting mainly for debugging purposes.

§ getMaxAppIDLength()

```
unsigned int getMaxAppIDLength ( void )
```

static

Returns

the maximum allowed length for the appID that gets passed to **connect()** in characters

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

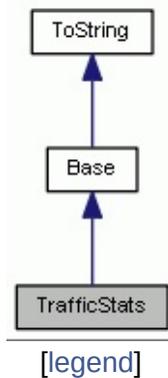
Client API 4.1.12.2

ExitGames > Photon > TrafficStats >

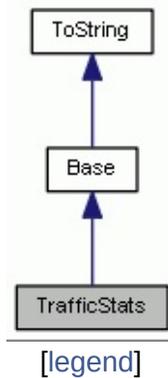
[Public Member Functions](#) | [List of all members](#)

TrafficStats Class Reference

Inheritance diagram for TrafficStats:



Collaboration diagram for TrafficStats:



Public Member Functions

virtual **~TrafficStats** (void)

int **getPackageHeaderSize** (void) const

int **getReliableCommandCount** (void) const

int **getUnreliableCommandCount** (void)
const

int **getFragmentCommandCount** (void)
const

int **getControlCommandCount** (void) const

int **getTotalPacketCount** (void) const

int **getTotalCommandsInPackets** (void)
const

int **getReliableCommandBytes** (void) const

int **getUnreliableCommandBytes** (void)
const

int **getFragmentCommandBytes** (void)
const

int **getControlCommandBytes** (void) const

int **getTotalCommandCount** (void) const

int **getTotalCommandBytes** (void) const

int **getTotalPacketBytes** (void) const

int **getTimestampOfLastAck** (void) const

int **getTimestampOfLastReliableCommand**
(void) const

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool
withTypes=false) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This class provides network traffic statistics.

See also

**PhotonPeer::getTrafficStatsEnabled(),
PhotonPeer::setTrafficStatsEnabled(),
PhotonPeer::getTrafficStatsElapsedMs(),
PhotonPeer::getTrafficStatsIncoming(),
PhotonPeer::getTrafficStatsOutgoing()**

Constructor & Destructor Documentation

§ ~TrafficStats()

`~TrafficStats (void)`

virtual

Destructor.

Member Function Documentation

§ getPackageHeaderSize()

```
int getPackageHeaderSize ( void ) const
```

Returns

the byte-size of per-package headers.

§ getReliableCommandCount()

```
int getReliableCommandCount ( void ) const
```

Returns

the reliable commands that are created/received by this client, ignoring repeats (the out command count can be higher than this due to repeats).

§ getUnreliableCommandCount()

```
int getUnreliableCommandCount ( void ) const
```

Returns

the unreliable commands that are created/received by this client.

§ getFragmentCommandCount()

```
int getFragmentCommandCount ( void ) const
```

Remarks

Commands get fragmented, when UDP is used and they are too big to fit into a single UDP packet.

Returns

the number of fragments for fragmented commands that are created/received by this client.

§ getControlCommandCount()

```
int getControlCommandCount ( void ) const
```

Remarks

The returned value includes connect, disconnect, verify connect, pings and acknowledgments for reliable commands.

Returns

the control commands that are created/received by this client under the hood to administer the connection.

§ getTotalPacketCount()

```
int getTotalPacketCount ( void ) const
```

Returns

the overall packets that are in created/received by this client.

§ getTotalCommandsInPackets()

```
int getTotalCommandsInPackets ( void ) const
```

Returns

the overall commands that are created/received by this client. For fragmented commands each fragment counts separately.

§ getReliableCommandBytes()

```
int getReliableCommandBytes ( void ) const
```

Returns

the bytes of the reliable commands that are created/received by this client, ignoring repeats (the count of actually outgoing bytes can be higher than this due to repeats).

§ getUnreliableCommandBytes()

```
int getUnreliableCommandBytes ( void ) const
```

Returns

the bytes of the unreliable commands that are created/received by this client.

§ getFragmentCommandBytes()

```
int getFragmentCommandBytes ( void ) const
```

Remarks

Commands get fragmented, when UDP is used and they are too big to fit into a single UDP packet.

Returns

the bytes of the fragments for fragmented commands that are created/received by this client.

§ getControlCommandBytes()

```
int getControlCommandBytes ( void ) const
```

Remarks

Control commands include connect, disconnect, verify connect, pings and acknowledgments for reliable commands.

Returns

the bytes of the control commands that are created/received by this client under the hood to administer the connection.

§ getTotalCommandCount()

```
int getTotalCommandCount ( void ) const
```

Returns

the sum of the return values of [getReliableCommandCount\(\)](#), [getUnreliableCommandCount\(\)](#), [getFragmentCommandCount\(\)](#) and [getControlCommandCount\(\)](#)

§ getTotalCommandBytes()

```
int getTotalCommandBytes ( void ) const
```

Returns

the sum of the return values of [getReliableCommandBytes\(\)](#), [getUnreliableCommandBytes\(\)](#), [getFragmentCommandBytes\(\)](#) and [getControlCommandBytes\(\)](#)

§ getTotalPacketBytes()

```
int getTotalPacketBytes ( void ) const
```

Returns

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

§ `getTimestampOfLastAck()`

```
int getTimestampOfLastAck ( void ) const
```

Returns

the timestamp of the last incoming ACK that has been read (every **`PhotonPeer::getTimePingInterval()`** milliseconds this client sends a PING which must be ACKd by the server).

§ getTimestampOfLastReliableCommand()

```
int getTimestampOfLastReliableCommand ( void ) const
```

Returns

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

§ toString()

```
JString & toString ( Common::JString & retStr,  
                    bool                               withTypes = false  
                    )                               const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

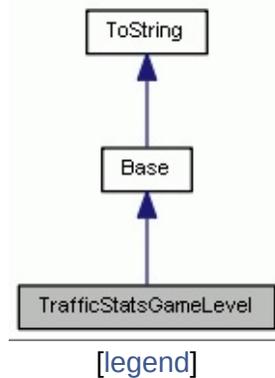
Returns

a JString representation of the instance and its contents for debugging purposes.

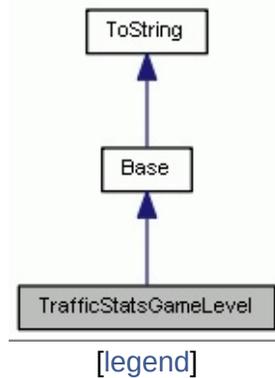
Implements **ToString**.

TrafficStatsGameLevel Class Reference

Inheritance diagram for TrafficStatsGameLevel:



Collaboration diagram for TrafficStatsGameLevel:



Public Member Functions

virtual **~TrafficStatsGameLevel** (void)

int **getOperationByteCount** (void) const

int **getOperationCount** (void) const

int **getResultByteCount** (void) const

int **getResultCount** (void) const

int **getEventByteCount** (void) const

int **getEventCount** (void) const

int **getLongestOpResponseCallback** (void)
const

nByte **getLongestOpResponseCallbackOpCod**
(void) const

int **getLongestEventCallback** (void) const

nByte **getLongestEventCallbackCode** (void)
const

int **getLongestDeltaBetweenDispatching**
(void) const

int **getLongestDeltaBetweenSending** (void)
const

int **getDispatchIncomingCommandsCalls**
(void) const

int **getSendOutgoingCommandsCalls** (void)
const

int **getTotalByteCount** (void) const

int **getTotalMessageCount** (void) const

int **getTotalIncomingByteCount** (void) const

int **getTotalIncomingMessageCount** (void)
const

int **getTotalOutgoingByteCount** (void) const

int **getTotalOutgoingMessageCount** (void)
const

void **resetMaximumCounters** (void)

virtual **Common::JString** & **toString** (**Common::JString** &retStr, bool
withTypes=false) const

virtual **Common::JString** **toStringVitalStats** (void) const

▶ **Public Member Functions inherited from Base**

virtual **~Base** (void)

▶ **Public Member Functions inherited from ToString**

virtual **~ToString** (void)

virtual **JString** **typeToString** (void) const

JString **toString** (bool withTypes=false) const

Additional Inherited Members

▶ Static Public Member Functions inherited from **Base**

static void **setListener** (const **BaseListener**
*baseListener)

static int **getDebugOutputLevel** (void)

static bool **setDebugOutputLevel** (int
debugLevel)

static const **LogFormatOptions** & **getLogFormatOptions** (void)

static void **setLogFormatOptions** (const
LogFormatOptions &options)

Detailed Description

This class provides game level traffic statistics.

See also

PhotonPeer::getTrafficStatsEnabled(),
PhotonPeer::setTrafficStatsEnabled(),
PhotonPeer::getTrafficStatsElapsedMs(),
PhotonPeer::getTrafficStatsGameLevel()

Constructor & Destructor Documentation

§ ~TrafficStatsGameLevel()

~TrafficStatsGameLevel (void)

virtual

Destructor.

Member Function Documentation

§ getOperationByteCount()

```
int getOperationByteCount ( void ) const
```

Returns

the sum of outgoing operations in bytes

§ getOperationCount()

```
int getOperationCount ( void ) const
```

Returns

the count of outgoing operations.

§ getResultByteCount()

```
int getResultByteCount ( void ) const
```

Returns

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

§ getResultCount()

```
int getResultCount ( void ) const
```

Returns

the count of incoming operation-results.

§ getEventByteCount()

```
int getEventByteCount ( void ) const
```

Returns

the sum of byte-cost of incoming events.

§ `getEventCount()`

```
int getEventCount ( void ) const
```

Returns

the count of incoming events.

§ getLongestOpResponseCallback()

```
int getLongestOpResponseCallback ( void ) const
```

Note

If such a callback takes long, it will lower the network performance and might lead to timeouts.

Returns

the longest time it took to complete a call to OnOperationResponse (in your code).

§ getLongestOpResponseCallbackOpCode()

```
nByte getLongestOpResponseCallbackOpCode ( void ) const
```

Returns

the OperationCode that causes the LongestOpResponseCallback.
See that description.

§ getLongestEventCallback()

```
int getLongestEventCallback ( void ) const
```

Note

If such a callback takes long, it will lower the network performance and might lead to timeouts.

Returns

the longest time a call to OnEvent (in your code) took.

§ getLongestEventCallbackCode()

```
nByte getLongestEventCallbackCode ( void ) const
```

Returns

the EventCode that caused the LongestEventCallback. See that description.

§ getLongestDeltaBetweenDispatching()

```
int getLongestDeltaBetweenDispatching ( void ) const
```

Note

This is not a crucial timing for networking. Long gaps just add "local lag" to events that are available already.

Returns

the longest time between subsequent calls to **PhotonPeer::dispatchIncomingCommands()** in milliseconds.

§ getLongestDeltaBetweenSending()

```
int getLongestDeltaBetweenSending ( void ) const
```

Note

This is a crucial value for network stability. Without calling **PhotonPeer::sendOutgoingCommands()**, nothing will be sent to the server, which might time out this client.

Returns

the longest time between subsequent calls to **PhotonPeer::sendOutgoingCommands()** in milliseconds.

§ getDispatchIncomingCommandsCalls()

```
int getDispatchIncomingCommandsCalls ( void ) const
```

Returns

the number of calls of

PhotonPeer::dispatchIncomingCommands().

§ getSendOutgoingCommandsCalls()

```
int getSendOutgoingCommandsCalls ( void ) const
```

Returns

the number of calls of **PhotonPeer::sendOutgoingCommands()**.

§ getTotalByteCount()

```
int getTotalByteCount ( void ) const
```

Returns

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

§ getTotalMessageCount()

```
int getTotalMessageCount ( void ) const
```

Returns

the sum of counted "logic level" messages.

§ getTotalIncomingByteCount()

```
int getTotalIncomingByteCount ( void ) const
```

Returns

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

§ getTotalIncomingMessageCount()

```
int getTotalIncomingMessageCount ( void ) const
```

Returns

the sum of counted incoming "logic level" messages.

§ getTotalOutgoingByteCount()

```
int getTotalOutgoingByteCount ( void ) const
```

Returns

the sum of byte-cost of all outgoing "logic level" messages (= OperationByteCount).

§ getTotalOutgoingMessageCount()

```
int getTotalOutgoingMessageCount ( void ) const
```

Returns

the sum of counted outgoing "logic level" messages (= OperationCount).

§ resetMaximumCounters()

```
void resetMaximumCounters ( void )
```

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

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

§ toString()

```
JString & toString ( Common::JString & retStr,  
                  bool withTypes = false  
                  ) const
```

virtual

Remarks

The cost of this function depends a lot on implementation details of the implementing subclasses, but for container classes this function can become quite expensive, if the instance contains huge amounts of data, as its cost for many container class implementations increases disproportionately high to the size of the payload.

Parameters

retStr reference to a string, to store the return-value in; the information, which is generated by this function, will be attached at the end of any eventually existing previous content of the string

withTypes set to true, to include type information in the generated string

Returns

a JString representation of the instance and its contents for debugging purposes.

Implements **ToString**.

§ toStringVitalStats()

JString toStringVitalStats (void) const

virtual

Returns

a JString representation of the vital stats for debugging purposes.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Class Index

[a](#) | [b](#) | [c](#) | [d](#) | [e](#) | [f](#) | [h](#) | [j](#) | [k](#) | [l](#) | [m](#) | [o](#) | [p](#) | [r](#) | [s](#) | [t](#) | [u](#) | [v](#) | [w](#)

a

[AllocatorInterface \(ExitGames::Common::MemoryManagement\)](#)
[ANSIString \(ExitGames::Common\)](#)
[AuthenticationValues \(ExitGames::LoadBalancing\)](#)
[AuthenticationValues \(ExitGames::Chat\)](#)

b

[Base \(ExitGames::Common\)](#)
[BaseCharString \(ExitGames::Common\)](#)
[BaseListener \(ExitGames::Common\)](#)

c

[Channel \(ExitGames::Chat\)](#)
[Client \(ExitGames::LoadBalancing\)](#)
[Client \(ExitGames::Chat\)](#)
[CustomType \(ExitGames::Common\)](#)
[CustomTypeBase \(ExitGames::Common\)](#)
[CustomTypeFactory \(ExitGames::Common\)](#)

d

[DeSerializer \(ExitGames::Common\)](#)

[a](#) | [b](#) | [c](#) | [d](#) | [e](#) | [f](#) | [h](#) | [j](#) | [k](#) | [l](#) | [m](#) | [o](#) | [p](#) | [r](#) | [s](#) | [t](#) | [u](#) | [v](#) | [w](#)

Photon C++

Client API 4.1.12.2

Class Hierarchy

[Go to the graphical class hierarchy](#)

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

[detail level 1 2 3 4]

 **AllocatorInterface**

▼  **BaseListener**

 **Listener**

 **Listener**

▼  **PhotonListener**

 **Client**

 **Client**

 **EventData**

 **OperationRequest**

 **OperationResponse**

▼  **PhotonPeer**

 **Peer**

 **LitePeer**

 **Peer**

▼  **Protocol**

 **TCP**

 **UDP**

 **UDPAlternative**

 **WS**

 **WSS**

🔗 Puncher

🔗 PunchListener

🔗 RelayClient

▼ 🔗 ToString

▼ 🔗 Base

🔗 AuthenticationValues

🔗 Channel

▼ 🔗 BaseCharString

🔗 ANSIStrng

🔗 UTF8String

▼ 🔗 CustomTypeBase

🔗 CustomType< typeCode >

🔗 CustomTypeFactory< typeCode >

🔗 DeSerializer

▼ 🔗 DictionaryBase

🔗 Dictionary< nByte, Common::ExitGames::Common::Object

🔗 Dictionary< nByte, Common::Object >

🔗 Dictionary< EKeyType, EValueType >

🔗 EGTime

🔗 Hashtable

🔗 JVector< Etype >

▼ 🔗 Object

🔗 KeyObject< Etype >

🔗 ValueObject< Etype >

🔗 Serializer

🔗 AuthenticationValues

🔗 FriendInfo

🔗 LobbyStatsRequest

🔗 LobbyStatsResponse

▼ 🔗 Player

- 🔗 [MutablePlayer](#)
- 🔗 [RaiseEventOptions](#)
- ▼ 🔗 [Room](#)
 - 🔗 [MutableRoom](#)
 - 🔗 [RoomOptions](#)
 - 🔗 [WebFlags](#)
 - 🔗 [TrafficStats](#)
 - 🔗 [TrafficStatsGameLevel](#)
 - 🔗 [JVector< Common::ExitGames::Common::JString >](#)
 - 🔗 [JVector< Common::ExitGames::Common::JVector< unsigned >](#)
 - 🔗 [JVector< Common::ExitGames::Common::Object >](#)
 - 🔗 [JVector< ExitGames::Chat::Channel *>](#)
 - 🔗 [JVector< ExitGames::Common::Object >](#)
 - 🔗 [JVector< ExitGames::LoadBalancing::FriendInfo >](#)
 - 🔗 [JVector< ExitGames::LoadBalancing::LobbyStatsRequest >](#)
 - 🔗 [JVector< ExitGames::LoadBalancing::Player *>](#)
 - 🔗 [JVector< ExitGames::LoadBalancing::Room *>](#)
 - 🔗 [JVector< int >](#)
 - 🔗 [JVector< nByte >](#)
- 🔗 [JString](#)
- 🔗 [LogFormatOptions](#)
- 🔗 [Logger](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- a -

- [addElement\(\)](#) : [JVector< Etype >](#)
 - [addElements\(\)](#) : [JVector< Etype >](#)
 - [alloc\(\)](#) : [AllocatorInterface](#)
 - [ANSIRepresentation\(\)](#) : [JString](#)
 - [ANSIString\(\)](#) : [ANSIString](#)
 - [AuthenticationValues\(\)](#) : [AuthenticationValues](#)
-

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- **b** -

- BaseCharString() : [BaseCharString](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- C -

- [capacity\(\)](#) : [JString](#)
- [charAt\(\)](#) : [JString](#)
- [cleanup\(\)](#) : [CustomTypeBase](#)
- [Client\(\)](#) : [Client](#)
- [compare\(\)](#) : [CustomTypeBase](#)
- [compareTo\(\)](#) : [JString](#)
- [concat\(\)](#) : [JString](#)
- [connect\(\)](#) : [Client](#) , [PhotonPeer](#)
- [connectReturn\(\)](#) : [Listener](#)
- [constructClass\(\)](#) : [CustomType< typeCode >](#)
- [contains\(\)](#) : [Dictionary< EKeyType, EValueType >](#) , [DictionaryBase](#) , [Hashtable](#) , [JVector< Etype >](#)
- [copy\(\)](#) : [CustomTypeFactory< typeCode >](#)
- [copyFactory\(\)](#) : [CustomTypeFactory< typeCode >](#)
- [copyInto\(\)](#) : [JVector< Etype >](#)
- [create\(\)](#) : [CustomTypeFactory< typeCode >](#)
- [cstr\(\)](#) : [BaseCharString](#) , [JString](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- d -

- dealloc() : [AllocatorInterface](#)
- debugReturn() : [Listener](#) , [BaseListener](#) , [Listener](#)
- deconstructClass() : [CustomType< typeCode >](#)
- deleteChars() : [JString](#)
- deserialize() : [CustomTypeBase](#)
- DeSerializer() : [DeSerializer](#)
- destroy() : [CustomTypeFactory< typeCode >](#)
- destroyFactory() : [CustomTypeFactory< typeCode >](#)
- Dictionary() : [Dictionary< EKeyType, EValueType >](#)
- DictionaryBase() : [DictionaryBase](#)
- disconnect() : [Client](#) , [Peer](#) , [PhotonPeer](#)
- disconnectReturn() : [Listener](#)
- dispatchIncomingCommands() : [Client](#) , [PhotonPeer](#)
- duplicate() : [CustomTypeBase](#)

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- e -

- [EGTime\(\)](#) : [EGTime](#)
- [endsWith\(\)](#) : [JString](#)
- [ensureCapacity\(\)](#) : [JString](#) , [JVector< Etype >](#)
- [equals\(\)](#) : [JString](#)
- [equalsIgnoreCase\(\)](#) : [JString](#)
- [establishEncryption\(\)](#) : [PhotonPeer](#)
- [EventData\(\)](#) : [EventData](#)

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- f -

- `fetchServerTimestamp()` : [Client](#) , [PhotonPeer](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- g -

- [get\(\)](#) : [AllocatorInterface](#)
- [getAddDateTime\(\)](#) : [LogFormatOptions](#)
- [getAddFile\(\)](#) : [LogFormatOptions](#)
- [getAddFunction\(\)](#) : [LogFormatOptions](#)
- [getAddLevel\(\)](#) : [LogFormatOptions](#)
- [getAddLine\(\)](#) : [LogFormatOptions](#)
- [getAutoJoinLobby\(\)](#) : [Client](#)
- [getByteCountCurrentDispatch\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getByteCountLastOperation\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getBytesIn\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getBytesOut\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getCacheSliceIndex\(\)](#) : [RaiseEventOptions](#)
- [getCapacity\(\)](#) : [JVector< Etype >](#)
- [getCArray\(\)](#) : [JVector< Etype >](#)
- [getChannelCountUserChannels\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getChannelID\(\)](#) : [RaiseEventOptions](#)
- [getCode\(\)](#) : [EventData](#)
- [getConnectionProtocol\(\)](#) : [PhotonPeer](#)
- [getControlCommandBytes\(\)](#) : [TrafficStats](#)
- [getControlCommandCount\(\)](#) : [TrafficStats](#)
- [getCountGamesRunning\(\)](#) : [Client](#)
- [getCountPlayersIngame\(\)](#) : [Client](#)
- [getCountPlayersOnline\(\)](#) : [Client](#)
- [getCRCEnabled\(\)](#) : [Client](#) , [PhotonPeer](#)
- [getCurrentlyJoinedRoom\(\)](#) : [Client](#)
- [getCustomProperties\(\)](#) : [Player](#) , [Room](#)
- [getCustomRoomProperties\(\)](#) : [RoomOptions](#)
- [getCustomType\(\)](#) : [Object](#)
- [getData\(\)](#) : [AuthenticationValues](#) , [Serializer](#) , [AuthenticationValues](#)

- `getDataAddress()` : **KeyObject< Etype > , ValueObject< Etype >**
- `getDataCopy()` : **KeyObject< Etype > , ValueObject< Etype >**
- `getDebugMessage()` : **OperationResponse**
- `getDebugOutputLevel()` : **Client , Base , Logger , Client , PhotonPeer**
- `getDimensions()` : **Object**
- `getDirectMode()` : **Room , RoomOptions**
- `getDisconnectedCause()` : **Client**
- `getDisconnectTimeout()` : **Client , PhotonPeer**
- `getDispatchIncomingCommandsCalls()` : **TrafficStatsGameLevel**
- `getElementAt()` : **JVector< Etype >**
- `getEmptyRoomTtl()` : **RoomOptions**
- `getEventByteCount()` : **TrafficStatsGameLevel**
- `getEventCaching()` : **RaiseEventOptions**
- `getEventCount()` : **TrafficStatsGameLevel**
- `getFirstElement()` : **JVector< Etype >**
- `getFlags()` : **WebFlags**
- `getFormatOptions()` : **Logger**
- `getFragmentCommandBytes()` : **TrafficStats**
- `getFragmentCommandCount()` : **TrafficStats**
- `getFriendList()` : **Client**
- `getFriendListAge()` : **Client**
- `getHashtable()` : **DictionaryBase**
- `getHttpForward()` : **WebFlags**
- `getIncomingReliableCommandsCount()` : **Client , PhotonPeer**
- `getIndexOf()` : **JVector< Etype >**
- `getInterestGroup()` : **RaiseEventOptions**
- `getIsEmpty()` : **JVector< Etype >**
- `getIsEncryptionAvailable()` : **Client , PhotonPeer**
- `getIsInactive()` : **Player**
- `getIsInGameRoom()` : **Client**
- `getIsInLobby()` : **Client**
- `getIsInRoom()` : **Client , FriendInfo**
- `getIsMasterClient()` : **Player**
- `getIsOnline()` : **FriendInfo**
- `getIsOpen()` : **Room , RoomOptions**
- `getIsPayloadEncryptionAvailable()` : **Client , PhotonPeer**
- `getIsVisible()` : **RoomOptions**
- `getKeys()` : **Dictionary< EKeyType, EValueType > , DictionaryBase , Hashtable**

- getKeyTypes() : **Dictionary< EKeyType, EValueType > , DictionaryBase**
- getLastElement() : **JVector< Etype >**
- getLastIndexOf() : **JVector< Etype >**
- getLimitOfUnreliableCommands() : **Client , PhotonPeer**
- getListener() : **PhotonPeer**
- getLobbyName() : **RoomOptions**
- getLobbyType() : **RoomOptions**
- getLocalPlayer() : **Client**
- getLogFormatOptions() : **Client , Base , Client , PhotonPeer**
- getLongestDeltaBetweenDispatching() : **TrafficStatsGameLevel**
- getLongestDeltaBetweenSending() : **TrafficStatsGameLevel**
- getLongestEventCallback() : **TrafficStatsGameLevel**
- getLongestEventCallbackCode() : **TrafficStatsGameLevel**
- getLongestOpResponseCallback() : **TrafficStatsGameLevel**
- getLongestOpResponseCallbackOpCode() : **TrafficStatsGameLevel**
- getMasterserverAddress() : **Client**
- getMaxAppIDLength() : **PhotonPeer**
- getMaxNumberOfNamespaces() : **LogFormatOptions**
- getMaxPlayers() : **Room , RoomOptions**
- getName() : **LobbyStatsRequest , LobbyStatsResponse , Player , Room**
- getNumber() : **Player**
- getNumTargetPlayers() : **RaiseEventOptions**
- getOperationByteCount() : **TrafficStatsGameLevel**
- getOperationCode() : **OperationRequest , OperationResponse**
- getOperationCount() : **TrafficStatsGameLevel**
- getPackageHeaderSize() : **TrafficStats**
- getPacketLossByCRC() : **Client , PhotonPeer**
- getParameterForCode() : **EventData , OperationRequest , OperationResponse**
- getParameters() : **AuthenticationValues , EventData , OperationRequest , OperationResponse**
- getPeerCount() : **Client , LobbyStatsResponse , PhotonPeer**
- getPeerID() : **Client , PhotonPeer**
- getPeerState() : **PhotonPeer**
- getPlayerCount() : **MutableRoom , Room**
- getPlayerTtl() : **RoomOptions**
- getPlugins() : **RoomOptions**

- `getPrivateChannel()` : **Client**
- `getPrivateChannels()` : **Client**
- `getPropsListedInLobby()` : **RoomOptions**
- `getPublicChannel()` : **Client**
- `getPublicChannels()` : **Client**
- `getPublishUserID()` : **RoomOptions**
- `getQueuedIncomingCommands()` : **Client** , **PhotonPeer**
- `getQueuedOutgoingCommands()` : **Client** , **PhotonPeer**
- `getQuickResendAttempts()` : **Client** , **PhotonPeer**
- `getReceiverGroup()` : **RaiseEventOptions**
- `getRegion()` : **Client**
- `getRegionWithBestPing()` : **Client**
- `getReliableCommandBytes()` : **TrafficStats**
- `getReliableCommandCount()` : **TrafficStats**
- `getResentReliableCommands()` : **Client** , **PhotonPeer**
- `getResultByteCount()` : **TrafficStatsGameLevel**
- `getResultCount()` : **TrafficStatsGameLevel**
- `getReturnCode()` : **OperationResponse**
- `getRoom()` : **FriendInfo**
- `getRoomCount()` : **LobbyStatsResponse**
- `getRoomList()` : **Client**
- `getRoomNameList()` : **Client**
- `getRoundTripTime()` : **Client** , **PhotonPeer**
- `getRoundTripTimeVariance()` : **Client** , **PhotonPeer**
- `getSecret()` : **AuthenticationValues**
- `getSendAuthCookie()` : **WebFlags**
- `getSendOutgoingCommandsCalls()` : **TrafficStatsGameLevel**
- `getSendState()` : **WebFlags**
- `getSendSync()` : **WebFlags**
- `getSentCountAllowance()` : **Client** , **PhotonPeer**
- `getServerAddress()` : **PhotonPeer**
- `getServerTime()` : **Client** , **PhotonPeer**
- `getServerTimeOffset()` : **Client** , **PhotonPeer**
- `getSize()` : **DictionaryBase** , **Hashtable** , **JVector< Etype >** , **Serializer**
- `getSizes()` : **Object**
- `getState()` : **Client**
- `getSuppressRoomEvents()` : **RoomOptions**
- `getTargetPlayers()` : **RaiseEventOptions**
- `getTimePingInterval()` : **Client** , **PhotonPeer**

- `getTimestampOfLastAck()` : **TrafficStats**
- `getTimestampOfLastReliableCommand()` : **TrafficStats**
- `getTimestampOfLastSocketReceive()` : **Client** , **PhotonPeer**
- `getTotalByteCount()` : **TrafficStatsGameLevel**
- `getTotalCommandBytes()` : **TrafficStats**
- `getTotalCommandCount()` : **TrafficStats**
- `getTotalCommandsInPackets()` : **TrafficStats**
- `getTotalIncomingByteCount()` : **TrafficStatsGameLevel**
- `getTotalIncomingMessageCount()` : **TrafficStatsGameLevel**
- `getTotalMessageCount()` : **TrafficStatsGameLevel**
- `getTotalOutgoingByteCount()` : **TrafficStatsGameLevel**
- `getTotalOutgoingMessageCount()` : **TrafficStatsGameLevel**
- `getTotalPacketBytes()` : **TrafficStats**
- `getTotalPacketCount()` : **TrafficStats**
- `getTrafficStatsElapsedMs()` : **Client** , **PhotonPeer**
- `getTrafficStatsEnabled()` : **Client** , **PhotonPeer**
- `getTrafficStatsGameLevel()` : **Client** , **PhotonPeer**
- `getTrafficStatsIncoming()` : **Client** , **PhotonPeer**
- `getTrafficStatsOutgoing()` : **Client** , **PhotonPeer**
- `getType()` : **AuthenticationValues** , **Object** , **AuthenticationValues** , **LobbyStatsRequest** , **LobbyStatsResponse**
- `getUnreliableCommandBytes()` : **TrafficStats**
- `getUnreliableCommandCount()` : **TrafficStats**
- `getUserID()` : **AuthenticationValues** , **Client** , **AuthenticationValues** , **Client** , **FriendInfo** , **Player**
- `getValue()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **Hashtable**
- `getValueDimensions()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase**
- `getValueSizes()` : **DictionaryBase**
- `getValueTypes()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase**
- `getWebFlags()` : **RaiseEventOptions**



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- **h** -

- Hashtable() : [Hashtable](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- i -

- `indexOf()` : [JString](#)
- `initUDPEncryption()` : [PhotonPeer](#)
- `initUserDataEncryption()` : [PhotonPeer](#)
- `insertElementAt()` : [JVector< Etype >](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- j -

- [JString\(\)](#) : [JString](#)
 - [JStringRepresentation\(\)](#) : [ANSIString](#) , [BaseCharString](#) , [UTF8String](#)
 - [JVector\(\)](#) : [JVector< Etype >](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- k -

- `KeyObject()` : [KeyObject< Etype >](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- | -

- [lastIndexOf\(\)](#) : [JString](#)
- [length\(\)](#) : [BaseCharString](#) , [JString](#)
- [LitePeer\(\)](#) : [LitePeer](#)
- [LobbyStatsRequest\(\)](#) : [LobbyStatsRequest](#)
- [log\(\)](#) : [Logger](#)
- [Logger\(\)](#) : [Logger](#)

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- 0 -

- Object() : [Object](#)
- onEvent() : [PhotonListener](#)
- onGetMessages() : [Listener](#)
- onOperationResponse() : [PhotonListener](#)
- onPingResponse() : [PhotonListener](#)
- onPrivateMessage() : [Listener](#)
- onStateChange() : [Listener](#)
- onStatusChanged() : [PhotonListener](#)
- onStatusUpdate() : [Listener](#)
- opAddFriends() : [Client](#)
- opChangeGroups() : [LitePeer](#) , [Client](#)
- opCreateRoom() : [Client](#)
- opCustom() : [Client](#) , [PhotonPeer](#)
- opCustomAuthenticationSendNextStepData() : [Client](#)
- OperationRequest() : [OperationRequest](#)
- OperationResponse() : [OperationResponse](#)
- operator const char *() : [ANSIString](#) , [BaseCharString](#) , [UTF8String](#)
- operator const EG_CHAR *() : [JString](#)
- operator JString() : [ANSIString](#) , [BaseCharString](#) , [UTF8String](#)
- operator!==() : [Dictionary< EKeyType, EValueType >](#) , [DictionaryBase](#) , [EGTime](#) , [Hashtable](#) , [JString](#) , [JVector< Etype >](#) , [Object](#)
- operator+() : [EGTime](#) , [JString](#)
- operator+=() : [EGTime](#) , [JString](#)
- operator-() : [EGTime](#)
- operator-=() : [EGTime](#)
- operator<() : [EGTime](#) , [JString](#)
- operator<<() : [JString](#)
- operator<=() : [EGTime](#) , [JString](#)

- operator=() : **ANSIString** , **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **EGTime** , **Hashtable** , **JString** , **JVector< Etype >** , **KeyObject< Etype >** , **Object** , **UTF8String** , **ValueObject< Etype >** , **MutablePlayer** , **MutableRoom** , **Player** , **RaiseEventOptions** , **Room** , **RoomOptions** , **EventData** , **OperationRequest** , **OperationResponse**
 - operator==() : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **EGTime** , **Hashtable** , **JString** , **JVector< Etype >** , **Object** , **Player** , **Room**
 - operator>() : **EGTime** , **JString**
 - operator>=() : **EGTime** , **JString**
 - operator[]() : **Dictionary< EKeyType, EValueType >** , **Hashtable** , **JString** , **JVector< Etype >** , **EventData** , **OperationRequest** , **OperationResponse**
 - opFindFriends() : **Client**
 - opGetProperties() : **LitePeer**
 - opGetPropertiesOfActor() : **LitePeer**
 - opGetPropertiesOfGame() : **LitePeer**
 - opJoin() : **LitePeer**
 - opJoinLobby() : **Client**
 - opJoinOrCreateRoom() : **Client**
 - opJoinRandomRoom() : **Client**
 - opJoinRoom() : **Client**
 - opLeave() : **LitePeer**
 - opLeaveLobby() : **Client**
 - opLeaveRoom() : **Client**
 - opLobbyStats() : **Client**
 - opPublishMessage() : **Client**
 - opRaiseEvent() : **LitePeer** , **Client**
 - opRemoveFriends() : **Client**
 - opSendPrivateMessage() : **Client**
 - opSetOnlineStatus() : **Client**
 - opSetPropertiesOfActor() : **LitePeer**
 - opSetPropertiesOfGame() : **LitePeer**
 - opSubscribe() : **Client**
 - opUnsubscribe() : **Client**
 - opWebRpc() : **Client**
 - overflowed() : **EGTime**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- p -

- PhotonPeer() : [PhotonPeer](#)
 - pingServer() : [PhotonPeer](#)
 - Player() : [Player](#)
 - pop() : [DeSerializer](#)
 - push() : [Serializer](#)
 - put() : [Dictionary< EKeyType, EValueType >](#) , [Hashtable](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- r -

- [RaiseEventOptions\(\)](#) : [RaiseEventOptions](#)
- [reconnectAndRejoin\(\)](#) : [Client](#)
- [remove\(\)](#) : [Dictionary< EKeyType, EValueType >](#) , [DictionaryBase](#) , [Hashtable](#)
- [removeAllElements\(\)](#) : [DictionaryBase](#) , [Hashtable](#) , [JVector< Etype >](#)
- [removeElement\(\)](#) : [JVector< Etype >](#)
- [removeElementAt\(\)](#) : [JVector< Etype >](#)
- [replace\(\)](#) : [JString](#)
- [resetMaximumCounters\(\)](#) : [TrafficStatsGameLevel](#)
- [resetTrafficStats\(\)](#) : [Client](#) , [PhotonPeer](#)
- [resetTrafficStatsMaximumCounters\(\)](#) : [Client](#) , [PhotonPeer](#)
- [resize\(\)](#) : [AllocatorInterface](#)
- [Room\(\)](#) : [Room](#)
- [RoomOptions\(\)](#) : [RoomOptions](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- S -

- [selectRegion\(\)](#) : [Client](#)
- [sendAcksOnly\(\)](#) : [Client](#) , [PhotonPeer](#)
- [sendDirect\(\)](#) : [Client](#)
- [sendOutgoingCommands\(\)](#) : [Client](#) , [PhotonPeer](#)
- [serialize\(\)](#) : [CustomTypeBase](#)
- [service\(\)](#) : [Client](#) , [PhotonPeer](#)
- [serviceBasic\(\)](#) : [Client](#) , [PhotonPeer](#)
- [setAddDateTime\(\)](#) : [LogFormatOptions](#)
- [setAddFile\(\)](#) : [LogFormatOptions](#)
- [setAddFunction\(\)](#) : [LogFormatOptions](#)
- [setAddLevel\(\)](#) : [LogFormatOptions](#)
- [setAddLine\(\)](#) : [LogFormatOptions](#)
- [setAutoJoinLobby\(\)](#) : [Client](#)
- [setCacheSliceIndex\(\)](#) : [RaiseEventOptions](#)
- [setChannelID\(\)](#) : [RaiseEventOptions](#)
- [setConnectionProtocol\(\)](#) : [PhotonPeer](#)
- [setCRCEnabled\(\)](#) : [Client](#) , [PhotonPeer](#)
- [setCustomRoomProperties\(\)](#) : [RoomOptions](#)
- [setData\(\)](#) : [AuthenticationValues](#)
- [setDebugOutputLevel\(\)](#) : [Client](#) , [Base](#) , [Logger](#) , [Client](#) , [PhotonPeer](#)
- [setDirectMode\(\)](#) : [RoomOptions](#)
- [setDisconnectTimeout\(\)](#) : [Client](#) , [PhotonPeer](#)
- [setElementAt\(\)](#) : [JVector< Etype >](#)
- [setEmptyRoomTtl\(\)](#) : [RoomOptions](#)
- [setEventCaching\(\)](#) : [RaiseEventOptions](#)
- [setFlags\(\)](#) : [WebFlags](#)
- [setFormatOptions\(\)](#) : [Logger](#)
- [setHttpForward\(\)](#) : [WebFlags](#)
- [setInterestGroup\(\)](#) : [RaiseEventOptions](#)

- `setIsOpen()` : **RoomOptions**
- `setIsVisible()` : **RoomOptions**
- `setLimitOfUnreliableCommands()` : **Client** , **PhotonPeer**
- `setListener()` : **Base** , **Logger**
- `setLobbyName()` : **RoomOptions**
- `setLobbyType()` : **RoomOptions**
- `setLogFormatOptions()` : **Client** , **Base** , **Client** , **PhotonPeer**
- `setMaxAllocSize()` : **AllocatorInterface**
- `setMaximumNumberOfNamespaces()` : **LogFormatOptions**
- `setMaxPlayers()` : **RoomOptions**
- `setParameters()` : **AuthenticationValues** , **OperationRequest**
- `setParametersWithUsernameAndToken()` : **AuthenticationValues**
- `setPlayerTtl()` : **RoomOptions**
- `setPlugins()` : **RoomOptions**
- `setPropsListedInLobby()` : **RoomOptions**
- `setPublishUserID()` : **RoomOptions**
- `setQuickResendAttempts()` : **Client** , **PhotonPeer**
- `setReceiverGroup()` : **RaiseEventOptions**
- `setRegion()` : **Client**
- `setSendAuthCookie()` : **WebFlags**
- `setSendState()` : **WebFlags**
- `setSendSync()` : **WebFlags**
- `setSentCountAllowance()` : **Client** , **PhotonPeer**
- `setSuppressRoomEvents()` : **RoomOptions**
- `setTargetPlayers()` : **RaiseEventOptions**
- `setTimePingInterval()` : **Client** , **PhotonPeer**
- `setTrafficStatsEnabled()` : **Client** , **PhotonPeer**
- `setType()` : **AuthenticationValues**
- `setUserID()` : **AuthenticationValues**
- `setWebFlags()` : **RaiseEventOptions**
- `size()` : **ANSIString** , **BaseCharString** , **UTF8String**
- `sizeof()` : **CustomTypeFactory< typeCode >**
- `startsWith()` : **JString**
- `subscribeReturn()` : **Listener**
- `substring()` : **JString**



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- t -

- [toInt\(\)](#) : [JString](#)
- [toLowerCase\(\)](#) : [JString](#)
- [toString\(\)](#) : [AuthenticationValues](#) , [Channel](#) , [BaseCharString](#) , [CustomTypeFactory< typeCode >](#) , [DeSerializer](#) , [Dictionary< EKeyType, EValueType >](#) , [DictionaryBase](#) , [EGTime](#) , [Hashtable](#) , [JString](#) , [JVector< Etype >](#) , [LogFormatOptions](#) , [Logger](#) , [Object](#) , [Serializer](#) , [ToString](#) , [AuthenticationValues](#) , [FriendInfo](#) , [LobbyStatsRequest](#) , [LobbyStatsResponse](#) , [Player](#) , [RaiseEventOptions](#) , [Room](#) , [RoomOptions](#) , [WebFlags](#) , [EventData](#) , [OperationRequest](#) , [OperationResponse](#) , [TrafficStats](#) , [TrafficStatsGameLevel](#)
- [toStringVitalStats\(\)](#) : [TrafficStatsGameLevel](#)
- [toUpperCase\(\)](#) : [JString](#)
- [trim\(\)](#) : [JString](#)
- [trimToSize\(\)](#) : [JVector< Etype >](#)
- [TypeCode](#) : [CustomType< typeCode >](#)
- [typeToString\(\)](#) : [Dictionary< EKeyType, EValueType >](#) , [DictionaryBase](#) , [ToString](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- u -

- unsubscribeReturn() : [Listener](#)
 - UTF8Representation() : [JString](#)
 - UTF8String() : [UTF8String](#)
-

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- v -

- `ValueObject()` : [ValueObject< Etype >](#)
 - `vitalStatsToString()` : [Client](#) , [PhotonPeer](#)
 - `vlog()` : [Logger](#)
-

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- W -

- WebFlags() : [WebFlags](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented class members with links to the class documentation for each member:

- ~ -

- [~AllocatorInterface\(\)](#) : **AllocatorInterface**
 - [~ANSIString\(\)](#) : **ANSIString**
 - [~Base\(\)](#) : **Base**
 - [~BaseCharString\(\)](#) : **BaseCharString**
 - [~Client\(\)](#) : **Client**
 - [~CustomTypeFactory\(\)](#) : **CustomTypeFactory< typeCode >**
 - [~Dictionary\(\)](#) : **Dictionary< EKeyType, EValueType >**
 - [~DictionaryBase\(\)](#) : **DictionaryBase**
 - [~EGTime\(\)](#) : **EGTime**
 - [~EventData\(\)](#) : **EventData**
 - [~Hashtable\(\)](#) : **Hashtable**
 - [~JString\(\)](#) : **JString**
 - [~JVector\(\)](#) : **JVector< Etype >**
 - [~KeyObject\(\)](#) : **KeyObject< Etype >**
 - [~LitePeer\(\)](#) : **LitePeer**
 - [~Object\(\)](#) : **Object**
 - [~OperationRequest\(\)](#) : **OperationRequest**
 - [~OperationResponse\(\)](#) : **OperationResponse**
 - [~PhotonListener\(\)](#) : **PhotonListener**
 - [~PhotonPeer\(\)](#) : **PhotonPeer**
 - [~Player\(\)](#) : **Player**
 - [~RaiseEventOptions\(\)](#) : **RaiseEventOptions**
 - [~Room\(\)](#) : **Room**
 - [~RoomOptions\(\)](#) : **RoomOptions**
 - [~ToString\(\)](#) : **ToString**
 - [~TrafficStats\(\)](#) : **TrafficStats**
 - [~TrafficStatsGameLevel\(\)](#) : **TrafficStatsGameLevel**
 - [~UTF8String\(\)](#) : **UTF8String**
 - [~ValueObject\(\)](#) : **ValueObject< Etype >**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- a -

- `addElement()` : **JVector< Etype >**
- `addElements()` : **JVector< Etype >**
- `alloc()` : **AllocatorInterface**
- `ANSIRepresentation()` : **JString**
- `ANSIString()` : **ANSIString**
- `AuthenticationValues()` : **AuthenticationValues**

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

- b -

- BaseCharString() : **BaseCharString**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

- C -

- `capacity()` : **JString**
- `charAt()` : **JString**
- `cleanup()` : **CustomTypeBase**
- `Client()` : **Client**
- `compare()` : **CustomTypeBase**
- `compareTo()` : **JString**
- `concat()` : **JString**
- `connect()` : **Client** , **PhotonPeer**
- `connectReturn()` : **Listener**
- `constructClass()` : **CustomType< typeCode >**
- `contains()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **Hashtable** , **JVector< Etype >**
- `copy()` : **CustomTypeFactory< typeCode >**
- `copyFactory()` : **CustomTypeFactory< typeCode >**
- `copyInto()` : **JVector< Etype >**
- `create()` : **CustomTypeFactory< typeCode >**
- `cstr()` : **BaseCharString** , **JString**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

- d -

- dealloc() : **AllocatorInterface**
- debugReturn() : **Listener** , **BaseListener** , **Listener**
- deconstructClass() : **CustomType< typeCode >**
- deleteChars() : **JString**
- deserialize() : **CustomTypeBase**
- DeSerializer() : **DeSerializer**
- destroy() : **CustomTypeFactory< typeCode >**
- destroyFactory() : **CustomTypeFactory< typeCode >**
- Dictionary() : **Dictionary< EKeyType, EValueType >**
- DictionaryBase() : **DictionaryBase**
- disconnect() : **Client** , **Peer** , **PhotonPeer**
- disconnectReturn() : **Listener**
- dispatchIncomingCommands() : **Client** , **PhotonPeer**
- duplicate() : **CustomTypeBase**

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

- e -

- EGTime() : **EGTime**
- endsWith() : **JString**
- ensureCapacity() : **JString** , **JVector< Etype >**
- equals() : **JString**
- equalsIgnoreCase() : **JString**
- establishEncryption() : **PhotonPeer**
- eventData() : **EventData**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- f -

- `fetchServerTimestamp()` : [Client](#) , [PhotonPeer](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- g -

- `get()` : **AllocatorInterface**
- `getAddDateTime()` : **LogFormatOptions**
- `getAddFile()` : **LogFormatOptions**
- `getAddFunction()` : **LogFormatOptions**
- `getAddLevel()` : **LogFormatOptions**
- `getAddLine()` : **LogFormatOptions**
- `getAutoJoinLobby()` : **Client**
- `getByteCountCurrentDispatch()` : **Client** , **PhotonPeer**
- `getByteCountLastOperation()` : **Client** , **PhotonPeer**
- `getBytesIn()` : **Client** , **PhotonPeer**
- `getBytesOut()` : **Client** , **PhotonPeer**
- `getCacheSliceIndex()` : **RaiseEventOptions**
- `getCapacity()` : **JVector< Etype >**
- `getCArray()` : **JVector< Etype >**
- `getChannelCountUserChannels()` : **Client** , **PhotonPeer**
- `getChannelID()` : **RaiseEventOptions**
- `getCode()` : **EventData**
- `getConnectionProtocol()` : **PhotonPeer**
- `getControlCommandBytes()` : **TrafficStats**
- `getControlCommandCount()` : **TrafficStats**
- `getCountGamesRunning()` : **Client**
- `getCountPlayersIngame()` : **Client**
- `getCountPlayersOnline()` : **Client**
- `getCRCEnabled()` : **Client** , **PhotonPeer**
- `getCurrentlyJoinedRoom()` : **Client**
- `getCustomProperties()` : **Player** , **Room**
- `getCustomRoomProperties()` : **RoomOptions**
- `getCustomType()` : **Object**
- `getData()` : **AuthenticationValues** , **Serializer** , **AuthenticationValues**
- `getDataAddress()` : **KeyObject< Etype >** , **ValueObject< Etype >**

- `getDataCopy()` : **KeyObject< Etype > , ValueObject< Etype >**
- `getDebugMessage()` : **OperationResponse**
- `getDebugOutputLevel()` : **Client , Base , Logger , Client , PhotonPeer**
- `getDimensions()` : **Object**
- `getDirectMode()` : **Room , RoomOptions**
- `getDisconnectedCause()` : **Client**
- `getDisconnectTimeout()` : **Client , PhotonPeer**
- `getDispatchIncomingCommandsCalls()` : **TrafficStatsGameLevel**
- `getElementAt()` : **JVector< Etype >**
- `getEmptyRoomTtl()` : **RoomOptions**
- `getEventByteCount()` : **TrafficStatsGameLevel**
- `getEventCaching()` : **RaiseEventOptions**
- `getEventCount()` : **TrafficStatsGameLevel**
- `getFirstElement()` : **JVector< Etype >**
- `getFlags()` : **WebFlags**
- `getFormatOptions()` : **Logger**
- `getFragmentCommandBytes()` : **TrafficStats**
- `getFragmentCommandCount()` : **TrafficStats**
- `getFriendList()` : **Client**
- `getFriendListAge()` : **Client**
- `getHashtable()` : **DictionaryBase**
- `getHttpForward()` : **WebFlags**
- `getIncomingReliableCommandsCount()` : **Client , PhotonPeer**
- `getIndexOf()` : **JVector< Etype >**
- `getInterestGroup()` : **RaiseEventOptions**
- `getIsEmpty()` : **JVector< Etype >**
- `getIsEncryptionAvailable()` : **Client , PhotonPeer**
- `getIsInactive()` : **Player**
- `getIsInGameRoom()` : **Client**
- `getIsInLobby()` : **Client**
- `getIsInRoom()` : **Client , FriendInfo**
- `getIsMasterClient()` : **Player**
- `getIsOnline()` : **FriendInfo**
- `getIsOpen()` : **Room , RoomOptions**
- `getIsPayloadEncryptionAvailable()` : **Client , PhotonPeer**
- `getIsVisible()` : **RoomOptions**
- `getKeys()` : **Dictionary< EKeyType, EValueType > , DictionaryBase , Hashtable**
- `getKeyTypes()` : **Dictionary< EKeyType, EValueType > ,**

DictionaryBase

- getLastElement() : **JVector< Etype >**
- getLastIndexOf() : **JVector< Etype >**
- getLimitOfUnreliableCommands() : **Client , PhotonPeer**
- getListener() : **PhotonPeer**
- getLobbyName() : **RoomOptions**
- getLobbyType() : **RoomOptions**
- getLocalPlayer() : **Client**
- getLogFormatOptions() : **Client , Base , Client , PhotonPeer**
- getLongestDeltaBetweenDispatching() : **TrafficStatsGameLevel**
- getLongestDeltaBetweenSending() : **TrafficStatsGameLevel**
- getLongestEventCallback() : **TrafficStatsGameLevel**
- getLongestEventCallbackCode() : **TrafficStatsGameLevel**
- getLongestOpResponseCallback() : **TrafficStatsGameLevel**
- getLongestOpResponseCallbackOpCode() : **TrafficStatsGameLevel**
- getMasterserverAddress() : **Client**
- getMaxAppIDLength() : **PhotonPeer**
- getMaxNumberOfNamespaces() : **LogFormatOptions**
- getMaxPlayers() : **Room , RoomOptions**
- getName() : **LobbyStatsRequest , LobbyStatsResponse , Player , Room**
- getNumber() : **Player**
- getNumTargetPlayers() : **RaiseEventOptions**
- getOperationByteCount() : **TrafficStatsGameLevel**
- getOperationCode() : **OperationRequest , OperationResponse**
- getOperationCount() : **TrafficStatsGameLevel**
- getPackageHeaderSize() : **TrafficStats**
- getPacketLossByCRC() : **Client , PhotonPeer**
- getParameterForCode() : **EventData , OperationRequest , OperationResponse**
- getParameters() : **AuthenticationValues , EventData , OperationRequest , OperationResponse**
- getPeerCount() : **Client , LobbyStatsResponse , PhotonPeer**
- getPeerID() : **Client , PhotonPeer**
- getPeerState() : **PhotonPeer**
- getPlayerCount() : **MutableRoom , Room**
- getPlayerTtl() : **RoomOptions**
- getPlugins() : **RoomOptions**
- getPrivateChannel() : **Client**

- `getPrivateChannels()` : **Client**
- `getPropsListedInLobby()` : **RoomOptions**
- `getPublicChannel()` : **Client**
- `getPublicChannels()` : **Client**
- `getPublishUserID()` : **RoomOptions**
- `getQueuedIncomingCommands()` : **Client** , **PhotonPeer**
- `getQueuedOutgoingCommands()` : **Client** , **PhotonPeer**
- `getQuickResendAttempts()` : **Client** , **PhotonPeer**
- `getReceiverGroup()` : **RaiseEventOptions**
- `getRegion()` : **Client**
- `getRegionWithBestPing()` : **Client**
- `getReliableCommandBytes()` : **TrafficStats**
- `getReliableCommandCount()` : **TrafficStats**
- `getResentReliableCommands()` : **Client** , **PhotonPeer**
- `getResultByteCount()` : **TrafficStatsGameLevel**
- `getResultCount()` : **TrafficStatsGameLevel**
- `getReturnCode()` : **OperationResponse**
- `getRoom()` : **FriendInfo**
- `getRoomCount()` : **LobbyStatsResponse**
- `getRoomList()` : **Client**
- `getRoomNameList()` : **Client**
- `getRoundTripTime()` : **Client** , **PhotonPeer**
- `getRoundTripTimeVariance()` : **Client** , **PhotonPeer**
- `getSecret()` : **AuthenticationValues**
- `getSendAuthCookie()` : **WebFlags**
- `getSendOutgoingCommandsCalls()` : **TrafficStatsGameLevel**
- `getSendState()` : **WebFlags**
- `getSendSync()` : **WebFlags**
- `getSentCountAllowance()` : **Client** , **PhotonPeer**
- `getServerAddress()` : **PhotonPeer**
- `getServerTime()` : **Client** , **PhotonPeer**
- `getServerTimeOffset()` : **Client** , **PhotonPeer**
- `getSize()` : **DictionaryBase** , **Hashtable** , **JVector< Etype >** , **Serializer**
- `getSizes()` : **Object**
- `getState()` : **Client**
- `getSuppressRoomEvents()` : **RoomOptions**
- `getTargetPlayers()` : **RaiseEventOptions**
- `getTimePingInterval()` : **Client** , **PhotonPeer**
- `getTimestampOfLastAck()` : **TrafficStats**

- `getTimestampOfLastReliableCommand()` : **TrafficStats**
- `getTimestampOfLastSocketReceive()` : **Client** , **PhotonPeer**
- `getTotalByteCount()` : **TrafficStatsGameLevel**
- `getTotalCommandBytes()` : **TrafficStats**
- `getTotalCommandCount()` : **TrafficStats**
- `getTotalCommandsInPackets()` : **TrafficStats**
- `getTotalIncomingByteCount()` : **TrafficStatsGameLevel**
- `getTotalIncomingMessageCount()` : **TrafficStatsGameLevel**
- `getTotalMessageCount()` : **TrafficStatsGameLevel**
- `getTotalOutgoingByteCount()` : **TrafficStatsGameLevel**
- `getTotalOutgoingMessageCount()` : **TrafficStatsGameLevel**
- `getTotalPacketBytes()` : **TrafficStats**
- `getTotalPacketCount()` : **TrafficStats**
- `getTrafficStatsElapsedMs()` : **Client** , **PhotonPeer**
- `getTrafficStatsEnabled()` : **Client** , **PhotonPeer**
- `getTrafficStatsGameLevel()` : **Client** , **PhotonPeer**
- `getTrafficStatsIncoming()` : **Client** , **PhotonPeer**
- `getTrafficStatsOutgoing()` : **Client** , **PhotonPeer**
- `getType()` : **AuthenticationValues** , **Object** , **AuthenticationValues** , **LobbyStatsRequest** , **LobbyStatsResponse**
- `getUnreliableCommandBytes()` : **TrafficStats**
- `getUnreliableCommandCount()` : **TrafficStats**
- `getUserID()` : **AuthenticationValues** , **Client** , **AuthenticationValues** , **Client** , **FriendInfo** , **Player**
- `getValue()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **Hashtable**
- `getValueDimensions()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase**
- `getValueSizes()` : **DictionaryBase**
- `getValueTypes()` : **Dictionary< EKeyType, EValueType >** , **DictionaryBase**
- `getWebFlags()` : **RaiseEventOptions**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

- h -

- Hashtable() : **Hashtable**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- i -

- `indexOf()` : **JString**
 - `initUDPEncryption()` : **PhotonPeer**
 - `initUserDataEncryption()` : **PhotonPeer**
 - `insertElementAt()` : **JVector< Etype >**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

- j -

- `JString()` : **JString**
 - `JStringRepresentation()` : **ANSIString** , **BaseCharString** , **UTF8String**
 - `JVector()` : **JVector< Etype >**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

- k -

- KeyObject() : **KeyObject< Etype >**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- | -

- `lastIndexOf()` : **JString**
- `length()` : **BaseCharString** , **JString**
- `LitePeer()` : **LitePeer**
- `LobbyStatsRequest()` : **LobbyStatsRequest**
- `log()` : **Logger**
- `Logger()` : **Logger**

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- 0 -

- Object() : **Object**
- onEvent() : **PhotonListener**
- onGetMessages() : **Listener**
- onOperationResponse() : **PhotonListener**
- onPingResponse() : **PhotonListener**
- onPrivateMessage() : **Listener**
- onStateChange() : **Listener**
- onStatusChanged() : **PhotonListener**
- onStatusUpdate() : **Listener**
- opAddFriends() : **Client**
- opChangeGroups() : **LitePeer** , **Client**
- opCreateRoom() : **Client**
- opCustom() : **Client** , **PhotonPeer**
- opCustomAuthenticationSendNextStepData() : **Client**
- OperationRequest() : **OperationRequest**
- OperationResponse() : **OperationResponse**
- operator const char *() : **ANSIString** , **BaseCharString** , **UTF8String**
- operator const EG_CHAR *() : **JString**
- operator JString() : **ANSIString** , **BaseCharString** , **UTF8String**
- operator!==() : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **EGTime** , **Hashtable** , **JString** , **JVector< Etype >** , **Object**
- operator+() : **EGTime** , **JString**
- operator+=() : **EGTime** , **JString**
- operator-() : **EGTime**
- operator-=() : **EGTime**
- operator<() : **EGTime** , **JString**
- operator<<() : **JString**
- operator<=() : **EGTime** , **JString**
- operator==() : **ANSIString** , **Dictionary< EKeyType, EValueType >** ,

- DictionaryBase , EGTime , Hashtable , JString , JVector< Etype > , KeyObject< Etype > , Object , UTF8String , ValueObject< Etype > , MutablePlayer , MutableRoom , Player , RaiseEventOptions , Room , RoomOptions ,EventData , OperationRequest , OperationResponse**
- **operator==() : Dictionary< EKeyType, EValueType > , DictionaryBase , EGTime , Hashtable , JString , JVector< Etype > , Object , Player , Room**
 - **operator>() : EGTime , JString**
 - **operator>=() : EGTime , JString**
 - **operator[]() : Dictionary< EKeyType, EValueType > , Hashtable , JString , JVector< Etype > , EventData , OperationRequest , OperationResponse**
 - **opFindFriends() : Client**
 - **opGetProperties() : LitePeer**
 - **opGetPropertiesOfActor() : LitePeer**
 - **opGetPropertiesOfGame() : LitePeer**
 - **opJoin() : LitePeer**
 - **opJoinLobby() : Client**
 - **opJoinOrCreateRoom() : Client**
 - **opJoinRandomRoom() : Client**
 - **opJoinRoom() : Client**
 - **opLeave() : LitePeer**
 - **opLeaveLobby() : Client**
 - **opLeaveRoom() : Client**
 - **opLobbyStats() : Client**
 - **opPublishMessage() : Client**
 - **opRaiseEvent() : LitePeer , Client**
 - **opRemoveFriends() : Client**
 - **opSendPrivateMessage() : Client**
 - **opSetOnlineStatus() : Client**
 - **opSetPropertiesOfActor() : LitePeer**
 - **opSetPropertiesOfGame() : LitePeer**
 - **opSubscribe() : Client**
 - **opUnsubscribe() : Client**
 - **opWebRpc() : Client**
 - **overflowed() : EGTime**

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- p -

- PhotonPeer() : **PhotonPeer**
- pingServer() : **PhotonPeer**
- Player() : **Player**
- pop() : **DeSerializer**
- push() : **Serializer**
- put() : **Dictionary< EKeyType, EValueType > , Hashtable**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

- r -

- RaiseEventOptions() : **RaiseEventOptions**
- reconnectAndRejoin() : **Client**
- remove() : **Dictionary< EKeyType, EValueType > , DictionaryBase , Hashtable**
- removeAllElements() : **DictionaryBase , Hashtable , JVector< Etype >**
- removeElement() : **JVector< Etype >**
- removeElementAt() : **JVector< Etype >**
- replace() : **JString**
- resetMaximumCounters() : **TrafficStatsGameLevel**
- resetTrafficStats() : **Client , PhotonPeer**
- resetTrafficStatsMaximumCounters() : **Client , PhotonPeer**
- resize() : **AllocatorInterface**
- Room() : **Room**
- RoomOptions() : **RoomOptions**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- S -

- selectRegion() : **Client**
- sendAcksOnly() : **Client** , **PhotonPeer**
- sendDirect() : **Client**
- sendOutgoingCommands() : **Client** , **PhotonPeer**
- serialize() : **CustomTypeBase**
- service() : **Client** , **PhotonPeer**
- serviceBasic() : **Client** , **PhotonPeer**
- setAddDateTime() : **LogFormatOptions**
- setAddFile() : **LogFormatOptions**
- setAddFunction() : **LogFormatOptions**
- setAddLevel() : **LogFormatOptions**
- setAddLine() : **LogFormatOptions**
- setAutoJoinLobby() : **Client**
- setCacheSliceIndex() : **RaiseEventOptions**
- setChannelID() : **RaiseEventOptions**
- setConnectionProtocol() : **PhotonPeer**
- setCRCEnabled() : **Client** , **PhotonPeer**
- setCustomRoomProperties() : **RoomOptions**
- setData() : **AuthenticationValues**
- setDebugOutputLevel() : **Client** , **Base** , **Logger** , **Client** , **PhotonPeer**
- setDirectMode() : **RoomOptions**
- setDisconnectTimeout() : **Client** , **PhotonPeer**
- setElementAt() : **JVector< Etype >**
- setEmptyRoomTtl() : **RoomOptions**
- setEventCaching() : **RaiseEventOptions**
- setFlags() : **WebFlags**
- setFormatOptions() : **Logger**
- setHttpForward() : **WebFlags**
- setInterestGroup() : **RaiseEventOptions**
- setIsOpen() : **RoomOptions**

- `setIsVisible()` : **RoomOptions**
- `setLimitOfUnreliableCommands()` : **Client** , **PhotonPeer**
- `setListener()` : **Base** , **Logger**
- `setLobbyName()` : **RoomOptions**
- `setLobbyType()` : **RoomOptions**
- `setLogFormatOptions()` : **Client** , **Base** , **Client** , **PhotonPeer**
- `setMaxAllocSize()` : **AllocatorInterface**
- `setMaxNumberOfNamespaces()` : **LogFormatOptions**
- `setMaxPlayers()` : **RoomOptions**
- `setParameters()` : **AuthenticationValues** , **OperationRequest**
- `setParametersWithUsernameAndToken()` : **AuthenticationValues**
- `setPlayerTtl()` : **RoomOptions**
- `setPlugins()` : **RoomOptions**
- `setPropsListedInLobby()` : **RoomOptions**
- `setPublishUserID()` : **RoomOptions**
- `setQuickResendAttempts()` : **Client** , **PhotonPeer**
- `setReceiverGroup()` : **RaiseEventOptions**
- `setRegion()` : **Client**
- `setSendAuthCookie()` : **WebFlags**
- `setSendState()` : **WebFlags**
- `setSendSync()` : **WebFlags**
- `setSentCountAllowance()` : **Client** , **PhotonPeer**
- `setSuppressRoomEvents()` : **RoomOptions**
- `setTargetPlayers()` : **RaiseEventOptions**
- `setTimePingInterval()` : **Client** , **PhotonPeer**
- `setTrafficStatsEnabled()` : **Client** , **PhotonPeer**
- `setType()` : **AuthenticationValues**
- `setUserID()` : **AuthenticationValues**
- `setWebFlags()` : **RaiseEventOptions**
- `size()` : **ANSIString** , **BaseCharString** , **UTF8String**
- `sizeof()` : **CustomTypeFactory< typeCode >**
- `startsWith()` : **JString**
- `subscribeReturn()` : **Listener**
- `substring()` : **JString**

- t -

- toInt() : **JString**
- toLowerCase() : **JString**
- toString() : **AuthenticationValues** , **Channel** , **BaseCharString** , **CustomTypeFactory< typeCode >** , **DeSerializer** , **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **EGTime** , **Hashtable** , **JString** , **JVector< Etype >** , **LogFormatOptions** , **Logger** , **Object** , **Serializer** , **ToString** , **AuthenticationValues** , **FriendInfo** , **LobbyStatsRequest** , **LobbyStatsResponse** , **Player** , **RaiseEventOptions** , **Room** , **RoomOptions** , **WebFlags** , **EventData** , **OperationRequest** , **OperationResponse** , **TrafficStats** , **TrafficStatsGameLevel**
- toStringVitalStats() : **TrafficStatsGameLevel**
- toUpperCase() : **JString**
- trim() : **JString**
- trimToSize() : **JVector< Etype >**
- typeToString() : **Dictionary< EKeyType, EValueType >** , **DictionaryBase** , **ToString**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- u -

- unsubscribeReturn() : **Listener**
 - UTF8Representation() : **JString**
 - UTF8String() : **UTF8String**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- v -

- ValueObject() : **ValueObject< Etype >**
 - vitalStatsToString() : **Client** , **PhotonPeer**
 - vlog() : **Logger**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

- W -

- `WebFlags()` : [WebFlags](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- ~ -

- ~AllocatorInterface() : **AllocatorInterface**
 - ~ANSIString() : **ANSIString**
 - ~Base() : **Base**
 - ~BaseCharString() : **BaseCharString**
 - ~Client() : **Client**
 - ~CustomTypeFactory() : **CustomTypeFactory< typeCode >**
 - ~Dictionary() : **Dictionary< EKeyType, EValueType >**
 - ~DictionaryBase() : **DictionaryBase**
 - ~EGTime() : **EGTime**
 - ~EventData() : **EventData**
 - ~Hashtable() : **Hashtable**
 - ~JString() : **JString**
 - ~JVector() : **JVector< Etype >**
 - ~KeyObject() : **KeyObject< Etype >**
 - ~LitePeer() : **LitePeer**
 - ~Object() : **Object**
 - ~OperationRequest() : **OperationRequest**
 - ~OperationResponse() : **OperationResponse**
 - ~PhotonListener() : **PhotonListener**
 - ~PhotonPeer() : **PhotonPeer**
 - ~Player() : **Player**
 - ~RaiseEventOptions() : **RaiseEventOptions**
 - ~Room() : **Room**
 - ~RoomOptions() : **RoomOptions**
 - ~ToString() : **ToString**
 - ~TrafficStats() : **TrafficStats**
 - ~TrafficStatsGameLevel() : **TrafficStatsGameLevel**
 - ~UTF8String() : **UTF8String**
 - ~ValueObject() : **ValueObject< Etype >**
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- TypeCode : [CustomType< typeCode >](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

File List

Here is a list of all documented files with brief descriptions:

Allocate.h	
AllocatorInterface.h	
ANSIString.h	
LoadBalancing-cpp/inc/AuthenticationValues.h	
Chat-cpp/inc/AuthenticationValues.h	
Base.h	
BaseCharString.h	
BaseListener.h	
Channel.h	
LoadBalancing-cpp/inc/Client.h	
Chat-cpp/inc/Client.h	
ClientState.h	
Common.h	
ConnectionProtocol.h	
LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h	
Chat-cpp/inc/Enums/CustomAuthenticationType.h	
CustomType.h	
CustomTypeBase.h	
CustomTypeFactory.h	
DebugLevel.h	
DeSerializer.h	
Dictionary.h	
DictionaryBase.h	
DirectMode.h	
LoadBalancing-cpp/inc/Enums/DisconnectCause.h	

Chat-cpp/inc/Enums/DisconnectCause.h
EGTime.h
Photon-cpp/inc/Enums/ErrorCode.h
LoadBalancing-cpp/inc/Enums/ErrorCode.h
Chat-cpp/inc/Enums/ErrorCode.h
EventCache.h
Photon-cpp/inc/Enums/EventCode.h
EventData.h
EventKey.h
FriendInfo.h
Hashtable.h
IsPrimitiveType.h
JString.h
JVector.h
KeyObject.h
LoadBalancing-cpp/inc/Listener.h
Chat-cpp/inc/Listener.h
LitePeer.h
LobbyStatsRequest.h
LobbyStatsResponse.h
LobbyType.h
Logger.h
MatchmakingMode.h
MutablePlayer.h
MutableRoom.h
NetworkPort.h
Object.h
Photon-cpp/inc/Enums/OperationCode.h
OperationRequest.h
OperationResponse.h
Photon-cpp/inc/Enums/ParameterCode.h
LoadBalancing-cpp/inc/Peer.h
Chat-cpp/inc/Peer.h

PeerState.h
PeerStates.h
PhotonListener.h
PhotonPeer.h
Player.h
RaiseEventOptions.h
ReceiverGroup.h
RegionSelectionMode.h
Room.h
RoomOptions.h
Serializer.h
ServerType.h
StatusCode.h
ToString.h
TrafficStats.h
TrafficStatsGameLevel.h
TypeCode.h
UserStatus.h
UTF8String.h
ValueObject.h
WebFlags.h

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

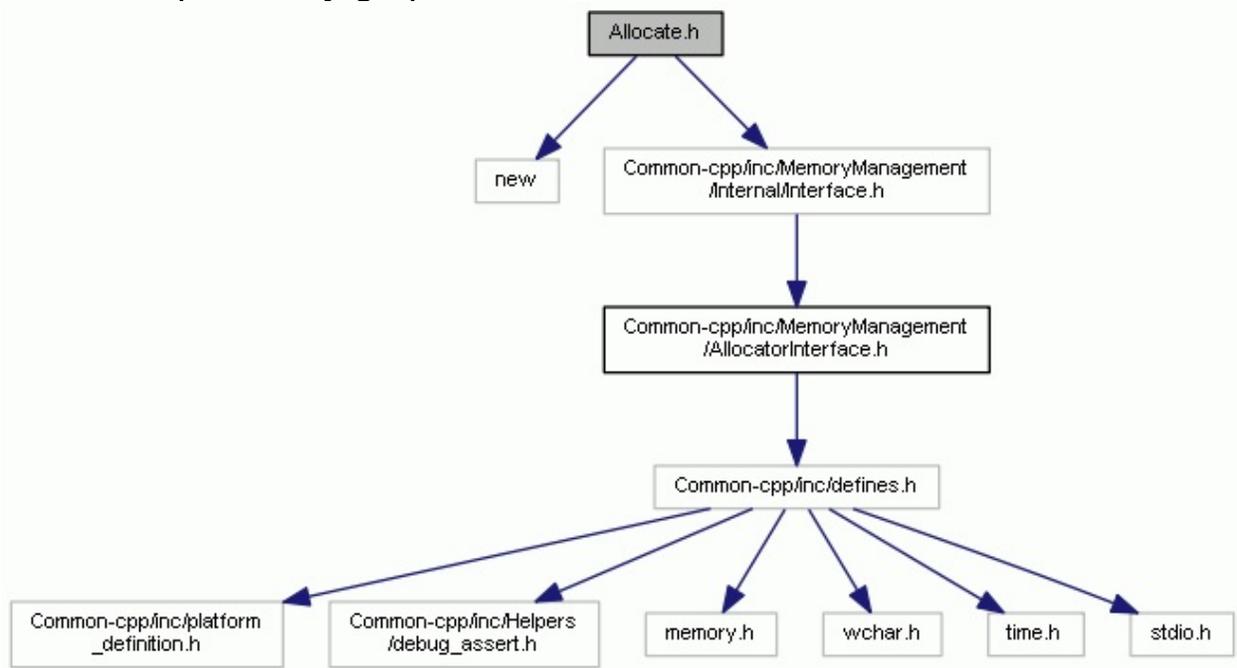
Client API 4.1.12.2

Common-cpp > inc > MemoryManagement

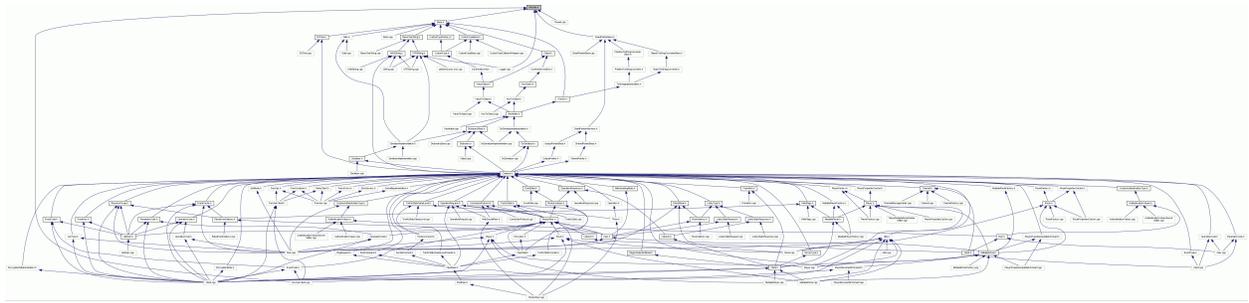
Namespaces | Macros

Allocate.h File Reference

Include dependency graph for Allocate.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Common

ExitGames::Common::MemoryManagement

Macros

```
#define EG_SIZE_T
```

LowLevelMemoryManagement

The macros in this section are an alternative for the C dynamic memory management functions `malloc()`, `free()`, `realloc()` and `calloc()`. They only work in C++, not in C, but same as the standard c library functions they don't call constructors/destructors, but only (de-)allocate the raw memory. This might be desired for high performance container implementations: it makes it possible to allocate storage for many elements at once, while still giving the option to wait with the construction until an element is used.

However in most scenarios it makes more sense to use the **High Level memory management functions**

The Memory Management API is optimized for frequent small-sized general purpose dynamic allocations (all dynamic memory allocations that are smaller than several megabytes per allocation and that can happen inside the main loop). The concrete implementation may vary between platforms and configurations, but will usually be a lot faster for this purpose than the standard C functions `malloc()`, `free()`, `realloc()` and `calloc()` and the standard C++ operators `new`, `new[]`, `delete` and `delete[]`.

```
#define EG_MALLOC
```

```
#define EG_FREE
```

```
#define EG_REALLOC
```

```
#define EG_CALLOC
```

HighLevelMemoryManagement

The template functions in this section are an alternative for the C++ dynamic memory management operators `new`, `new[]`, `delete` and `delete[]`.

They are implemented in terms of enhancing the **Low Level Memory Management macros** and for this reason offer similar advantages over `new` and `co` like those macros offer over `malloc` and `co`.

However same as `new` and `co` they also construct and destruct the object that they allocate and deallocate.

```
#define ALLOCATE(type, p, ...)
```

```
#define ALLOCATE_ARRAY(type, p, count, ...)
```

```
#define REALLOCATE_ARRAY(type, p, count, ...)
```

```
#define DEALLOCATE(type, p)
```

```
#define DEALLOCATE_ARRAY(type, p)
```

```
void setMaxAllocSize (size_t maxAllocSize)
```

```
void setMaxSizeForAllocatorUsage (size_t maxSizeForAllocatorUs
```

```
void setAllocator  
  (ExitGames::Common::MemoryManagement::AllocatorInter  
   &allocator)
```

```
void setAllocatorToDefault (void)
```

```
template<typename Ftype >  
Ftype * allocate (void)
```

```
template<typename Ftype >
```

```
Ftype * allocateArray (size_t count)
```

```
template<typename Ftype >
```

```
Ftype * reallocArray (Ftype *p, size_t count)
```

```
template<typename Ftype >
```

```
void deallocate (const Ftype *p)
```

```
template<typename Ftype >
```

```
void deallocateArray (const Ftype *p)
```

Macro Definition Documentation

§ EG_MALLOC

```
#define EG_MALLOC
```

This macro allocates the requested amount of bytes as a single continuous block from dynamic memory and returns the address of the first byte of that block.

Blocks of memory that have been allocated with **EG_MALLOC()**, have to be deallocated with **EG_FREE()**, when they are no longer needed.

If the requested amount of bytes is 0, then this macro will do nothing and return a NULL pointer.

§ EG_FREE

```
#define EG_FREE
```

Pass the address of memory, that has previously been returned by **EG_MALLOC()**, **EG_REALLOC()** or **EG_CALLOC()** to this function, to deallocate it.

If the passed address is NULL, then this macro will do nothing.

If a passed non-NULL address was not previously returned by **EG_MALLOC()**, **EG_REALLOC()** or **EG_CALLOC()**, then the behavior is undefined.

§ EG_REALLOC

```
#define EG_REALLOC
```

This macro resizes the block of memory at the passed address to the passed size and returns the new address of this block of memory.

The returned address isn't guaranteed to match the passed one. Depending on the old and new size of the memory block, resizing the block may include moving it to a new location. When a block gets moved, is an implementation detail, that could be different between implementations on different platforms and can change without notice. Notably block-movements might happen in the case of an increase as well as of a decrease of the block size.

If a block of memory gets moved to a new location, then the content of all bytes that fit in both, the old and the new block size, is copied from the old to the new location by a call to `memcpy()`. For this reason calls to **EG_REALLOC()** can be expensive for huge blocks of memory.

If the new block size is smaller than the old one, then all content at the surplus bytes will get lost.

If the passed address is `NULL`, then this macro will behave just like **EG_MALLOC()**.

If a passed non-`NULL` address was not previously returned by **EG_MALLOC()**, **EG_REALLOC()** or **EG_CALLOC()**, then the behavior is undefined.

§ EG_CALLOC

```
#define EG_CALLOC
```

This macro allocates memory for the requested amount of array elements of the specified element size as a single continuous block from dynamic memory, initializes all its bytes to 0 and returns the address of the first byte of that block.

Blocks of memory that have been allocated with **EG_CALLOC()**, have to be deallocated with **EG_FREE()**, when they are no longer needed.

If the requested amount of bytes is 0, then this macro will do nothing and return a NULL pointer.

§ ALLOCATE

```
#define ALLOCATE ( type,  
                  p,  
                  ...  
                )
```

This is the macro version of the `allocate()` template function.

Normally the template version should be preferred, but using the macro instead can be needed, if you want to pass more than 10 parameters to the constructor or if you want to call a private or protected constructor to which your class has (friend-/subclass-)access.

Parameters

- type** the data type of the instance to create
- p** a pointer, in which the macro will store the address of the freshly created instance
- ...** optional arguments to pass to the constructor

§ ALLOCATE_ARRAY

```
#define ALLOCATE_ARRAY ( type,  
                        p,  
                        count,  
                        ...  
                        )
```

This is the macro version of the `allocateArray()` template function.

Normally the template version should be preferred, but using the macro instead can be needed, if you want to pass more than 10 parameters to the constructor or if you want to call a private or protected constructor to which your class has (friend-/subclass-)access.

Parameters

- type** the data type of the instance to create
- p** a pointer, in which the macro will store the address of the freshly created instance
- count** the number of the elements to create
- ...** optional arguments to pass to the constructor

§ REALLOCATE_ARRAY

```
#define REALLOCATE_ARRAY ( type,  
                           p,  
                           count,  
                           ...  
                           )
```

This is the macro version of the `reallocateArray()` template function.

Normally the template version should be preferred, but using the macro instead can be needed, if you want to pass more than 10 parameters to the constructor or if you want to call a private or protected constructor to which your class has (friend-/subclass-)access.

Parameters

- type** the data type of the instance to create
- p** a pointer, in which the macro will store the address of the freshly created instance
- count** the number of the elements to create
- ...** optional arguments to pass to the constructor

§ DEALLOCATE

```
#define DEALLOCATE ( type,  
                    p  
                    )
```

This is the macro version of the `deallocate()` template function.

Normally the template version should be preferred, but using the macro instead can make sense for consistency reasons when the macro version has been used for allocation.

Parameters

- type** the data type of the instance, to which `p` points
- p** a pointer to the instance to destroy

§ DEALLOCATE_ARRAY

```
#define DEALLOCATE_ARRAY ( type,  
                           p  
                           )
```

This is the macro version of the [deallocateArray\(\)](#) template function.

Normally the template version should be preferred, but using the macro instead can make sense for consistency reasons when the macro version has been used for allocation.

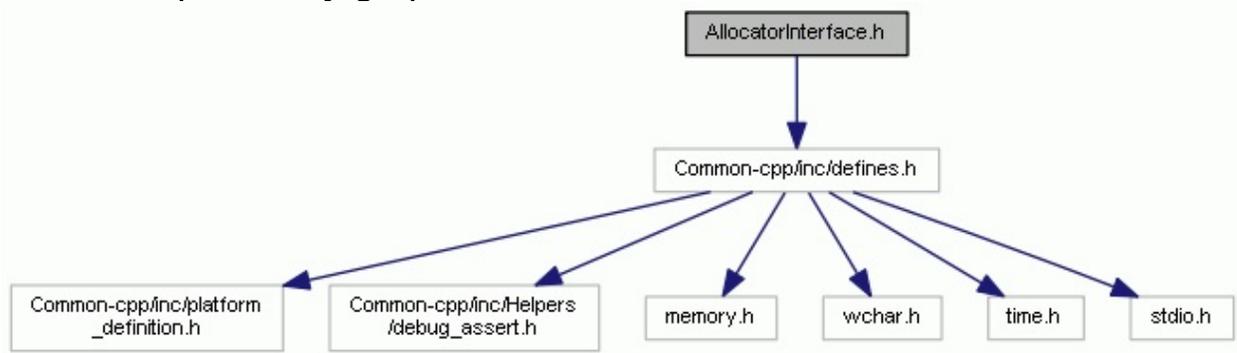
Parameters

- type** the data type of the instance, to which p points
- p** a pointer to the instance to destroy

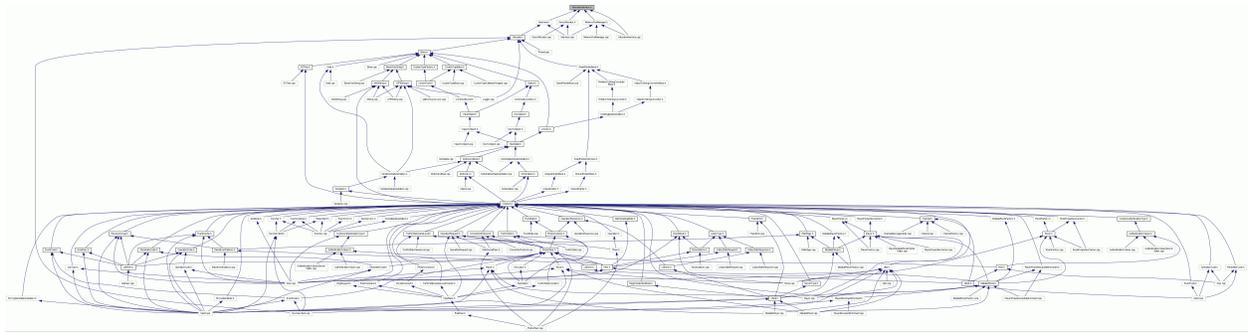
AllocatorInterface.h

File Reference

Include dependency graph for AllocatorInterface.h:



This graph shows which files directly or indirectly include this file:



Classes

class **AllocatorInterface**

Namespaces

ExitGames

ExitGames::Common

ExitGames::Common::MemoryManagement

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

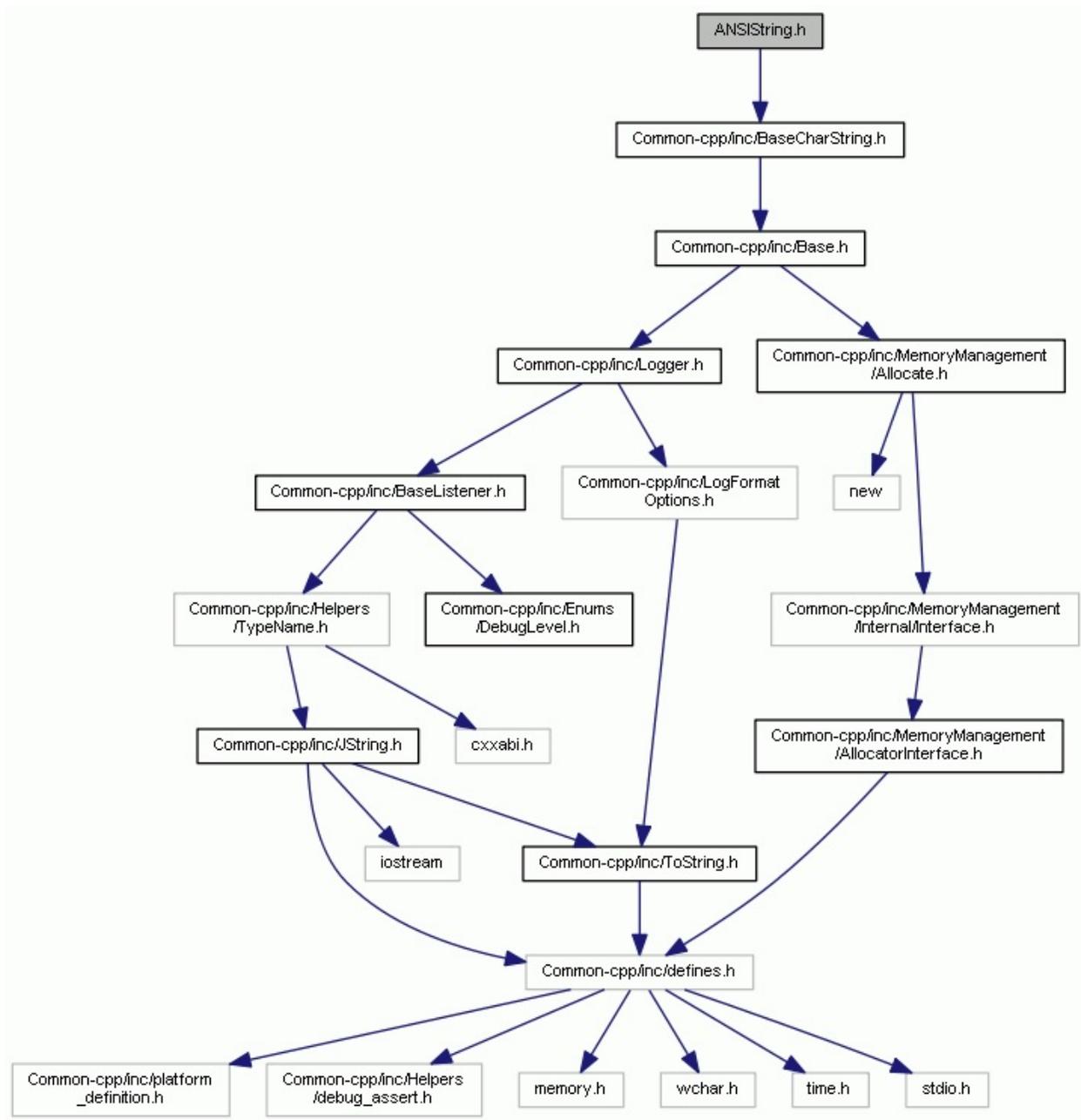
Client API 4.1.12.2

Common-cpp > inc >

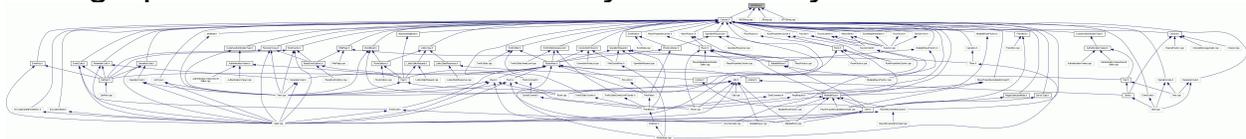
[Classes](#) | [Namespaces](#)

ANSIString.h File Reference

Include dependency graph for ANSIString.h:



This graph shows which files directly or indirectly include this file:



Classes

class **ANSIString**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

LoadBalancing-cpp/inc/AuthenticationValues.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/AuthenticationValues.h:

Classes

class **AuthenticationValues**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

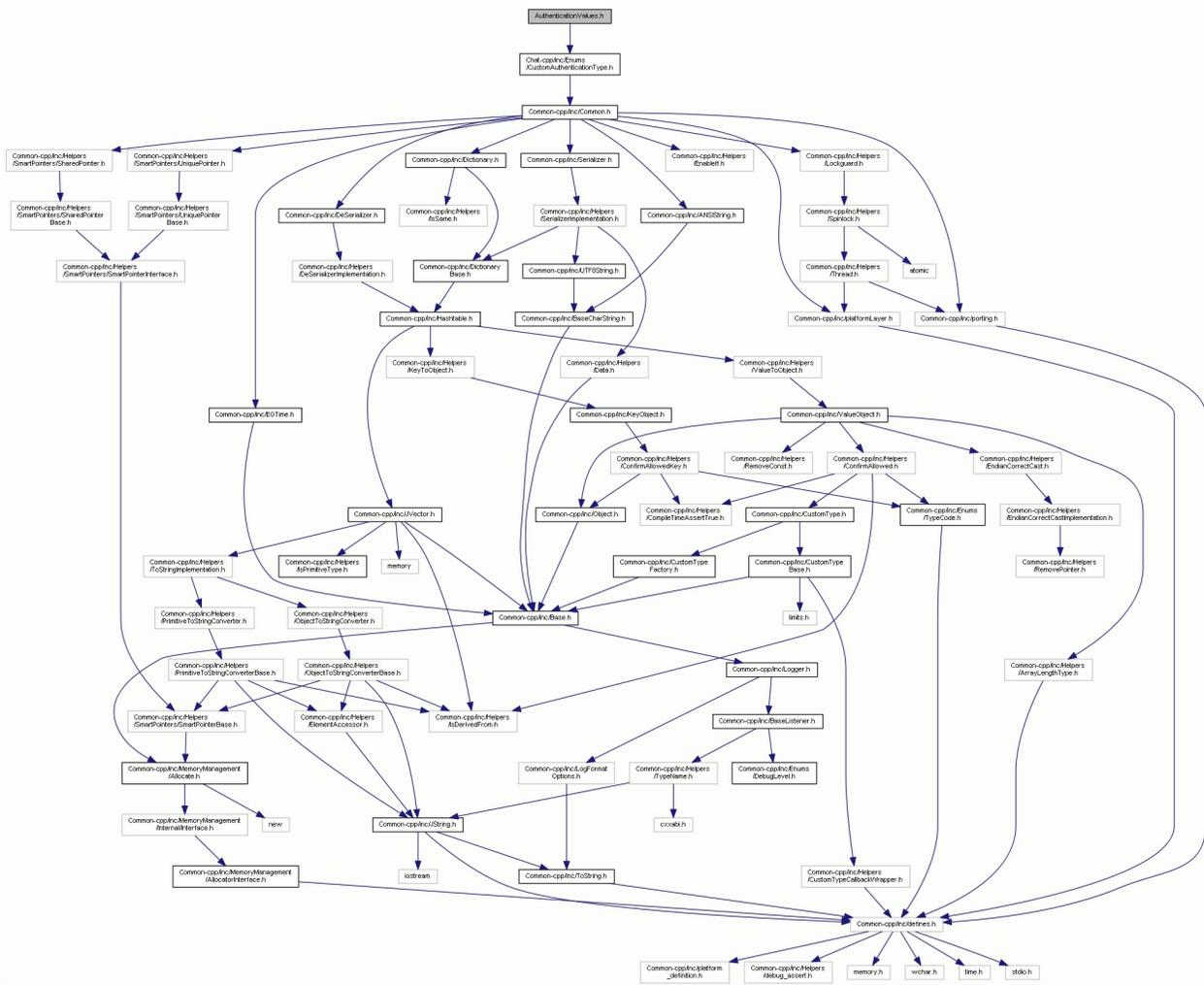
Client API 4.1.12.2

Chat-cpp > inc >

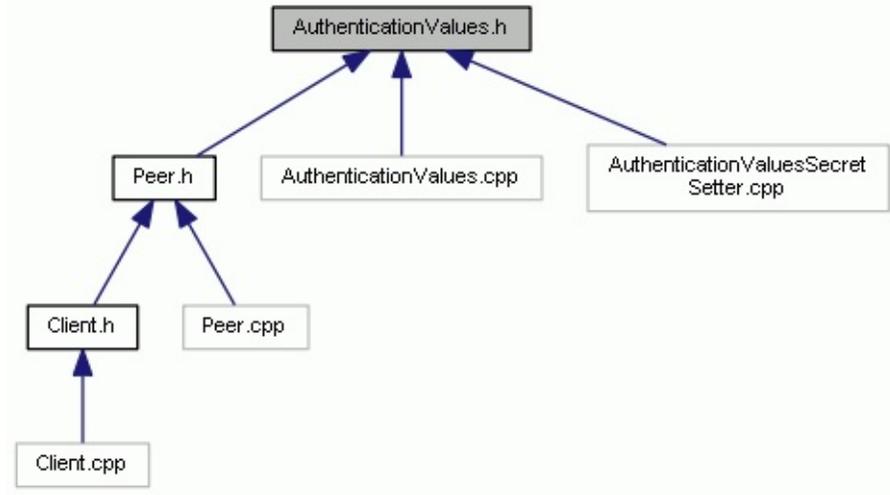
[Classes](#) | [Namespaces](#)

Chat-cpp/inc/AuthenticationValues.h File Reference

Include dependency graph for Chat-cpp/inc/AuthenticationValues.h:



This graph shows which files directly or indirectly include this file:



Classes

class **AuthenticationValues**

Namespaces

ExitGames

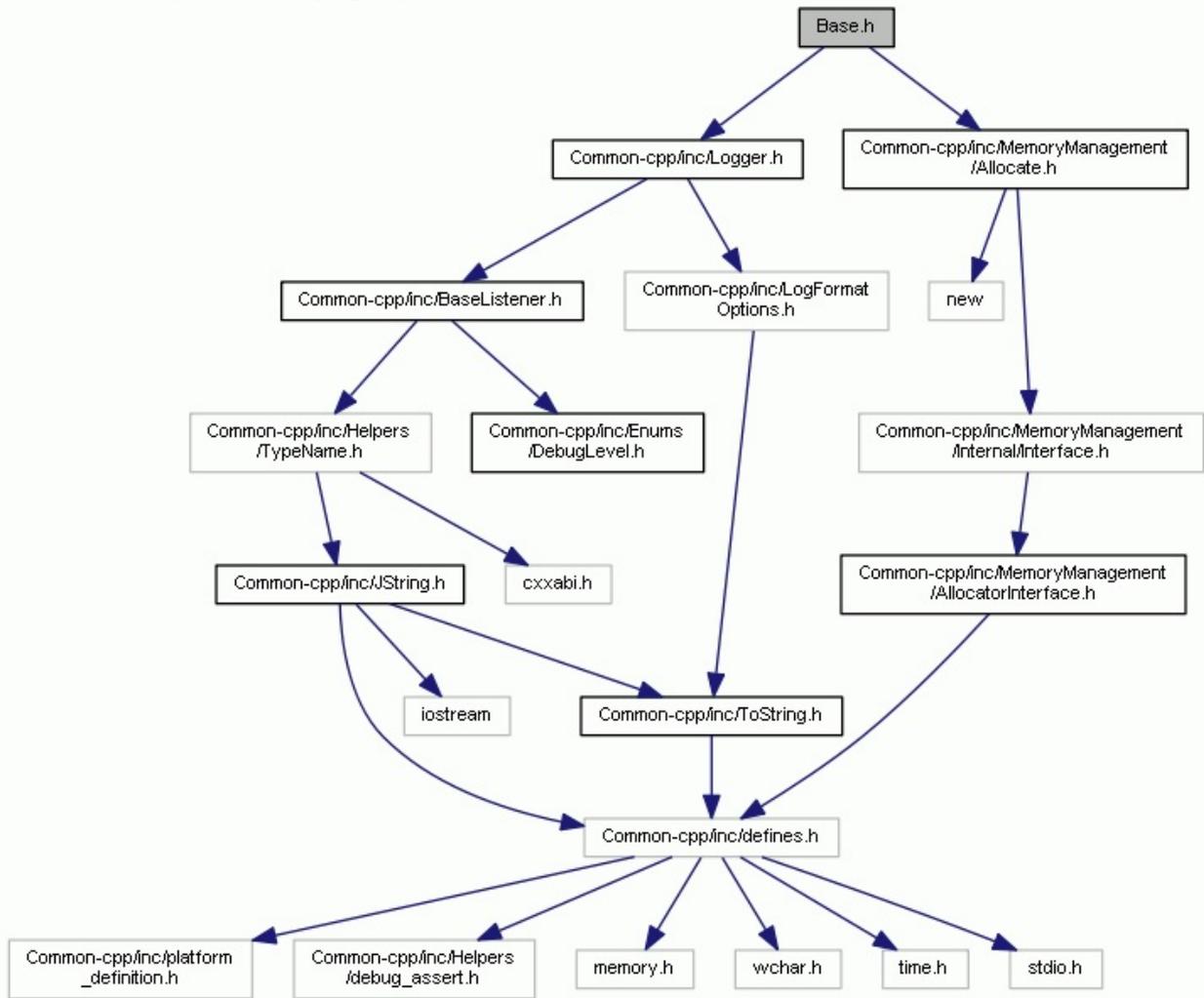
ExitGames::Chat

© Exit Games ®, all rights reserved.

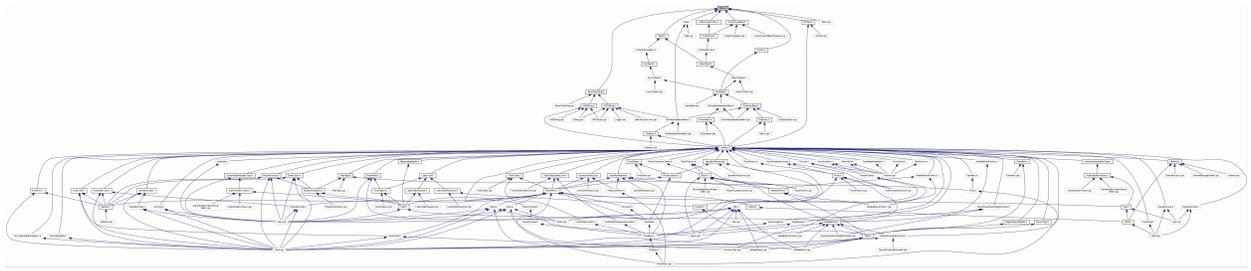
[Photon Documentation](#) | [Contact](#) | [Terms](#)

Base.h File Reference

Include dependency graph for Base.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Base**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



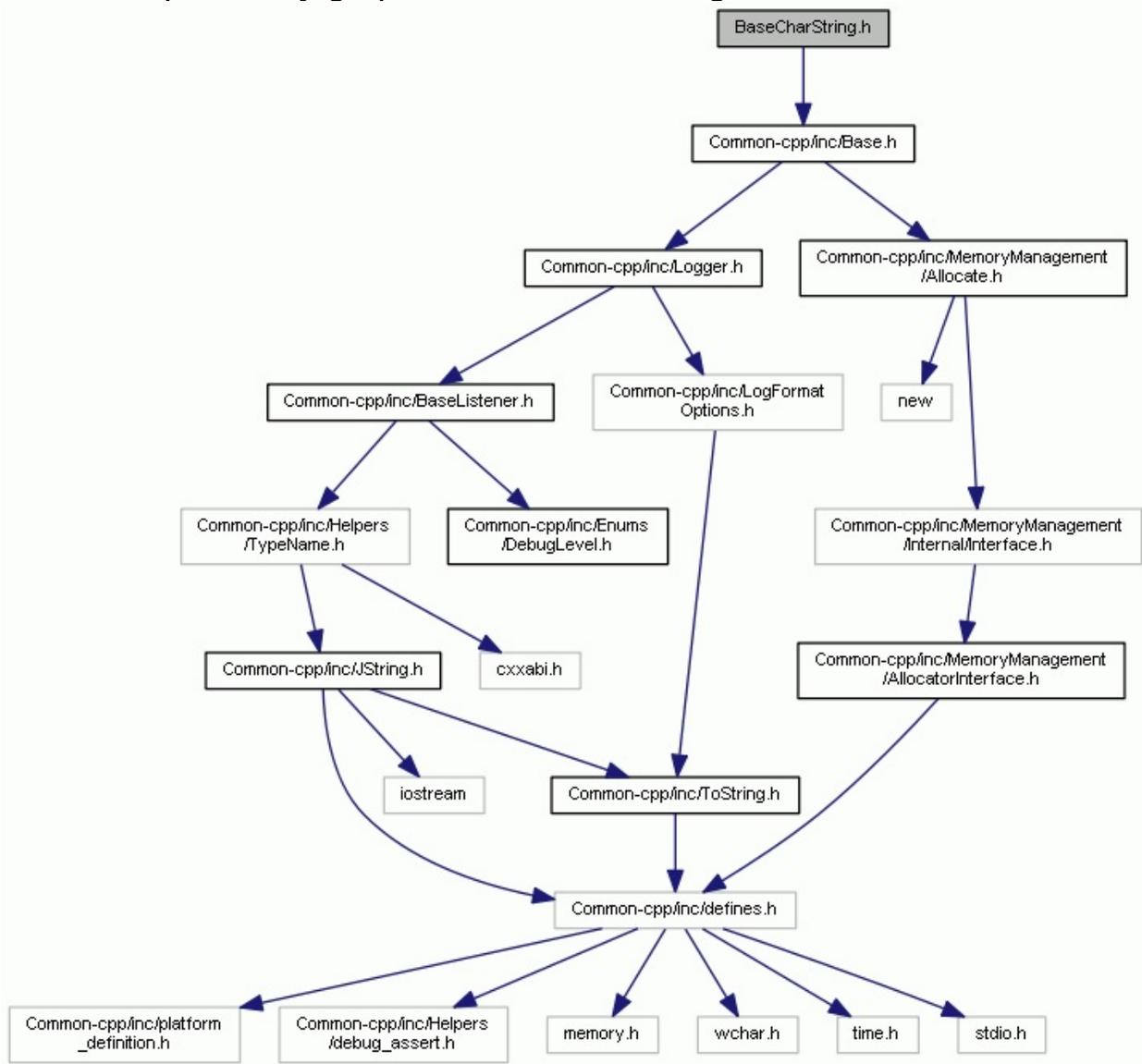
Photon C++ Client API 4.1.12.2

Common-cpp > inc >

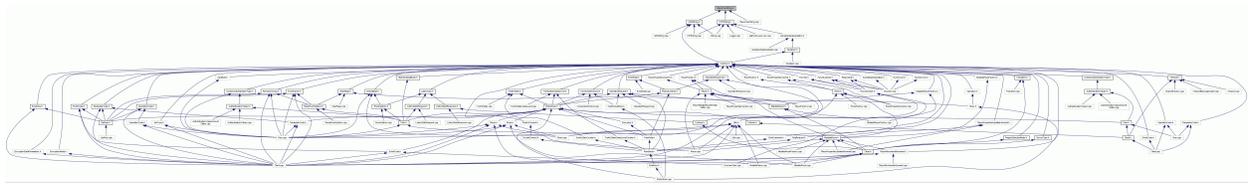
Classes | Namespaces

BaseCharString.h File Reference

Include dependency graph for BaseCharString.h:



This graph shows which files directly or indirectly include this file:



Classes

class **BaseCharString**

Namespaces

ExitGames

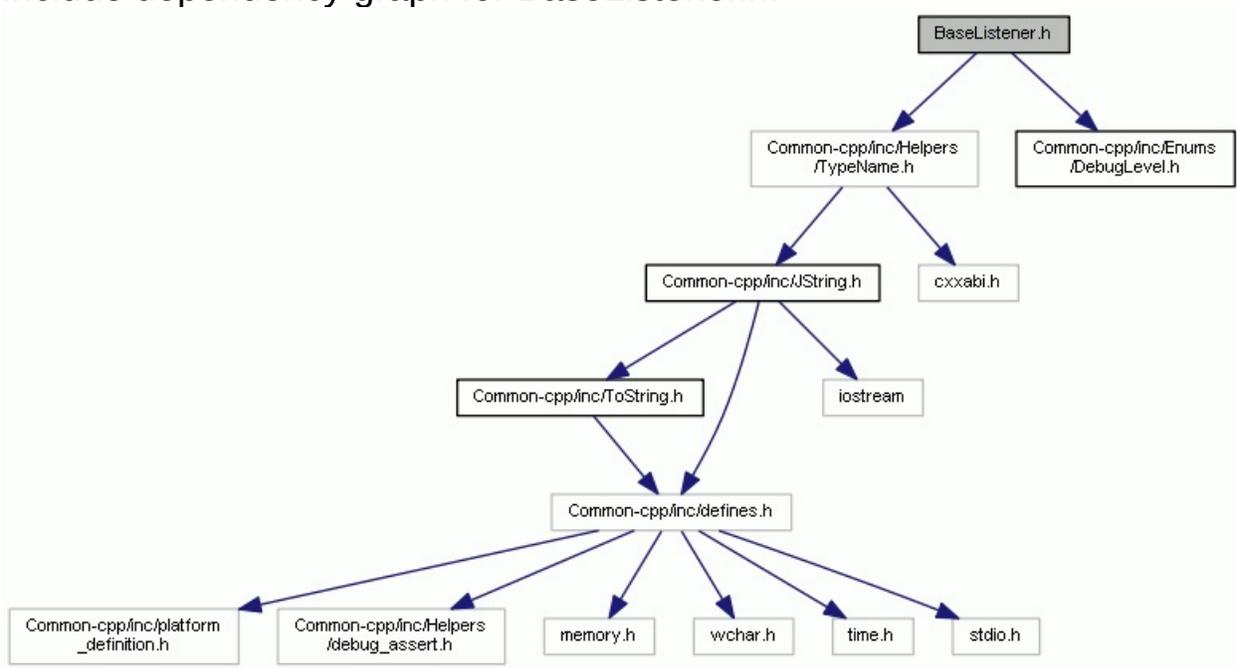
ExitGames::Common

© Exit Games ®, all rights reserved.

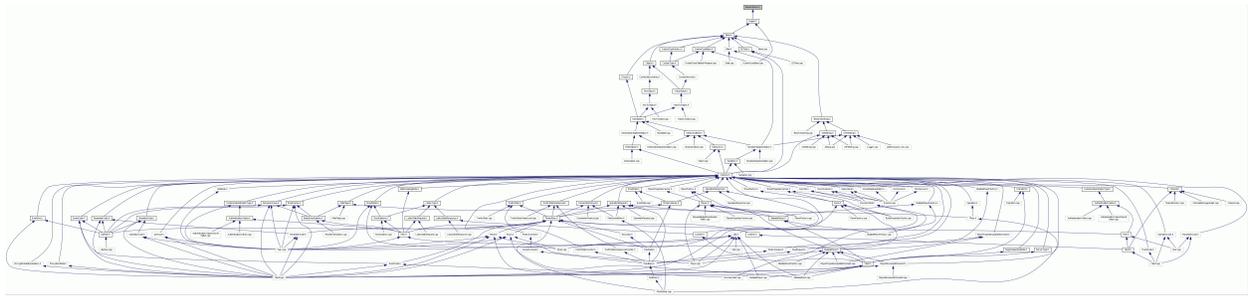
[Photon Documentation](#) | [Contact](#) | [Terms](#)

BaseListener.h File Reference

Include dependency graph for BaseListener.h:



This graph shows which files directly or indirectly include this file:



Classes

class **BaseListener**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Chat-cpp > inc >

[Classes](#) | [Namespaces](#)

Channel.h File Reference

Include dependency graph for Channel.h:

Classes

class **Channel**

Namespaces

ExitGames

ExitGames::Chat

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

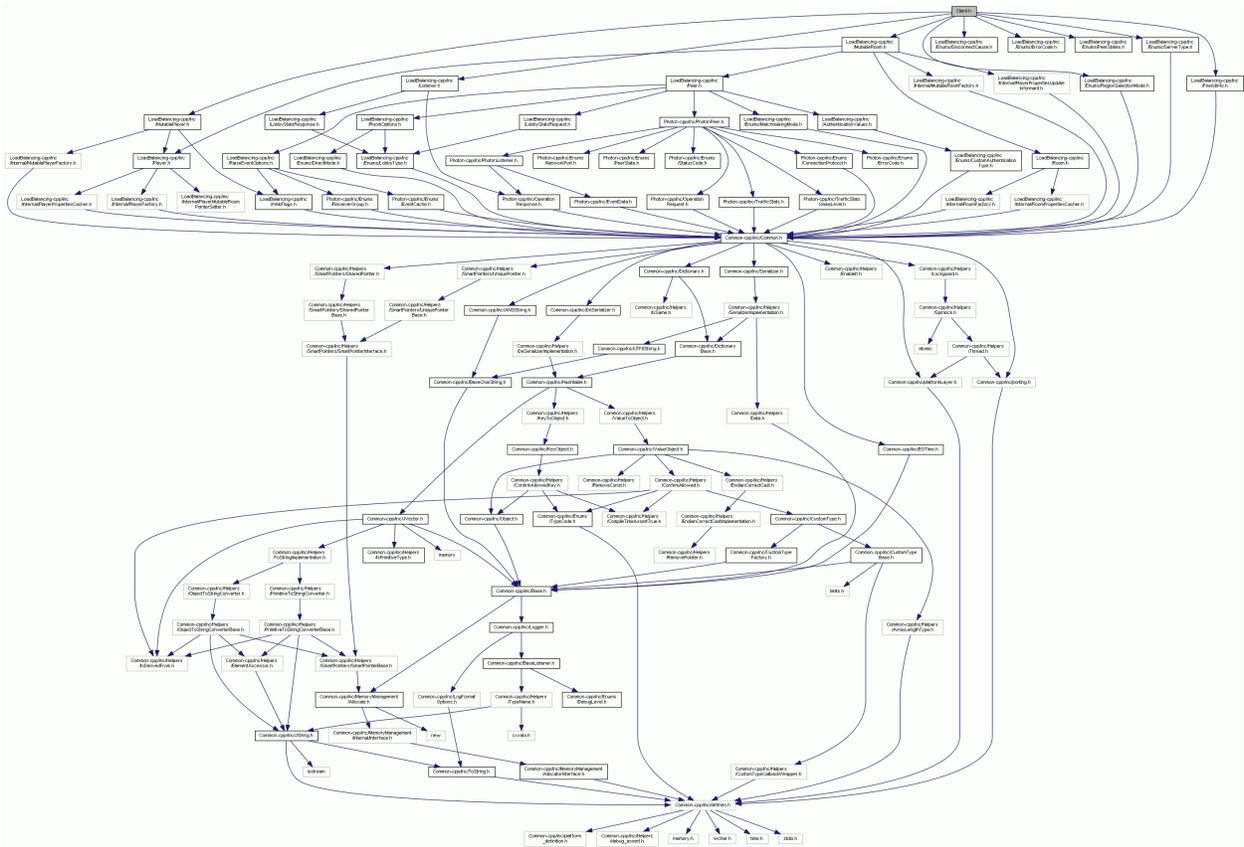
Client API 4.1.12.2

LoadBalancing-cpp > inc >

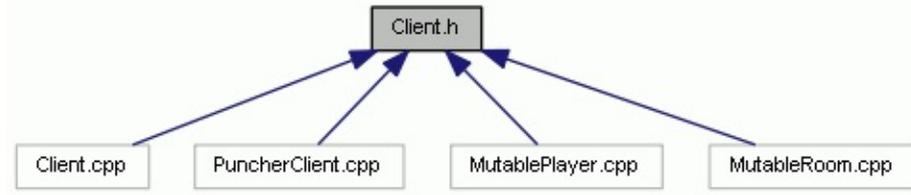
[Classes](#) | [Namespaces](#)

LoadBalancing-cpp/inc/Client.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/Client.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Client**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

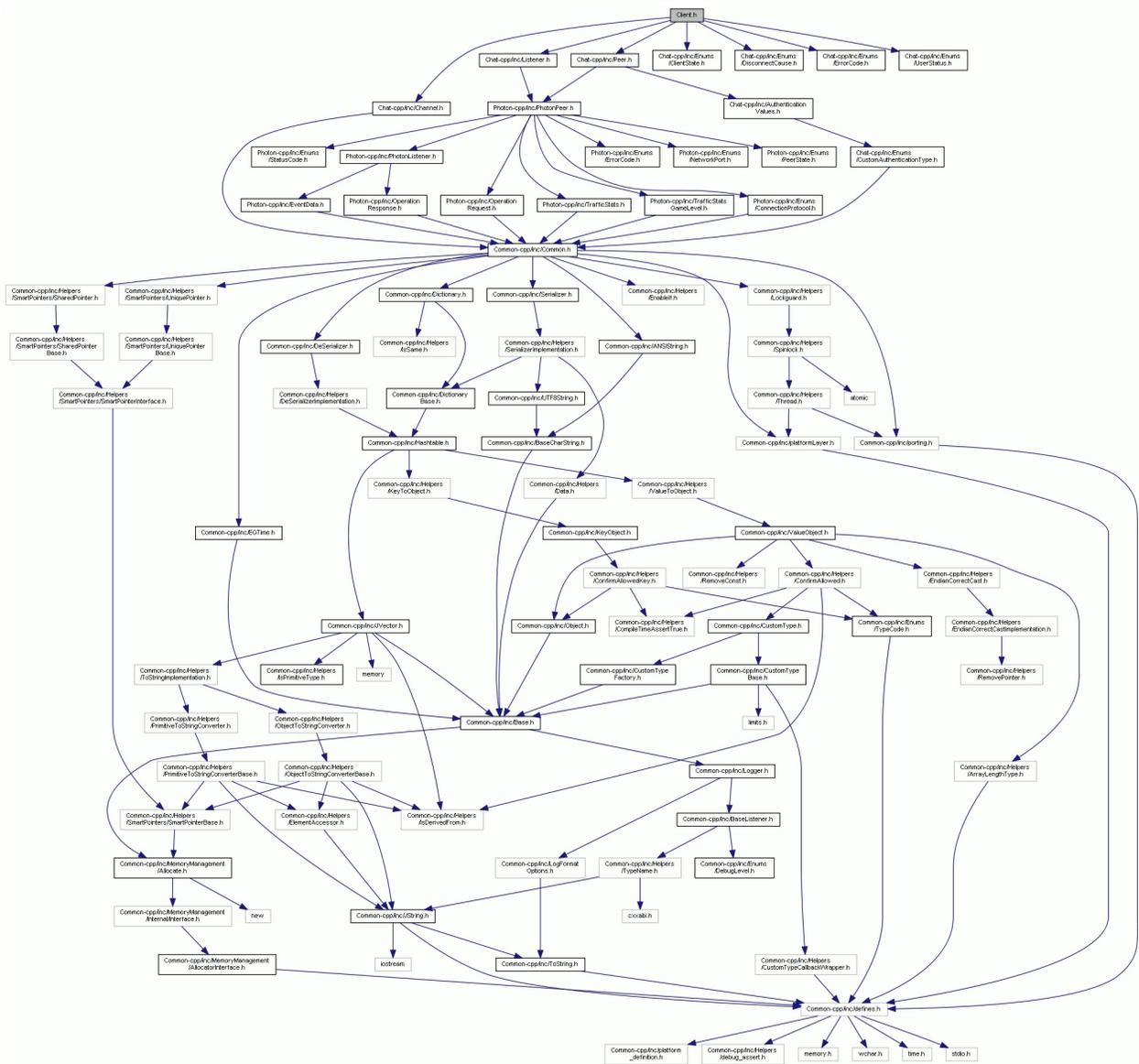
Chat-cpp > inc >

[Classes](#) | [Namespaces](#)

Chat-cpp/inc/Client.h

File Reference

Include dependency graph for Chat-cpp/inc/Client.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Client**

Namespaces

ExitGames

ExitGames::Chat

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

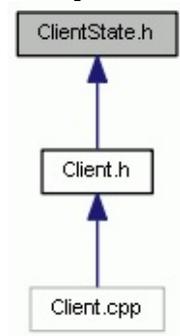
Client API 4.1.12.2

Chat-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

ClientState.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Chat

ExitGames::Chat::ClientState

Variables

static const int **Uninitialized**
Peer is created but not used yet.

static const int **ConnectingToNameServer**
Connecting to Name Server (includes connect
authenticate and joining the lobby)

static const int **ConnectedToNameServer**
Connected to Name Server.

static const int **Authenticating**
Authenticating.

static const int **Authenticated**
Authenticated.

static const int **DisconnectingFromNameServer**
Transition from Name to **Chat** Server.

static const int **ConnectingToFrontEnd**
Transition to **Chat** Server.

static const int **ConnectedToFrontEnd**
Connected to **Chat** Server. Subscribe to channels and
chat here.

static const int **Disconnecting**
The client disconnects (from any server).

static const int **Disconnected**
The client is no longer connected (to any server).
Connect to Name Server to go on.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Common-cpp > inc >

Common.h File Reference

Include dependency graph for Common.h:



Photon C++

Client API 4.1.12.2

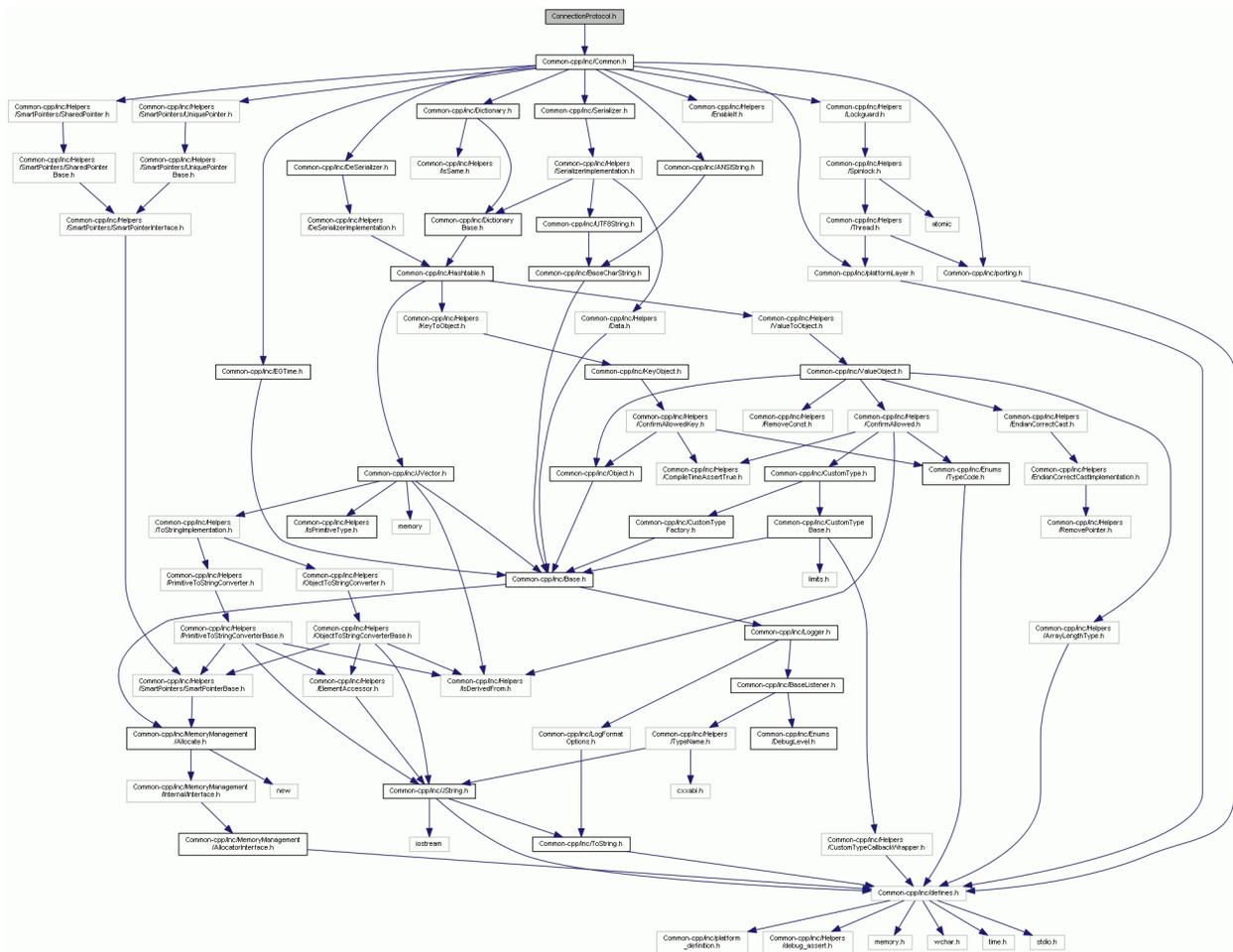
[Photon-cpp](#) > [inc](#) > [Enums](#)

[Namespaces](#) | [Functions](#) | [Variables](#)

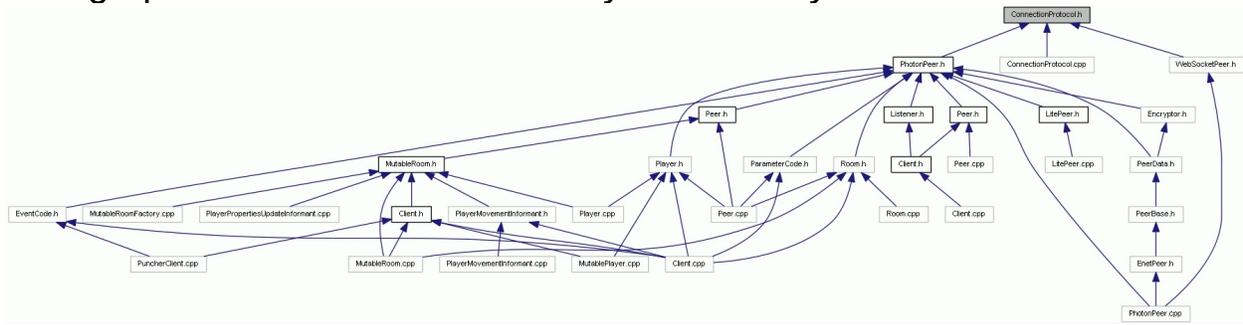
ConnectionProtocol.h

File Reference

Include dependency graph for ConnectionProtocol.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Photon

ExitGames::Photon::ConnectionProtocol

Functions

bool **getIsUDP** (nByte connectionProtocol)

bool **getIsTCP** (nByte connectionProtocol)

bool **getIsWebSocket** (nByte connectionProtocol)

bool **getIsSecure** (nByte connectionProtocol)

Variables

static const nByte **UDP**
Use UDP to connect to **Photon**, which allows you to send operations reliable or unreliable on demand.

static const nByte **TCP**
Use TCP to connect to **Photon**.

static const nByte **WS**
Use websockets to connect to **Photon**.

static const nByte **WSS**
Use secure websockets to connect to **Photon**.

static const nByte **DEFAULT**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

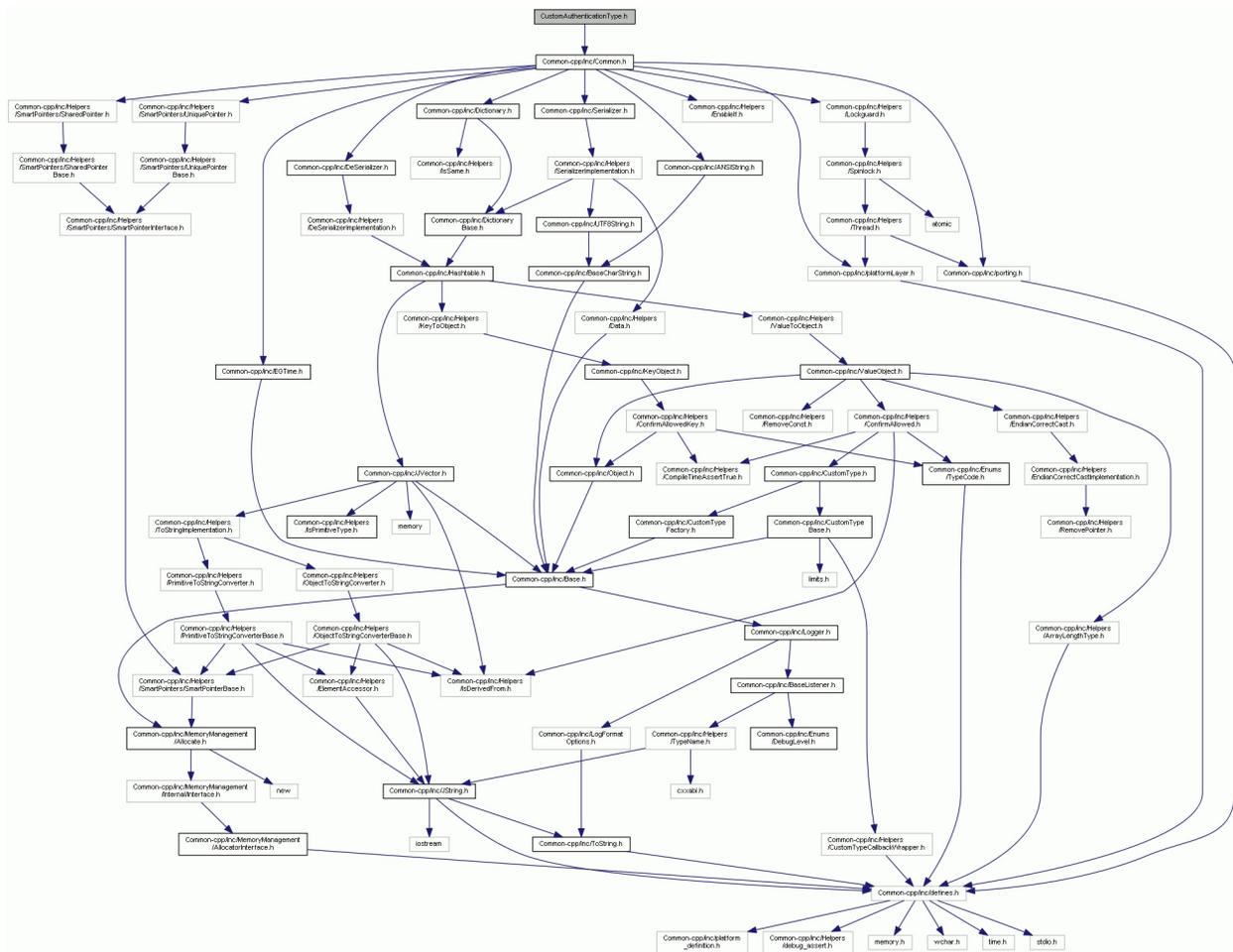
Client API 4.1.12.2

[LoadBalancing-cpp](#) [inc](#) [Enums](#)

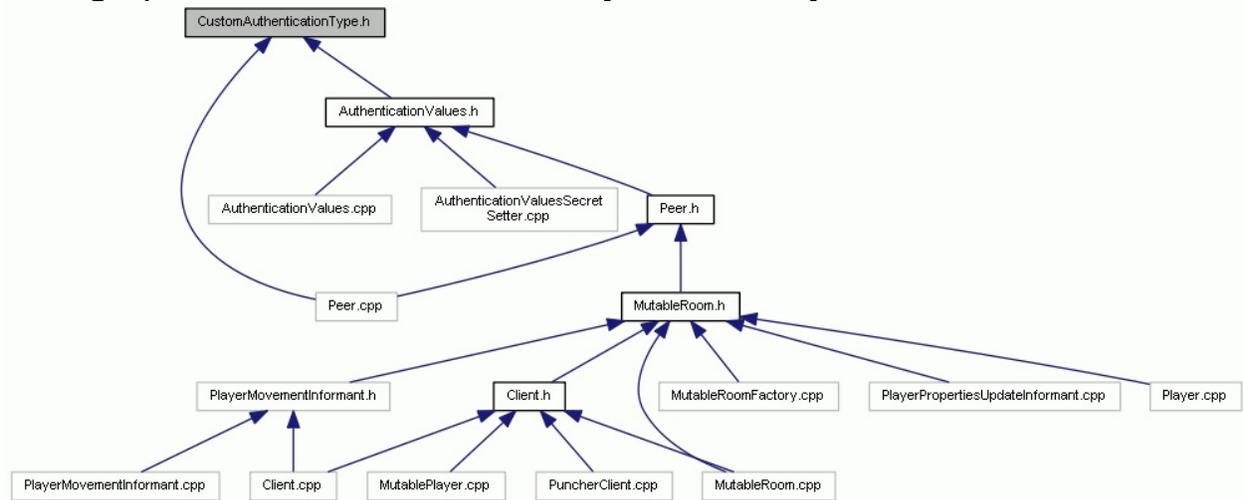
[Namespaces](#) | [Variables](#)

LoadBalancing- cpp/inc/Enums/CustomAuthenticationType.h File Reference

Include dependency graph for LoadBalancing-
cpp/inc/Enums/CustomAuthenticationType.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::CustomAuthenticationType

Variables

static const nByte **CUSTOM**
Use a custom authentication service.

static const nByte **STEAM**
Authenticates users by their Steam Account. Set auth values accordingly!

static const nByte **FACEBOOK**
Authenticates users by their Facebook Account. Set auth values accordingly!

static const nByte **OCULUS**
Authenticates users by their Oculus Account. Set auth values accordingly!

static const nByte **PLAYSTATION**
Authenticates users by their PSN Account. Set auth values accordingly!

static const nByte **XBOX**
Authenticates users by their XBox Network Account. Set auth values accordingly!

static const nByte **NONE**
Disables custom authentication.



Photon C++

Client API 4.1.12.2

[Chat-cpp](#)

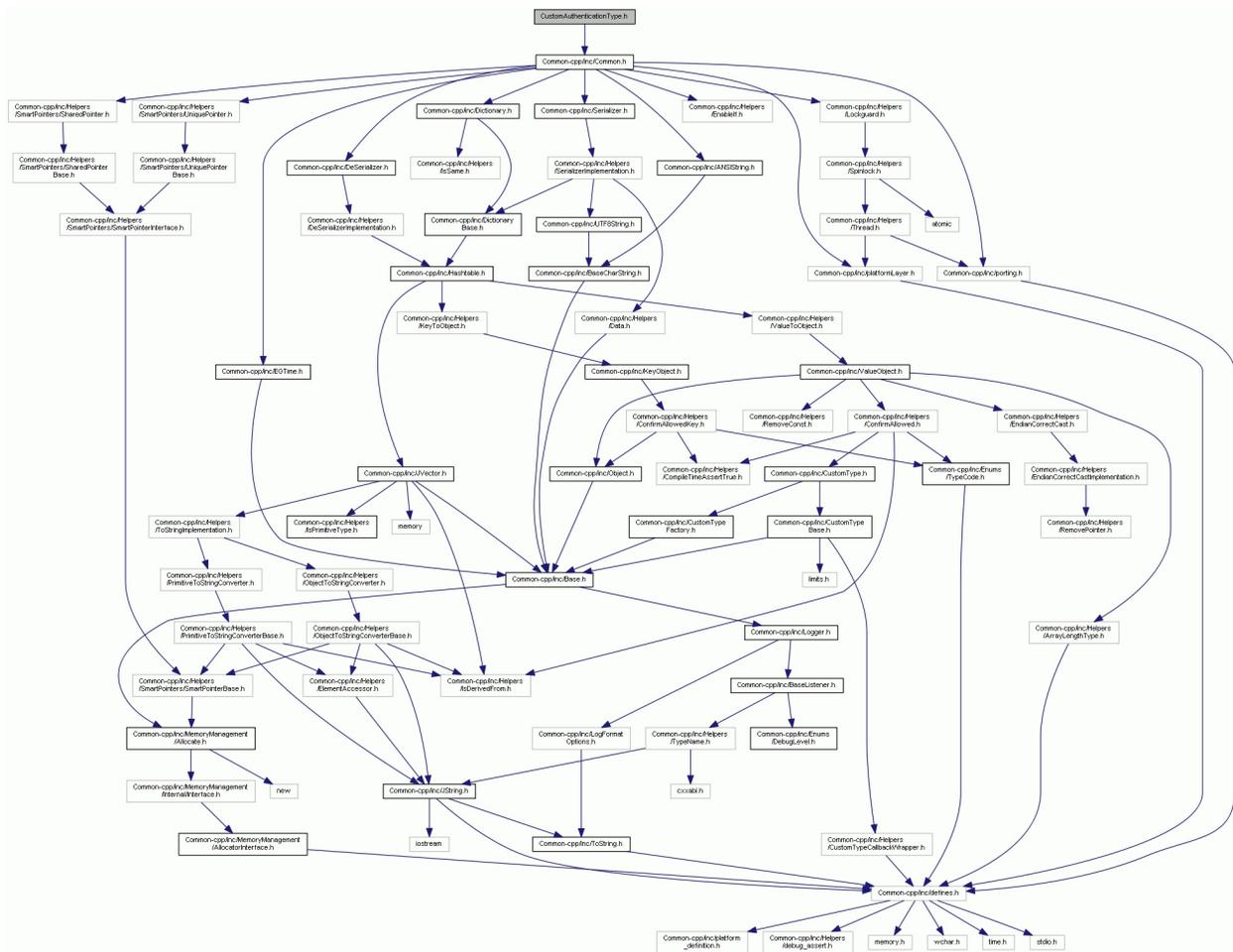
[inc](#)

[Enums](#)

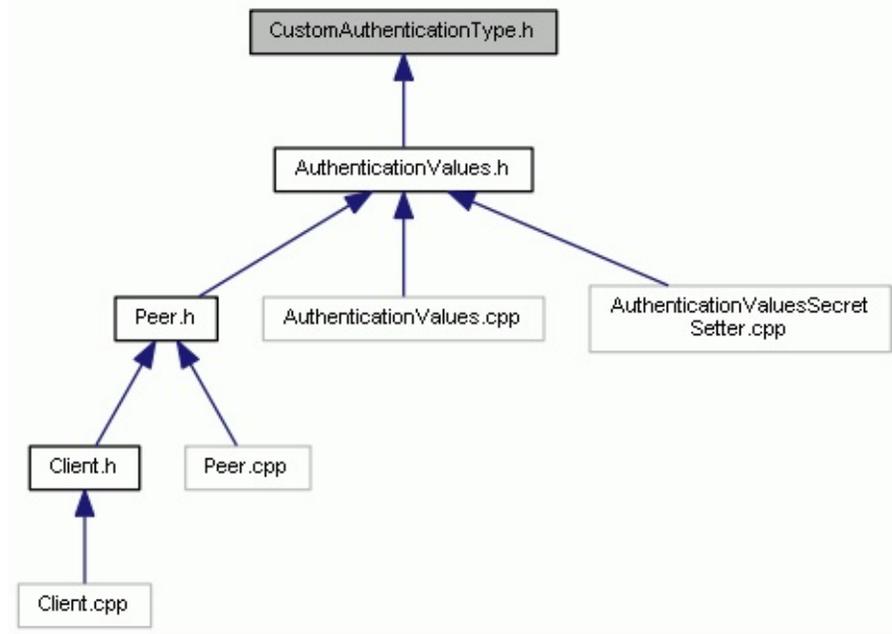
[Namespaces](#) | [Variables](#)

Chat- cpp/inc/Enums/CustomAuthenticationType.h File Reference

Include dependency graph for Chat-
cpp/inc/Enums/CustomAuthenticationType.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Chat

ExitGames::Chat::CustomAuthenticationType

Variables

static const nByte **CUSTOM**
Use a custom authentication service.

static const nByte **STEAM**
Authenticates users by their Steam Account. Set auth values accordingly!

static const nByte **FACEBOOK**
Authenticates users by their Facebook Account. Set auth values accordingly!

static const nByte **OCULUS**
Authenticates users by their Oculus Account. Set auth values accordingly!

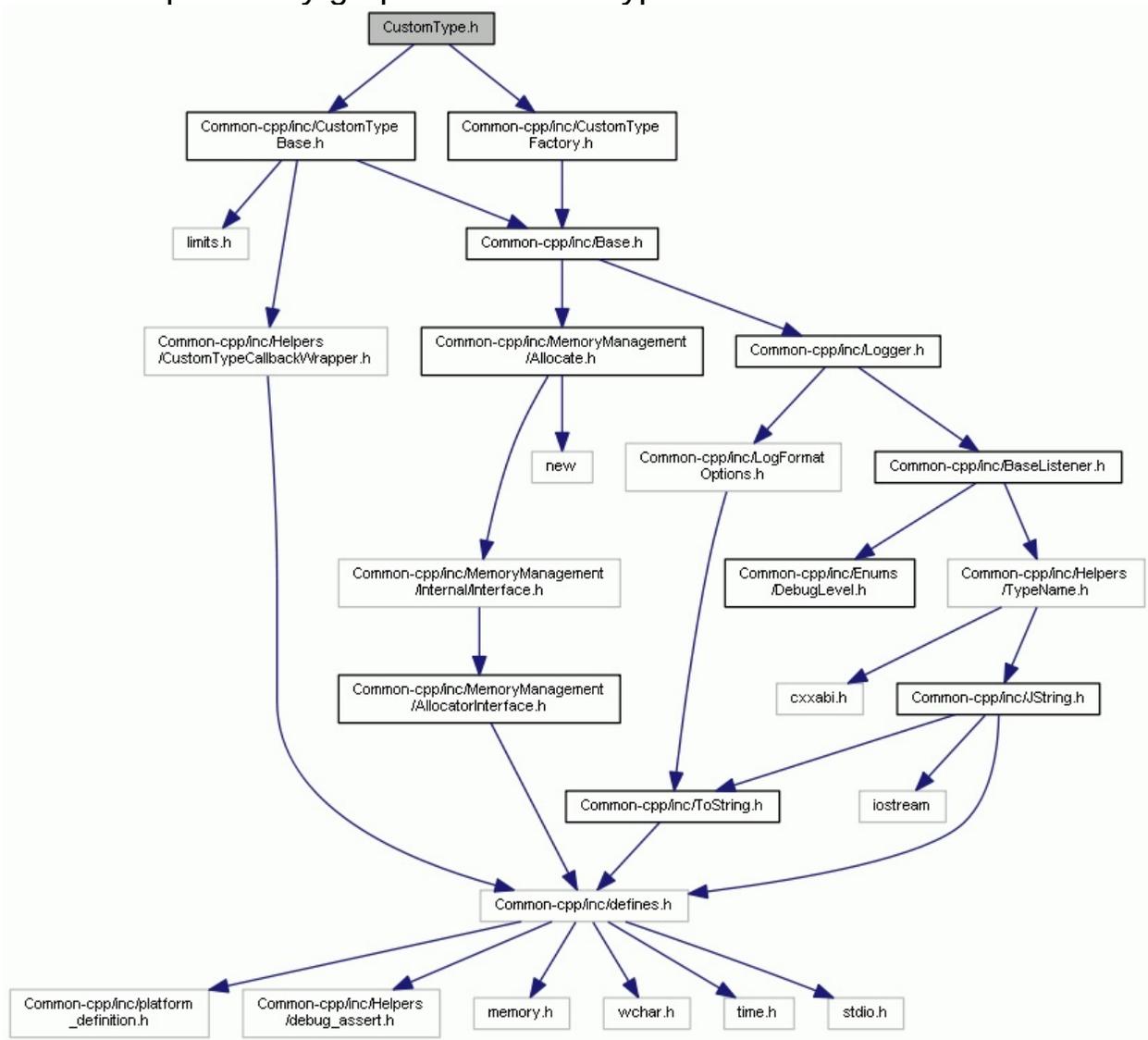
static const nByte **PLAYSTATION**
Authenticates users by their PSN Account. Set auth values accordingly!

static const nByte **XBOX**
Authenticates users by their XBox Network Account. Set auth values accordingly!

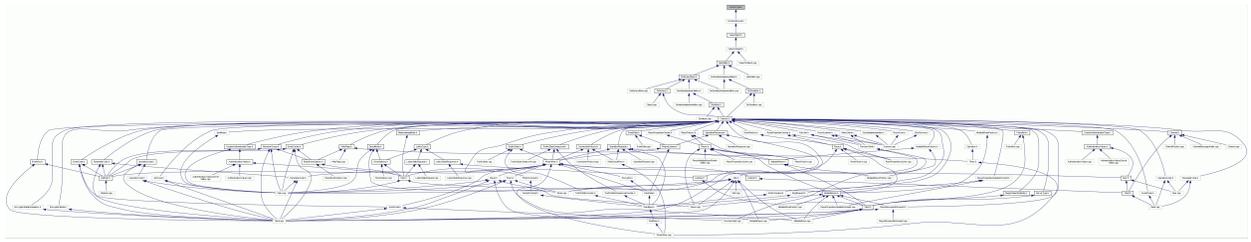
static const nByte **NONE**
Disables custom authentication.

CustomType.h File Reference

Include dependency graph for CustomType.h:



This graph shows which files directly or indirectly include this file:



Classes

```
class CustomType< typeCode >
```

Namespaces

ExitGames

ExitGames::Common

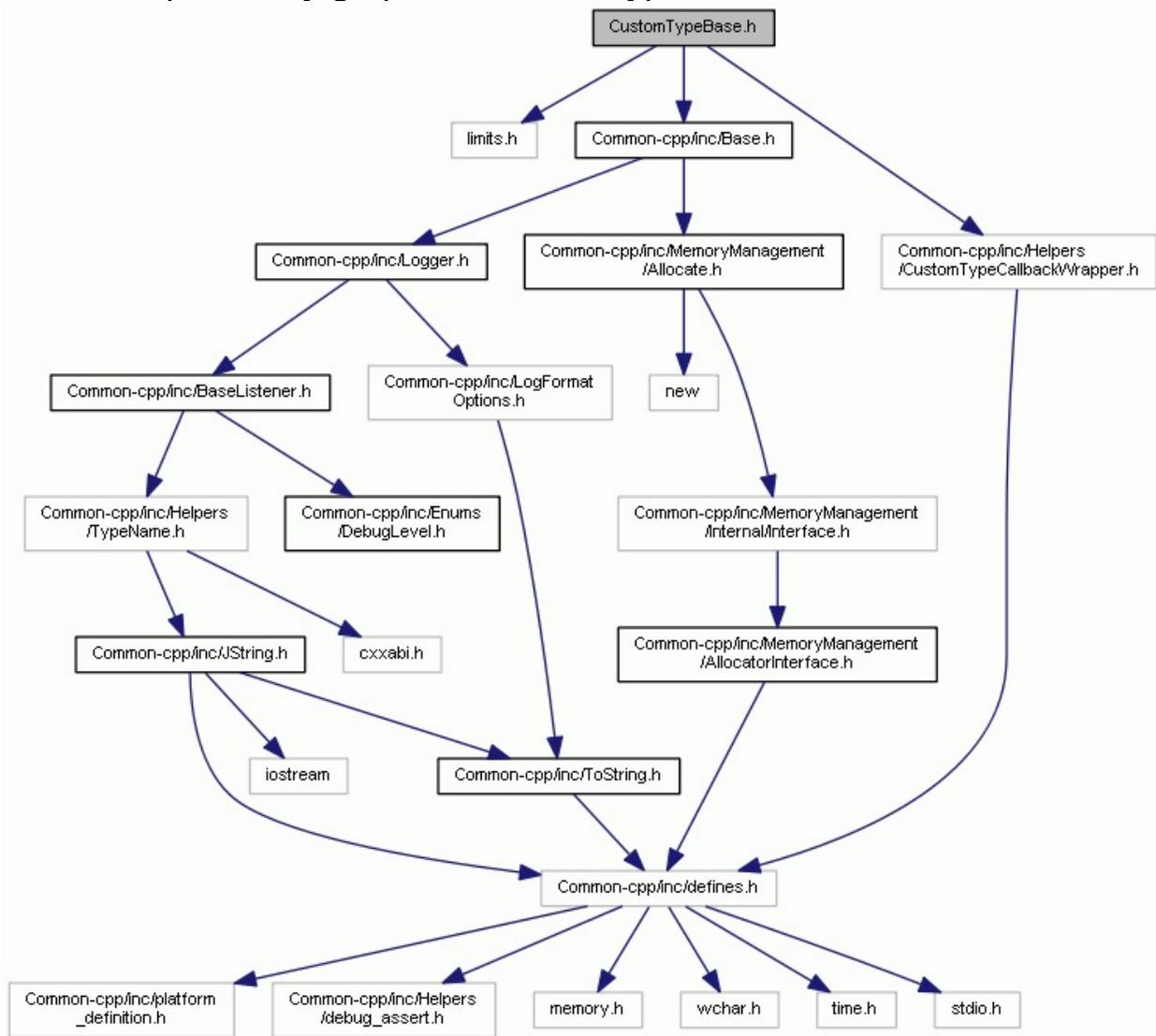
© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

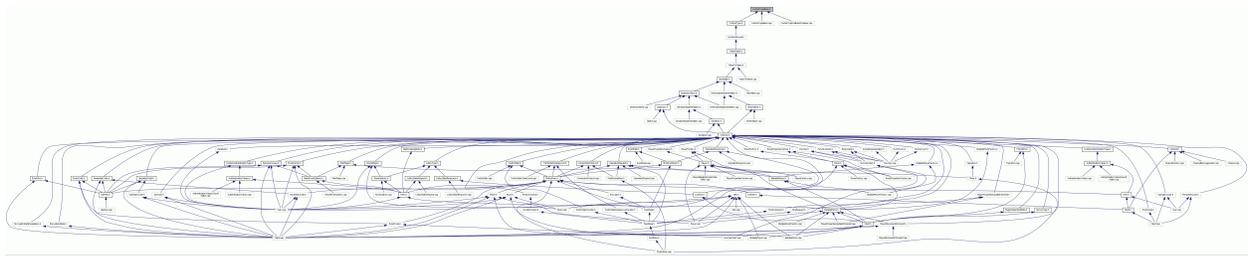
CustomTypeBase.h

File Reference

Include dependency graph for CustomTypeBase.h:



This graph shows which files directly or indirectly include this file:



Classes

class **CustomTypeBase**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

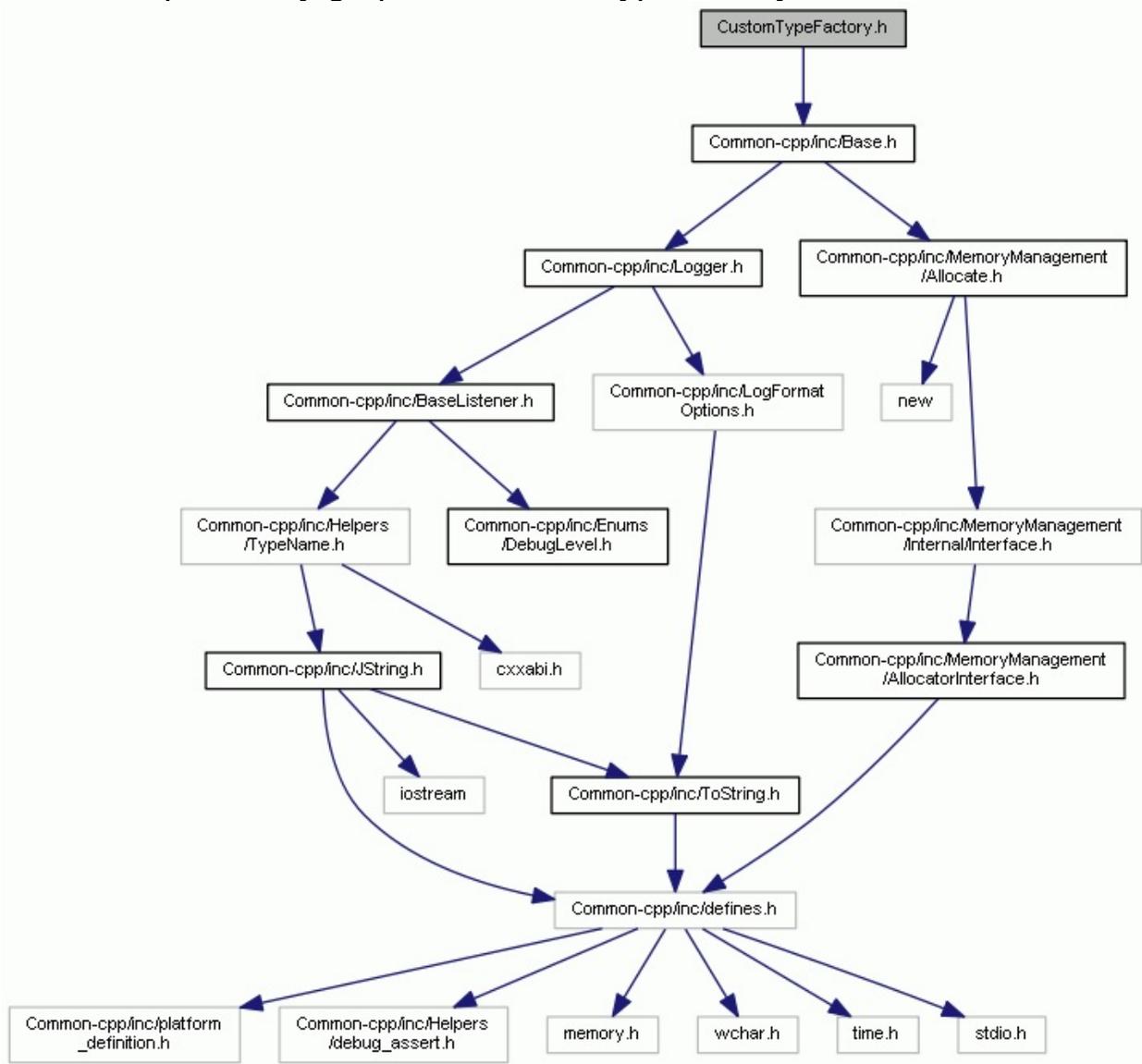
Common-cpp > inc >

Classes | Namespaces

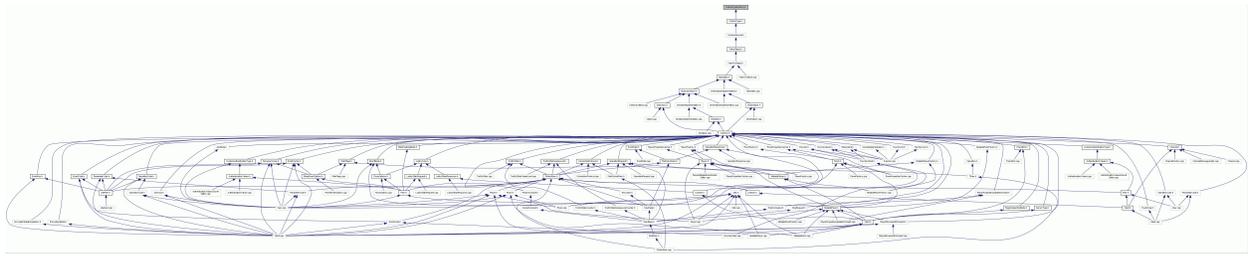
CustomTypeFactory.h

File Reference

Include dependency graph for CustomTypeFactory.h:



This graph shows which files directly or indirectly include this file:



Classes

class **CustomType**< typeCode >

class **CustomTypeFactory**< typeCode >

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Common-cpp

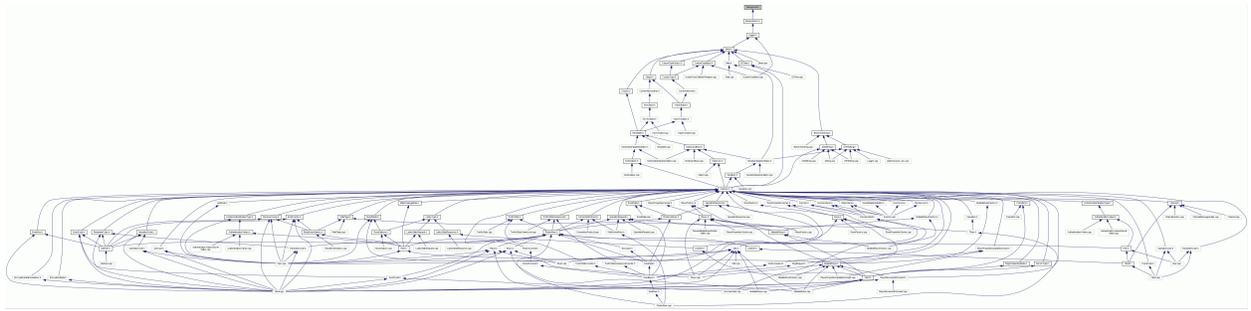
inc

Enums

[Namespaces](#) | [Variables](#)

DebugLevel.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Common

ExitGames::Common::DebugLevel

Variables

static const int **OFF**
No debug out.

static const int **ERRORS**
Only error descriptions.

static const int **WARNINGS**
Warnings and errors.

static const int **INFO**
Information about internal workflows, warnings and errors.

static const int **ALL**
Most complete workflow description (but lots of debug output), info, warnings and errors.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

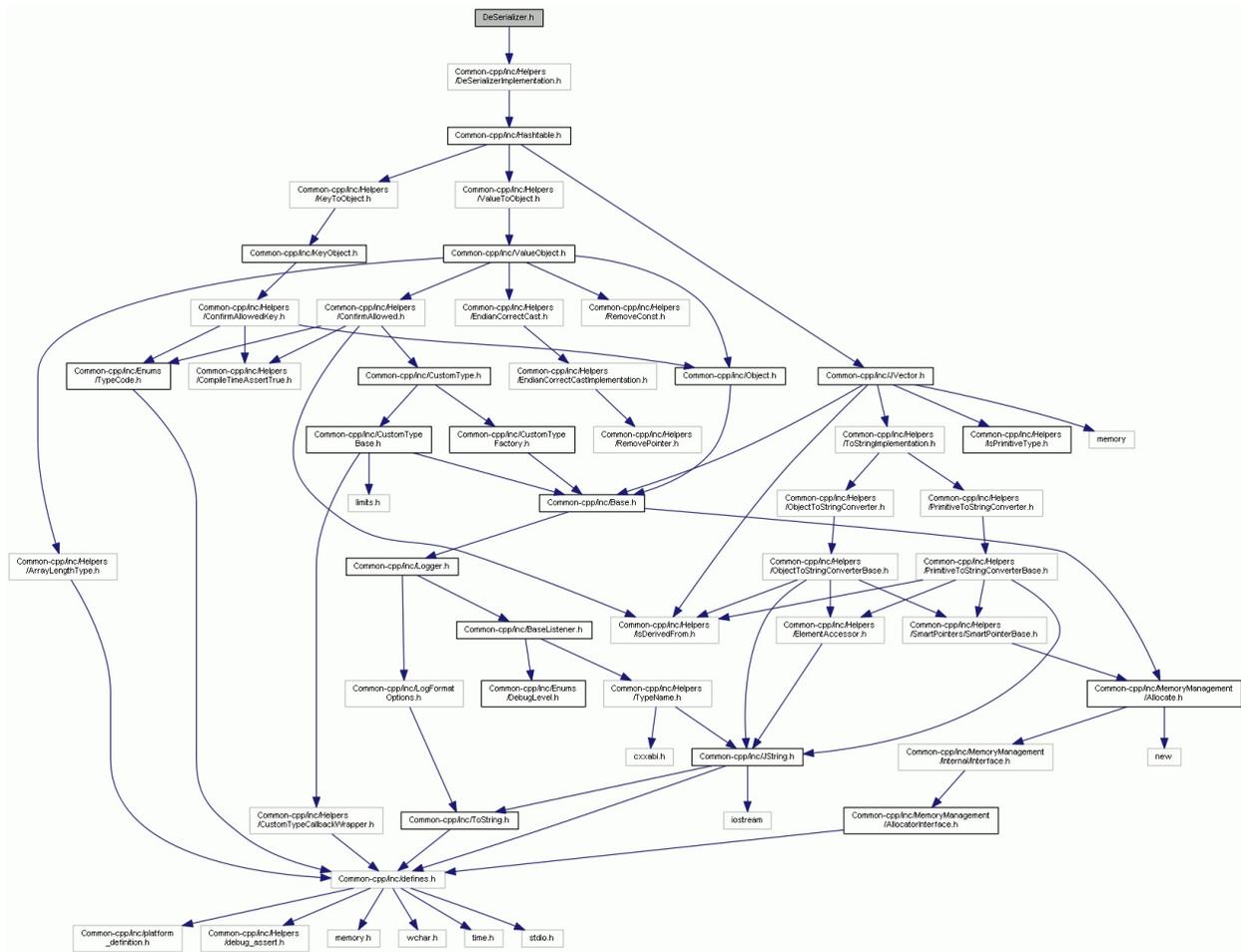
Common-cpp

inc

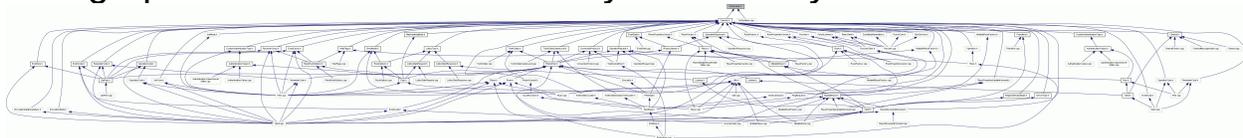
[Classes](#) | [Namespaces](#)

DeSerializer.h File Reference

Include dependency graph for DeSerializer.h:



This graph shows which files directly or indirectly include this file:



Classes

class **DeSerializer**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

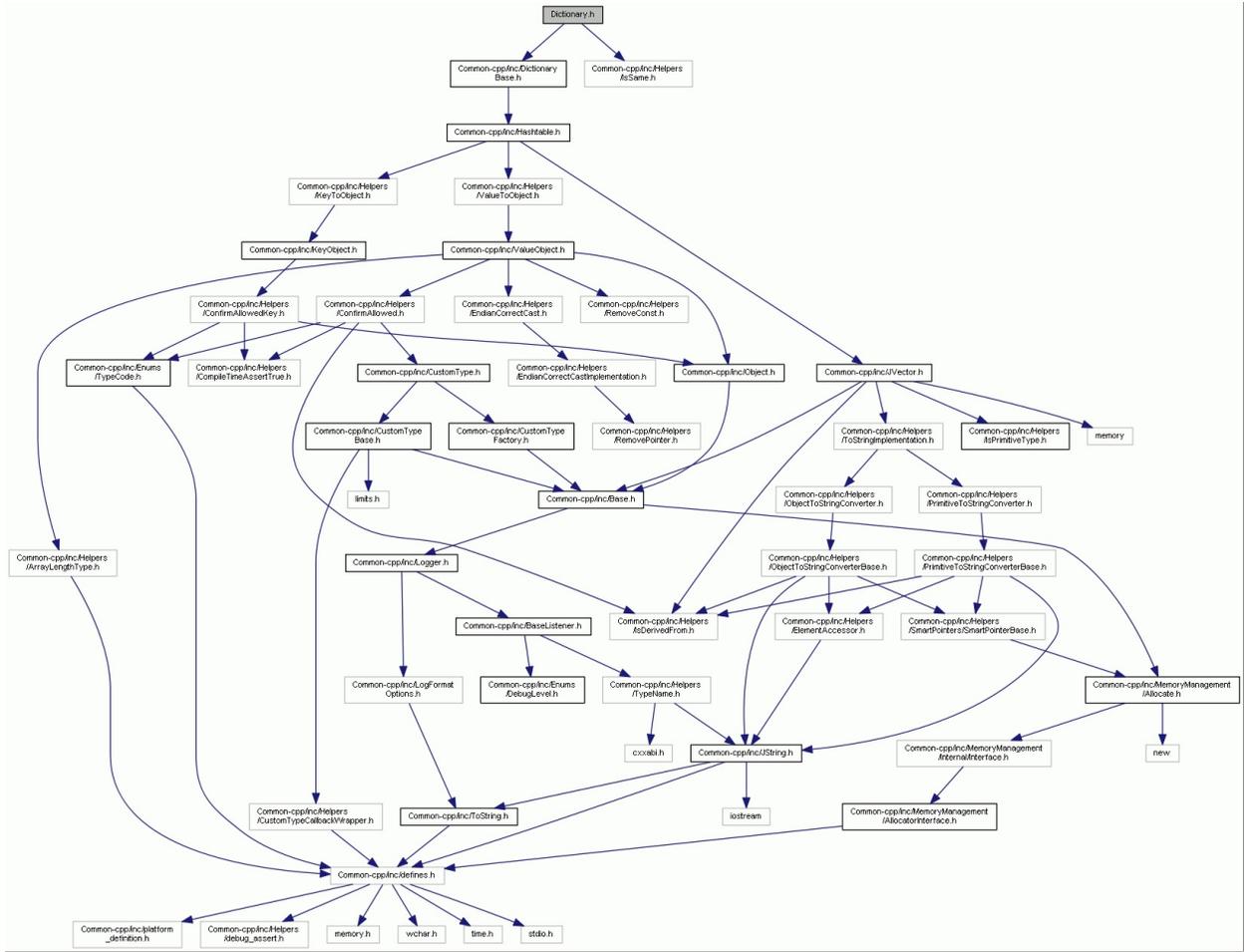
Client API 4.1.12.2

Common-cpp > inc >

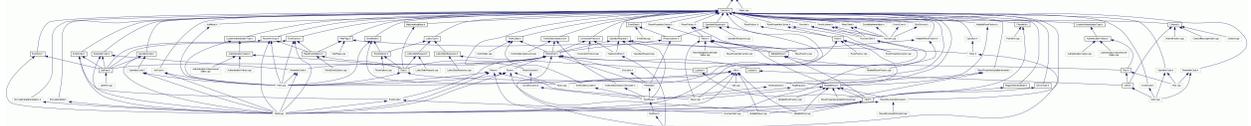
[Classes](#) | [Namespaces](#)

Dictionary.h File Reference

Include dependency graph for Dictionary.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Dictionary**< EKeyType, EValueType >

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Common-cpp

inc

[Classes](#) | [Namespaces](#)

DictionaryBase.h File Reference

Include dependency graph for DictionaryBase.h:

Classes

class **DictionaryBase**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

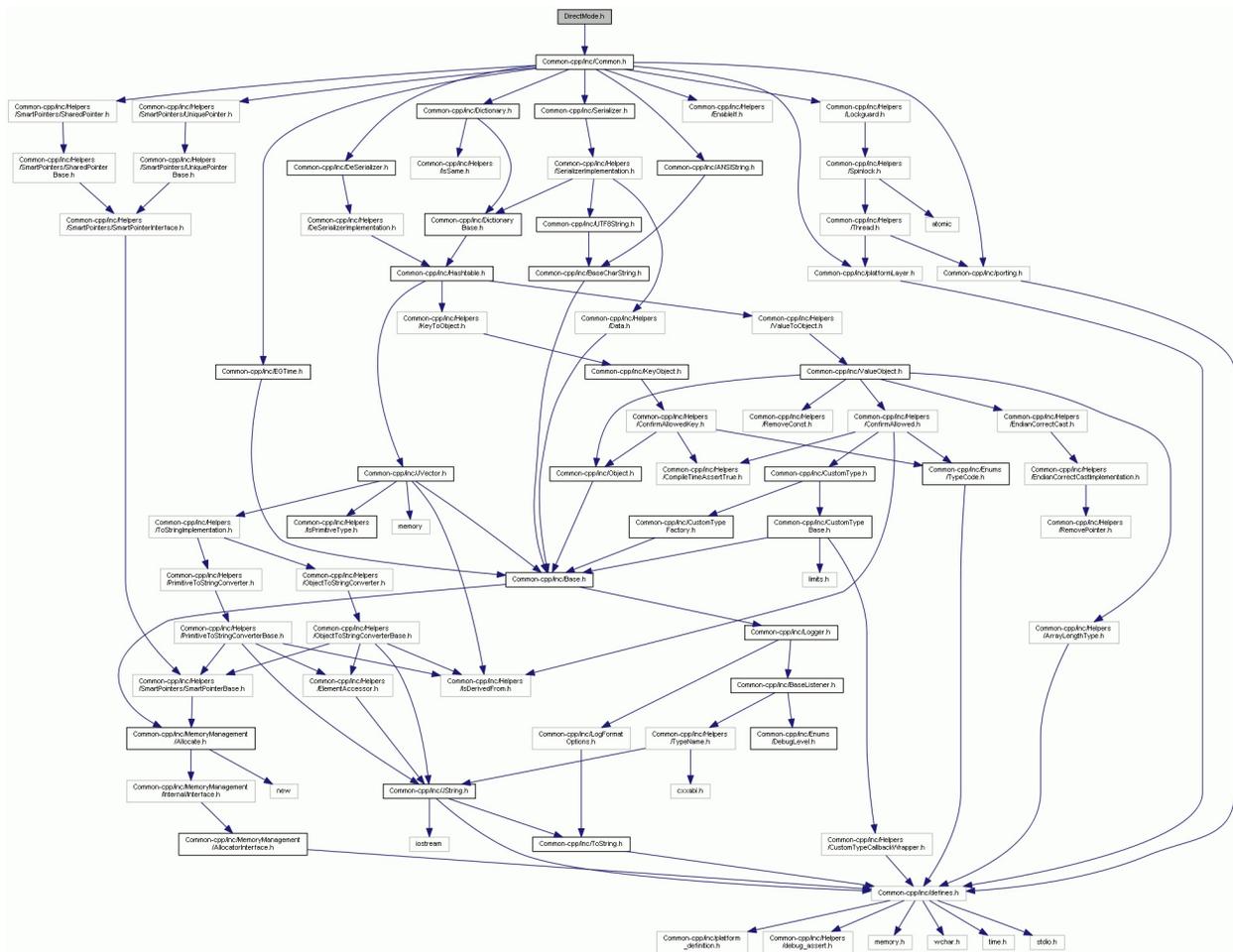
Client API 4.1.12.2

[LoadBalancing-cpp](#) > [inc](#) > [Enums](#) >

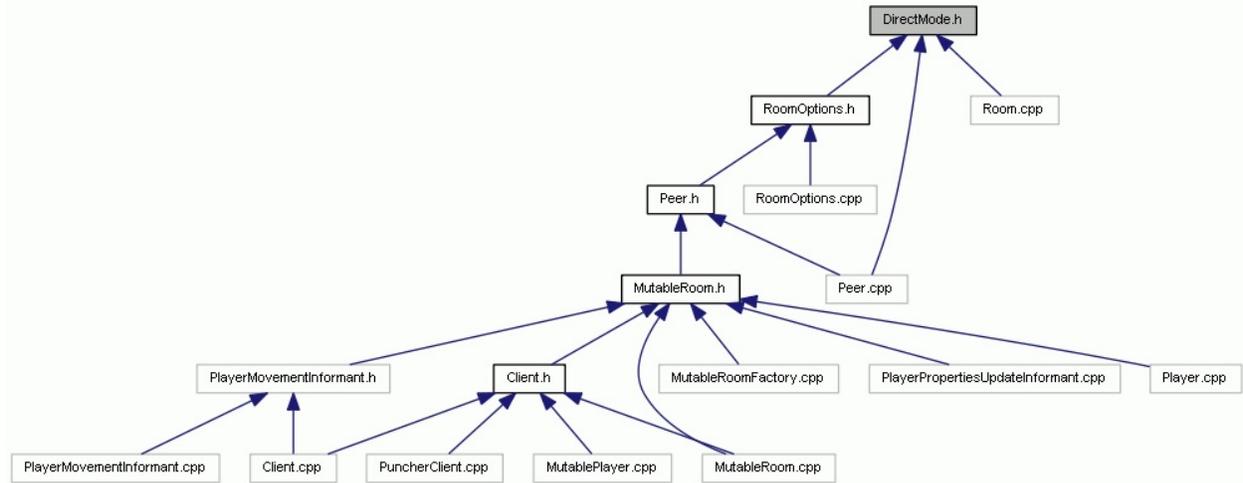
[Namespaces](#) | [Variables](#)

DirectMode.h File Reference

Include dependency graph for DirectMode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::DirectMode

Variables

static const nByte **NONE**
Do not create any 2p2 connections between the clients. This is the default.

static const nByte **ALL_TO_ALL**
Each client establishes a direct connection with every other client inside the room.

static const nByte **MASTER_TO_ALL**
The master client establishes a direct connection with every other client inside the room. All other clients only establish a direct connection with the master client but not with each other.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

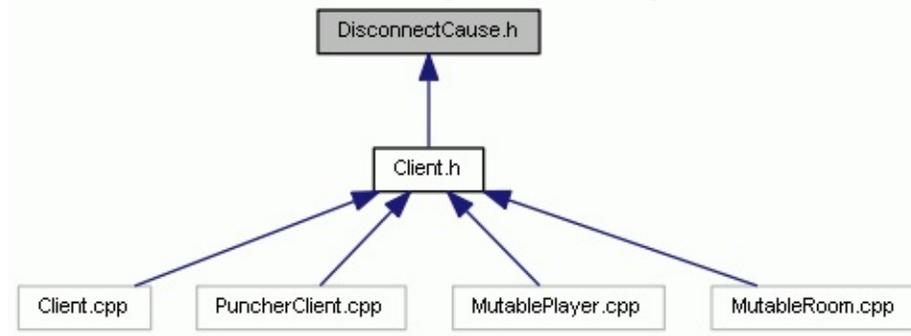
Client API 4.1.12.2

LoadBalancing-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

LoadBalancing-cpp/inc/Enums/DisconnectCause.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::DisconnectCause

Variables

static const int **NONE**
No error was tracked.

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
OnStatusChanged: The CCUs count of your **Photon** Server License is exhausted (temporarily).

static const int **EXCEPTION_ON_CONNECT**
OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

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

static const int **DISCONNECT_BY_SERVER_LOGIC**
OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).

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

static const int **EXCEPTION**
OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

static const int **INVALID_AUTHENTICATION**
OnOperationResponse: Authenticate in the **Photon** Cloud with invalid AppId. Update your subscription or contact Exit Games.

static const int **MAX_CCU_REACHED**

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

static const int **INVALID_REGION**

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

static const int **OPERATION_NOT_ALLOWED_IN_CURRENT_STATE**

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

static const int **CUSTOM_AUTHENTICATION_FAILED**

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

Photon C++

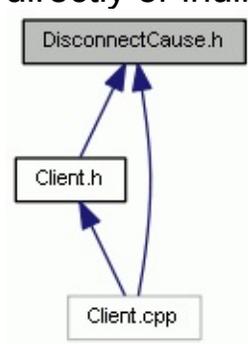
Client API 4.1.12.2

Chat-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

Chat-cpp/inc/Enums/DisconnectCause.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Chat

ExitGames::Chat::DisconnectCause

Variables

static const int **NONE**
No error was tracked.

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
OnStatusChanged: The CCUs count of your **Photon** Server License is exhausted (temporarily).

static const int **EXCEPTION_ON_CONNECT**
OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.

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

static const int **DISCONNECT_BY_SERVER_LOGIC**
OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).

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

static const int **EXCEPTION**
OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.

static const int **INVALID_AUTHENTICATION**
OnOperationResponse: Authenticate in the **Photon** Cloud with invalid AppId. Update your subscription or contact Exit Games.

static const int **MAX_CCU_REACHED**

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

static const int **INVALID_REGION**

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

static const int **OPERATION_NOT_ALLOWED_IN_CURRENT_STATE**

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

static const int **CUSTOM_AUTHENTICATION_FAILED**

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

Photon C++

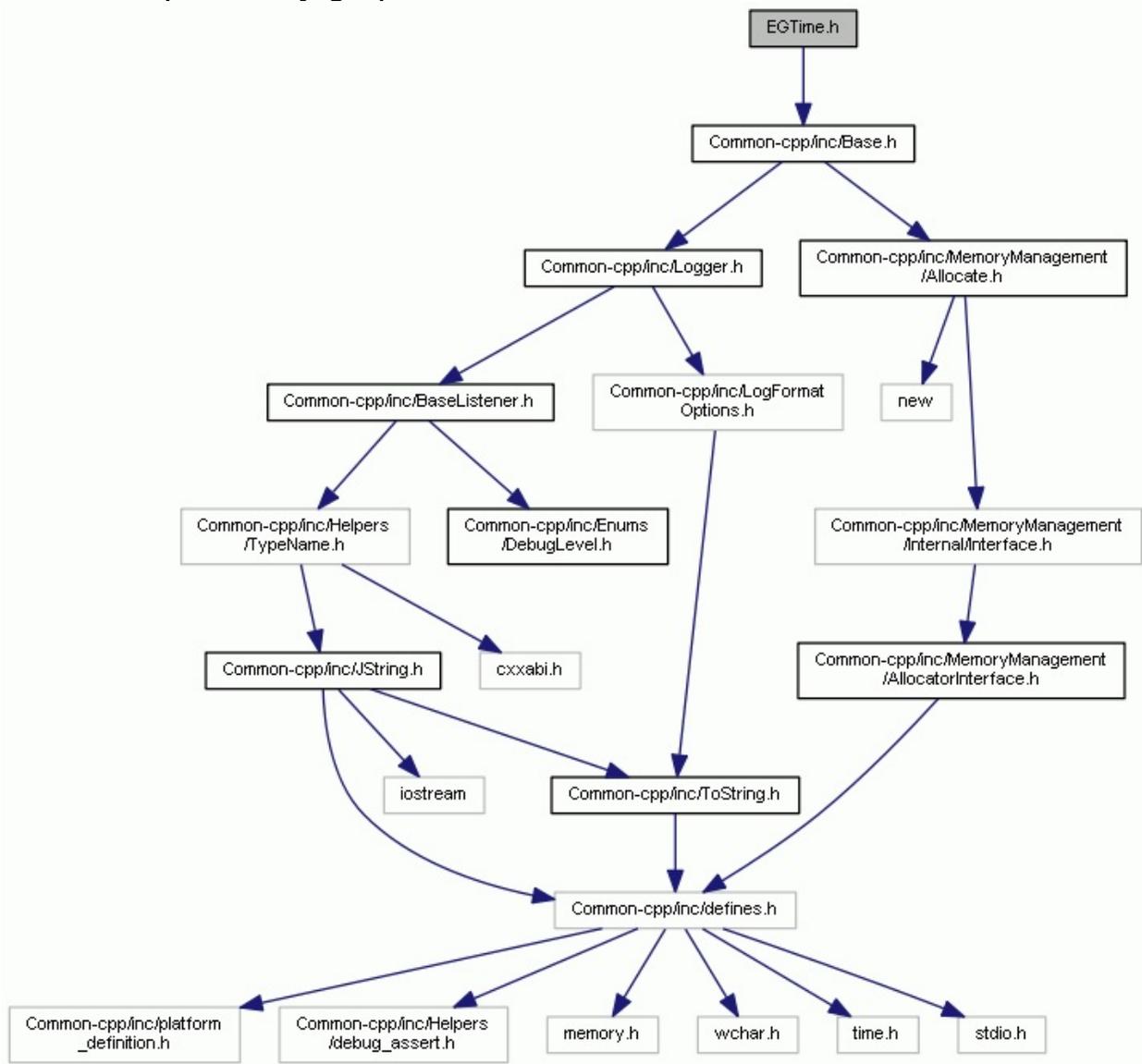
Client API 4.1.12.2

Common-cpp > inc >

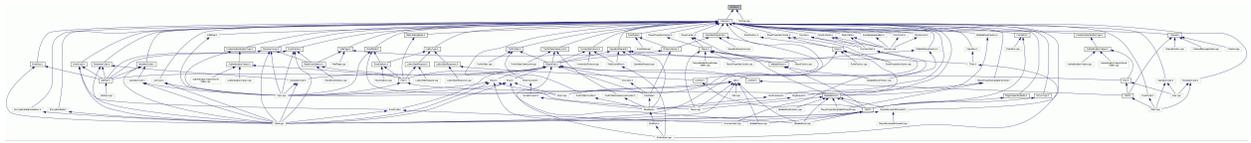
Classes | Namespaces

EGTime.h File Reference

Include dependency graph for EGTime.h:



This graph shows which files directly or indirectly include this file:



Classes

class **EGTime**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



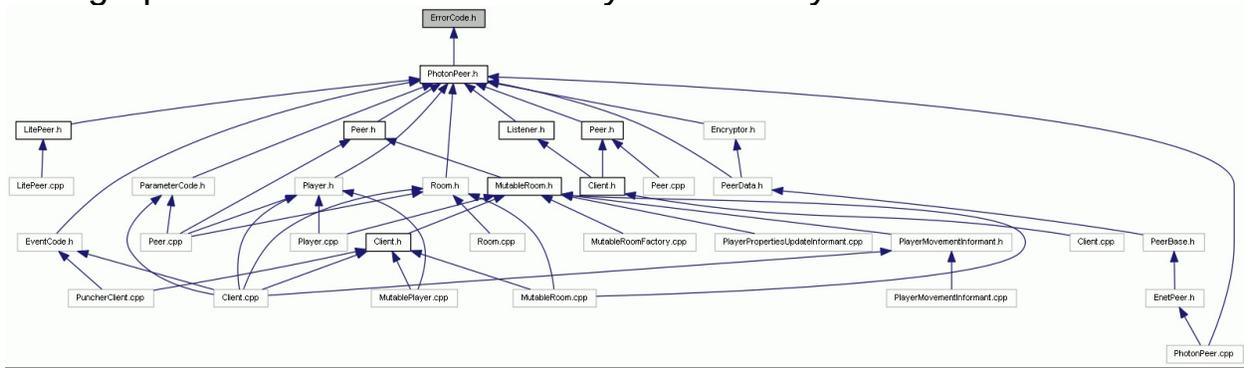
Photon C++ Client API 4.1.12.2

Photon-cpp > inc > Enums

Namespaces | Variables

Photon-cpp/inc/Enums/ErrorCode.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Photon

ExitGames::Photon::ErrorCode

Variables

static const int **SUCCESS**
No error.

static const int **EFAILED**
General failure.

static const int **ENOMEMORY**
Out of memory.

static const int **EBADCLASS**
NULL class object.

static const int **EBADPARAM**
Invalid parameter.

static const int **EITEMBUSY**
Context (system, interface, etc.) is busy.

static const int **NET_SUCCESS**
No network error, successful operation.

static const int **NET_ERROR**
Unsuccessful operation.

static const int **NET_ENETNONET**
Network subsystem unavailable.

static const int **NET_MSGSIZE**
Message too long. A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.

static const int **NET_ENOTCONN**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

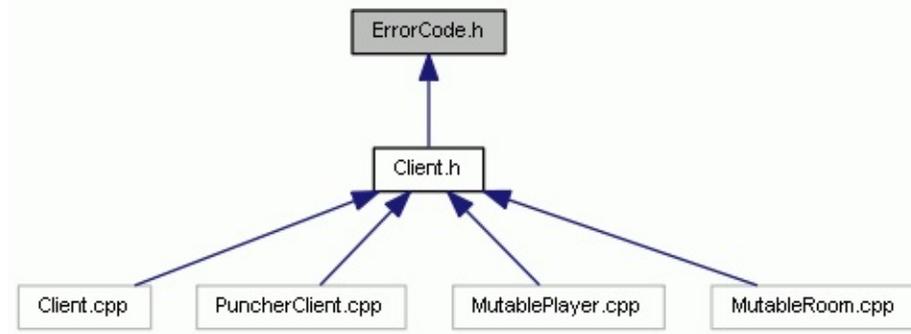
Client API 4.1.12.2

LoadBalancing-cpp > inc > Enums

[Namespaces](#) | [Variables](#)

LoadBalancing-cpp/inc/Enums/ErrorCode.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::ErrorCode

Variables

static const int **OPERATION_DENIED**

static const int **OPERATION_INVALID**

static const int **INTERNAL_SERVER_ERROR**

static const int **OK**

static const int **INVALID_AUTHENTICATION**

static const int **GAME_ID_ALREADY_EXISTS**

static const int **GAME_FULL**

static const int **GAME_CLOSED**

static const int **ALREADY_MATCHED**

static const int **SERVER_FULL**

static const int **USER_BLOCKED**

static const int **NO_MATCH_FOUND**

static const int **GAME_DOES_NOT_EXIST**

static const int **MAX_CCU_REACHED**

static const int **INVALID_REGION**

static const int **CUSTOM_AUTHENTICATION_FAILED**

static const int **AUTHENTICATION_TOKEN_EXPIRED**

static const int **PLUGIN_REPORTED_ERROR**

static const int **PLUGIN_MISMATCH**

static const int **JOIN_FAILED_PEER_ALREADY_JOINED**

static const int **JOIN_FAILED_FOUND_INACTIVE_JOINER**

static const int **JOIN_FAILED_WITH_REJOINER_NOT_FOUND**

static const int **JOIN_FAILED_FOUND_EXCLUDED_USER_ID**

static const int **JOIN_FAILED_FOUND_ACTIVE_JOINER**

static const int **HTTP_LIMIT_REACHED**

static const int **EXTERNAL_HTTP_CALL_FAILED**

static const int **SLOT_ERROR**

static const int **INVALID_ENCRYPTION_PARAMETERS**

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

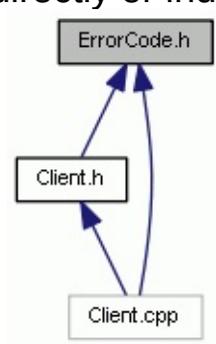
Client API 4.1.12.2

Chat-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

Chat-cpp/inc/Enums/ErrorCode.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Chat

ExitGames::Chat::ErrorCode

Variables

static const int **OPERATION_DENIED**

static const int **OPERATION_INVALID**

static const int **INTERNAL_SERVER_ERROR**

static const int **OK**

static const int **INVALID_AUTHENTICATION**

static const int **GAME_ID_ALREADY_EXISTS**

static const int **GAME_FULL**

static const int **GAME_CLOSED**

static const int **ALREADY_MATCHED**

static const int **SERVER_FULL**

static const int **USER_BLOCKED**

static const int **NO_MATCH_FOUND**

static const int **GAME_DOES_NOT_EXIST**

static const int **MAX_CCU_REACHED**

static const int **INVALID_REGION**

static const int **CUSTOM_AUTHENTICATION_FAILED**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

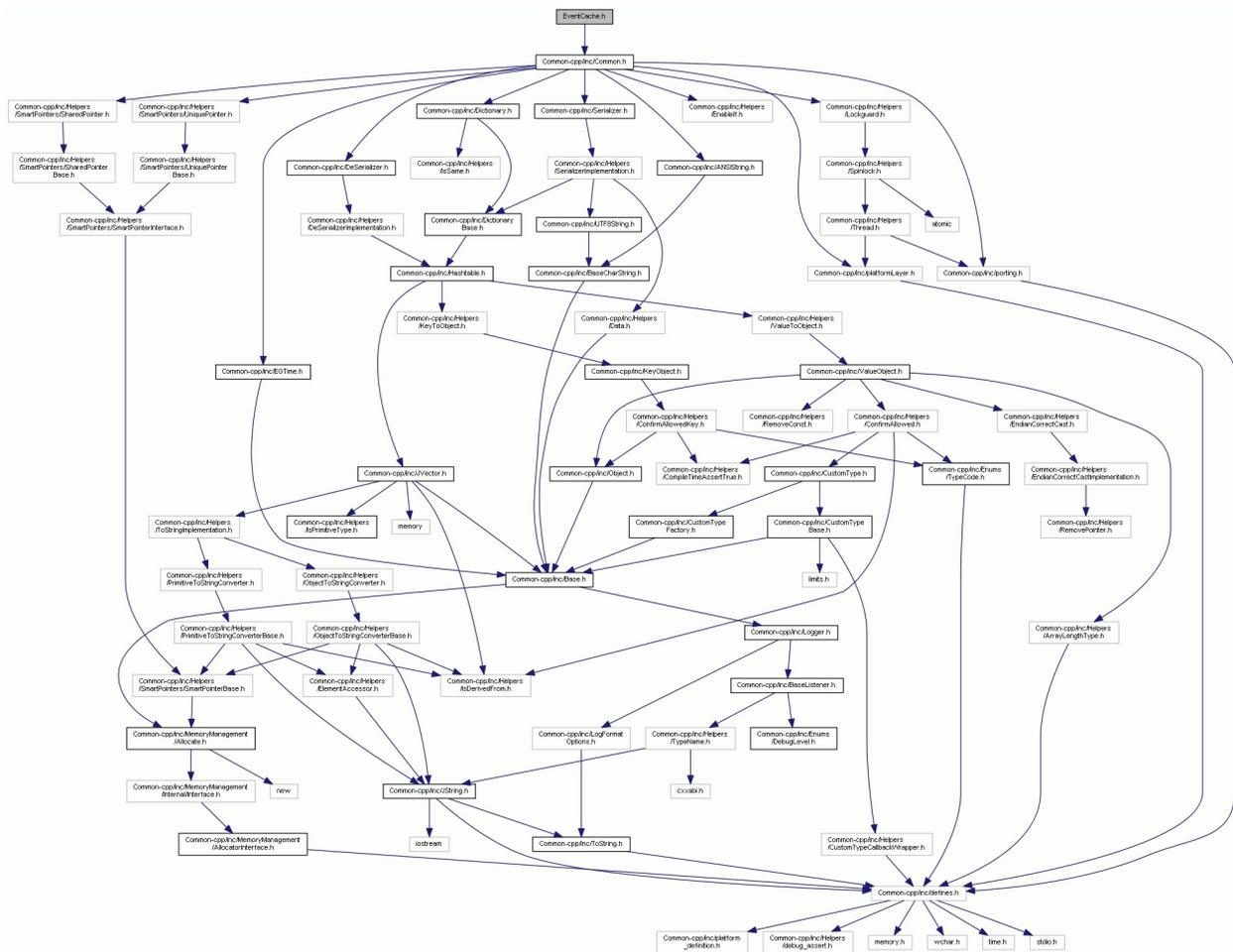
Client API 4.1.12.2

[Photon-cpp](#) > [inc](#) > [Enums](#)

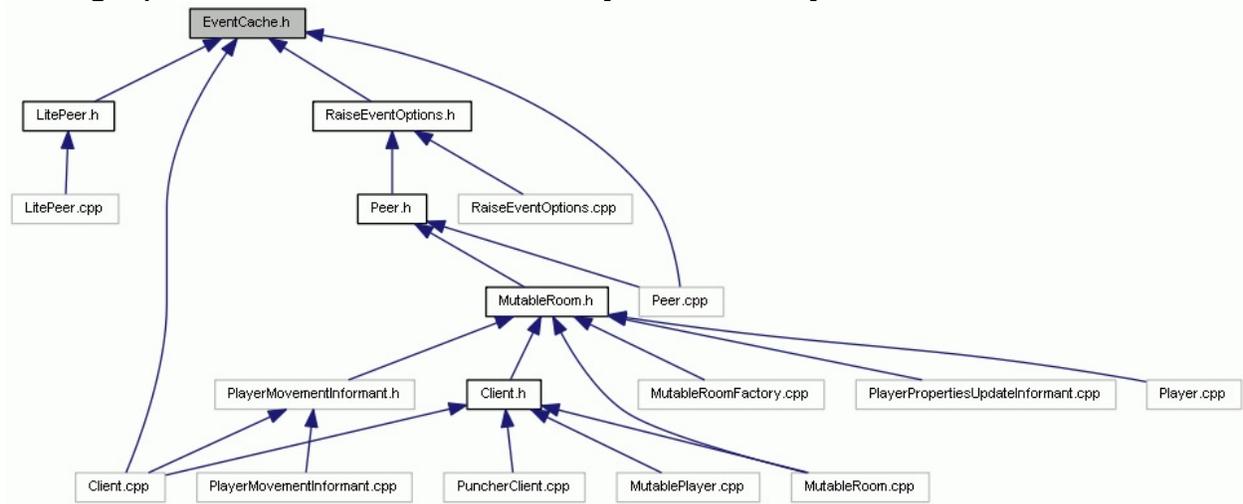
[Namespaces](#) | [Variables](#)

EventCache.h File Reference

Include dependency graph for EventCache.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::EventCache

Variables

static const nByte **DO_NOT_CACHE**

static const nByte **MERGE_CACHE**

static const nByte **REPLACE_CACHE**

static const nByte **REMOVE_CACHE**

static const nByte **ADD_TO_ROOM_CACHE**

static const nByte **ADD_TO_ROOM_CACHE_GLOBAL**

static const nByte **REMOVE_FROM_ROOM_CACHE**

static const nByte **REMOVE_FROM_ROOM_CACHE_FOR_ACTORS_**

static const nByte **SLICE_INC_INDEX**

static const nByte **SLICE_SET_INDEX**

static const nByte **SLICE_PURGE_INDEX**

static const nByte **SLICE_PURGE_UP_TO_INDEX**



Photon C++

Client API 4.1.12.2

[Photon-cpp](#)

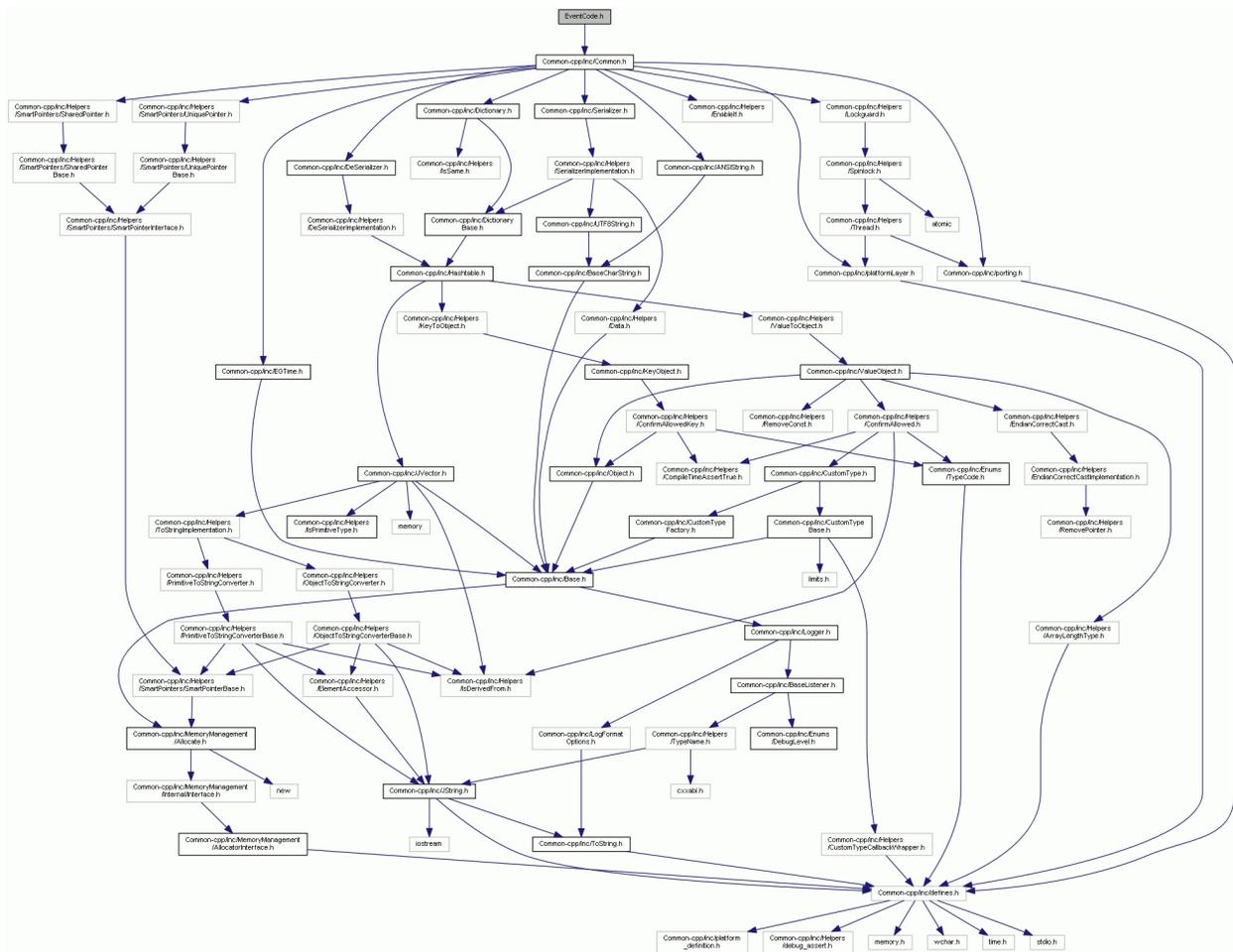
[inc](#)

[Enums](#)

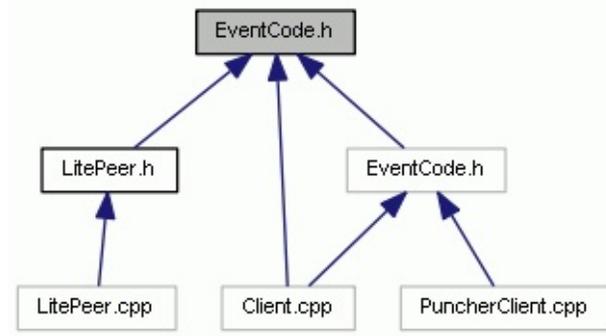
[Namespaces](#) | [Variables](#)

Photon- cpp/inc/Enums/EventCode.h File Reference

Include dependency graph for Photon-cpp/inc/Enums/EventCode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::EventCode

Variables

static const nByte **JOIN**

static const nByte **LEAVE**

static const nByte **PROPERTIES_CHANGED**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Photon-cpp > inc >

[Classes](#) | [Namespaces](#)

EventData.h File Reference

Include dependency graph for EventData.h:

Classes

class **EventData**

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[Photon-cpp](#)

[inc](#)

[Enums](#)

[Namespaces](#) | [Variables](#)

EventKey.h File Reference

Include dependency graph for EventKey.h:

Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::EventKey

Variables

static const nByte **ACTORNR**

static const nByte **TARGET_ACTORNR**

static const nByte **ACTORLIST**

static const nByte **PROPERTIES**

static const nByte **ACTORPROPERTIES**

static const nByte **GAMEPROPERTIES**

static const nByte **DATA**

© [Exit Games](#)®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

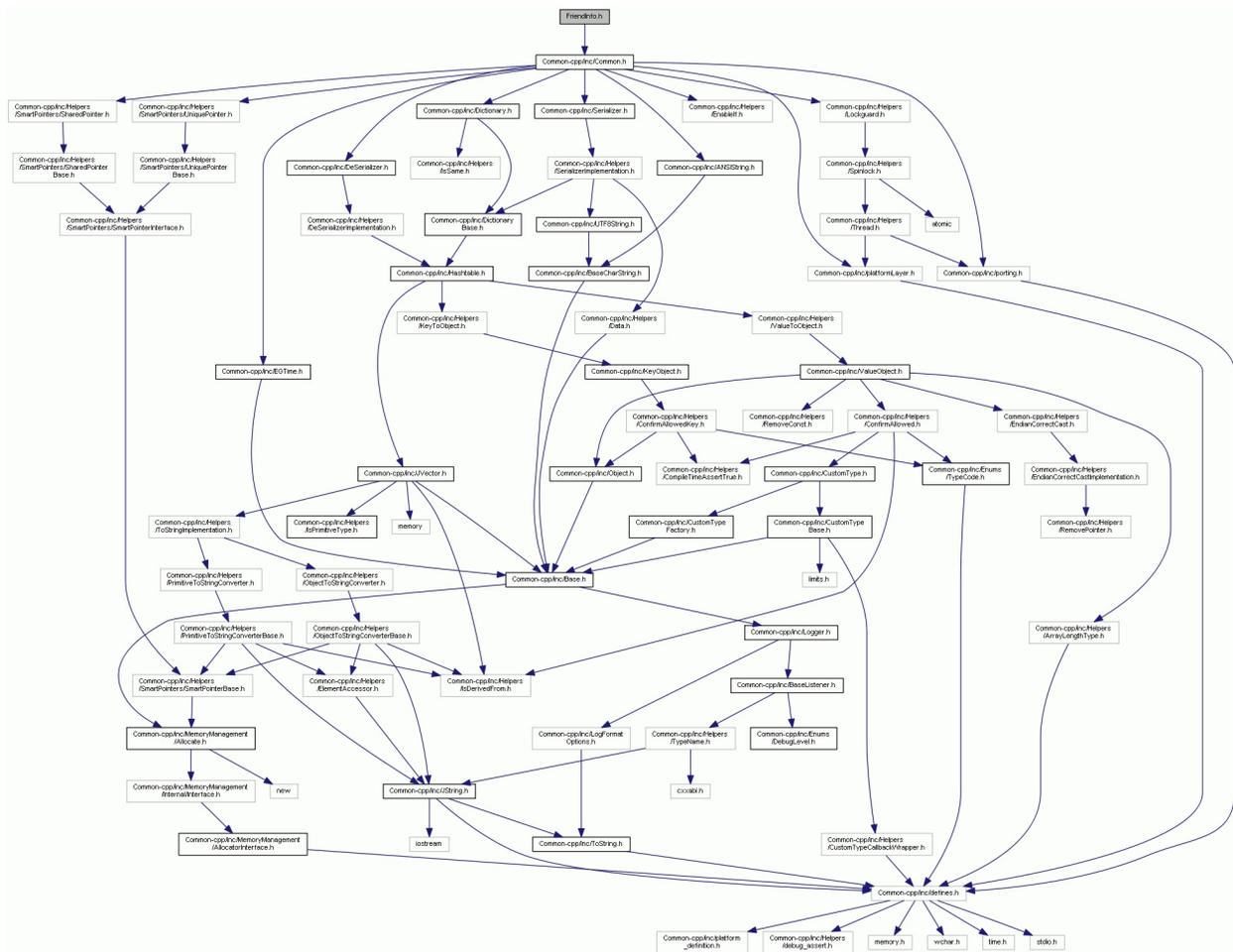
Client API 4.1.12.2

LoadBalancing-cpp > inc >

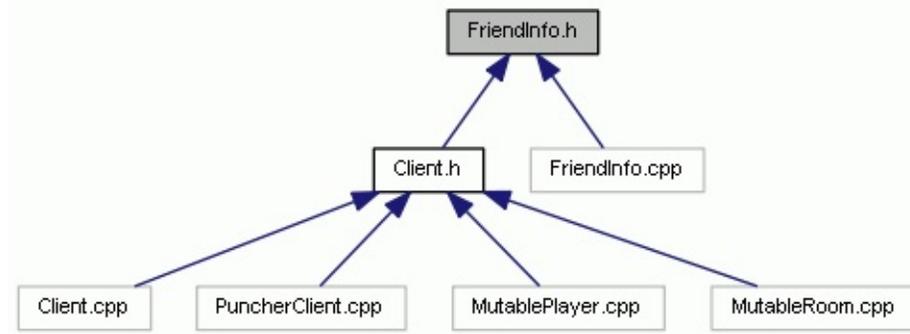
[Classes](#) | [Namespaces](#)

FriendInfo.h File Reference

Include dependency graph for FriendInfo.h:



This graph shows which files directly or indirectly include this file:



Classes

class **FriendInfo**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

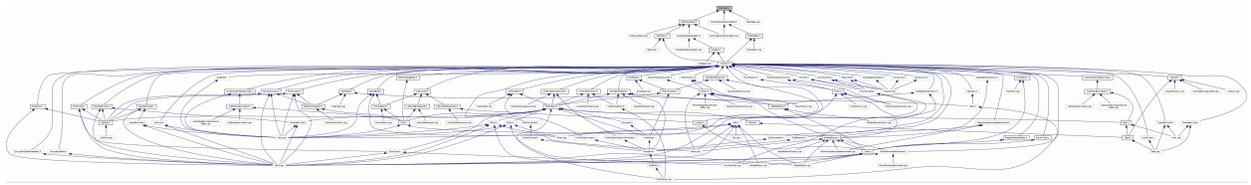
Client API 4.1.12.2

Common-cpp > inc >

[Classes](#) | [Namespaces](#)

Hashtable.h File Reference

Include dependency graph for Hashtable.h:



Classes

class **Hashtable**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

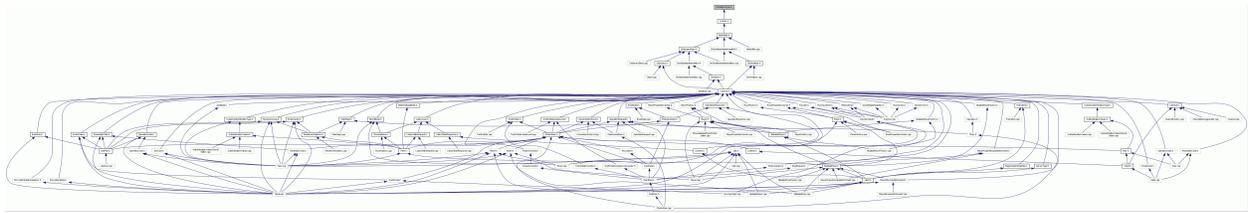
Client API 4.1.12.2

Common-cpp > inc > Helpers >

[Namespaces](#)

IsPrimitiveType.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

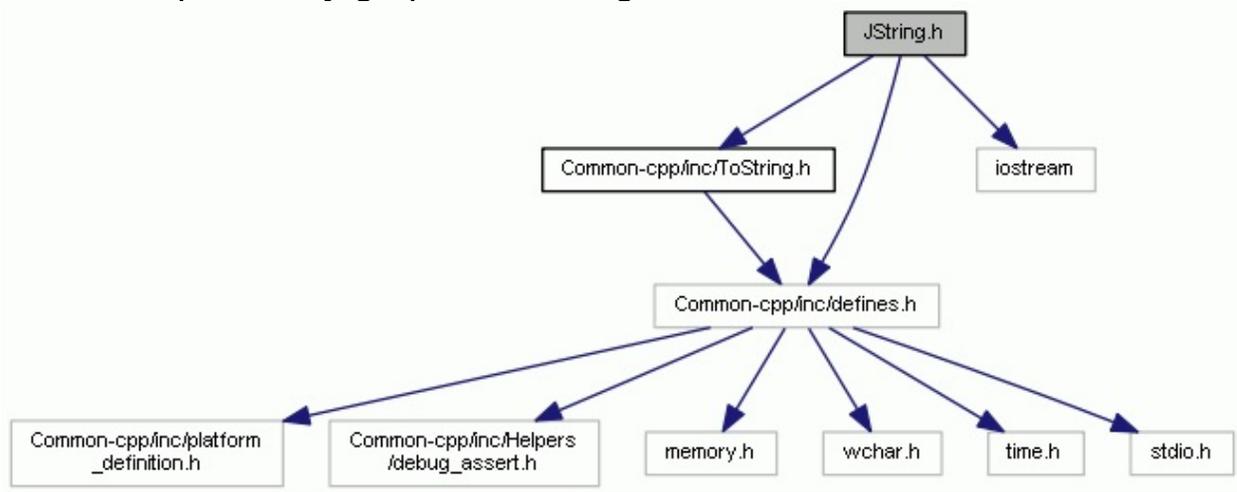
ExitGames::Common

© Exit Games ®, all rights reserved.

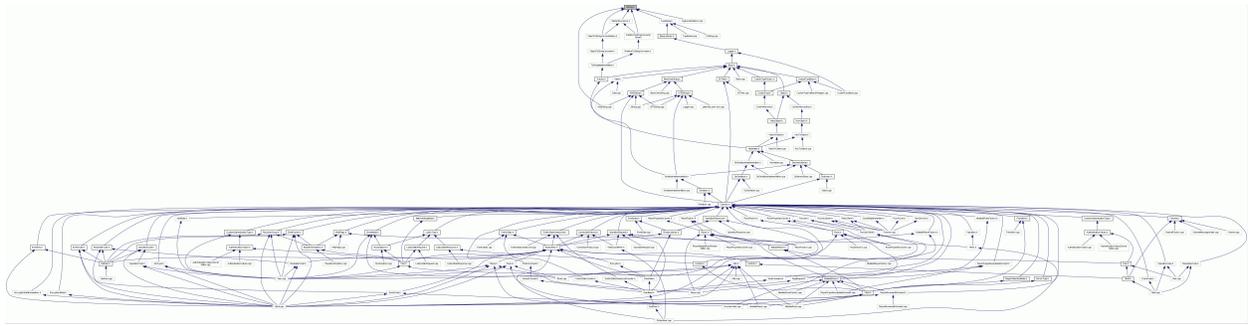
[Photon Documentation](#) | [Contact](#) | [Terms](#)

JString.h File Reference

Include dependency graph for JString.h:



This graph shows which files directly or indirectly include this file:



Classes

class **JString**

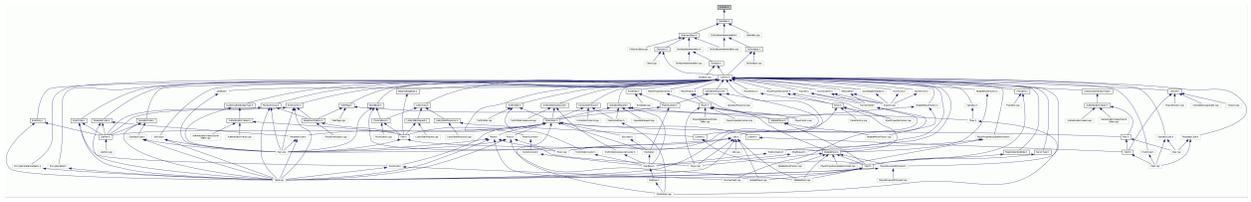
Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Classes

class **JVector< Etype >**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

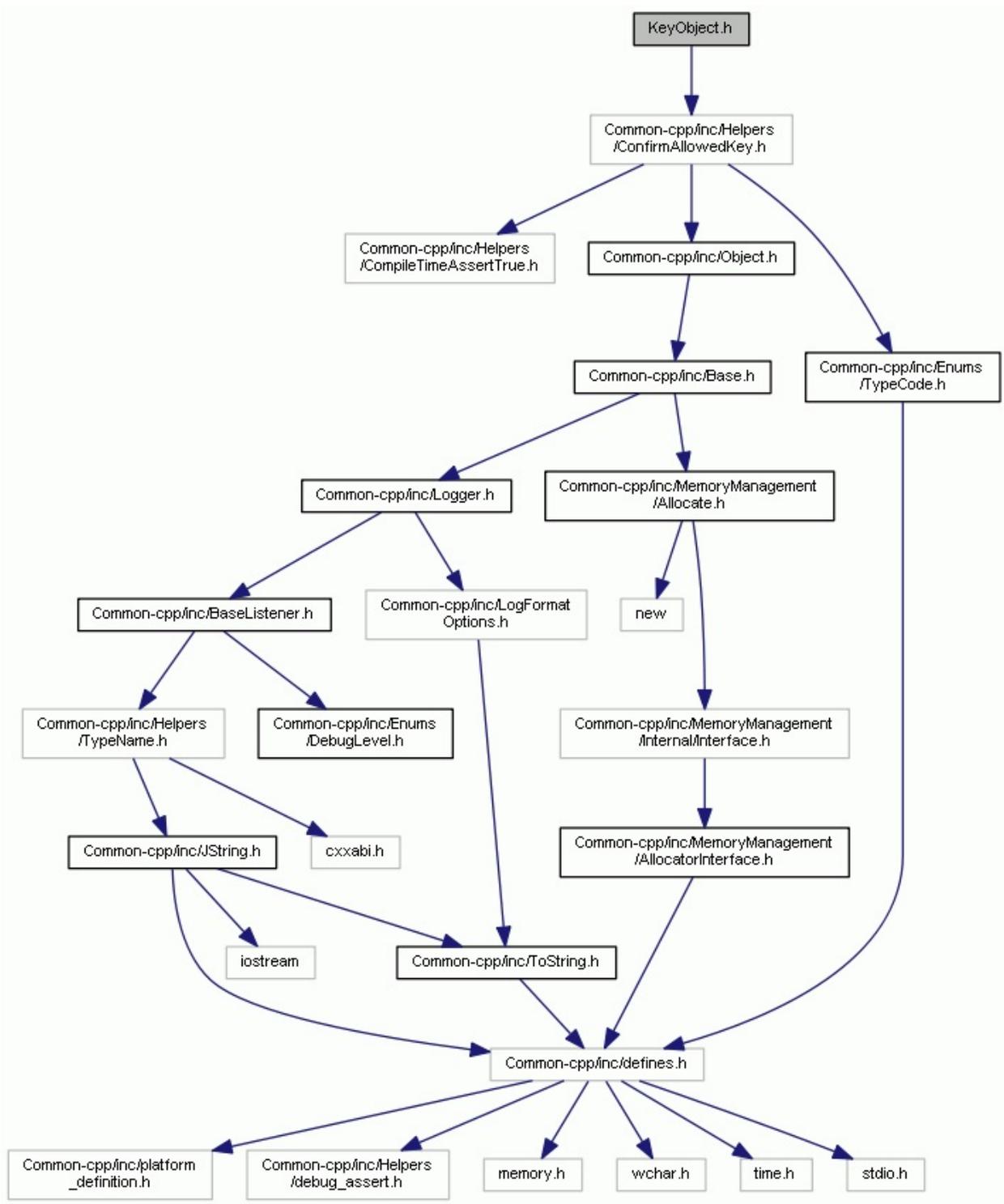
Client API 4.1.12.2

Common-cpp > inc >

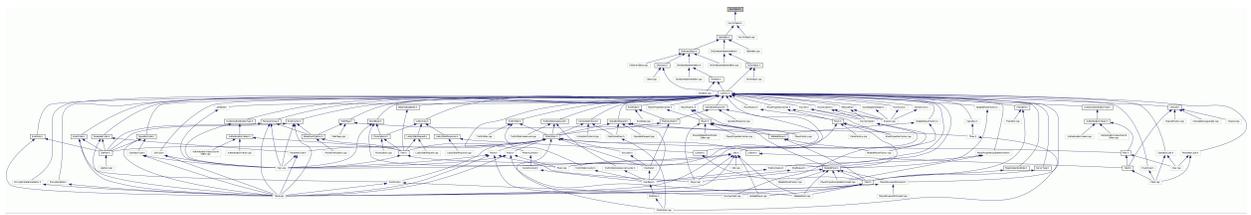
[Classes](#) | [Namespaces](#)

KeyObject.h File Reference

Include dependency graph for KeyObject.h:



This graph shows which files directly or indirectly include this file:



Classes

class **KeyObject**< Etype >

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

LoadBalancing-cpp/inc/Listener.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/Listener.h:

Classes

class **Listener**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Chat-cpp > inc >

[Classes](#) | [Namespaces](#)

Chat- cpp/inc/Listener.h File Reference

Include dependency graph for Chat-cpp/inc/Listener.h:

Classes

class **Listener**

Namespaces

ExitGames

ExitGames::Chat

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

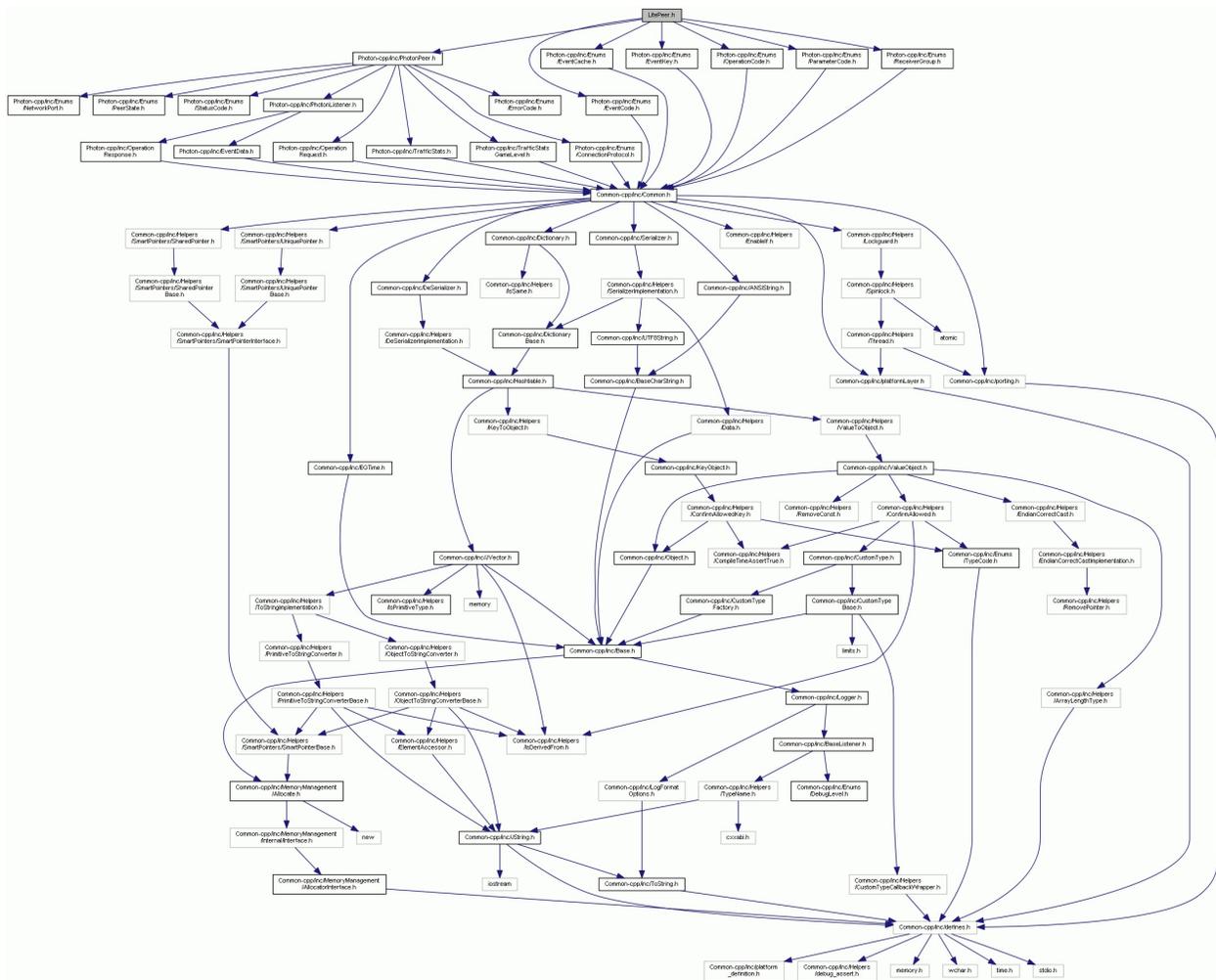
Client API 4.1.12.2

Photon-cpp > inc >

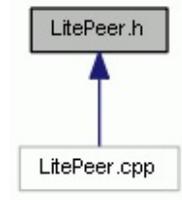
[Classes](#) | [Namespaces](#)

LitePeer.h File Reference

Include dependency graph for LitePeer.h:



This graph shows which files directly or indirectly include this file:



Classes

class **LitePeer**

Namespaces

ExitGames

ExitGames::Lite

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

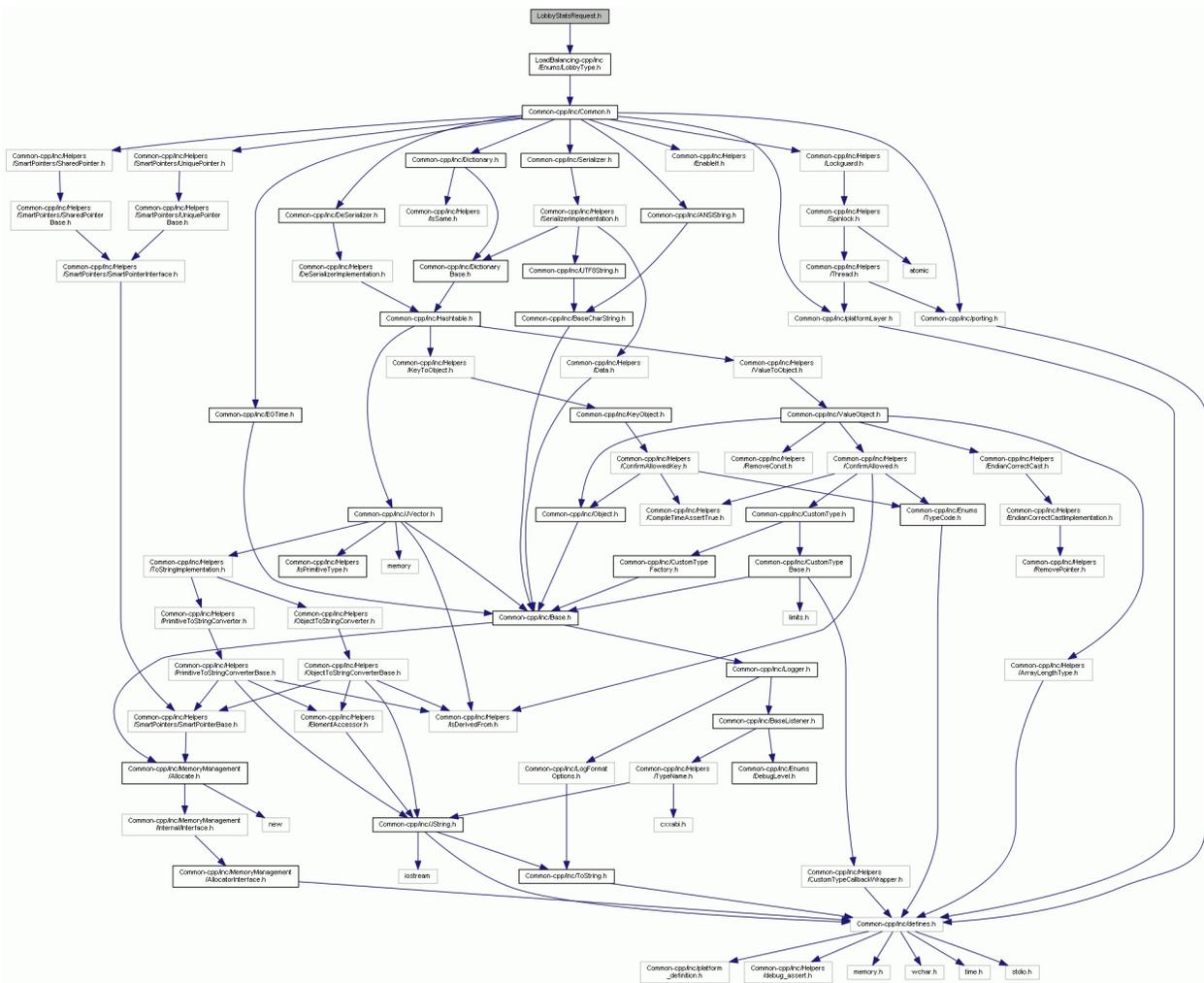
LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

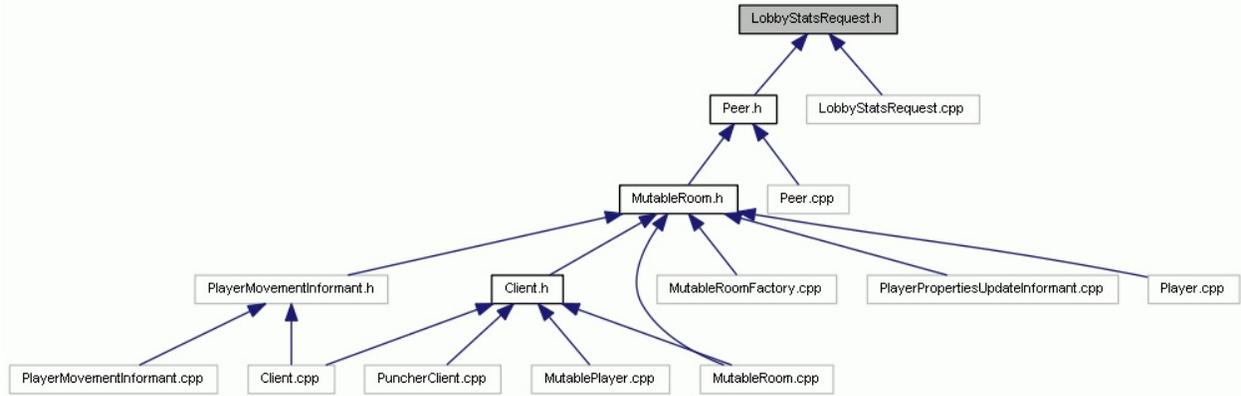
LobbyStatsRequest.h

File Reference

Include dependency graph for LobbyStatsRequest.h:



This graph shows which files directly or indirectly include this file:



Classes

class **LobbyStatsRequest**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

LobbyStatsResponse.h File Reference

Include dependency graph for LobbyStatsResponse.h:

Classes

class **LobbyStatsResponse**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[LoadBalancing-cpp](#)

[inc](#)

[Enums](#)

[Namespaces](#) | [Variables](#)

LobbyType.h File Reference

Include dependency graph for LobbyType.h:

Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::LobbyType

Variables

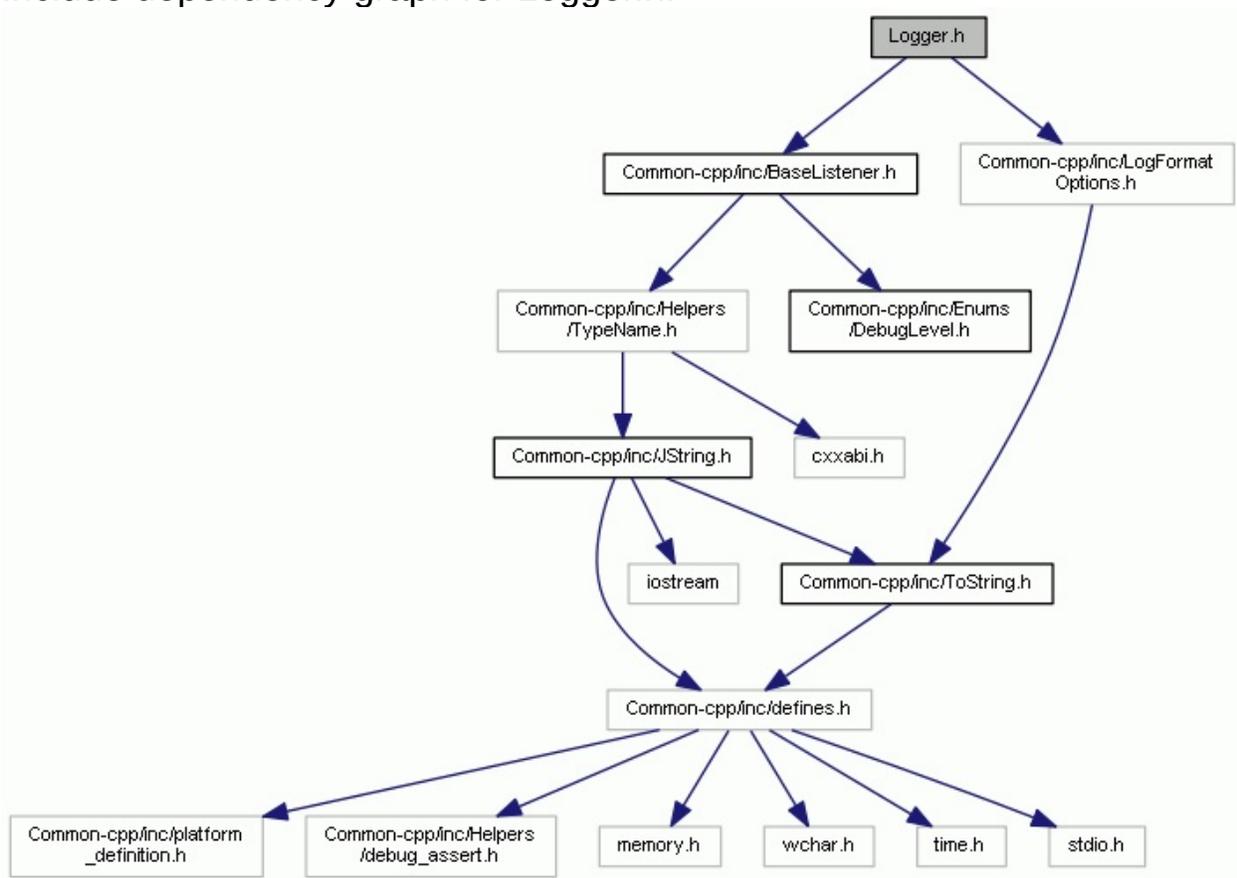
static const nByte **DEFAULT**
This lobby type is used unless another lobby type is specified. **Room** lists will be sent and **Client::opJoinRandomRoom()** can filter by matching properties.

static const nByte **SQL_LOBBY**
This lobby type lists rooms like type DEFAULT but SQL-like "where" clauses for filtering can be used with **Client::opJoinRandomRoom()**. This allows 'bigger', 'less', 'or' and 'and' combinations.

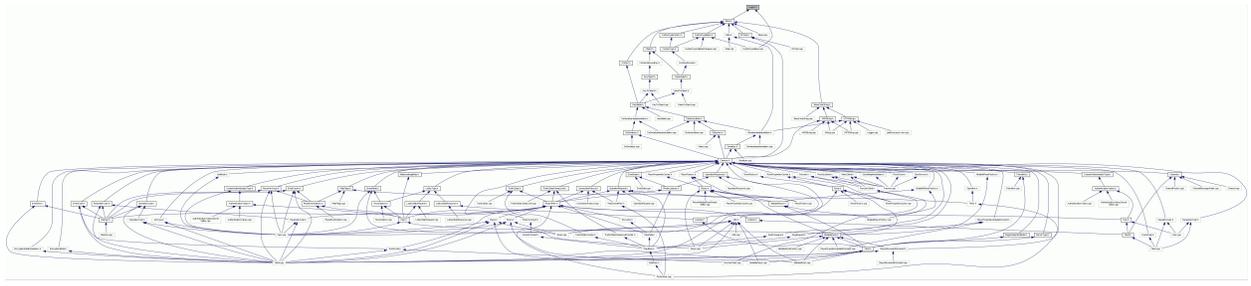
static const nByte **ASYNC_RANDOM_LOBBY**
This lobby does not send room lists. It is only used for **Client::opJoinRandomRoom()**. It keeps rooms available for matchmaking for a while even when there are only inactive users left.

Logger.h File Reference

Include dependency graph for Logger.h:



This graph shows which files directly or indirectly include this file:



Classes

```
class Logger
```

Namespaces

ExitGames

ExitGames::Common

Macros

```
#define EGLOG(debugLevel, ...)
```

Macro Definition Documentation

§ EGLOG

```
#define EGLOG ( debugLevel,  
              ...  
              )
```

With debug builds of the Photon client this macro will call **debugReturn()**, if the passed debug level is of the same or a higher priority than the one returned by **getDebugOutputLevel()**. In case of a call to **debugReturn()** it will pass a nicely formatted string consisting of the debug message, a timestamp of the calling time and the filename, function name and line number of the code, from which it has been called. With release builds this macro won't add any code to the resulting binary and therefor not do anything at all.

Remarks

EGLOG() always operates on a variable of type **Logger**, named **mLogger**. If no such variable is available in the scope of the call, then calling this macro won't succeed. Until a listener is specified for a logger instance, **EGLOG()** calls, that operate on that instance, won't log anything. What actually gets printed and to which output device (for example stdout/stderr or a certain file or stream) is up to the implementation of the specified listener.

Parameters

debugLevel the debug output level of the message, which in combination with the level, which is passed to **setDebugOutputLevel()** will determine, if the message will be passed to **debugReturn()** or not

... the debug format string + optional arguments (format specifiers for optional arguments work the same way like in the printf family of functions)

See also

getDebugOutputLevel(), **setDebugOutputLevel()**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

LoadBalancing-cpp

inc

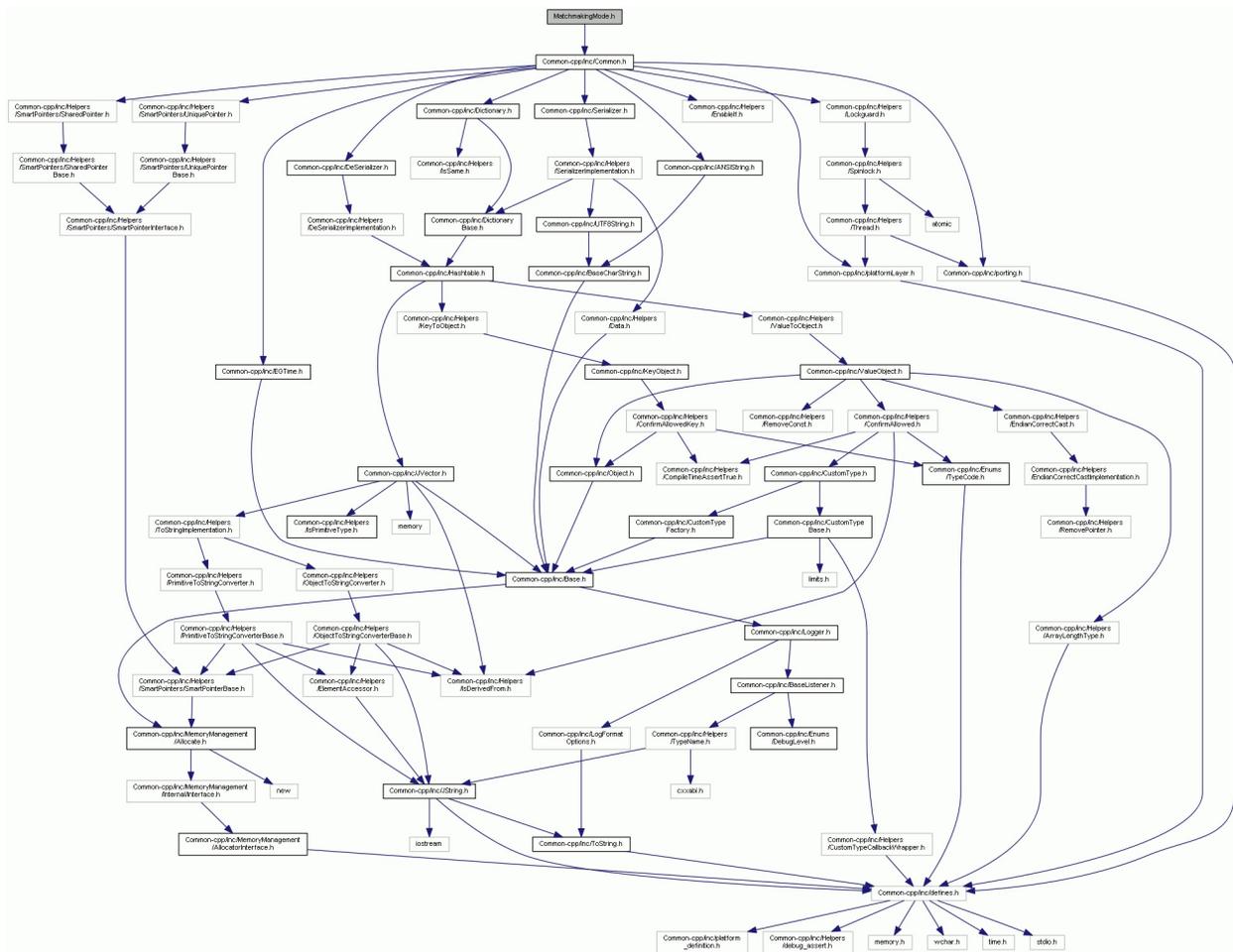
Enums

[Namespaces](#) | [Variables](#)

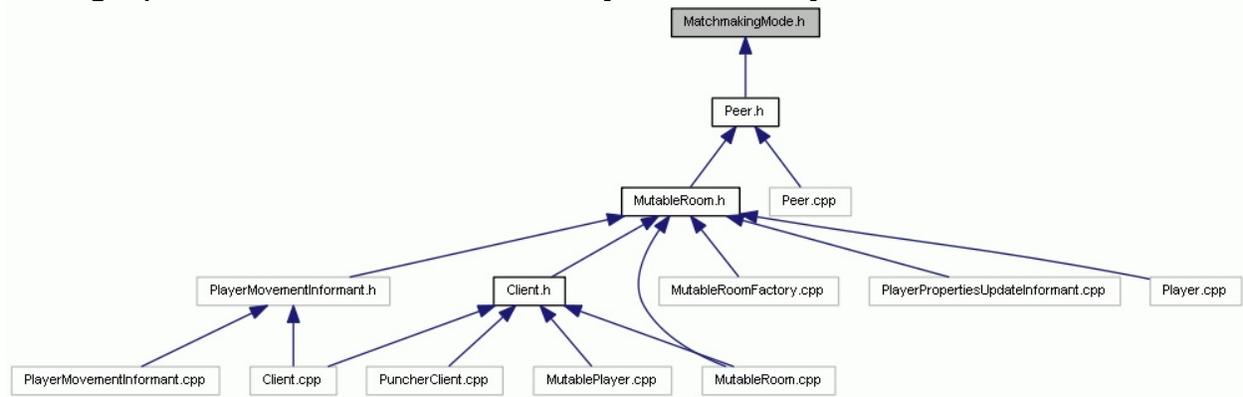
MatchmakingMode.h

File Reference

Include dependency graph for MatchmakingMode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::MatchmakingMode

Variables

static const nByte **FILL_ROOM**
Fills up rooms (oldest first) to get players together as fast as possible. Default. Makes most sense with MaxPlayers > 0 and games that can only start with more players.

static const nByte **SERIAL_MATCHING**
Distributes players across available rooms sequentially but takes filters into account. Without filters, rooms get players evenly distributed.

static const nByte **RANDOM_MATCHING**
Joins a (fully) random room. Expected properties must match, but aside from this, any available room might be selected.



Photon C++

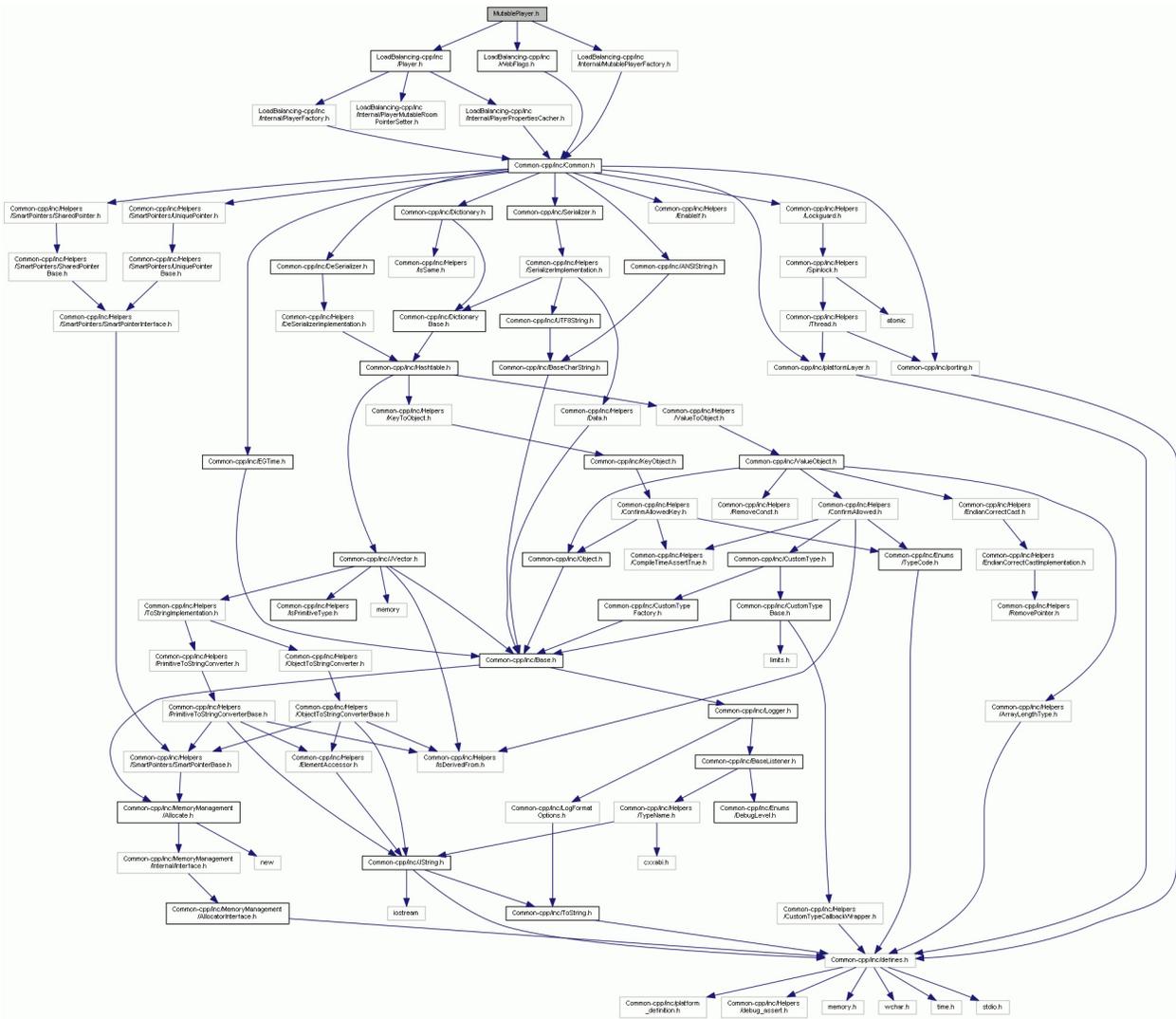
Client API 4.1.12.2

LoadBalancing-cpp > inc >

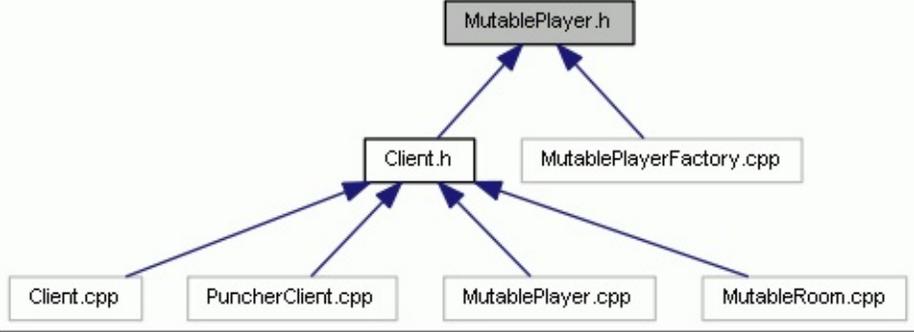
[Classes](#) | [Namespaces](#)

MutablePlayer.h File Reference

Include dependency graph for MutablePlayer.h:



This graph shows which files directly or indirectly include this file:



Classes

class **MutablePlayer**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

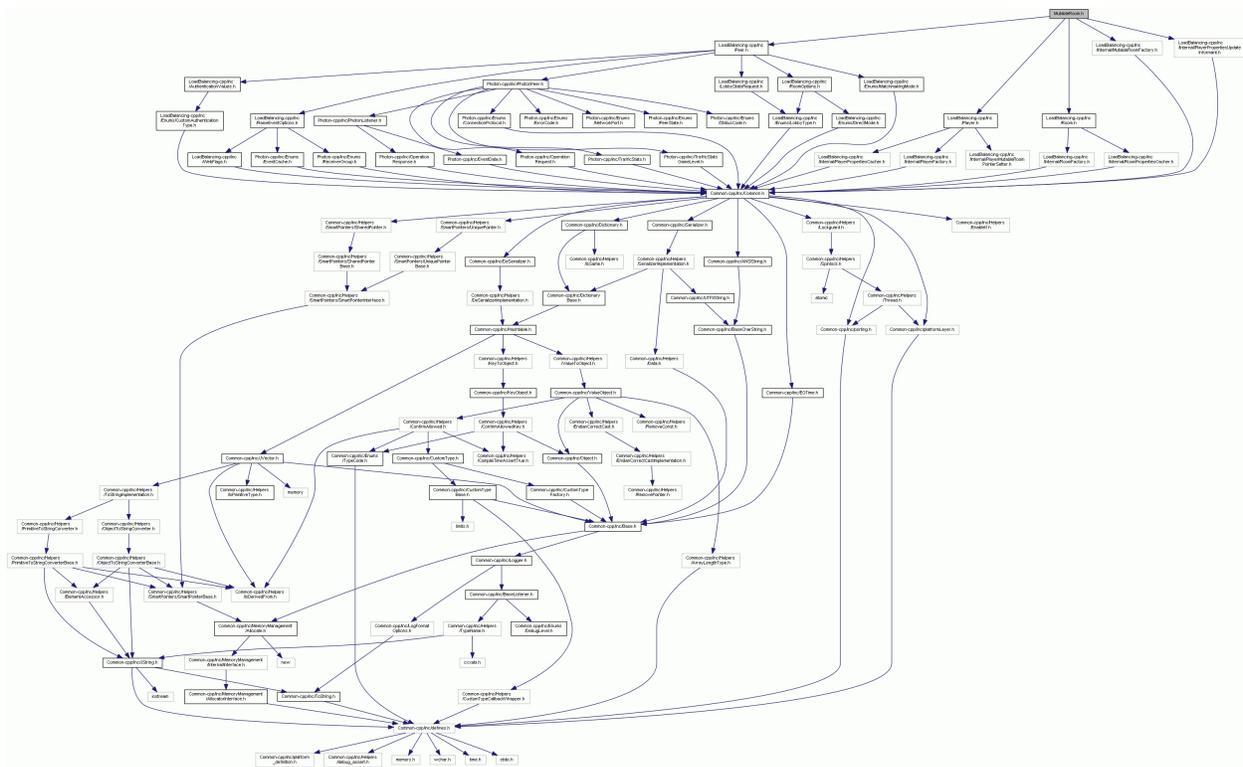
Client API 4.1.12.2

LoadBalancing-cpp > inc >

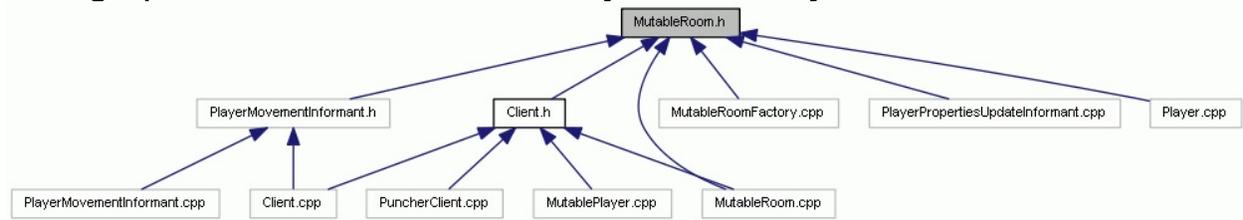
[Classes](#) | [Namespaces](#)

MutableRoom.h File Reference

Include dependency graph for MutableRoom.h:



This graph shows which files directly or indirectly include this file:



Classes

```
class MutableRoom
```

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Classes

struct **Protocol**

struct **UDP**

struct **UDPAlternative**

struct **TCP**

struct **WS**

struct **WSS**

Namespaces

ExitGames

ExitGames::Photon

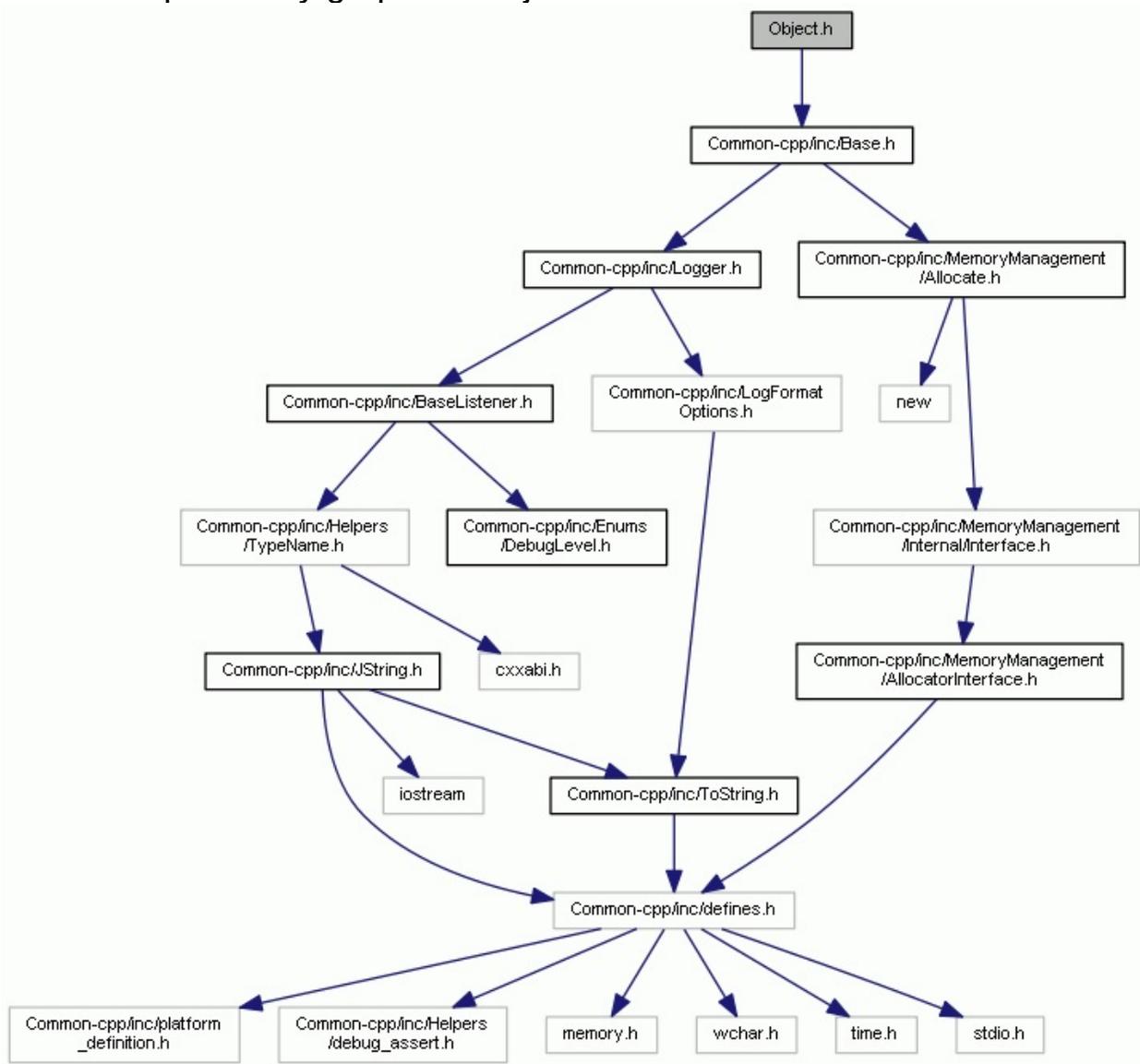
ExitGames::Photon::NetworkPort

© Exit Games ®, all rights reserved.

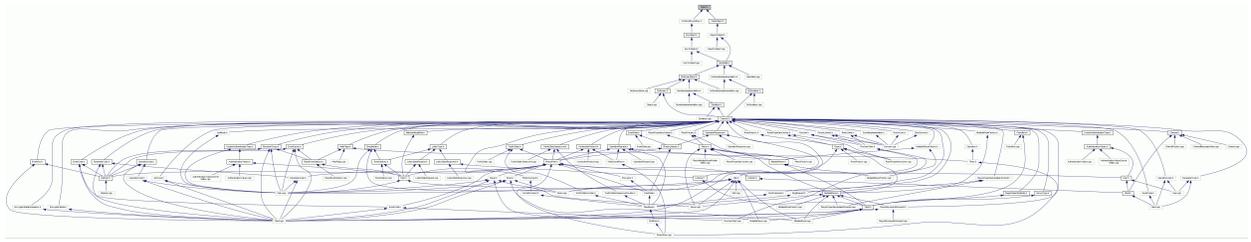
[Photon Documentation](#) | [Contact](#) | [Terms](#)

Object.h File Reference

Include dependency graph for Object.h:



This graph shows which files directly or indirectly include this file:



Classes

```
class Object
```

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

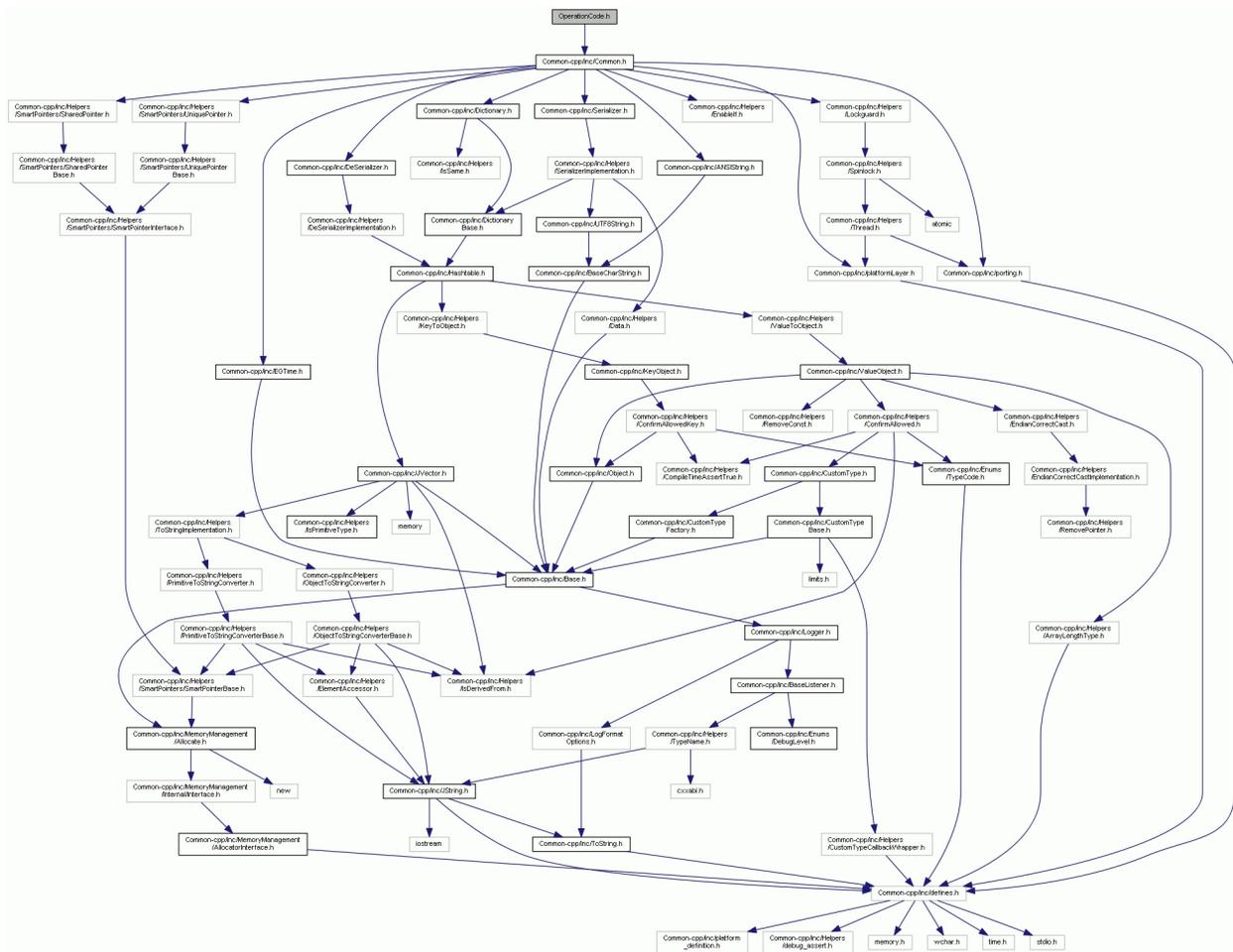
Client API 4.1.12.2

[Photon-cpp](#) > [inc](#) > [Enums](#)

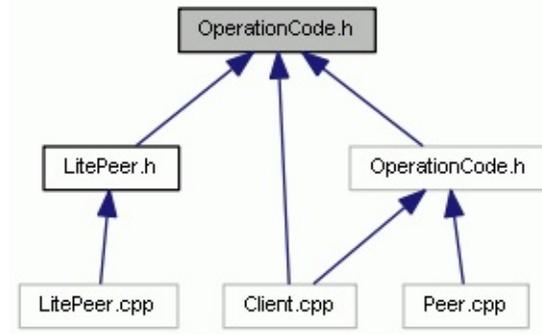
[Namespaces](#) | [Variables](#)

Photon- cpp/inc/Enums/OperationCode.h File Reference

Include dependency graph for Photon-cpp/inc/Enums/OperationCode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::OperationCode

Variables

static const nByte **JOIN**

static const nByte **LEAVE**

static const nByte **RAISE_EV**

static const nByte **SETPROPERTIES**

static const nByte **GETPROPERTIES**

static const nByte **CHANGE_GROUPS**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Photon-cpp > inc >

[Classes](#) | [Namespaces](#) | [Typedefs](#)

OperationRequest.h

File Reference

Include dependency graph for OperationRequest.h:

Classes

class **OperationRequest**

Namespaces

ExitGames

ExitGames::Photon

Typedefs

```
typedef Common::Dictionary< nByte, Common::Object > OperationReq
```

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Photon-cpp > inc >

[Classes](#) | [Namespaces](#)

OperationResponse.h

File Reference

Include dependency graph for OperationResponse.h:

Classes

```
class OperationResponse
```

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

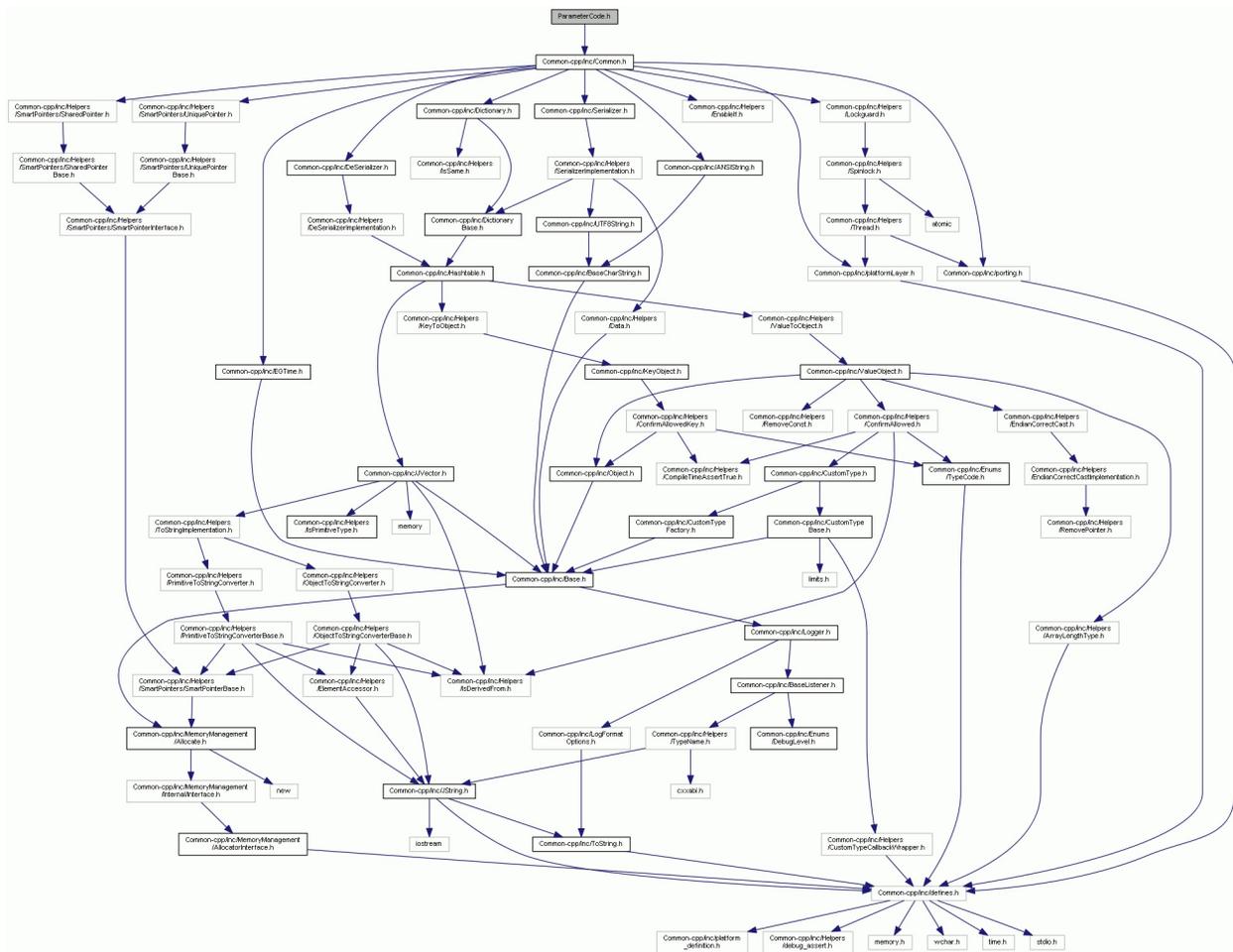
Client API 4.1.12.2

[Photon-cpp](#) > [inc](#) > [Enums](#)

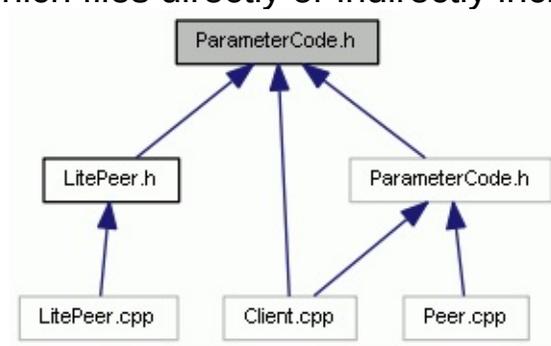
[Namespaces](#) | [Variables](#)

Photon-cpp/inc/Enums/ParameterCode.h File Reference

Include dependency graph for Photon-cpp/inc/Enums/ParameterCode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::ParameterCode

Variables

static const nByte **GAMEID**

static const nByte **ACTORNR**

static const nByte **TARGET_ACTORNR**

static const nByte **ACTOR_LIST**

static const nByte **PROPERTIES**

static const nByte **BROADCAST**

static const nByte **ACTOR_PROPERTIES**

static const nByte **GAME_PROPERTIES**

static const nByte **CACHE**

static const nByte **RECEIVER_GROUP**

static const nByte **DATA**

static const nByte **CODE**

static const nByte **GROUP**

static const nByte **REMOVE**

static const nByte **ADD**



Photon C++

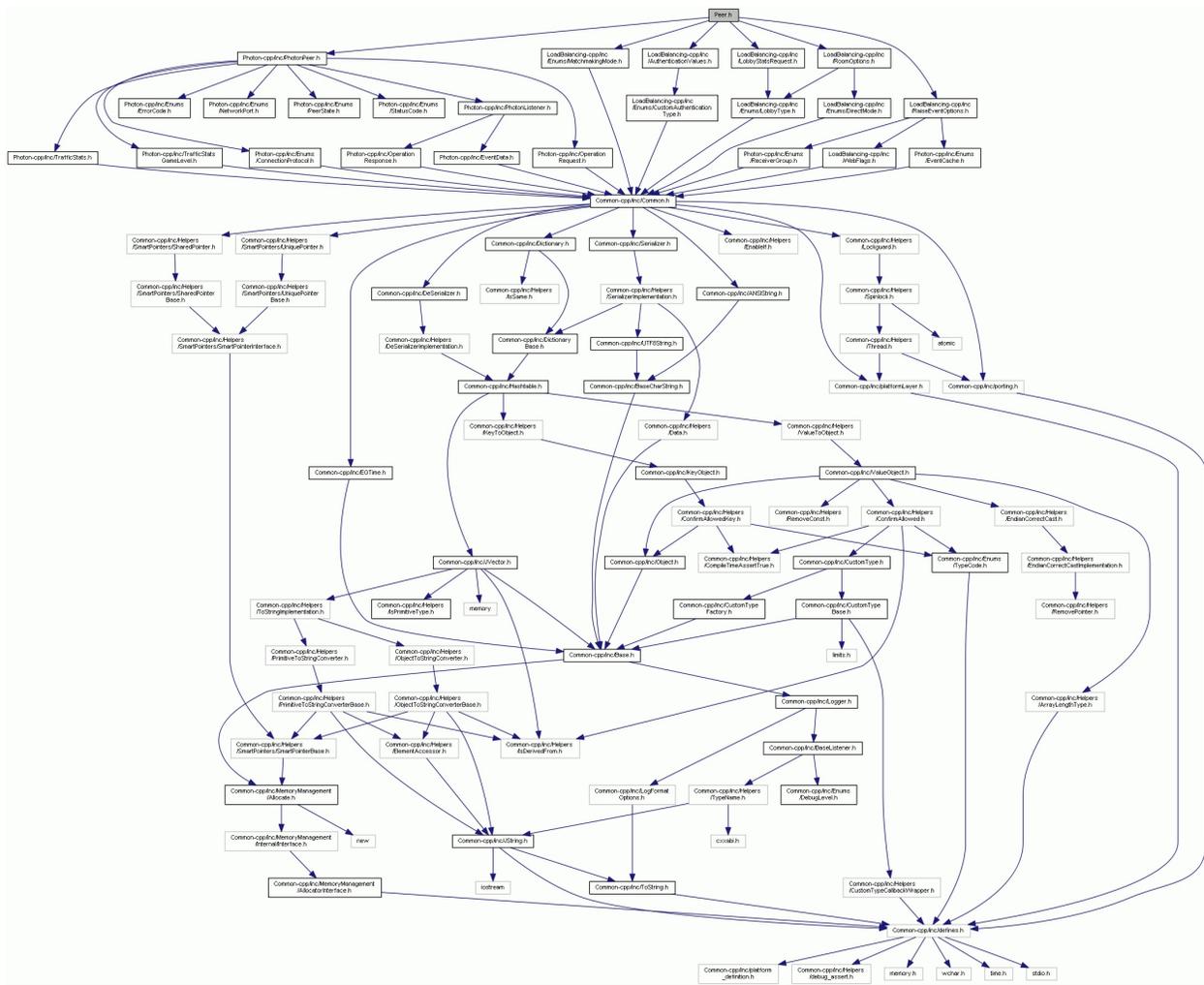
Client API 4.1.12.2

LoadBalancing-cpp > inc >

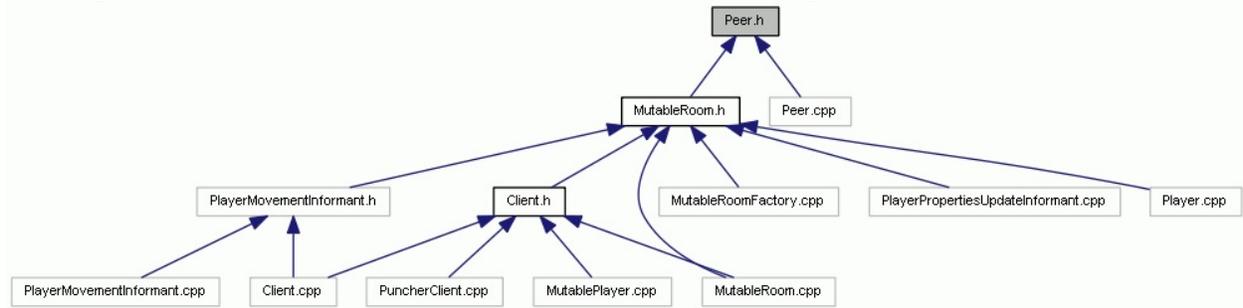
[Classes](#) | [Namespaces](#)

LoadBalancing-cpp/inc/Peer.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/Peer.h:



This graph shows which files directly or indirectly include this file:



Classes

```
class Peer
```

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Chat-cpp > inc >

[Classes](#) | [Namespaces](#)

Chat-cpp/inc/Peer.h

File Reference

Include dependency graph for Chat-cpp/inc/Peer.h:

Classes

class **Peer**

Namespaces

ExitGames

ExitGames::Chat

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

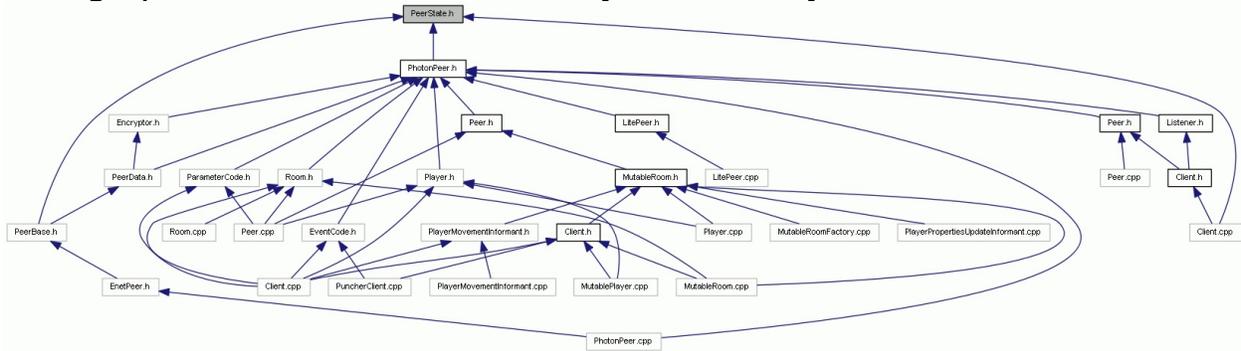
Client API 4.1.12.2

Photon-cpp > inc > Enums

[Namespaces](#) | [Variables](#)

PeerState.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Photon

ExitGames::Photon::PeerState

Variables

static const int **DISCONNECTED**
The peer is disconnected and can't call Operations. Call PhotonPeer_connect().

static const int **CONNECTING**
The peer is establishing the connection: opening a socket, exchanging packages with **Photon**.

static const int **INITIALIZING_APPLICATION**
The connection is established and now sends the application name to **Photon**. You set the "application name" by calling PhotonPeer_connect().

static const int **CONNECTED**
The peer is connected and initialized (selected an application). You can now use operations.

static const int **DISCONNECTING**
The peer is disconnecting. It sent a disconnect to the server, which will acknowledge closing the connection.

Photon C++

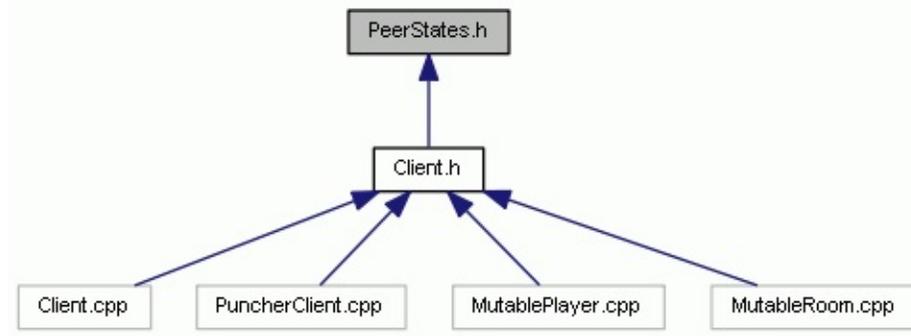
Client API 4.1.12.2

LoadBalancing-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

PeerStates.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

ExitGames::LoadBalancing::PeerStates

Variables

static const int **Uninitialized**

static const int **PeerCreated**

static const int **ConnectingToNameserver**

static const int **ConnectedToNameserver**

static const int **DisconnectingFromNameserver**

static const int **Connecting**

static const int **Connected**

static const int **WaitingForCustomAuthenticationNextStepCall**

static const int **Authenticated**

static const int **JoinedLobby**

static const int **DisconnectingFromMasterserver**

static const int **ConnectingToGameserver**

static const int **ConnectedToGameserver**

static const int **AuthenticatedOnGameServer**

static const int **Joining**

static const int **Joined**

static const int **Leaving**

static const int **Left**

static const int **DisconnectingFromGameserver**

static const int **ConnectingToMasterserver**

static const int **ConnectedComingFromGameserver**

static const int **AuthenticatedComingFromGameserver**

static const int **Disconnecting**

static const int **Disconnected**

© [Exit Games](#) ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

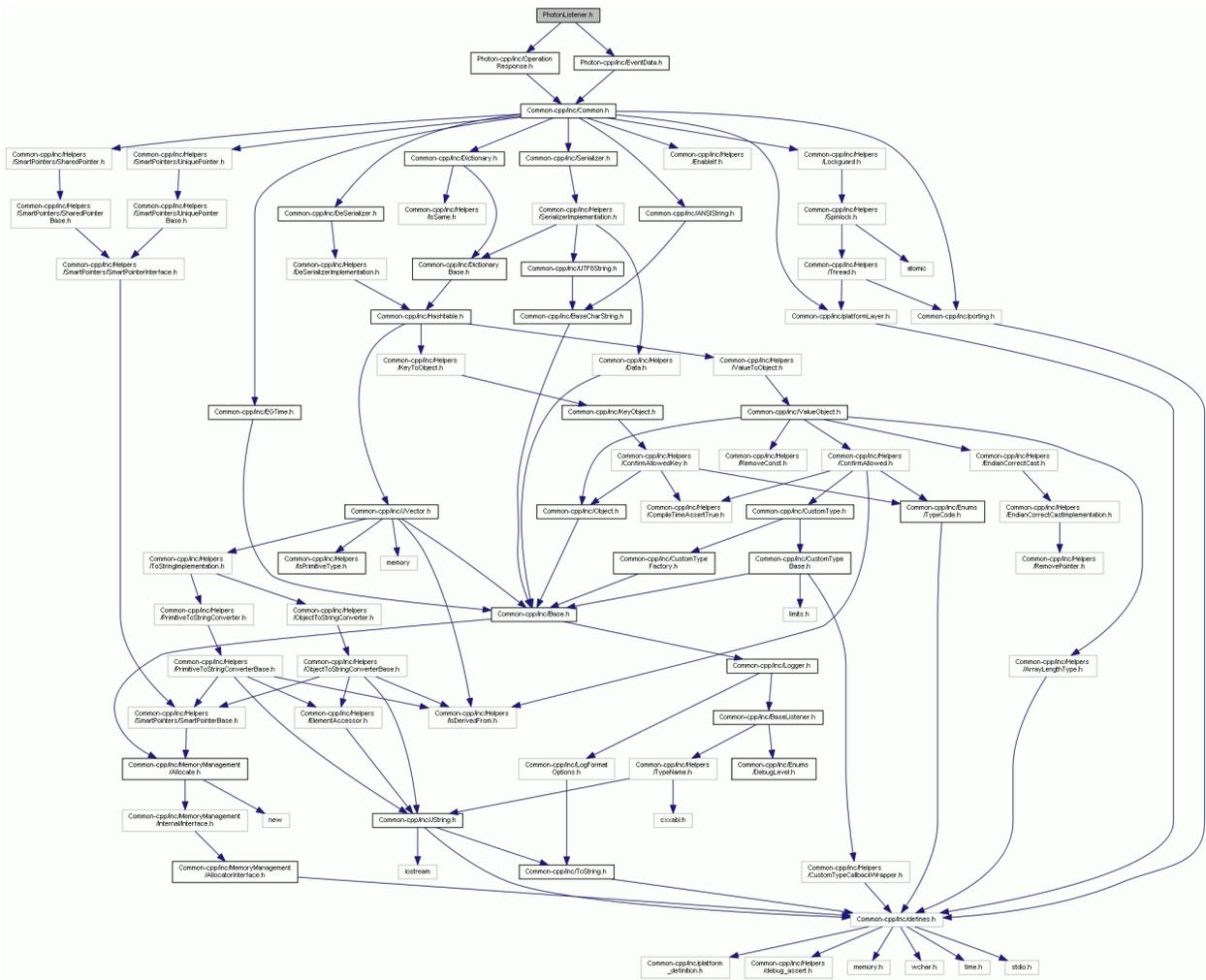
Client API 4.1.12.2

Photon-cpp > inc >

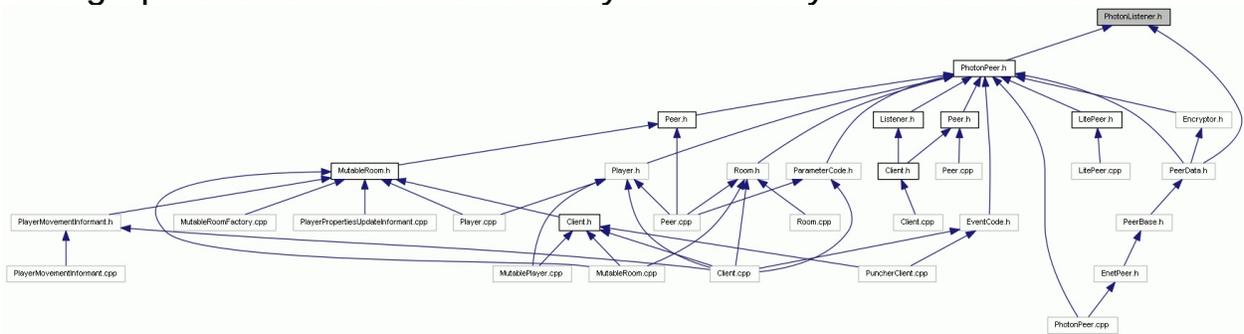
[Classes](#) | [Namespaces](#)

PhotonListener.h File Reference

Include dependency graph for PhotonListener.h:



This graph shows which files directly or indirectly include this file:



Classes

class **PhotonListener**

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Photon-cpp > inc >

[Classes](#) | [Namespaces](#) | [Macros](#)

PhotonPeer.h File Reference

Include dependency graph for PhotonPeer.h:

Classes

class **PhotonPeer**

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

Player.h File Reference

Include dependency graph for Player.h:

Classes

```
class Player
```

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

RaiseEventOptions.h

File Reference

Include dependency graph for RaiseEventOptions.h:

Classes

class **RaiseEventOptions**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[Photon-cpp](#)

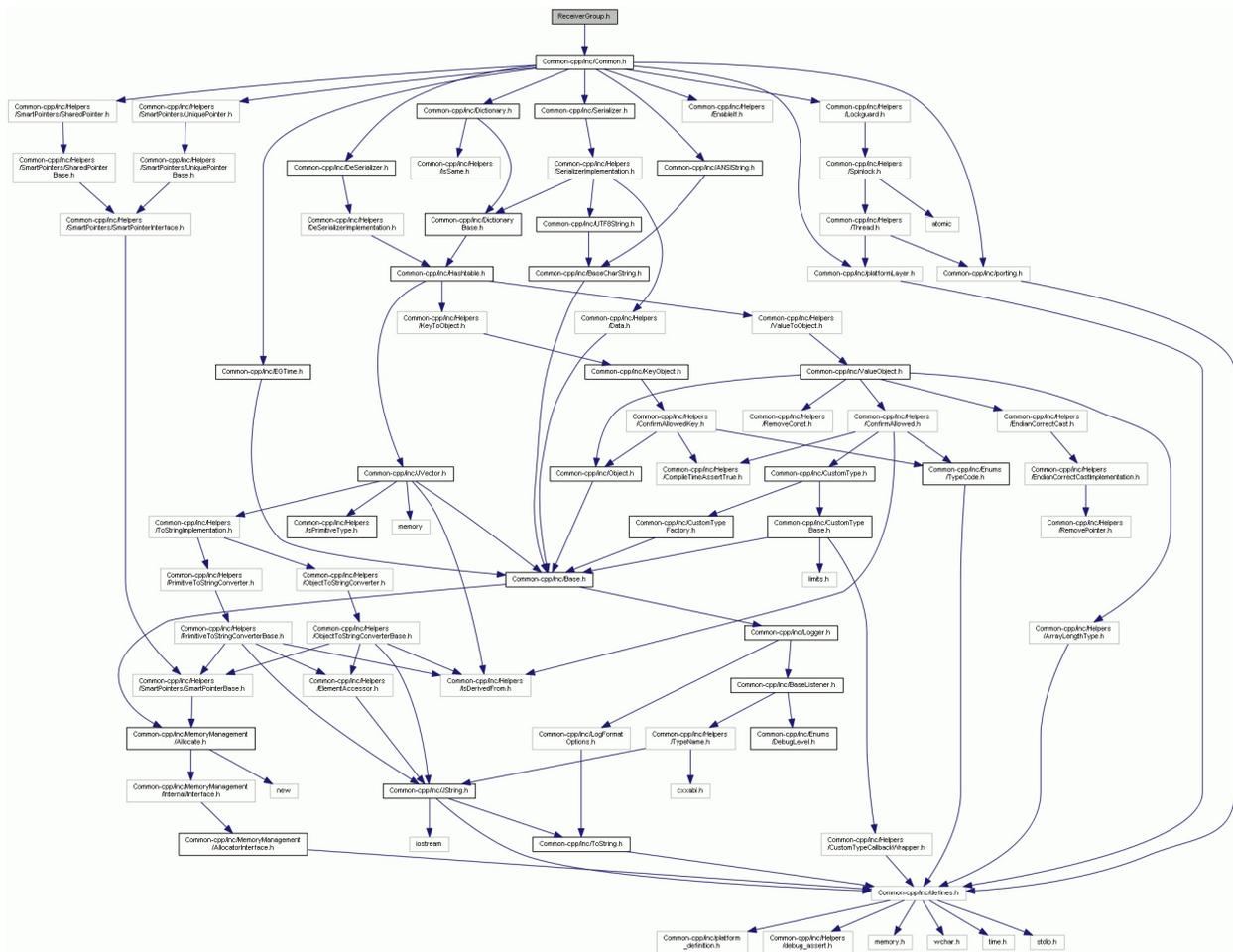
[inc](#)

[Enums](#)

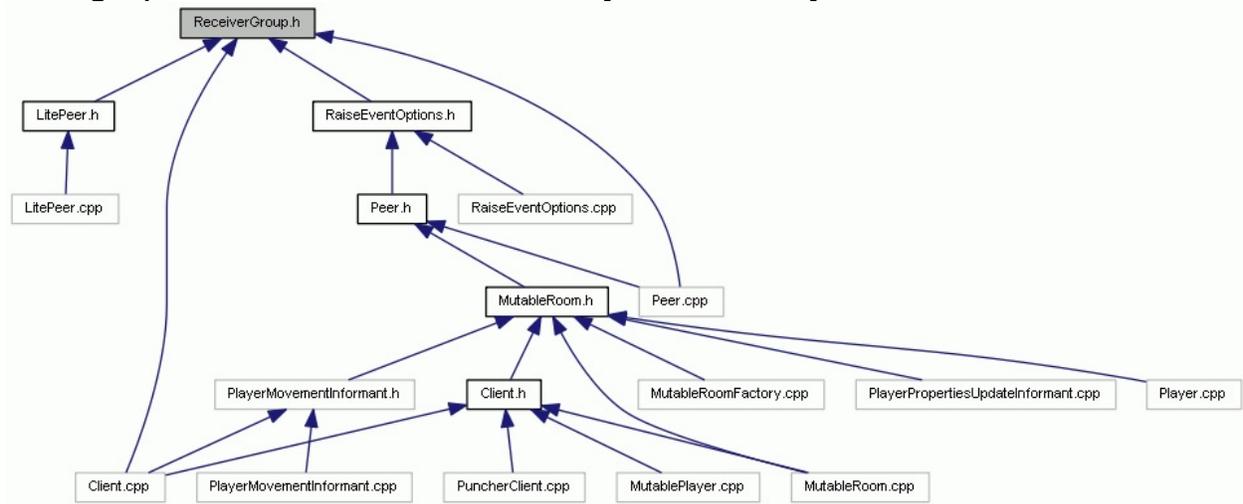
[Namespaces](#) | [Variables](#)

ReceiverGroup.h File Reference

Include dependency graph for ReceiverGroup.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Lite

ExitGames::Lite::ReceiverGroup

Variables

static const nByte **OTHERS**

static const nByte **ALL**

static const nByte **MASTER_CLIENT**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[LoadBalancing-cpp](#)

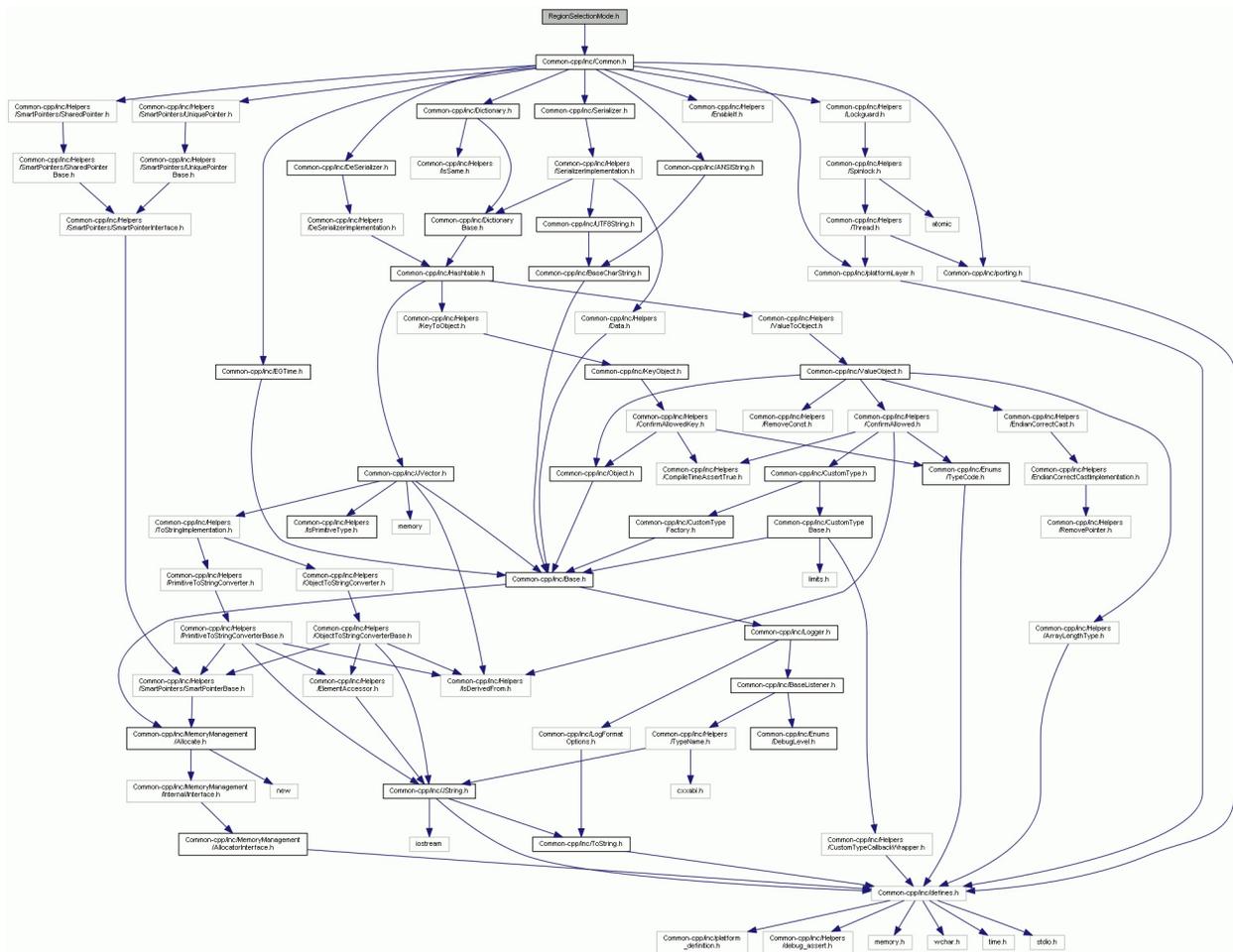
[inc](#)

[Enums](#)

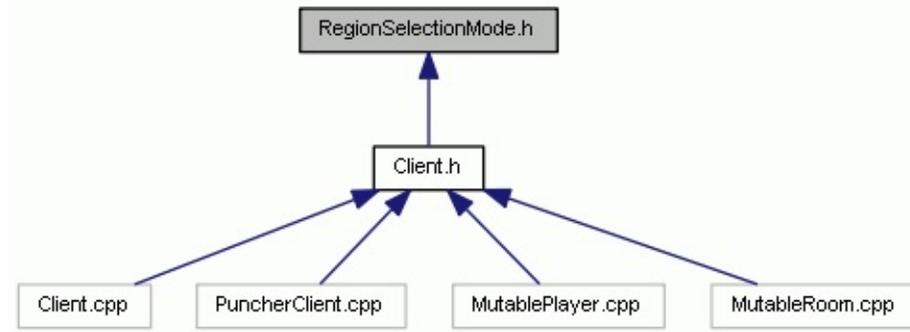
[Namespaces](#) | [Variables](#)

RegionSelectionMode.h File Reference

Include dependency graph for RegionSelectionMode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

Variables

static const nByte **DEFAULT**

static const nByte **SELECT**

static const nByte **BEST**

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

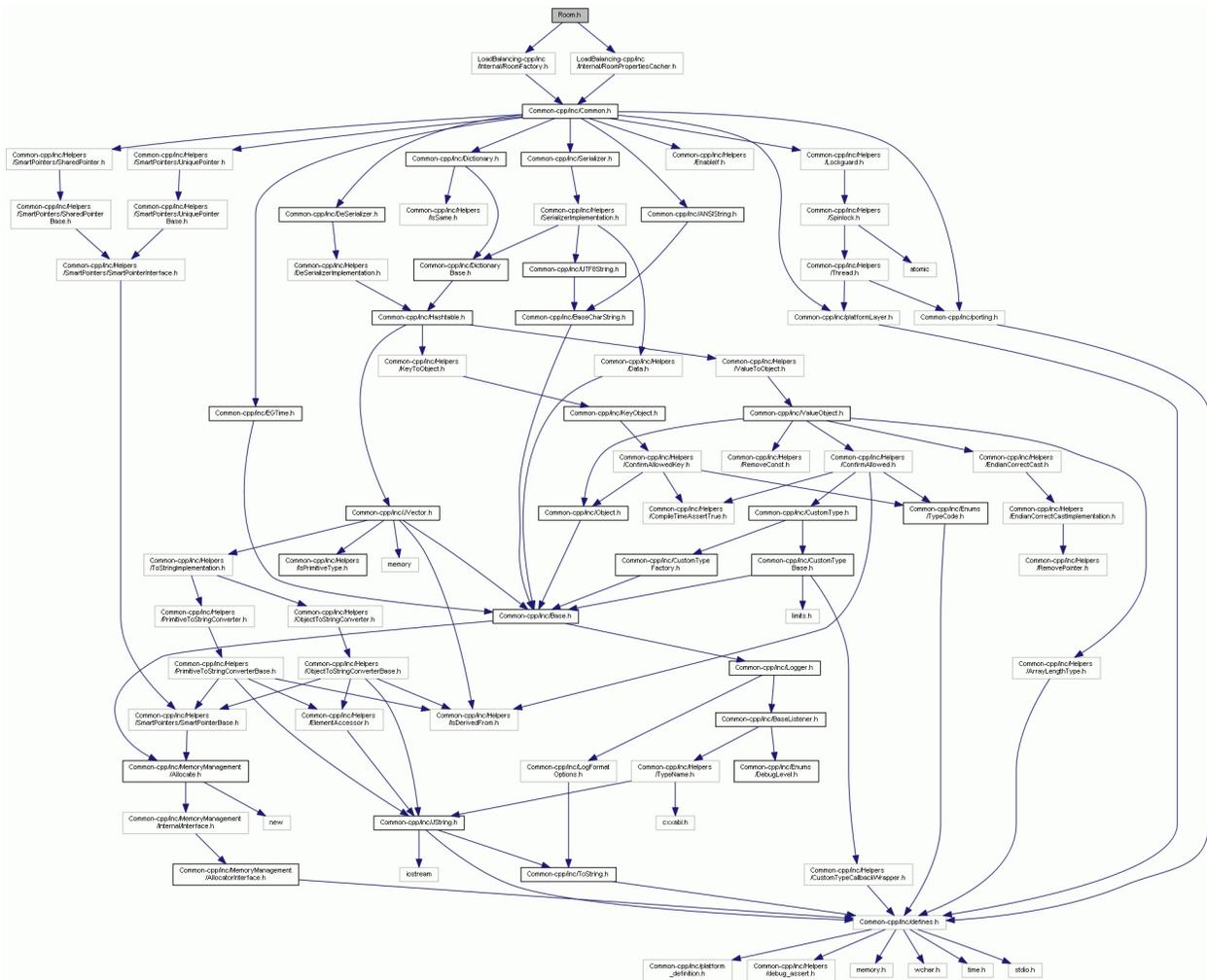
Client API 4.1.12.2

LoadBalancing-cpp > inc >

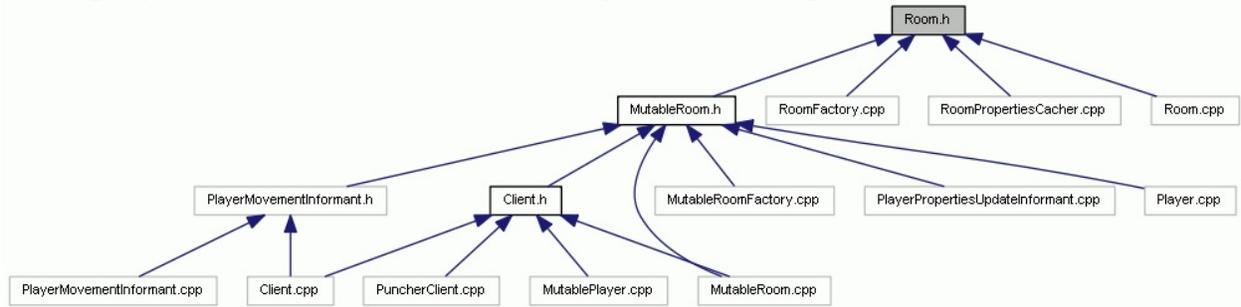
[Classes](#) | [Namespaces](#)

Room.h File Reference

Include dependency graph for Room.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Room**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

LoadBalancing-cpp > inc >

[Classes](#) | [Namespaces](#)

RoomOptions.h File Reference

Include dependency graph for RoomOptions.h:

Classes

class **RoomOptions**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

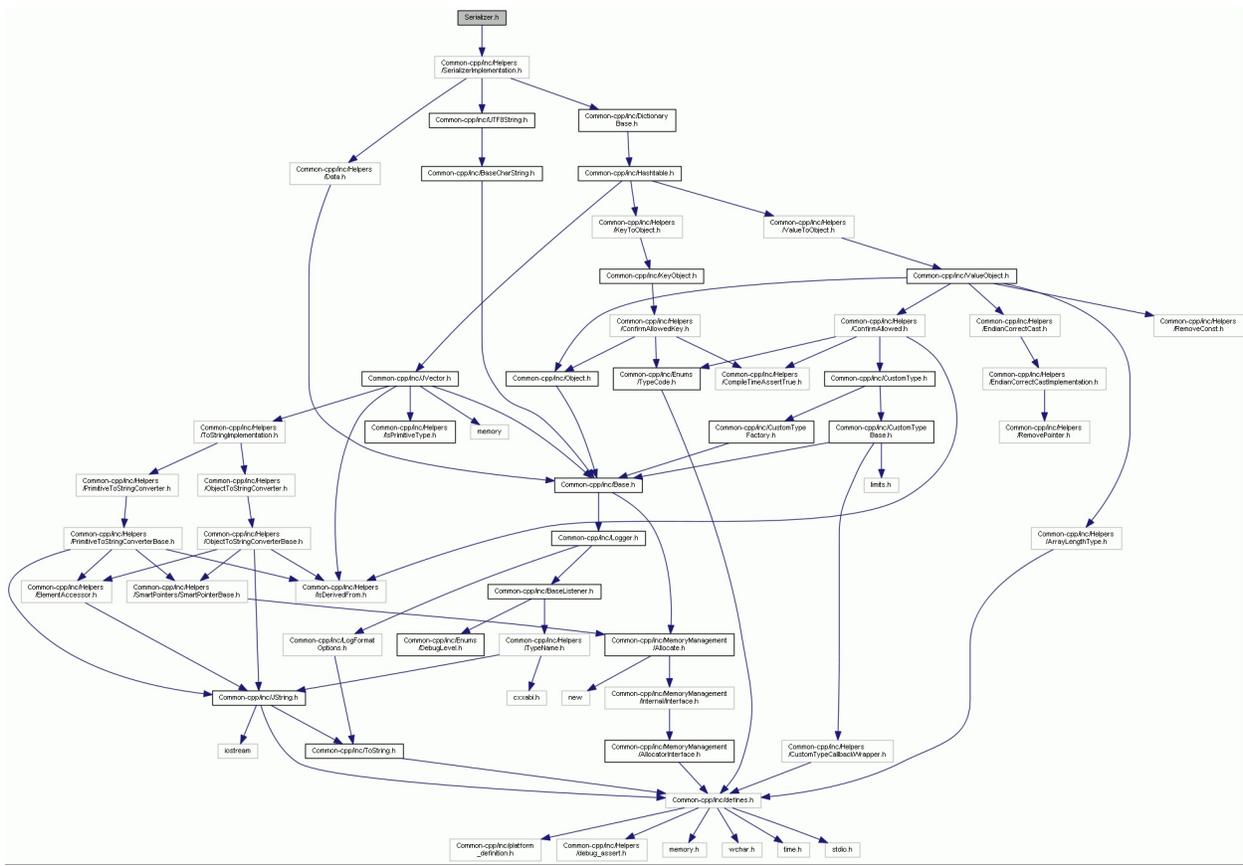
Client API 4.1.12.2

Common-cpp > inc >

[Classes](#) | [Namespaces](#)

Serializer.h File Reference

Include dependency graph for Serializer.h:



This graph shows which files directly or indirectly include this file:



Classes

class **Serializer**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

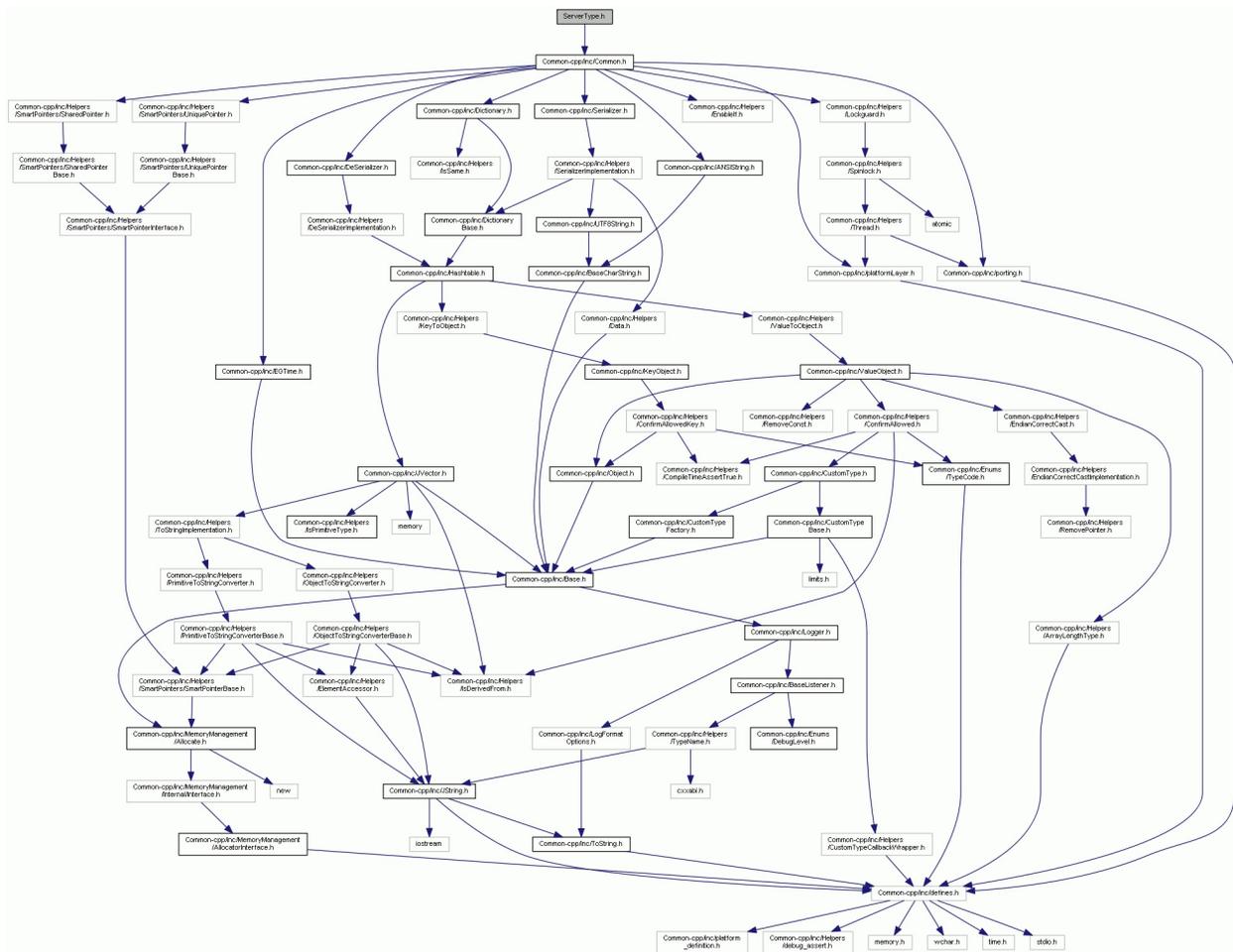
Client API 4.1.12.2

LoadBalancing-cpp > inc > Enums >

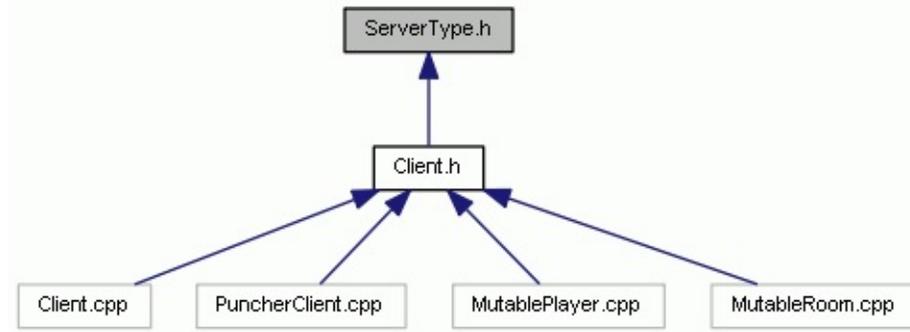
[Namespaces](#) | [Variables](#)

ServerType.h File Reference

Include dependency graph for ServerType.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::LoadBalancing

Variables

```
static const nByte NAME_SERVER
```

```
static const nByte MASTER_SERVER
```

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Namespaces

ExitGames

ExitGames::Photon

ExitGames::Photon::StatusCode

Variables

static const int **EXCEPTION_ON_CONNECT**
the **PhotonPeer** encountered an exception while opening the incoming connection to the server. The server could be down / not running.

static const int **CONNECT**
the **PhotonPeer** is connected.

static const int **DISCONNECT**
the **PhotonPeer** just disconnected.

static const int **EXCEPTION**
the **PhotonPeer** encountered an exception and will disconnect, too.

static const int **QUEUE_OUTGOING_RELIABLE_WARNING**
PhotonPeer outgoing queue is filling up. send more often.

static const int **QUEUE_OUTGOING_UNRELIABLE_WARNING**
PhotonPeer outgoing queue is filling up. send more often.

static const int **SEND_ERROR**
Sending command failed. Either not connected, or the requested channel is bigger than the number of initialized channels.

static const int **QUEUE_OUTGOING_ACKS_WARNING**
PhotonPeer outgoing queue is filling up. Send more often.

static const int **QUEUE_INCOMING_RELIABLE_WARNING**
PhotonPeer incoming reliable queue is filling up.

Dispatch more often.

static const int **QUEUE_INCOMING_UNRELIABLE_WARNING**
PhotonPeer incoming unreliable queue is filling up.
Dispatch more often.

static const int **QUEUE_SENT_WARNING**
PhotonPeer sent queue is filling up. Check, why the server does not acknowledge your sent reliable commands.

static const int **INTERNAL_RECEIVE_EXCEPTION**
Exception, if a server cannot be connected. Most likely, the server is not responding. Ask the user to try again later.

static const int **TIMEOUT_DISCONNECT**
Disconnection due to a timeout (client did no longer receive ACKs from server).

static const int **DISCONNECT_BY_SERVER**
Disconnect by server due to timeout (received a disconnect command, cause server misses ACKs of client).

static const int **DISCONNECT_BY_SERVER_USER_LIMIT**
Disconnect by server due to concurrent user limit reached (received a disconnect command).

static const int **DISCONNECT_BY_SERVER_LOGIC**
Disconnect by server due to server's logic (received a disconnect command).

static const int **ENCRYPTION_ESTABLISHED**
The encryption-setup for secure communication finished successfully.

static const int **ENCRYPTION_FAILED_TO_ESTABLISH**

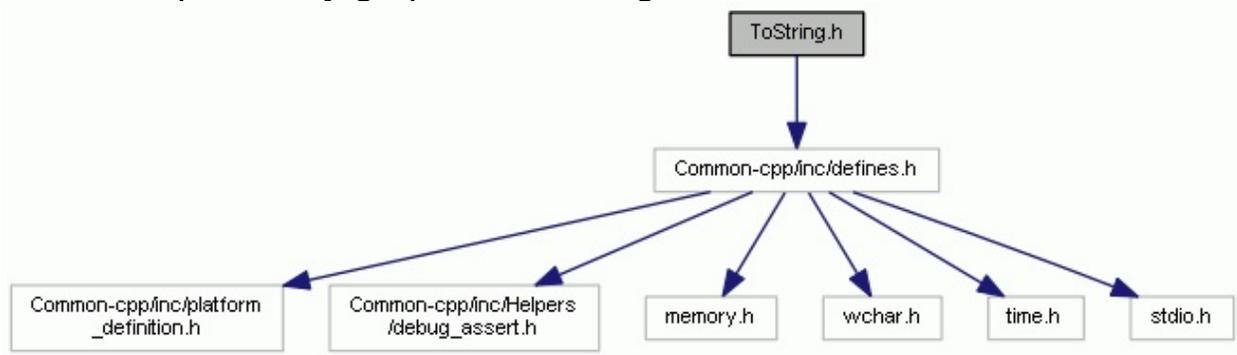
The encryption-setup failed for some reason. Check debug logs.

© Exit Games ®, all rights reserved.

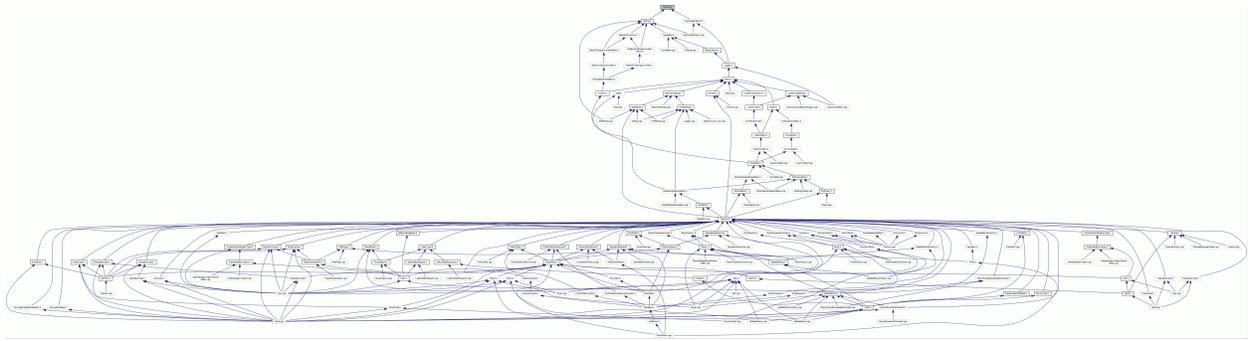
[Photon Documentation](#) | [Contact](#) | [Terms](#)

ToString.h File Reference

Include dependency graph for ToString.h:



This graph shows which files directly or indirectly include this file:



Classes

class **ToString**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

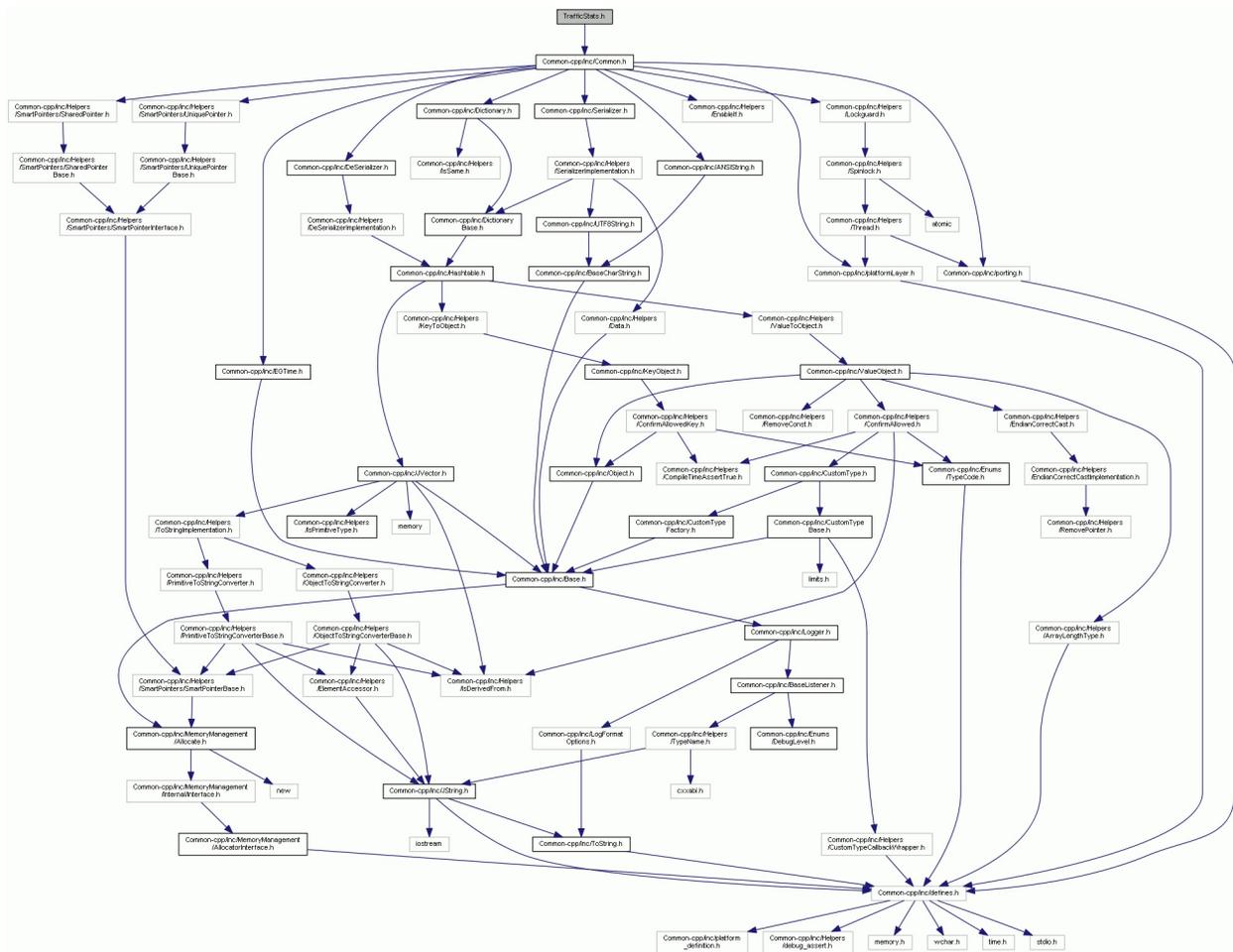
Client API 4.1.12.2

Photon-cpp > inc >

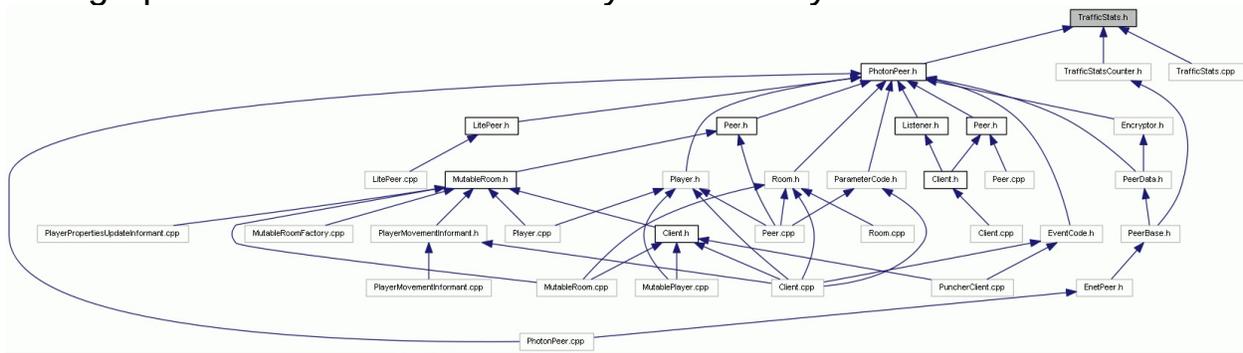
[Classes](#) | [Namespaces](#)

TrafficStats.h File Reference

Include dependency graph for TrafficStats.h:



This graph shows which files directly or indirectly include this file:



Classes

class **TrafficStats**

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Photon-cpp > inc >

[Classes](#) | [Namespaces](#)

TrafficStatsGameLevel.h File Reference

Include dependency graph for TrafficStatsGameLevel.h:

Classes

class **TrafficStatsGameLevel**

Namespaces

ExitGames

ExitGames::Photon

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

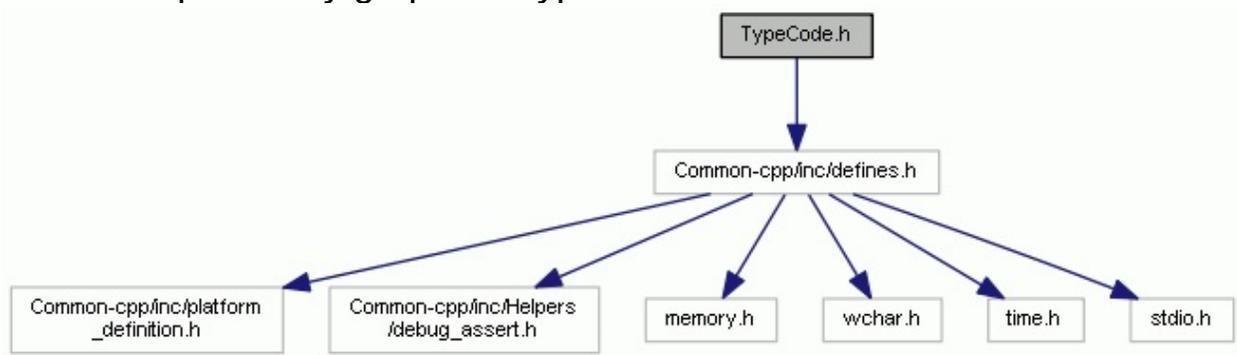
Client API 4.1.12.2

Common-cpp > inc > Enums

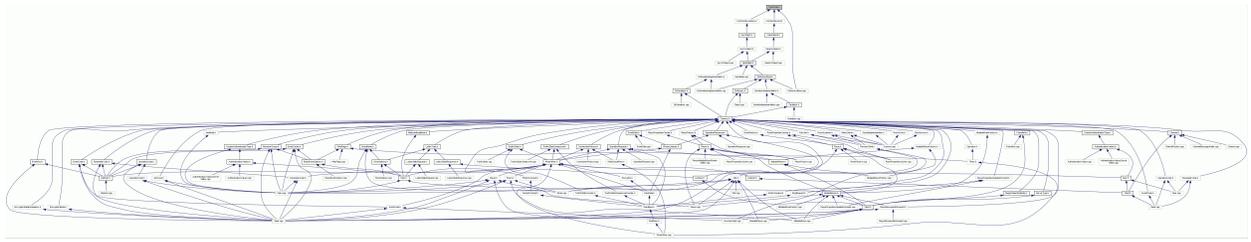
[Namespaces](#) | [Variables](#)

TypeCode.h File Reference

Include dependency graph for TypeCode.h:



This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Common

ExitGames::Common::TypeCode

Variables

static const nByte **BYTE**
nByte

static const nByte **SHORT**
short

static const nByte **INTEGER**
int

static const nByte **LONG**
int64

static const nByte **FLOAT**
float

static const nByte **DOUBLE**
double

static const nByte **BOOLEAN**
bool

static const nByte **STRING**
JString.

static const nByte **HASHTABLE**
Hashtable.

static const nByte **DICTIONARY**
Dictionary.

static const nByte **OBJECT**
Object, only allowed for arrays!

static const nByte **ARRAY**
internal only

static const nByte **BYTEARRAY**
internal only

static const nByte **PHOTON_COMMAND**
internal only

static const nByte **EG_NULL**
internal only

static const nByte **CUSTOM**
internal only

static const nByte **UNKNOWN**
internal only

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

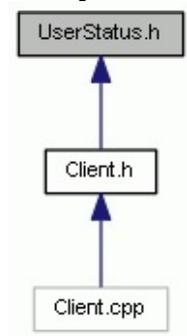
Client API 4.1.12.2

Chat-cpp > inc > Enums >

[Namespaces](#) | [Variables](#)

UserStatus.h File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

ExitGames

ExitGames::Chat

ExitGames::Chat::UserStatus

Variables

static const int **OFFLINE**
Offline.

static const int **INVISIBLE**
Be invisible to everyone. Sends no message.

static const int **ONLINE**
Online and available.

static const int **AWAY**
Online but not available.

static const int **DND**
Do not disturb.

static const int **LFG**
Looking For Game/Group. Could be used when you want to be invited or do matchmaking.

static const int **PLAYING**
Could be used when in a room, playing.



Photon C++

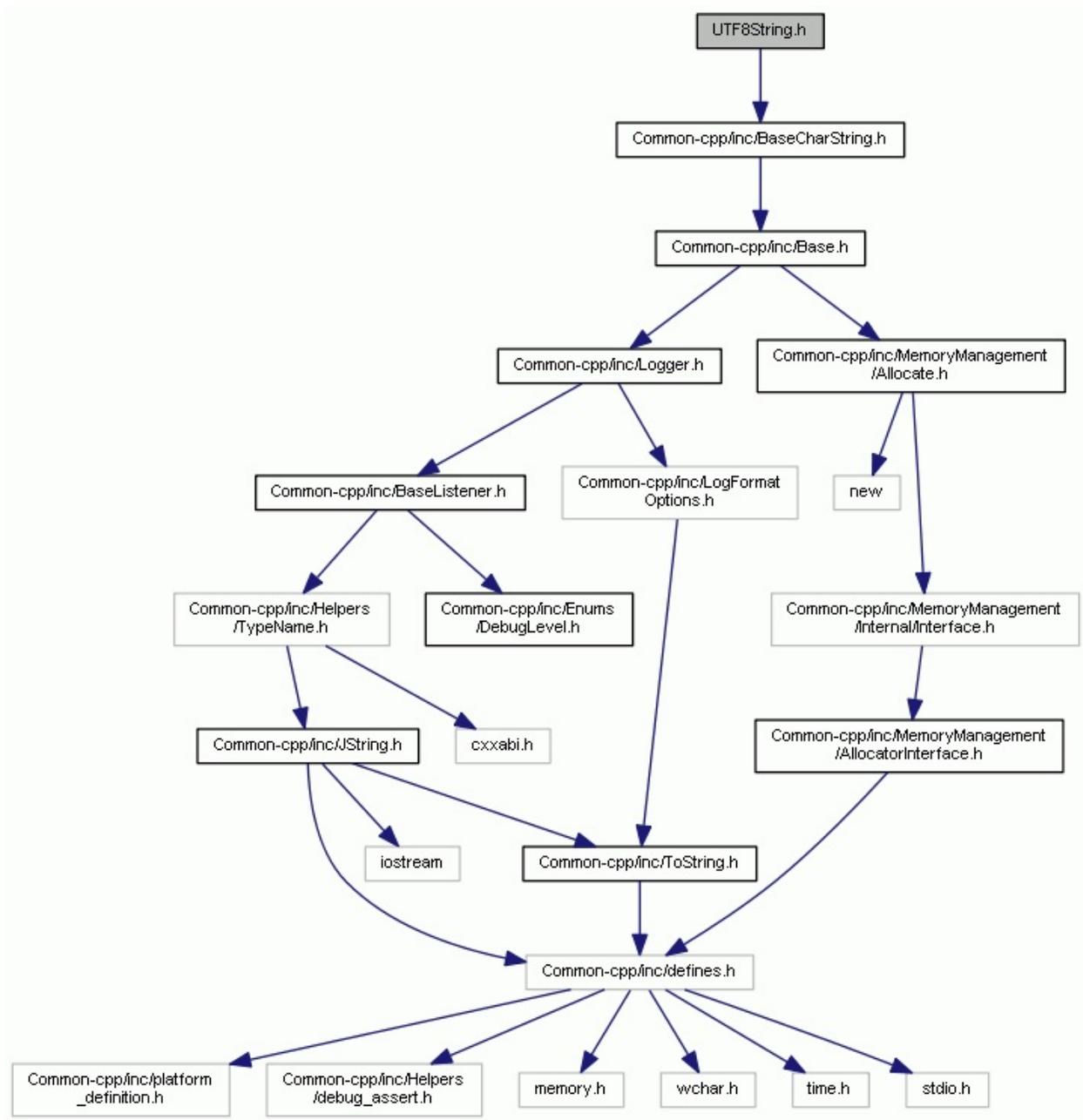
Client API 4.1.12.2

Common-cpp > inc >

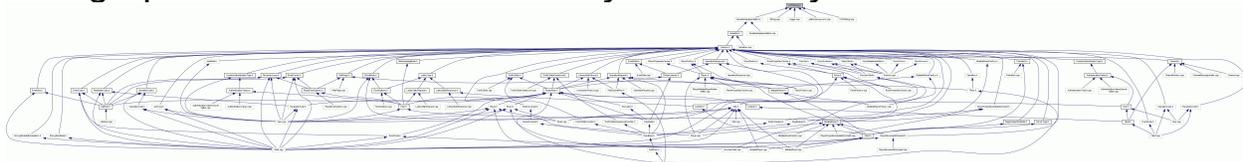
[Classes](#) | [Namespaces](#)

UTF8String.h File Reference

Include dependency graph for UTF8String.h:



This graph shows which files directly or indirectly include this file:



Classes

class **UTF8String**

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

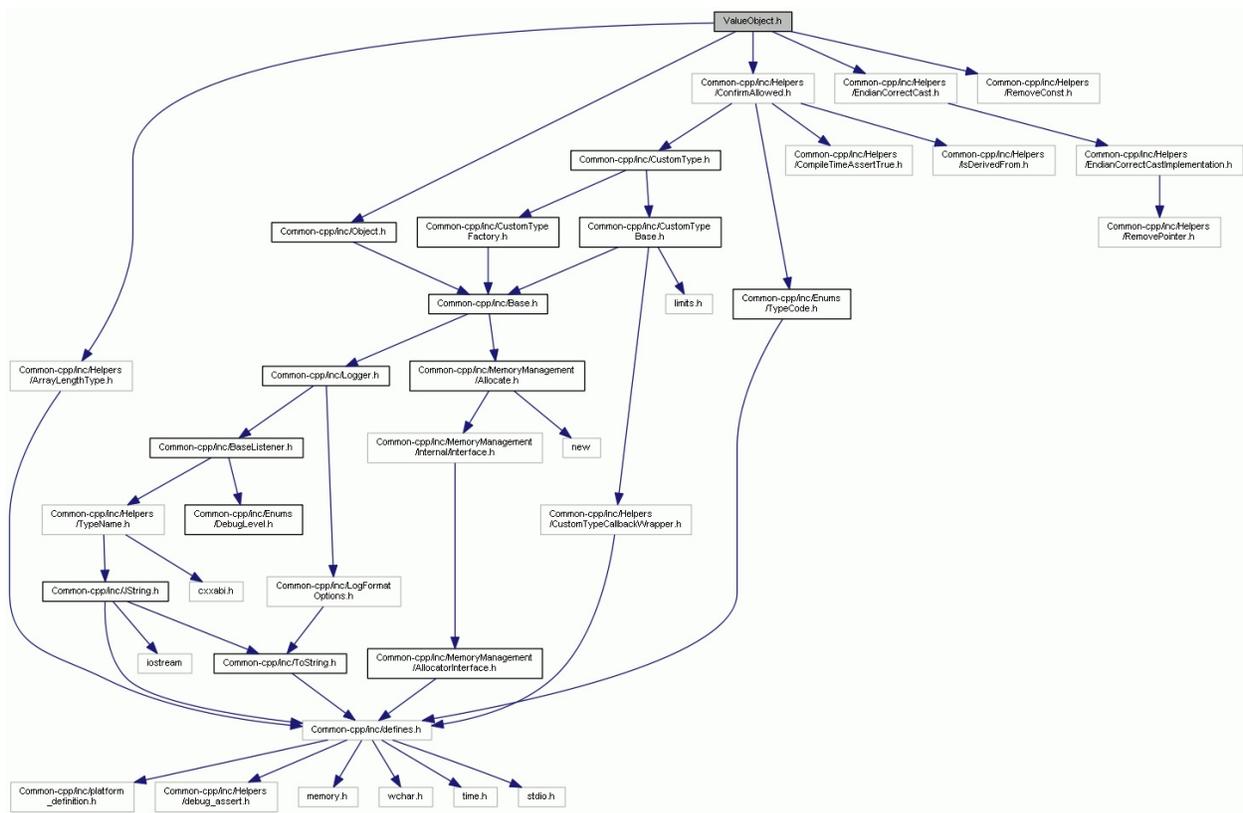
Client API 4.1.12.2

Common-cpp > inc >

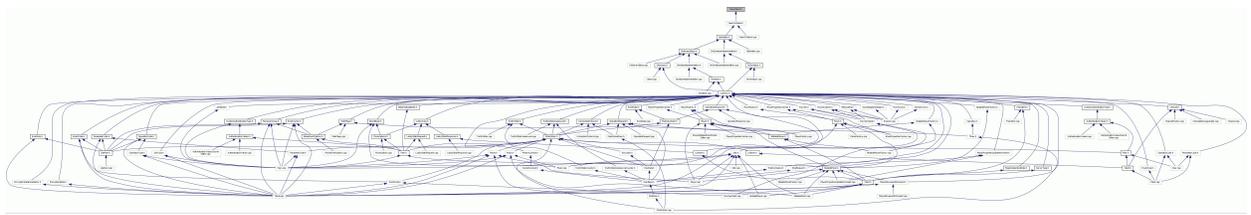
[Classes](#) | [Namespaces](#)

ValueObject.h File Reference

Include dependency graph for ValueObject.h:



This graph shows which files directly or indirectly include this file:



Classes

class **ValueObject**< Etype >

Namespaces

ExitGames

ExitGames::Common

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

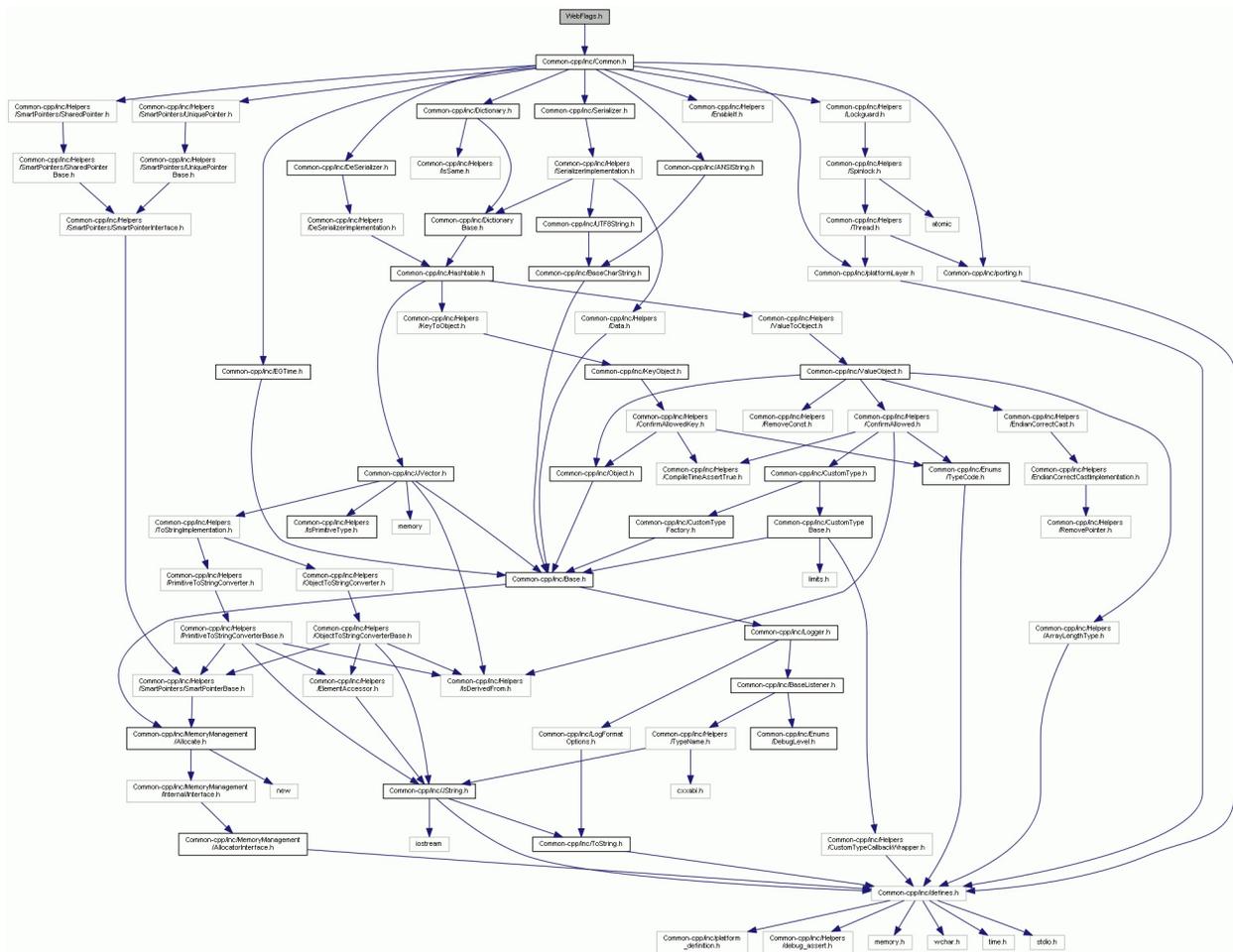
Client API 4.1.12.2

LoadBalancing-cpp > inc >

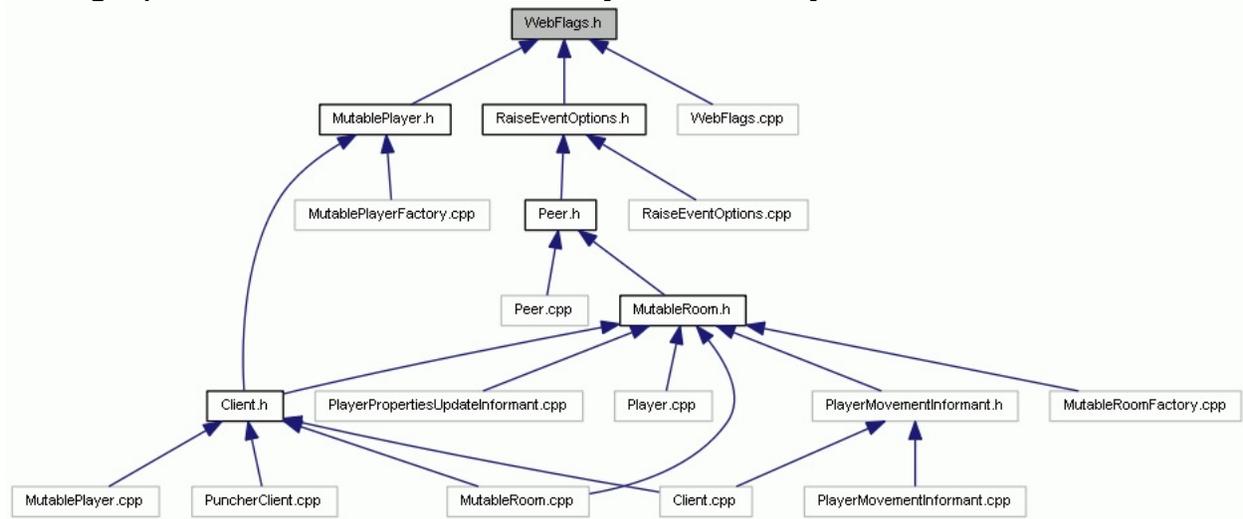
[Classes](#) | [Namespaces](#)

WebFlags.h File Reference

Include dependency graph for WebFlags.h:



This graph shows which files directly or indirectly include this file:



Classes

class **WebFlags**

Namespaces

ExitGames

ExitGames::LoadBalancing

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

Here is a list of all documented file members with links to the documentation:

- ALLOCATE : [Allocate.h](#)
 - ALLOCATE_ARRAY : [Allocate.h](#)
 - DEALLOCATE : [Allocate.h](#)
 - DEALLOCATE_ARRAY : [Allocate.h](#)
 - EG_CALLOC : [Allocate.h](#)
 - EG_FREE : [Allocate.h](#)
 - EG_MALLOC : [Allocate.h](#)
 - EG_REALLOC : [Allocate.h](#)
 - EGLOG : [Logger.h](#)
 - REALLOCATE_ARRAY : [Allocate.h](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

- ALLOCATE : [Allocate.h](#)
 - ALLOCATE_ARRAY : [Allocate.h](#)
 - DEALLOCATE : [Allocate.h](#)
 - DEALLOCATE_ARRAY : [Allocate.h](#)
 - EG_CALLOC : [Allocate.h](#)
 - EG_FREE : [Allocate.h](#)
 - EG_MALLOC : [Allocate.h](#)
 - EG_REALLOC : [Allocate.h](#)
 - EGLOG : [Logger.h](#)
 - REALLOCATE_ARRAY : [Allocate.h](#)
-

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



AuthenticationValues Member List

This is the complete list of members for **AuthenticationValues**, including all inherited members.

AuthenticationValues(void)

getData(void) const

getDebugOutputLevel(void)

getLogFormatOptions(void)

getParameters(void) const

getSecret(void) const

getType(void) const

getUserID(void) const

setData(const Common::JVector< nByte > &data)

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

setParameters(const Common::JString ¶meters)

setParametersWithUsernameAndToken(const Common::JString &user

setType(nByte type)

setUserID(const Common::JString &userID)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Graph Legend

This page explains how to interpret the graphs that are generated by doxygen.

Consider the following example:

```
/*! Invisible class because of truncation */  
class Invisible { };  
  
/*! Truncated class, inheritance relation is hidden  
    */  
class Truncated : public Invisible { };  
  
/* Class not documented with doxygen comments */  
class Undocumented { };  
  
/*! Class that is inherited using public inheritance  
    */  
class PublicBase : public Truncated { };  
  
/*! A template class */  
template<class T> class Templ { };  
  
/*! Class that is inherited using protected  
    inheritance */  
class ProtectedBase { };  
  
/*! Class that is inherited using private inheritance  
    */  
class PrivateBase { };
```

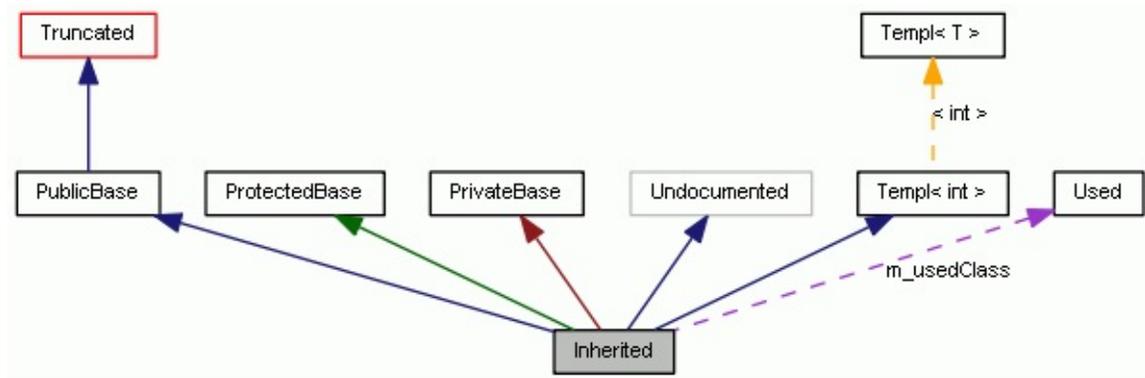
```

/*! Class that is used by the Inherited class */
class Used { };

/*! Super class that inherits a number of other
    classes */
class Inherited : public PublicBase,
                 protected ProtectedBase,
                 private PrivateBase,
                 public Undocumented,
                 public Templ<int>
{
private:
    Used *m_usedClass;
};

```

This will result in the following graph:



The boxes in the above graph have the following meaning:

- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a gray border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class for which not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

The arrows have the following meaning:

- A dark blue arrow is used to visualize a public inheritance relation

between two classes.

- A dark green arrow is used for protected inheritance.
- A dark red arrow is used for private inheritance.
- A purple dashed arrow is used if a class is contained or used by another class. The arrow is labelled with the variable(s) through which the pointed class or struct is accessible.
- A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labelled with the template parameters of the instance.

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Channel Member List

This is the complete list of members for **Channel**, including all inherited members.

clearMessages (void) (defined in Channel)	Cha
getDebugOutputLevel (void)	Bas
getIsPrivate (void) const (defined in Channel)	Cha
getLogFormatOptions (void)	Bas
getMessageCount (void) const (defined in Channel)	Cha
getMessages (void) const (defined in Channel)	Cha
getName (void) const (defined in Channel)	Cha
getSenders (void) const (defined in Channel)	Cha
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (Common::JString &retStr, bool withTypes=false) const	Cha
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~ToString (void)	ToS



Client Member List

This is the complete list of members for **Client**, including all inherited members.

Client(Listener &listener, const Common::JString &applicationID, const C

connect(const AuthenticationValues &authenticationValues=Authenticatic

disconnect(void)

dispatchIncomingCommands(void)

fetchServerTimestamp(void)

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getCRCEnabled(void) const

getDebugOutputLevel(void) const

getDisconnectedCause(void) const

getDisconnectTimeout(void) const

getIncomingReliableCommandsCount(void) const

getIsPayloadEncryptionAvailable(void) const

getLimitOfUnreliableCommands(void) const

getLogFormatOptions(void) const

getPacketLossByCRC(void) const

getPeerCount(void)

getPeerID(void) const

getPrivateChannel(const Common::JString &userName) const
getPrivateChannels(void) const
getPublicChannel(const Common::JString &channelName) const
getPublicChannels(void) const
getQueuedIncomingCommands(void) const
getQueuedOutgoingCommands(void) const
getQuickResendAttempts(void) const
getRegion(void) const
getResentReliableCommands(void) const
getRoundTripTime(void) const
getRoundTripTimeVariance(void) const
getSentCountAllowance(void) const
getServerTime(void) const
getServerTimeOffset(void) const
getState(void) const
getTimePingInterval(void) const
getTimestampOfLastSocketReceive(void) const
getTrafficStatsElapsedMs(void) const
getTrafficStatsEnabled(void) const
getTrafficStatsGameLevel(void) const
getTrafficStatsIncoming(void) const
getTrafficStatsOutgoing(void) const
getUserID(void) const
onPingResponse(const Common::JString &address, unsigned int pingRe
opAddFriends(const Common::JVector< Common::JString > &userIDs)
opPublishMessage(const Common::JString &channelName, const Ftype
opPublishMessage(const Common::JString &channelName, const Ftype
opPublishMessage(const Common::JString &channelName, const Ftype
opRemoveFriends(const Common::JVector< Common::JString > &userI
opSendPrivateMessage(const Common::JString &userName, const Ftyp
opSendPrivateMessage(const Common::JString &userName, const Ftyp

opSendMessage(const Common::JString &userName, const Ftype
opSetOnlineStatus(int status)
opSetOnlineStatus(int status, const Ftype &message) (defined in **Client**)
opSetOnlineStatus(int status, const Ftype pMessageArray, typename C
opSetOnlineStatus(int status, const Ftype pMessageArray, const short *|
opSubscribe(const Common::JVector< Common::JString > &channels, ir
opUnsubscribe(const Common::JVector< Common::JString > &channels
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)
sendAcksOnly(void)
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)
setLogFormatOptions(const Common::LogFormatOptions &formatOptio
setQuickResendAttempts(nByte quickResendAttempts)
setRegion(const Common::JString ®ion)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStatsEnabled)
vitalStatsToString(bool all) const
~BaseListener() (defined in **BaseListener**)
~Client(void)
~PhotonListener(void)



Listener Member List

This is the complete list of members for [Listener](#), including all inherited members.

[clientErrorReturn](#)(int errorCode)=0 (defined in [Listener](#))

[connectionErrorReturn](#)(int errorCode)=0 (defined in [Listener](#))

[connectReturn](#)(int errorCode, const Common::JString &errorString)=0

[debugReturn](#)(int debugLevel, const Common::JString &string)=0

[disconnectReturn](#)(void)=0

[onGetMessages](#)(const Common::JString &channelName, const Common::JString &message)

[onPrivateMessage](#)(const Common::JString &sender, const Common::Ok)

[onStateChange](#)(int state)=0

[onStatusUpdate](#)(const Common::JString &user, int status, bool gotMess)

[serverErrorReturn](#)(int errorCode)=0 (defined in [Listener](#))

[subscribeReturn](#)(const Common::JVector< Common::JString > &channe

[unsubscribeReturn](#)(const Common::JVector< Common::JString > &char

[warningReturn](#)(int warningCode)=0 (defined in [Listener](#))

[~BaseListener](#)() (defined in [BaseListener](#))

[~Listener](#)(void) (defined in [Listener](#))



Photon C++

Client API 4.1.12.2

[ExitGames](#)[Chat](#)[Peer](#)

Peer Member List

This is the complete list of members for **Peer**, including all inherited members.

connect(const Common::JString &ipAddr, const Common::JString &applI

disconnect(void)

dispatchIncomingCommands(void)

establishEncryption(void)

fetchServerTimestamp(void)

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getConnectionProtocol(void) const

getCRCEnabled(void) const

getDebugOutputLevel(void) const

getDisconnectTimeout(void) const

getIncomingReliableCommandsCount(void) const

getIsEncryptionAvailable(void) const

getIsPayloadEncryptionAvailable(void) const

getLimitOfUnreliableCommands(void) const

`getListener(void)`
`getLogFormatOptions(void) const`
`getMaxAppIDLength(void)`
`getPacketLossByCRC(void) const`
`getPeerCount(void)`
`getPeerID(void) const`
`getPeerState(void) const`
`getQueuedIncomingCommands(void) const`
`getQueuedOutgoingCommands(void) const`
`getQuickResendAttempts(void) const`
`getResentReliableCommands(void) const`
`getRoundTripTime(void) const`
`getRoundTripTimeVariance(void) const`
`getSentCountAllowance(void) const`
`getServerAddress(void) const`
`getServerTime(void) const`
`getServerTimeOffset(void) const`
`getTimePingInterval(void) const`
`getTimestampOfLastSocketReceive(void) const`
`getTrafficStatsElapsedMs(void) const`
`getTrafficStatsEnabled(void) const`
`getTrafficStatsGameLevel(void) const`
`getTrafficStatsIncoming(void) const`
`getTrafficStatsOutgoing(void) const`
`initUDPEncryption(const Common::JVector< nByte > &encryptSecret, co`
`initUserDataEncryption(const Common::JVector< nByte > &secret)`
`opAddFriends(const Common::JVector< Common::JString > &userIDs) (`
`opAuthenticateOnFrontEnd(const Common::JString &secret) (defined in`
`opAuthenticateOnNameServer(const Common::JString &appID, const C`
`opCustom(const OperationRequest &operationRequest, bool sendReliab`
`opPublishMessage(const Common::JString &channelName, const Ftype`

opPublishMessage(const Common::JString &channelName, const Ftype
opPublishMessage(const Common::JString &channelName, const Ftype
opRemoveFriends(const Common::JVector< Common::JString > &userll
opSendPrivateMessage(const Common::JString &userName, const Ftyp
opSendPrivateMessage(const Common::JString &userName, const Ftyp
opSendPrivateMessage(const Common::JString &userName, const Ftyp
opSetOnlineStatus(int status) (defined in **Peer**)
opSetOnlineStatus(int status, const Ftype &message) (defined in **Peer**)
opSetOnlineStatus(int status, const Ftype pMessageArray, typename C
opSetOnlineStatus(int status, const Ftype pMessageArray, const short *|
opSubscribe(const Common::JVector< Common::JString > &channels, ir
opUnsubscribe(const Common::JVector< Common::JString > &channels
Peer(Photon::PhotonListener &listener, nByte connectionProtocol=Photor
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=Conne
pingServer(const Common::JString &address, unsigned int pingAttempts
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)
sendAcksOnly(void)
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setConnectionProtocol(nByte connectionProtocol)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)
setLogFormatOptions(const Common::LogFormatOptions &formatOptio
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStasEnabled)

vitalStatsToString(bool all) const

~**Peer**(void) (defined in **Peer**)

~**PhotonPeer**(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



AllocatorInterface Member List

This is the complete list of members for **AllocatorInterface**, including all inherited members.

<code>alloc(size_t size)=0</code>	AllocatorInterface	pure virtual
<code>dealloc(void *p)=0</code>	AllocatorInterface	pure virtual
<code>get(void)</code>	AllocatorInterface	static
<code>resize(void *p, size_t size)=0</code>	AllocatorInterface	pure virtual
<code>setMaxAllocSize(size_t maxAllocSize)=0</code>	AllocatorInterface	pure virtual
<code>~AllocatorInterface(void)</code>	AllocatorInterface	virtual



ANSIString Member List

This is the complete list of members for **ANSIString**, including all inherited members.

ANSIString (void)	ANSIString
ANSIString (const ANSIString &str)	ANSIString
ANSIString (const JString &str)	ANSIString
ANSIString (const char *str)	ANSIString
ANSIString (const EG_CHAR *str)	ANSIString
BaseCharString ()	BaseCharString
cstr (void) const	BaseCharString
getDebugOutputLevel (void)	BaseCharString
getLogFormatOptions (void)	BaseCharString
JStringRepresentation (void) const	ANSIString
length (void) const	BaseCharString
operator const char * (void) const	ANSIString
operator JString (void) const	ANSIString
operator= (const ANSIString &Rhs)	ANSIString
operator= (const JString &Rhs)	ANSIString
operator= (const char *Rhs)	ANSIString
operator= (const EG_CHAR *Rhs)	ANSIString
setDebugOutputLevel (int debugLevel)	BaseCharString
setListener (const BaseListener *baseListener)	BaseCharString
setLogFormatOptions (const LogFormatOptions &options)	BaseCharString
size (void) const	ANSIString

toString (JString &retStr, bool withTypes=false) const	Bas
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~ANSIString (void)	ANS
~Base (void)	Bas
~BaseCharString (void)	Bas
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++ Client API 4.1.12.2

[ExitGames](#)[Common](#)[Base](#)

Base Member List

This is the complete list of members for **Base**, including all inherited members.

getDebugOutputLevel (void)	Base	sta
getLogFormatOptions (void)	Base	sta
setDebugOutputLevel (int debugLevel)	Base	sta
setListener (const BaseListener *baseListener)	Base	sta
setLogFormatOptions (const LogFormatOptions &options)	Base	sta
toString (JString &retStr, bool withTypes=false) const =0	Tostring	pur
toString (bool withTypes=false) const	Tostring	
typeToString (void) const	Tostring	virt
~Base (void)	Base	virt
~ToString (void)	Tostring	virt

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



BaseCharString Member List

This is the complete list of members for **BaseCharString**, including all inherited members.

BaseCharString()	Bas
cstring (void) const	Bas
getDebugOutputLevel (void)	Bas
getLogFormatOptions (void)	Bas
JStringRepresentation (void) const =0	Bas
length (void) const	Bas
operator const char * (void) const =0	Bas
operator JString (void) const =0	Bas
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
size (void) const =0	Bas
toString (JString &retStr, bool withTypes=false) const	Bas
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~BaseCharString (void)	Bas
~ToString (void)	ToS

Photon C++

Client API 4.1.12.2

[ExitGames](#) > [Common](#) > [BaseListener](#)

BaseListener Member List

This is the complete list of members for **BaseListener**, including all inherited members.

debugReturn (int debugLevel, const JString &string)=0	BaseListener	pu
~BaseListener () (defined in BaseListener)	BaseListener	vir

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



CustomType< typeCode > Member List

This is the complete list of members for **CustomType< typeCode >**, including all inherited members.

cleanup(void)=0

compare(const CustomTypeBase &other) const =0

constructClass(const CustomTypeFactory< typeCode > &factory)

deconstructClass(void)

deserialize(const nByte *pData, short length)=0

duplicate(CustomTypeBase *pRetVal) const =0

getDebugOutputLevel(void)

getLogFormatOptions(void)

serialize(nByte *pRetVal) const =0

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

ExitGames::Common::Base::toString(JString &retStr, bool withTypes=true)

ExitGames::Common::Base::toString(bool withTypes=false) const

TypeCode

typeToString(void) const

~Base(void)

~ToString(void)



CustomTypeBase Member List

This is the complete list of members for **CustomTypeBase**, including all inherited members.

cleanup(void)=0

compare(const CustomTypeBase &other) const =0

deserialize(const nByte *pData, short length)=0

duplicate(CustomTypeBase *pRetVal) const =0

getDebugOutputLevel(void)

getLogFormatOptions(void)

serialize(nByte *pRetVal) const =0

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

ExitGames::Common::Base::toString(JString &retStr, bool withTypes=true)

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~ToString(void)



CustomTypeFactory< typeCode > Member List

This is the complete list of members for **CustomTypeFactory< typeCode >**, including all inherited members.

copy(const CustomType< typeCode > *pToCopy, short amount) const =0

copyFactory(void) const =0

create(short amount) const =0

destroy(const CustomType< typeCode > *pToDestroy) const =0

destroyFactory(void)=0

getDebugOutputLevel(void)

getLogFormatOptions(void)

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

sizeof(void) const =0

toString(JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~CustomTypeFactory(void)

~ToString(void)



DeSerializer Member List

This is the complete list of members for **DeSerializer**, including all inherited members.

DeSerializer (const nByte *data, int size)	DeS
getDebugOutputLevel (void)	Bas
getLogFormatOptions (void)	Bas
pop (Object &object)	DeS
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (JString &retStr, bool withTypes=false) const	DeS
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~ToString (void)	ToS



Dictionary< EKeyType, EValueType > Member List

This is the complete list of members for **Dictionary< EKeyType, EValueType >**, including all inherited members.

contains(const EKeyType &key) const

ExitGames::Common::DictionaryBase::contains(const FKeyType &key) const
Dictionary(void)

Dictionary(const Dictionary< EKeyType, EValueType > &toCopy)

DictionaryBase(const DictionaryBase &toCopy)

getDebugOutputLevel(void)

getHashtable(void) const

getKeys(void) const

ExitGames::Common::DictionaryBase::getKeys(const FKeyType *) const

ExitGames::Common::DictionaryBase::getKeys(const Object *) const

getKeyTypes(void) const

getLogFormatOptions(void)

getSize(void) const

getValue(const EKeyType &key) const

ExitGames::Common::DictionaryBase::getValue(const FKeyType &key) const

ExitGames::Common::DictionaryBase::getValue(const FKeyType &key) const

getValueDimensions(void) const

getValueSizes(const FKeyType &key) const

getValueTypes(void) const

operator!=(const Dictionary< EKeyType, EValueType > &toCompare) const

ExitGames::Common::DictionaryBase::operator!=(const DictionaryBase &other) const
operator=(const Dictionary< EKeyType, EValueType > &toCopy)
ExitGames::Common::DictionaryBase::operator=(const DictionaryBase &other)
operator==(const Dictionary< EKeyType, EValueType > &toCompare) const
ExitGames::Common::DictionaryBase::operator==(const DictionaryBase &other) const
operator[](unsigned int index) const
operator[](unsigned int index)
put(const Dictionary< EKeyType, EValueType > &src)
put(const EKeyType &key, const EValueType &val)
put(const EKeyType &key)
put(const EKeyType &key, const EValueType pVal, typename Common::T)
put(const EKeyType &key, const EValueType pVal, const short *sizes)
remove(const EKeyType &key)
ExitGames::Common::DictionaryBase::remove(const FKeyType &key)
removeAllElements(void)
setDebugOutputLevel(int debugLevel)
setListener(const BaseListener *baseListener)
setLogFormatOptions(const LogFormatOptions &options)
toString(JString &retStr, bool withTypes=false) const
ExitGames::Common::Base::toString(bool withTypes=false) const
typeToString(void) const
~Base(void)
~Dictionary(void)
~DictionaryBase(void)
~ToString(void)



DictionaryBase Member List

This is the complete list of members for **DictionaryBase**, including all inherited members.

contains (const FKeyType &key) const	Dic
DictionaryBase (const DictionaryBase &toCopy)	Dic
getDebugOutputLevel (void)	Bas
getHashtable (void) const	Dic
getKeys (const FKeyType *) const	Dic
getKeys (const Object *) const	Dic
getKeyTypes (void) const	Dic
getLogFormatOptions (void)	Bas
getSize (void) const	Dic
getValue (const FKeyType &key, const FValueType *) const	Dic
getValue (const FKeyType &key, const Object *) const	Dic
getValueDimensions (void) const	Dic
getValueSizes (const FKeyType &key) const	Dic
getValueTypes (void) const	Dic
operator!= (const DictionaryBase &toCompare) const	Dic
operator= (const DictionaryBase &toCopy)	Dic
operator== (const DictionaryBase &toCompare) const	Dic
remove (const FKeyType &key)	Dic
removeAllElements (void)	Dic
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas

setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (JString &retStr, bool withTypes=false) const	Dic
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	Dic
~Base (void)	Bas
~DictionaryBase (void)	Dic
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



EGTime Member List

This is the complete list of members for **EGTime**, including all inherited members.

EGTime (int time)	EGTime
EGTime (const EGTime &toCopy)	EGTime
getDebugOutputLevel (void)	BaseListener
getLogFormatOptions (void)	BaseListener
operator!= (const EGTime &time) const	EGTime
operator+ (const EGTime &time)	EGTime
operator+= (const EGTime &time)	EGTime
operator- (const EGTime &time)	EGTime
operator-= (const EGTime &time)	EGTime
operator< (const EGTime &time) const	EGTime
operator<= (const EGTime &time) const	EGTime
operator= (const EGTime &toCopy)	EGTime
operator= (const int &time)	EGTime
operator== (const EGTime &time) const	EGTime
operator> (const EGTime &time) const	EGTime
operator>= (const EGTime &time) const	EGTime
overflowed (const EGTime &time) const	EGTime
setDebugOutputLevel (int debugLevel)	BaseListener
setListener (const BaseListener *baseListener)	BaseListener
setLogFormatOptions (const LogFormatOptions &options)	BaseListener
toString (JString &retStr, bool withTypes=false) const	EGTime

ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~EGTime (void)	EG
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Hashtable Member List

This is the complete list of members for **Hashtable**, including all inherited members.

contains(const FKeyType &key) const

getDebugOutputLevel(void)

getKeys(void) const

getLogFormatOptions(void)

getSize(void) const

getValue(const FKeyType &key) const

Hashtable(void)

Hashtable(const Hashtable &toCopy)

operator!=(const Hashtable &toCompare) const

operator=(const Hashtable &toCopy)

operator==(const Hashtable &toCompare) const

operator[](unsigned int index) const

operator[](unsigned int index)

put(const Hashtable &src)

put(const FKeyType &key, const FValueType &val)

put(const FKeyType &key)

put(const FKeyType &key, const FValueType pVal, typename Common::F

put(const FKeyType &key, const FValueType pVal, const short *sizes)

remove(const FKeyType &key)

removeAllElements(void)

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

toString(JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~Hashtable(void)

~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



JString Member List

This is the complete list of members for **JString**, including all inherited members.

ANSIRepresentation(void) const

capacity(void) const

charAt(unsigned int index) const

compareTo(const JString &anotherString) const

concat(const JString &str)

cstr(void) const

deleteChars(unsigned int start, unsigned int length) const

endsWith(const JString &suffix) const

ensureCapacity(unsigned int minCapacity)

equals(const JString &anotherString) const

equalsIgnoreCase(const JString &anotherString) const

indexOf(char ch) const

indexOf(char ch, unsigned int fromIndex) const

indexOf(EG_CHAR ch) const

indexOf(EG_CHAR ch, unsigned int fromIndex) const

indexOf(const JString &str) const

indexOf(const JString &str, unsigned int fromIndex) const

JString(unsigned int bufferlen=0)

JString(const char *Value)

JString(const EG_CHAR *Value)

JString(const JString &Value)

JString(const UTF8String &Value)
JString(const ANSIStrng &Value)
lastIndexOf(char ch) const
lastIndexOf(char ch, unsigned int fromIndex) const
lastIndexOf(EG_CHAR ch) const
lastIndexOf(EG_CHAR ch, unsigned int fromIndex) const
lastIndexOf(const JString &str) const
lastIndexOf(const JString &str, unsigned int fromIndex) const
length(void) const
operator const EG_CHAR *(void) const
operator !=(const JString &Rhs) const
operator !=(const JString &Lsh, const Etype &Rsh)
operator !=(const Etype &Lsh, const JString &Rsh)
operator +(const JString &Lsh, const Etype &Rsh)
operator +(const Etype &Lsh, const JString &Rsh)
operator +(const JString &Lsh, const JString &Rsh)
operator +=(const JString &Rhs)
operator +=(const Etype &Rhs)
operator <(const JString &Rhs) const
operator <(const JString &Lsh, const Etype &Rsh)
operator <(const Etype &Lsh, const JString &Rsh)
operator <<(::std::basic_ostream< _Elem, _Traits > &stream, const JStrin
operator <=(const JString &Rhs) const
operator <=(const JString &Lsh, const Etype &Rsh)
operator <=(const Etype &Lsh, const JString &Rsh)
operator =(const JString &Rhs)
operator =(const char *Rhs)
operator =(const EG_CHAR *Rhs)
operator =(const UTF8String &Rhs)
operator =(const ANSIStrng &Rhs)
operator =(char Rhs)

operator=(signed char Rhs)
operator=(unsigned char Rhs)
operator=(EG_CHAR Rhs)
operator=(short aNum)
operator=(unsigned short aNum)
operator=(int aNum)
operator=(unsigned int aNum)
operator=(long aNum)
operator=(unsigned long aNum)
operator=(long long aNum)
operator=(unsigned long long aNum)
operator=(float aNum)
operator=(double aNum)
operator=(long double aNum)
operator=(bool aBool)
operator==(const JString &Rhs) const
operator==(const JString &Lsh, const Etype &Rsh)
operator==(const Etype &Lsh, const JString &Rsh)
operator>(const JString &Rhs) const
operator>(const JString &Lsh, const Etype &Rsh)
operator>(const Etype &Lsh, const JString &Rsh)
operator>=(const JString &Rhs) const
operator>=(const JString &Lsh, const Etype &Rsh)
operator>=(const Etype &Lsh, const JString &Rsh)
operator[](unsigned int Index) const
operator[](unsigned int Index)
replace(char oldChar, char newChar) const
replace(EG_CHAR oldChar, EG_CHAR newChar) const
replace(const JString &match, const JString &replacement) const
startsWith(const JString &prefix) const
startsWith(const JString &prefix, unsigned int offset) const

substring(unsigned int beginIndex) const
substring(unsigned int beginIndex, unsigned int endIndex) const
toInt(void) const
toLowerCase(void) const
toString(JString &retStr, bool withTypes=false) const
ExitGames::Common::ToString::toString(bool withTypes=false) const
toUpperCase(void) const
trim(void)
typeToString(void) const
UTF8Representation(void) const
~JString(void)
~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



JVector< Etype > Member List

This is the complete list of members for **JVector< Etype >**, including all inherited members.

addElement(const Etype &obj)

addElements(const JVector< Etype > &vector)

addElements(const Etype *carray, unsigned int elementCount)

contains(const Etype &elem) const

copyInto(Etype *array) const

ensureCapacity(unsigned int minCapacity)

getCapacity(void) const

getCArray(void) const

getDebugOutputLevel(void)

getElementAt(unsigned int index) const

getFirstElement(void) const

getIndexOf(const Etype &elem) const

getIsEmpty(void) const

getLastElement(void) const

getLastIndexOf(const Etype &elem) const

getLogFormatOptions(void)

getSize(void) const

insertElementAt(const Etype &obj, unsigned int index)

JVector(unsigned int initialCapacity=0, unsigned int capacityIncrement=1)

JVector(const Etype *carray, unsigned int elementCount, unsigned int init

JVector(const JVector< Etype > &rhv)

operator!=(const JVector< Etype > &toCompare) const
operator=(const JVector< Etype > &rhv)
operator==(const JVector< Etype > &toCompare) const
operator[](unsigned int index) const
operator[](unsigned int index)
removeAllElements(void)
removeElement(const Etype &obj)
removeElementAt(unsigned int index)
setDebugOutputLevel(int debugLevel)
setElementAt(const Etype &obj, unsigned int index)
setListener(const BaseListener *baseListener)
setLogFormatOptions(const LogFormatOptions &options)
toString(JString &retStr, bool withTypes=false) const
ExitGames::Common::Base::toString(bool withTypes=false) const
trimToSize(void)
typeToString(void) const
~Base(void)
~JVector(void)
~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



KeyObject< Etype > Member List

This is the complete list of members for **KeyObject< Etype >**, including all inherited members.

getCustomType(void) const

getDataAddress(void) const

getDataCopy(void) const

getDebugOutputLevel(void)

getDimensions(void) const

getLogFormatOptions(void)

getSizes(void) const

getType(void) const

KeyObject(const KeyObject< Etype > &toCopy)

KeyObject(const Object &obj)

KeyObject(const Object *obj)

KeyObject(const typename Helpers::ConfirmAllowedKey< Etype >::type &obj)
Object(void)

Object(const Object &toCopy)

operator!=(const Object &toCompare) const

operator=(const KeyObject< Etype > &toCopy)

operator=(const Object &toCopy)

operator==(const Object &toCompare) const

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

toString(JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~KeyObject(void)

~Object(void)

~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



LogFormatOptions Member List

This is the complete list of members for **LogFormatOptions**, including all inherited members.

getAddDateTime(void) const

getAddFile(void) const

getAddFunction(void) const

getAddLevel(void) const

getAddLine(void) const

getMaxNumberOfNamespaces(void) const

LogFormatOptions(void) (defined in **LogFormatOptions**)

setAddDateTime(bool addTime)

setAddFile(bool addFile)

setAddFunction(bool addFunction)

setAddLevel(bool addLevel)

setAddLine(bool addLine)

setMaxNumberOfNamespaces(unsigned int maxNumberOfNamespaces)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::ToString::toString(bool withTypes=false) const

typeToString(void) const

~ToString(void)



Logger Member List

This is the complete list of members for **Logger**, including all inherited members.

getDebugOutputLevel(void) const

getFormatOptions(void) const

log(int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool withTypes)

Logger(int debugLevel=DebugLevel::WARNINGS)

setDebugOutputLevel(int debugLevel)

setFormatOptions(const LogFormatOptions &formatOptions)

setListener(const BaseListener &listener)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::ToString::toString(bool withTypes=false) const

typeToString(void) const

vlog(int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool withTypes)

~Logger(void) (defined in **Logger**)

~ToString(void)



Object Member List

This is the complete list of members for **Object**, including all inherited members.

getCustomType (void) const	Obj
getDebugOutputLevel (void)	Bas
getDimensions (void) const	Obj
getLogFormatOptions (void)	Bas
getSizes (void) const	Obj
getType (void) const	Obj
Object (void)	Obj
Object (const Object &toCopy)	Obj
operator!= (const Object &toCompare) const	Obj
operator= (const Object &toCopy)	Obj
operator== (const Object &toCompare) const	Obj
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (JString &retStr, bool withTypes=false) const	Obj
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~Object (void)	Obj
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Serializer Member List

This is the complete list of members for **Serializer**, including all inherited members.

getData(void) const

getDebugOutputLevel(void)

getLogFormatOptions(void)

getSize(void) const

push(const T &data)

push(const T pData, typename Helpers::ArrayLengthType< T >::type arra

push(const T pData, const short *arraySizes)

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

toString(JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~ToString(void)



Photon C++

Client API 4.1.12.2

[ExitGames](#)[Common](#)[ToString](#)

ToString Member List

This is the complete list of members for **ToString**, including all inherited members.

toString (JString &retStr, bool withTypes=false) const =0	ToString	pure vir
toString (bool withTypes=false) const	ToString	
typeToString (void) const	ToString	virtual
~ToString (void)	ToString	virtual

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



UTF8String Member List

This is the complete list of members for **UTF8String**, including all inherited members.

BaseCharString()	Bas
cstring(void) const	Bas
getDebugOutputLevel(void)	Bas
getLogFormatOptions(void)	Bas
JStringRepresentation(void) const	UTF
length(void) const	Bas
operator const char *(void) const	UTF
operator JString(void) const	UTF
operator=(const UTF8String &Rhs)	UTF
operator=(const JString &Rhs)	UTF
operator=(const char *Rhs)	UTF
operator=(const EG_CHAR *Rhs)	UTF
setDebugOutputLevel(int debugLevel)	Bas
setListener(const BaseListener *baseListener)	Bas
setLogFormatOptions(const LogFormatOptions &options)	Bas
size(void) const	UTF
size(const JString &str)	UTF
toString(JString &retStr, bool withTypes=false) const	Bas
ExitGames::Common::Base::toString(bool withTypes=false) const	ToS
typeToString(void) const	ToS
UTF8String(void)	UTF

UTF8String (const UTF8String &str)	UTF
UTF8String (const JString &str)	UTF
UTF8String (const char *str)	UTF
UTF8String (const EG_CHAR *str)	UTF
~Base (void)	Bas
~BaseCharString (void)	Bas
~ToString (void)	ToS
~UTF8String (void)	UTF

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



ValueObject< Etype > Member List

This is the complete list of members for [ValueObject< Etype >](#), including all inherited members.

[getCustomType](#)(void) const

[getDataAddress](#)(void) const

[getDataCopy](#)(void) const

[getDebugOutputLevel](#)(void)

[getDimensions](#)(void) const

[getLogFormatOptions](#)(void)

[getSizes](#)(void) const (defined in [ValueObject< Etype >](#))

[getType](#)(void) const

[Object](#)(void)

[Object](#)(const Object &toCopy)

[operator!=](#)(const Object &toCompare) const

[operator=](#)(const ValueObject< Etype > &toCopy)

[operator=](#)(const Object &toCopy)

[operator==](#)(const Object &toCompare) const

[setDebugOutputLevel](#)(int debugLevel)

[setListener](#)(const BaseListener *baseListener)

[setLogFormatOptions](#)(const LogFormatOptions &options)

[toString](#)(JString &retStr, bool withTypes=false) const

[ExitGames::Common::Base::toString](#)(bool withTypes=false) const

[typeToString](#)(void) const

[ValueObject](#)(const ValueObject< Etype > &toCopy)

ValueObject(const Object &obj)

ValueObject(const Object *obj)

ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type &c

ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type pC

ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type pC

~**Base**(void)

~**Object**(void)

~**ToString**(void)

~**ValueObject**(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



LitePeer Member List

This is the complete list of members for **LitePeer**, including all inherited members.

connect(const Common::JString &ipAddr, const Common::JString &applI

disconnect(void)

dispatchIncomingCommands(void)

establishEncryption(void)

fetchServerTimestamp(void)

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getConnectionProtocol(void) const

getCRCEnabled(void) const

getDebugOutputLevel(void) const

getDisconnectTimeout(void) const

getIncomingReliableCommandsCount(void) const

getIsEncryptionAvailable(void) const

getIsPayloadEncryptionAvailable(void) const

getLimitOfUnreliableCommands(void) const

getListener(void)
getLogFormatOptions(void) const
getMaxAppIDLength(void)
getPacketLossByCRC(void) const
getPeerCount(void)
getPeerID(void) const
getPeerState(void) const
getQueuedIncomingCommands(void) const
getQueuedOutgoingCommands(void) const
getQuickResendAttempts(void) const
getResentReliableCommands(void) const
getRoundTripTime(void) const
getRoundTripTimeVariance(void) const
getSentCountAllowance(void) const
getServerAddress(void) const
getServerTime(void) const
getServerTimeOffset(void) const
getTimePingInterval(void) const
getTimestampOfLastSocketReceive(void) const
getTrafficStatsElapsedMs(void) const
getTrafficStatsEnabled(void) const
getTrafficStatsGameLevel(void) const
getTrafficStatsIncoming(void) const
getTrafficStatsOutgoing(void) const
initUDPEncryption(const Common::JVector< nByte > &encryptSecret, c
initUserDataEncryption(const Common::JVector< nByte > &secret)
LitePeer(Photon::PhotonListener &listener, nByte connectionProtocol=Ph
opChangeGroups(const Common::JVector< nByte > *pGroupsToRemov
opCustom(const OperationRequest &operationRequest, bool sendReliak
opGetProperties(nByte channelId=0)
opGetPropertiesOfActor(const Common::JString *properties, short num

opGetPropertiesOfActor(const nByte *properties, short numProperties, ...)
opGetPropertiesOfGame(const Common::JString *properties, short numProperties, ...)
opGetPropertiesOfGame(const nByte *properties, short numProperties, ...)
opJoin(const Common::JString &gameId, const Common::Hashtable &gameProperties)
opLeave(void)
opRaiseEvent(bool reliable, Ftype parameters, nByte eventCode, nByte ...)
opRaiseEvent(bool reliable, Ftype pParameterArray, typename Common::JString ...)
opRaiseEvent(bool reliable, Ftype pParameterArray, const short *pArrSize, ...)
opSetPropertiesOfActor(int actorNr, const Common::Hashtable &properties)
opSetPropertiesOfGame(const Common::Hashtable &properties, bool b ...)
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=ConnectionProtocol:: ...)
pingServer(const Common::JString &address, unsigned int pingAttempts)
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)
sendAcksOnly(void)
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setConnectionProtocol(nByte connectionProtocol)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)
setLogFormatOptions(const Common::LogFormatOptions &formatOptions)
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStatsEnabled)
vitalStatsToString(bool all) const
~LitePeer(void)
~PhotonPeer(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#)[LoadBalancing](#)[AuthenticationValues](#)

AuthenticationValues Member List

This is the complete list of members for **AuthenticationValues**, including all inherited members.

AuthenticationValues(void)

getData(void) const

getDebugOutputLevel(void)

getLogFormatOptions(void)

getParameters(void) const

getSecret(void) const

getType(void) const

getUserID(void) const

setData(const Common::JVector< nByte > &data)

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

setParameters(const Common::JString ¶meters)

setParametersWithUsernameAndToken(const Common::JString &useri

setType(nByte type)

setUserID(const Common::JString &userID)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Client Member List

This is the complete list of members for **Client**, including all inherited members.

Client(LoadBalancing::Listener &listener, const Common::JString &applic

connect(const AuthenticationValues &authenticationValues=Authenticatic

disconnect(void)

dispatchIncomingCommands(void)

fetchServerTimestamp(void)

getAutoJoinLobby(void) const

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getCountGamesRunning(void) const

getCountPlayersIngame(void) const

getCountPlayersOnline(void) const

getCRCEnabled(void) const

getCurrentlyJoinedRoom(void)

getDebugOutputLevel(void) const

getDisconnectedCause(void) const

getDisconnectTimeout(void) const

getFriendList(void) const

getFriendListAge(void) const

getIncomingReliableCommandsCount(void) const
getIsEncryptionAvailable(void) const
getIsInGameRoom(void) const
getIsInLobby(void) const
getIsInRoom(void) const
getIsPayloadEncryptionAvailable(void) const
getLimitOfUnreliableCommands(void) const
getLocalPlayer(void)
getLogFormatOptions(void) const
getMasterserverAddress(void) const
getPacketLossByCRC(void) const
getPeerCount(void)
getPeerID(void) const
getQueuedIncomingCommands(void) const
getQueuedOutgoingCommands(void) const
getQuickResendAttempts(void) const
getRegionWithBestPing(void) const
getResentReliableCommands(void) const
getRoomList(void) const
getRoomNameList(void) const
getRoundTripTime(void) const
getRoundTripTimeVariance(void) const
getSentCountAllowance(void) const
getServerTime(void) const
getServerTimeOffset(void) const
getState(void) const
getTimePingInterval(void) const
getTimestampOfLastSocketReceive(void) const
getTrafficStatsElapsedMs(void) const
getTrafficStatsEnabled(void) const
getTrafficStatsGameLevel(void) const

getTrafficStatsIncoming(void) const
getTrafficStatsOutgoing(void) const
getUserID(void) const
opChangeGroups(const Common::JVector< nByte > *pGroupsToRemove
opCreateRoom(const Common::JString &gameID, const RoomOptions &
opCustom(const Photon::OperationRequest &operationRequest, bool se
opCustomAuthenticationSendNextStepData(const AuthenticationValue
opFindFriends(const Common::JString *friendsToFind, short numFriends
opJoinLobby(const Common::JString &lobbyName=Common::JString(),
opJoinOrCreateRoom(const Common::JString &gameID, const RoomOp
opJoinRandomRoom(const Common::Hashtable &customRoomProperti
opJoinRoom(const Common::JString &gameID, bool rejoin=false, int cac
opLeaveLobby(void)
opLeaveRoom(bool willComeBack=false, bool sendAuthCookie=false)
opLobbyStats(const Common::JVector< LoadBalancing::LobbyStatsReq
opRaiseEvent(bool reliable, const Ftype ¶meters, nByte eventCode,
opRaiseEvent(bool reliable, const Ftype pParameterArray, typename Co
opRaiseEvent(bool reliable, const Ftype pParameterArray, const short *p
opWebRpc(const Common::JString &uriPath)
opWebRpc(const Common::JString &uriPath, const Ftype ¶meters, l
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArr
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArr
reconnectAndRejoin(void)
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)
selectRegion(const Common::JString &selectedRegion)
sendAcksOnly(void)
sendDirect(const Ftype ¶meters, int targetPlayer, bool fallbackRelay
sendDirect(const Ftype pParameterArray, typename Common::Helpers:::
sendDirect(const Ftype pParameterArray, const short *pArrSizes, int targ
sendDirect(const Ftype ¶meters, const Common::JVector< int > &ta

sendDirect(const Ftype pParameterArray, typename Common::Helpers::
sendDirect(const Ftype pParameterArray, const short *pArrSizes, const C
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setAutoJoinLobby(bool autoJoinLobby)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)
setLogFormatOptions(const Common::LogFormatOptions &formatOptio
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStatsEnabled)
vitalStatsToString(bool all) const
~**BaseListener**() (defined in **BaseListener**)
~**Client**(void)
~**PhotonListener**(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



FriendInfo Member List

This is the complete list of members for **FriendInfo**, including all inherited members.

getDebugOutputLevel (void)	Bas
getIsInRoom (void) const	Frie
getIsOnline (void) const	Frie
getLogFormatOptions (void)	Bas
getRoom (void) const	Frie
getUserID (void) const	Frie
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (Common::JString &retStr, bool withTypes=false) const	Frie
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~ToString (void)	ToS



Listener Member List

This is the complete list of members for **Listener**, including all inherited members.

clientErrorReturn(int errorCode)=0 (defined in **Listener**)

connectionErrorReturn(int errorCode)=0 (defined in **Listener**)

connectReturn(int errorCode, const Common::JString &errorString, cons

createRoomReturn(int localPlayerNr, const Common::Hashtable &roomF

customEventAction(int playerNr, nByte eventCode, const Common::Obj

debugReturn(int debugLevel, const Common::JString &string)=0

disconnectReturn(void)=0 (defined in **Listener**)

joinLobbyReturn(void)=0 (defined in **Listener**)

joinOrCreateRoomReturn(int localPlayerNr, const Common::Hashtable &

joinRandomRoomReturn(int localPlayerNr, const Common::Hashtable &

joinRoomEventAction(int playerNr, const Common::JVector< int > &play

joinRoomReturn(int localPlayerNr, const Common::Hashtable &roomPro

leaveLobbyReturn(void)=0 (defined in **Listener**)

leaveRoomEventAction(int playerNr, bool isInactive)=0 (defined in **Liste**

leaveRoomReturn(int errorCode, const Common::JString &errorString)=(

onAppStatsUpdate(void) (defined in **Listener**)

onAvailableRegions(const Common::JVector< Common::JString > &, co

onCacheSliceChanged(int) (defined in **Listener**)

onCustomAuthenticationIntermediateStep(const Common::Dictionary<

onCustomOperationResponse(const Photon::OperationResponse &ope

onDirectMessage(const Common::Object &, int, bool) (defined in **Listen**

onFindFriendsResponse(void) (defined in [Listener](#))

onLobbyStatsResponse(const Common::JVector< LobbyStatsResponse > &)

onLobbyStatsUpdate(const Common::JVector< LobbyStatsResponse > &)

onMasterClientChanged(int, int) (defined in [Listener](#))

onPlayerPropertiesChange(int, const Common::Hashtable &) (defined in [Listener](#))

onRoomListUpdate(void) (defined in [Listener](#))

onRoomPropertiesChange(const Common::Hashtable &) (defined in [Listener](#))

onSecretReceival(const Common::JString &) (defined in [Listener](#))

serverErrorReturn(int errorCode)=0 (defined in [Listener](#))

warningReturn(int warningCode)=0 (defined in [Listener](#))

webRpcReturn(int, const Common::JString &, const Common::JString &, const Common::JString &)

~BaseListener() (defined in [BaseListener](#))

~Listener(void) (defined in [Listener](#))

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



LobbyStatsRequest Member List

This is the complete list of members for **LobbyStatsRequest**, including all inherited members.

getDebugOutputLevel(void)

getLogFormatOptions(void)

getName(void) const

getType(void) const

LobbyStatsRequest(const Common::JString &name=Common::JString())

setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~ToString(void)



LobbyStatsResponse Member List

This is the complete list of members for **LobbyStatsResponse**, including all inherited members.

getDebugOutputLevel (void)	Bas
getLogFormatOptions (void)	Bas
getName (void) const	Lob
getPeerCount (void) const	Lob
getRoomCount (void) const	Lob
getType (void) const	Lob
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (Common::JString &retStr, bool withTypes=false) const	Lob
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~ToString (void)	ToS



MutablePlayer Member List

This is the complete list of members for **MutablePlayer**, including all inherited members.

addCustomProperties(const Common::Hashtable &customProperties, c
addCustomProperty(const ktype &key, const vtype &value, const WebFl
addCustomProperty(const ktype &key, const vtype pValueArray, typenar
addCustomProperty(const ktype &key, const vtype pValueArray, const s
getCustomProperties() const
getDebugOutputLevel(void)
getIsInactive(void) const
getIsMasterClient(void) const
getLogFormatOptions(void)
getName() const
getNumber(void) const
getUserID() const
mergeCustomProperties(const Common::Hashtable &customProperties
MutablePlayer(const MutablePlayer &toCopy) (defined in **MutablePlayer**
operator=(const Player &toCopy)
operator=(const MutablePlayer &toCopy) (defined in **MutablePlayer**)
operator==(const Player &player) const
Player(const Player &toCopy)
removeCustomProperties(const ktype *keys, unsigned int count, const \
removeCustomProperty(const ktype &key, const WebFlags &webflags=
setDebugOutputLevel(int debugLevel)

setListener(const BaseListener *baseListener)
setLogFormatOptions(const LogFormatOptions &options)
setName(const Common::JString &name, const WebFlags &webflags=W
toString(Common::JString &retStr, bool withTypes=false) const
toString(bool withTypes, bool withCustomProperties) const
ExitGames::Common::Base::toString(bool withTypes=false) const
typeToString(void) const
~Base(void)
~MutablePlayer(void) (defined in **MutablePlayer**)
~Player(void)
~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



MutableRoom Member List

This is the complete list of members for **MutableRoom**, including all inherited members.

addCustomProperties(const Common::Hashtable &customProperties, c
addCustomProperty(const ktype &key, const vtype &value, const Comm
addCustomProperty(const ktype &key, const vtype pValueArray, typenar
addCustomProperty(const ktype &key, const vtype pValueArray, const s
getCustomProperties(void) const
getDebugOutputLevel(void)
getDirectMode(void) const
getEmptyRoomTtl(void) const (defined in **MutableRoom**)
getExpectedUsers(void) const (defined in **MutableRoom**)
getIsOpen(void) const
getIsVisible(void) const (defined in **MutableRoom**)
getLogFormatOptions(void)
getMasterClientID(void) const (defined in **MutableRoom**)
getMaxPlayers(void) const
getName(void) const
getPlayerCount(void) const
getPlayerForNumber(int playerNumber) const (defined in **MutableRoom**)
getPlayers(void) const (defined in **MutableRoom**)
getPlayerTtl(void) const (defined in **MutableRoom**)
getPlugins(void) const (defined in **MutableRoom**)
getPropsListedInLobby(void) const (defined in **MutableRoom**)

getPublishUserID(void) const (defined in **MutableRoom**)
getSuppressRoomEvents(void) const (defined in **MutableRoom**)
mergeCustomProperties(const Common::Hashtable &customProperties
MutableRoom(const MutableRoom &toCopy) (defined in **MutableRoom**)
operator=(const Room &toCopy)
operator=(const MutableRoom &toCopy) (defined in **MutableRoom**)
operator==(const Room &room) const
removeCustomProperties(const ktype *keys, unsigned int count, const (&room)
removeCustomProperty(const ktype &key, const Common::Hashtable &
Room(const Room &toCopy)
setDebugOutputLevel(int debugLevel)
setExpectedUsers(const Common::JVector< Common::JString > &expect
setIsOpen(bool isOpen, const WebFlags &webflags=WebFlags()) (define
setIsVisible(bool isVisible, const WebFlags &webflags=WebFlags()) (defi
setListener(const BaseListener *baseListener)
setLogFormatOptions(const LogFormatOptions &options)
setMaxPlayers(nByte maxPlayers, const WebFlags &webflags=WebFlag
setPropsListedInLobby(const Common::JVector< Common::JString > &
toString(bool withTypes=false, bool withCustomProperties=false, bool wi
ExitGames::LoadBalancing::Room::toString(Common::JString &retStr
ExitGames::LoadBalancing::Room::toString(bool withTypes, bool with
ExitGames::Common::Base::toString(bool withTypes=false) const
typeToString(void) const
~Base(void)
~MutableRoom(void) (defined in **MutableRoom**)
~Room(void)
~ToString(void)



Peer Member List

This is the complete list of members for **Peer**, including all inherited members.

connect(const Common::JString &ipAddr, const Common::JString &applI

disconnect(void)

dispatchIncomingCommands(void)

establishEncryption(void)

fetchServerTimestamp(void)

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getConnectionProtocol(void) const

getCRCEnabled(void) const

getDebugOutputLevel(void) const

getDisconnectTimeout(void) const

getIncomingReliableCommandsCount(void) const

getIsEncryptionAvailable(void) const

getIsPayloadEncryptionAvailable(void) const

getLimitOfUnreliableCommands(void) const

getListener(void)
getLogFormatOptions(void) const
getMaxAppIDLength(void)
getPacketLossByCRC(void) const
getPeerCount(void)
getPeerID(void) const
getPeerState(void) const
getQueuedIncomingCommands(void) const
getQueuedOutgoingCommands(void) const
getQuickResendAttempts(void) const
getResentReliableCommands(void) const
getRoundTripTime(void) const
getRoundTripTimeVariance(void) const
getSentCountAllowance(void) const
getServerAddress(void) const
getServerTime(void) const
getServerTimeOffset(void) const
getTimePingInterval(void) const
getTimestampOfLastSocketReceive(void) const
getTrafficStatsElapsedMs(void) const
getTrafficStatsEnabled(void) const
getTrafficStatsGameLevel(void) const
getTrafficStatsIncoming(void) const
getTrafficStatsOutgoing(void) const
initUDPEncryption(const Common::JVector< nByte > &encryptSecret, c
initUserDataEncryption(const Common::JVector< nByte > &secret)
opAuthenticate(const Common::JString &appID, const Common::JString
opAuthenticateOnce(const Common::JString &appID, const Common::J
opChangeGroups(const Common::JVector< nByte > *pGroupsToRemov
opCreateRoom(const Common::JString &gameID, const RoomOptions &
opCustom(const OperationRequest &operationRequest, bool sendReliak

opFindFriends(const Common::JString *friendsToFind, short numFriends
opGetRegions(bool encrypted, const Common::JString &appId) (defined
opJoinLobby(const Common::JString &lobbyName=Common::JString(),
opJoinRandomRoom(const Common::Hashtable &customRoomProperti
opJoinRoom(const Common::JString &gameID, const RoomOptions &op
opLeaveLobby(void) (defined in **Peer**)
opLeaveRoom(bool willComeBack=false, bool sendAuthCookie=false) (c
opLobbyStats(const Common::JVector< LoadBalancing::LobbyStatsReq
opRaiseEvent(bool reliable, const Ftype ¶meters, nByte eventCode,
opRaiseEvent(bool reliable, const Ftype pParameterArray, typename Co
opRaiseEvent(bool reliable, const Ftype pParameterArray, const short *p
opSetPropertiesOfPlayer(int playerNr, const Common::Hashtable &prop
opSetPropertiesOfRoom(const Common::Hashtable &properties, const
opWebRpc(const Common::JString &uriPath) (defined in **Peer**)
opWebRpc(const Common::JString &uriPath, const Ftype ¶meters, l
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArr
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArr
Peer(Photon::PhotonListener &listener, nByte connectionProtocol=Photor
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=Conne
pingServer(const Common::JString &address, unsigned int pingAttempts
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)
sendAcksOnly(void)
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setConnectionProtocol(nByte connectionProtocol)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)

setLogFormatOptions(const Common::LogFormatOptions &formatOptio
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStasEnabled)
vitalStatsToString(bool all) const
~**Peer**(void) (defined in **Peer**)
~**PhotonPeer**(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Player Member List

This is the complete list of members for **Player**, including all inherited members.

getCustomProperties() const	Play
getDebugOutputLevel (void)	Bas
getIsInactive (void) const	Play
getIsMasterClient (void) const	Play
getLogFormatOptions (void)	Bas
getName () const	Play
getNumber (void) const	Play
getUserID () const	Play
operator= (const Player &toCopy)	Play
operator== (const Player &player) const	Play
Player (const Player &toCopy)	Play
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (Common::JString &retStr, bool withTypes=false) const	Play
toString (bool withTypes, bool withCustomProperties) const	Play
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~Player (void)	Play
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



RaiseEventOptions Member List

This is the complete list of members for **RaiseEventOptions**, including all inherited members.

getCacheSliceIndex(void) const

getChannelID(void) const

getDebugOutputLevel(void)

getEventCaching(void) const

getInterestGroup(void) const

getLogFormatOptions(void)

getNumTargetPlayers(void) const

getReceiverGroup(void) const

getTargetPlayers(void) const

getWebFlags(void) const

operator=(const RaiseEventOptions &toCopy)

RaiseEventOptions(nByte channelID=0, nByte eventCaching=Lite::Even

RaiseEventOptions(const RaiseEventOptions &toCopy)

setCacheSliceIndex(int cacheSliceIndex)

setChannelID(nByte channelID)

setDebugOutputLevel(int debugLevel)

setEventCaching(nByte eventCaching)

setInterestGroup(nByte interestGroup)

setListener(const BaseListener *baseListener)

setLogFormatOptions(const LogFormatOptions &options)

setReceiverGroup(nByte receiverGroup)

setTargetPlayers(const int *targetPlayers, short numTargetPlayers)

setWebFlags(const WebFlags &webFlags)

toString(Common::JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~RaiseEventOptions(void)

~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Room Member List

This is the complete list of members for **Room**, including all inherited members.

getCustomProperties (void) const	Room
getDebugOutputLevel (void)	Base
getDirectMode (void) const	Room
getIsOpen (void) const	Room
getLogFormatOptions (void)	Base
getMaxPlayers (void) const	Room
getName (void) const	Room
getPlayerCount (void) const	Room
operator= (const Room &toCopy)	Room
operator== (const Room &room) const	Room
Room (const Room &toCopy)	Room
setDebugOutputLevel (int debugLevel)	Base
setListener (const BaseListener *baseListener)	Base
setLogFormatOptions (const LogFormatOptions &options)	Base
toString (Common::JString &retStr, bool withTypes=false) const	Room
toString (bool withTypes, bool withCustomProperties) const	Room
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
~Base (void)	Bas
~Room (void)	Roc
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#)[LoadBalancing](#)[RoomOptions](#)

RoomOptions Member List

This is the complete list of members for **RoomOptions**, including all inherited members.

getCustomRoomProperties(void) const

getDebugOutputLevel(void)

getDirectMode(void) const

getEmptyRoomTtl(void) const

getIsOpen(void) const

getIsVisible(void) const

getLobbyName(void) const

getLobbyType(void) const

getLogFormatOptions(void)

getMaxPlayers(void) const

getPlayerTtl(void) const

getPlugins(void) const

getPropsListedInLobby(void) const

getPublishUserID(void) const

getSuppressRoomEvents(void) const

operator=(const RoomOptions &toCopy)

RoomOptions(bool isVisible=true, bool isOpen=true, nByte maxPlayers=

RoomOptions(const RoomOptions &toCopy)

setCustomRoomProperties(const Common::Hashtable &customRoomP

setDebugOutputLevel(int debugLevel)

setDirectMode(nByte directMode)

setEmptyRoomTtl(int emptyRoomTtl)
setIsOpen(bool isOpen)
setIsVisible(bool isVisible)
setListener(const BaseListener *baseListener)
setLobbyName(const Common::JString &lobbyName)
setLobbyType(nByte lobbyType)
setLogFormatOptions(const LogFormatOptions &options)
setMaxPlayers(nByte maxPlayers)
setPlayerTtl(int playerTtl)
setPlugins(const Common::JVector< Common::JString > *pPlugins)
setPropsListedInLobby(const Common::JVector< Common::JString > &
setPublishUserID(bool publishUserID)
suppressRoomEvents(bool suppressRoomEvents)
toString(Common::JString &retStr, bool withTypes=false) const
ExitGames::Common::Base::toString(bool withTypes=false) const
typeToString(void) const
~Base(void)
~RoomOptions(void)
~ToString(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



WebFlags Member List

This is the complete list of members for **WebFlags**, including all inherited members.

getDebugOutputLevel (void)	Base
getFlags (void) const	WebFlags
getHttpForward (void) const	WebFlags
getLogFormatOptions (void)	Base
getSendAuthCookie (void) const	WebFlags
getSendState (void) const	WebFlags
getSendSync (void) const	WebFlags
setDebugOutputLevel (int debugLevel)	Base
setFlags (nByte webFlags)	WebFlags
setHttpForward (bool httpWebForward)	WebFlags
setListener (const BaseListener *baseListener)	Base
setLogFormatOptions (const LogFormatOptions &options)	Base
setSendAuthCookie (bool sendAuthCookie)	WebFlags
setSendState (bool sendState)	WebFlags
setSendSync (bool sendSync)	WebFlags
toString (Common::JString &retStr, bool withTypes=false) const	WebFlags
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
typeToString (void) const	ToS
WebFlags (nByte webFlags=0)	WebFlags
~Base (void)	Base
~ToString (void)	ToS

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Protocol Member List

This is the complete list of members for **Protocol**, including all inherited members.

GAME (defined in Protocol)	Protocol
MASTER (defined in Protocol)	Protocol
NAME (defined in Protocol)	Protocol

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [NetworkPort](#) > [TCP](#)

TCP Member List

This is the complete list of members for **TCP**, including all inherited members.

GAME (defined in TCP)	TCP	static
MASTER (defined in TCP)	TCP	static
NAME (defined in TCP)	TCP	static
TCP (void) (defined in TCP)	TCP	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++

Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [NetworkPort](#) > [UDP](#)

UDP Member List

This is the complete list of members for **UDP**, including all inherited members.

GAME (defined in UDP)	UDP	static
MASTER (defined in UDP)	UDP	static
NAME (defined in UDP)	UDP	static
UDP (void) (defined in UDP)	UDP	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Photon C++

Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [NetworkPort](#) > [UDPAlternative](#)

UDPAlternative Member List

This is the complete list of members for **UDPAlternative**, including all inherited members.

GAME (defined in UDPAlternative)	UDPAlternative	static
MASTER (defined in UDPAlternative)	UDPAlternative	static
NAME (defined in UDPAlternative)	UDPAlternative	static
UDPAlternative (void) (defined in UDPAlternative)	UDPAlternative	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [NetworkPort](#) > [WS](#)

WS Member List

This is the complete list of members for **WS**, including all inherited members.

GAME (defined in WS)	WS	static
MASTER (defined in WS)	WS	static
NAME (defined in WS)	WS	static
WS (void) (defined in WS)	WS	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++ Client API 4.1.12.2

[ExitGames](#) > [Photon](#) > [NetworkPort](#) > [WSS](#) >

WSS Member List

This is the complete list of members for **WSS**, including all inherited members.

GAME (defined in WSS)	WSS	static
MASTER (defined in WSS)	WSS	static
NAME (defined in WSS)	WSS	static
WSS (void) (defined in WSS)	WSS	

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



Puncher Member List

This is the complete list of members for **Puncher**, including all inherited members.

clear(void) (defined in **Puncher**)

init(PunchListener *pPunchListener) (defined in **Puncher**)

processPackage(const Common::JVector< nByte > &packet, bool relay,

Puncher(RelayClient *pRelayClient, const Common::Logger &logger) (de

sendDirect(const Common::JVector< nByte > &buffer, int targetID, bool f

sendDirect(const Common::JVector< nByte > &buffer, const Common::JV

service(void) (defined in **Puncher**)

startPunch(int remoteID) (defined in **Puncher**)

~Puncher(void) (defined in **Puncher**)

Photon C++

Client API 4.1.12.2

[ExitGames](#)[Photon](#)[Punchthrough](#)[PunchListener](#)

PunchListener Member List

This is the complete list of members for **PunchListener**, including all inherited members.

onReceiveDirect(const Common::JVector< nByte > &inBuf, int remoteID,
~PunchListener(void) (defined in **PunchListener**)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

RelayClient Member List

This is the complete list of members for **RelayClient**, including all inherited members.

getLocalID(void)=0 (defined in **RelayClient**)

sendRelay(const Common::JVector< nByte > &buffer, const Common::JV

~RelayClient(void) (defined in **RelayClient**)



EventData Member List

This is the complete list of members for **EventData**, including all inherited members.

EventData(const EventData &toCopy)

getCode(void) const

getParameterForCode(nByte parameterCode) const

getParameters(void) const

operator=(const EventData &toCopy)

operator[](unsigned int index) const

toString(bool withParameters=false, bool withParameterTypes=false) const

~EventData(void)



OperationRequest Member List

This is the complete list of members for **OperationRequest**, including all inherited members.

getOperationCode(void) const

getParameterForCode(nByte parameterCode) const

getParameters(void) const

getParameters(void)

OperationRequest(nByte operationCode, const OperationRequestParam

OperationRequest(const OperationRequest &toCopy)

operator=(const OperationRequest &toCopy)

operator[](unsigned int index) const

setParameters(const OperationRequestParameters ¶meters)

toString(bool withParameters=false, bool withParameterTypes=false) cor

~OperationRequest(void)



OperationResponse Member List

This is the complete list of members for **OperationResponse**, including all inherited members.

getDebugMessage(void) const

getOperationCode(void) const

getParameterForCode(nByte parameterCode) const

getParameters(void) const

getReturnCode(void) const

OperationResponse(const OperationResponse &toCopy)

operator=(const OperationResponse &toCopy)

operator[](unsigned int index) const

toString(bool withDebugMessage=false, bool withParameters=false, bool

~OperationResponse(void)



Photon C++

Client API 4.1.12.2

[ExitGames](#)[Photon](#)[PhotonListener](#)

PhotonListener Member List

This is the complete list of members for **PhotonListener**, including all inherited members.

debugReturn(int debugLevel, const JString &string)=0

onEvent(const EventData &eventData)=0

onOperationResponse(const OperationResponse &operationResponse)

onPingResponse(const Common::JString &address, unsigned int pingRe

onStatusChanged(int statusCode)=0

~BaseListener() (defined in **BaseListener**)

~PhotonListener(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



PhotonPeer Member List

This is the complete list of members for **PhotonPeer**, including all inherited members.

connect(const Common::JString &ipAddr, const Common::JString &applI

disconnect(void)

dispatchIncomingCommands(void)

establishEncryption(void)

fetchServerTimestamp(void)

getByteCountCurrentDispatch(void) const

getByteCountLastOperation(void) const

getBytesIn(void) const

getBytesOut(void) const

getChannelCountUserChannels(void) const

getConnectionProtocol(void) const

getCRCEnabled(void) const

getDebugOutputLevel(void) const

getDisconnectTimeout(void) const

getIncomingReliableCommandsCount(void) const

getIsEncryptionAvailable(void) const

getIsPayloadEncryptionAvailable(void) const

getLimitOfUnreliableCommands(void) const

getListener(void)
getLogFormatOptions(void) const
getMaxAppIDLength(void)
getPacketLossByCRC(void) const
getPeerCount(void)
getPeerID(void) const
getPeerState(void) const
getQueuedIncomingCommands(void) const
getQueuedOutgoingCommands(void) const
getQuickResendAttempts(void) const
getResentReliableCommands(void) const
getRoundTripTime(void) const
getRoundTripTimeVariance(void) const
getSentCountAllowance(void) const
getServerAddress(void) const
getServerTime(void) const
getServerTimeOffset(void) const
getTimePingInterval(void) const
getTimestampOfLastSocketReceive(void) const
getTrafficStatsElapsedMs(void) const
getTrafficStatsEnabled(void) const
getTrafficStatsGameLevel(void) const
getTrafficStatsIncoming(void) const
getTrafficStatsOutgoing(void) const
initUDPEncryption(const Common::JVector< nByte > &encryptSecret, c
initUserDataEncryption(const Common::JVector< nByte > &secret)
opCustom(const OperationRequest &operationRequest, bool sendReliab
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=Conne
pingServer(const Common::JString &address, unsigned int pingAttempts
resetTrafficStats(void)
resetTrafficStatsMaximumCounters(void)

sendAcksOnly(void)
sendOutgoingCommands(void)
service(bool dispatchIncomingCommands=true)
serviceBasic(void)
setConnectionProtocol(nByte connectionProtocol)
setCRCEnabled(bool crcEnabled)
setDebugOutputLevel(int debugLevel)
setDisconnectTimeout(int disconnectTimeout)
setLimitOfUnreliableCommands(int value)
setLogFormatOptions(const Common::LogFormatOptions &formatOptio
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStasEnabled)
vitalStatsToString(bool all) const
~PhotonPeer(void)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



TrafficStats Member List

This is the complete list of members for **TrafficStats**, including all inherited members.

getControlCommandBytes (void) const	TrafficStats
getControlCommandCount (void) const	TrafficStats
getDebugOutputLevel (void)	BaseListener
getFragmentCommandBytes (void) const	TrafficStats
getFragmentCommandCount (void) const	TrafficStats
getLogFormatOptions (void)	BaseListener
getPackageHeaderSize (void) const	TrafficStats
getReliableCommandBytes (void) const	TrafficStats
getReliableCommandCount (void) const	TrafficStats
getTimestampOfLastAck (void) const	TrafficStats
getTimestampOfLastReliableCommand (void) const	TrafficStats
getTotalCommandBytes (void) const	TrafficStats
getTotalCommandCount (void) const	TrafficStats
getTotalCommandsInPackets (void) const	TrafficStats
getTotalPacketBytes (void) const	TrafficStats
getTotalPacketCount (void) const	TrafficStats
getUnreliableCommandBytes (void) const	TrafficStats
getUnreliableCommandCount (void) const	TrafficStats
setDebugOutputLevel (int debugLevel)	BaseListener
setListener (const BaseListener *baseListener)	BaseListener
setLogFormatOptions (const LogFormatOptions &options)	BaseListener

<code>toString(Common::JString &retStr, bool withTypes=false) const</code>	<code>TrafficStats</code>
<code>ExitGames::Common::Base::toString(bool withTypes=false) const</code>	<code>ToString</code>
<code>typeToString(void) const</code>	<code>ToString</code>
<code>~Base(void)</code>	<code>Base</code>
<code>~ToString(void)</code>	<code>ToString</code>
<code>~TrafficStats(void)</code>	<code>TrafficStats</code>

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)



TrafficStatsGameLevel Member List

This is the complete list of members for [TrafficStatsGameLevel](#), including all inherited members.

getDebugOutputLevel (void)	Base
getDispatchIncomingCommandsCalls (void) const	TrafficStatsGameLevel
getEventByteCount (void) const	TrafficStatsGameLevel
getEventCount (void) const	TrafficStatsGameLevel
getLogFormatOptions (void)	Base
getLongestDeltaBetweenDispatching (void) const	TrafficStatsGameLevel
getLongestDeltaBetweenSending (void) const	TrafficStatsGameLevel
getLongestEventCallback (void) const	TrafficStatsGameLevel
getLongestEventCallbackCode (void) const	TrafficStatsGameLevel
getLongestOpResponseCallback (void) const	TrafficStatsGameLevel
getLongestOpResponseCallbackOpCode (void) const	TrafficStatsGameLevel
getOperationByteCount (void) const	TrafficStatsGameLevel
getOperationCount (void) const	TrafficStatsGameLevel
getResultByteCount (void) const	TrafficStatsGameLevel
getResultCount (void) const	TrafficStatsGameLevel
getSendOutgoingCommandsCalls (void) const	TrafficStatsGameLevel
getTotalByteCount (void) const	TrafficStatsGameLevel
getTotalIncomingByteCount (void) const	TrafficStatsGameLevel
getTotalIncomingMessageCount (void) const	TrafficStatsGameLevel
getTotalMessageCount (void) const	TrafficStatsGameLevel
getTotalOutgoingByteCount (void) const	TrafficStatsGameLevel

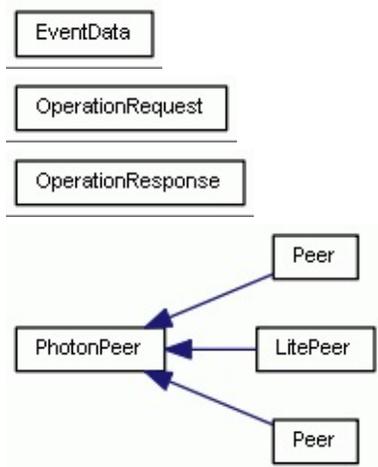
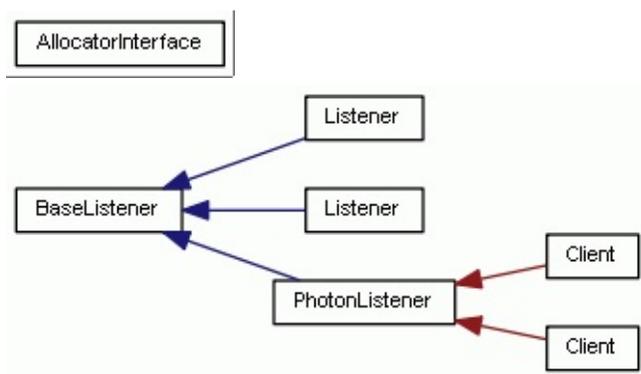
getTotalOutgoingMessageCount (void) const	Tra
resetMaximumCounters (void)	Tra
setDebugOutputLevel (int debugLevel)	Bas
setListener (const BaseListener *baseListener)	Bas
setLogFormatOptions (const LogFormatOptions &options)	Bas
toString (Common::JString &retStr, bool withTypes=false) const	Tra
ExitGames::Common::Base::toString (bool withTypes=false) const	ToS
toStringVitalStats (void) const	Tra
typeToString (void) const	ToS
~Base (void)	Bas
~ToString (void)	ToS
~TrafficStatsGameLevel (void)	Tra

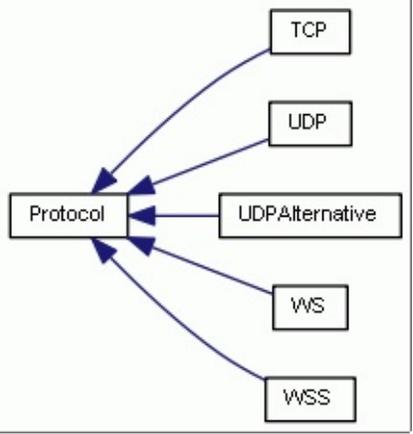
© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

Class Hierarchy

[Go to the textual class hierarchy](#)

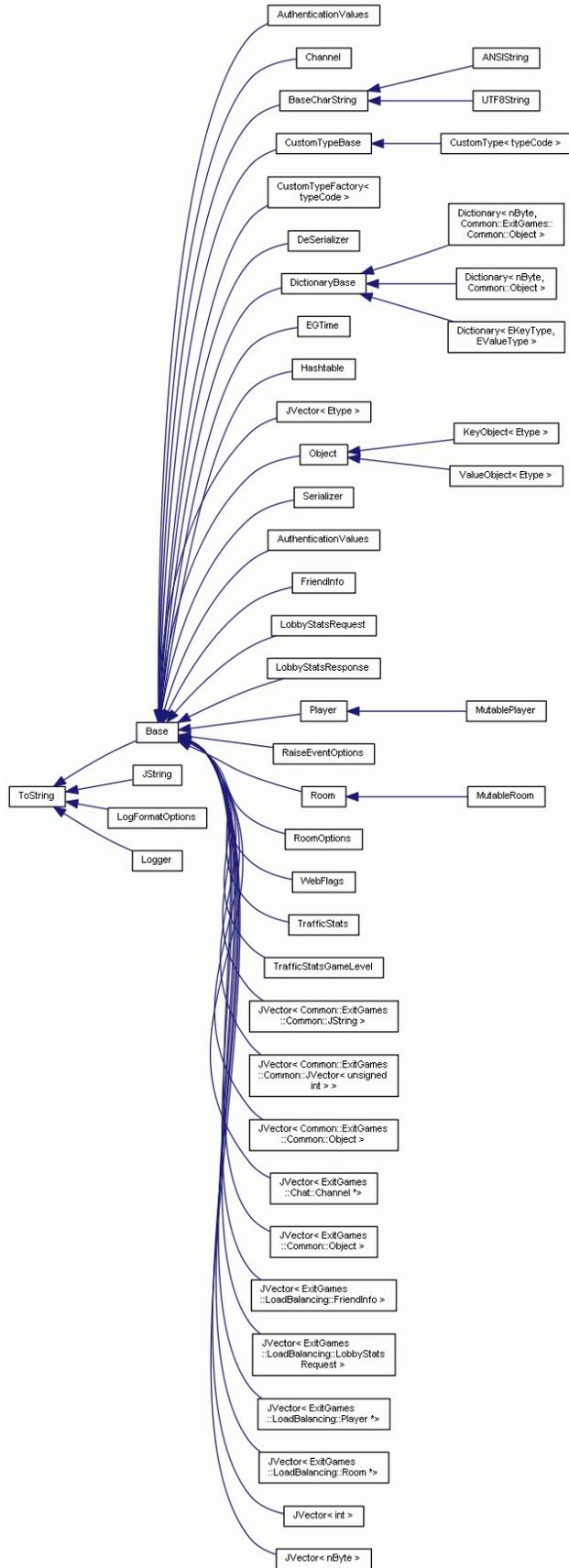




Puncher

PunchListener

RelayClient



© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

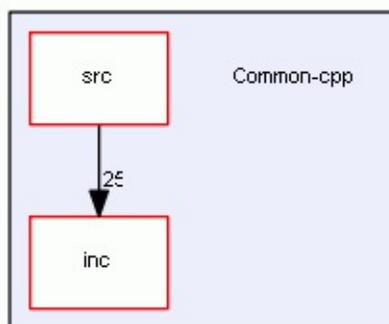
Photon C++

Client API 4.1.12.2

Common-cpp

Common-cpp Directory Reference

Directory dependency graph for Common-cpp:



Directories

© Exit Games ®, all rights reserved.

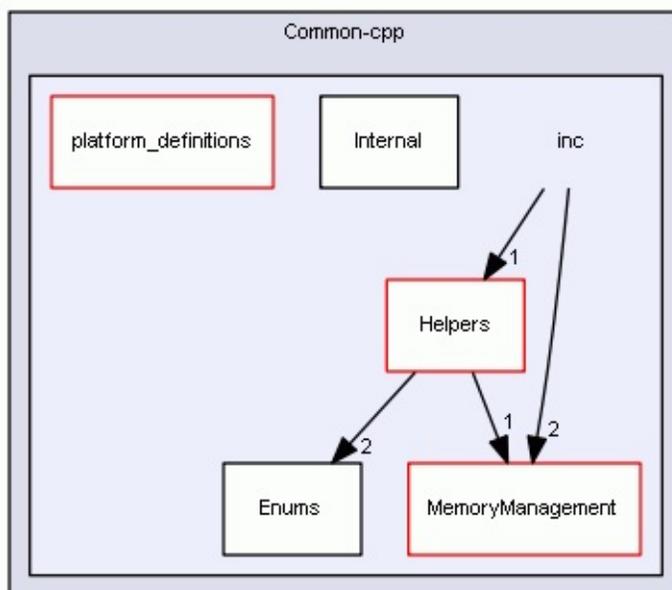
[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++ Client API 4.1.12.2

Common-cpp > inc >

inc Directory Reference

Directory dependency graph for inc:



Directories

directory **platform_definitions**

Files

file **ANSIString.h**

file **Base.h**

file **BaseCharString.h**

file **BaseListener.h**

file **Common.h**

file **CustomType.h**

file **CustomTypeBase.h**

file **CustomTypeFactory.h**

file **DeSerializer.h**

file **Dictionary.h**

file **DictionaryBase.h**

file **EGTime.h**

file **Hashtable.h**

file **JString.h**

file **JVector.h**

file **KeyObject.h**

file **Logger.h**

file [Object.h](#)

file [Serializer.h](#)

file [ToString.h](#)

file [UTF8String.h](#)

file [ValueObject.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

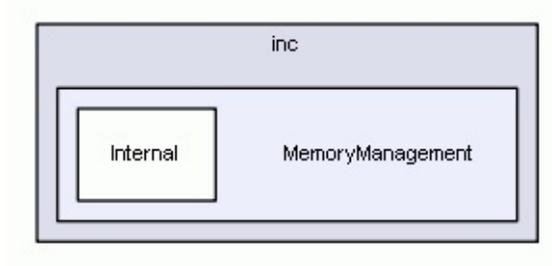
Photon C++

Client API 4.1.12.2

Common-cpp > inc > MemoryManagement

MemoryManagement Directory Reference

Directory dependency graph for MemoryManagement:



Directories

Files

file [Allocate.h](#)

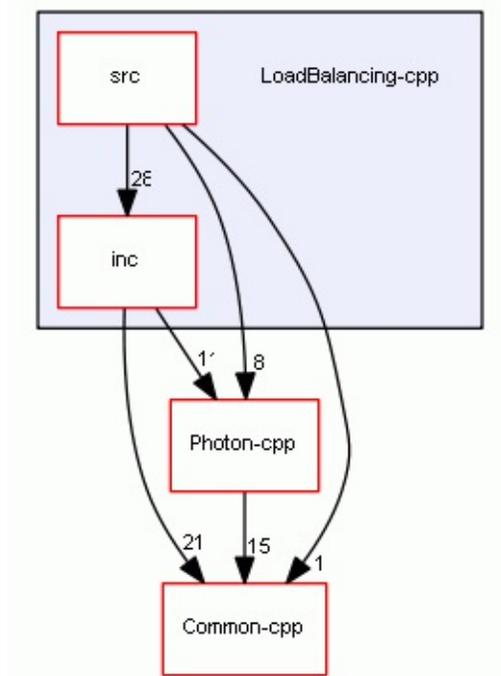
file [AllocatorInterface.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

LoadBalancing-cpp Directory Reference

Directory dependency graph for LoadBalancing-cpp:



Directories

© Exit Games ®, all rights reserved.

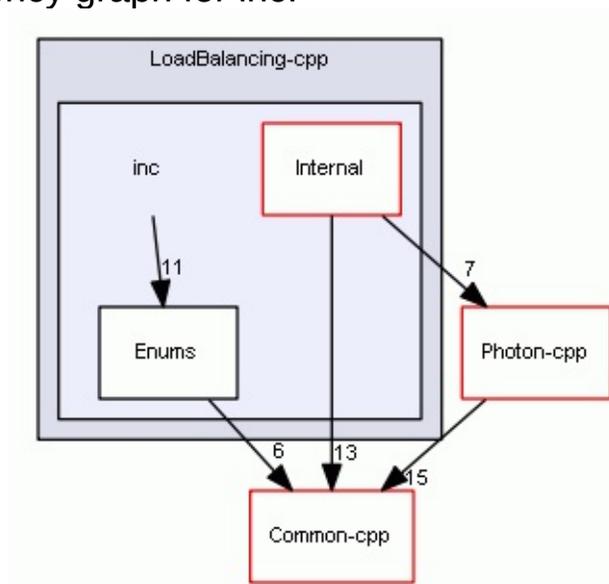
[Photon Documentation](#) | [Contact](#) | [Terms](#)

Photon C++ Client API 4.1.12.2

LoadBalancing-cpp > inc >

inc Directory Reference

Directory dependency graph for inc:



Directories

Files

file [LoadBalancing-cpp/inc/AuthenticationValues.h](#)

file [LoadBalancing-cpp/inc/Client.h](#)

file [FriendInfo.h](#)

file [LoadBalancing-cpp/inc/Listener.h](#)

file [LobbyStatsRequest.h](#)

file [LobbyStatsResponse.h](#)

file [MutablePlayer.h](#)

file [MutableRoom.h](#)

file [LoadBalancing-cpp/inc/Peer.h](#)

file [Player.h](#)

file [RaiseEventOptions.h](#)

file [Room.h](#)

file [RoomOptions.h](#)

file [WebFlags.h](#)

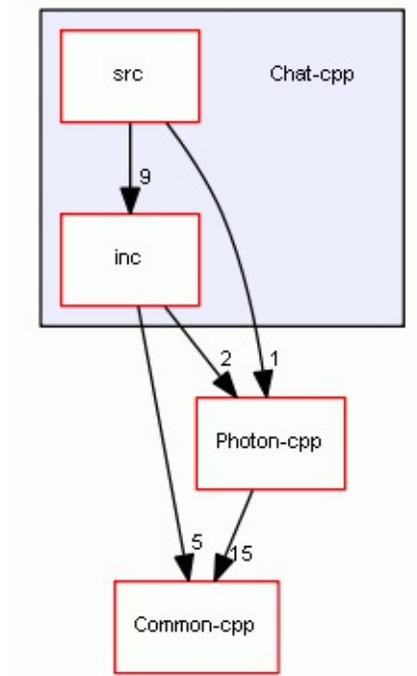
Photon C++

Client API 4.1.12.2

Chat-cpp

Chat-cpp Directory Reference

Directory dependency graph for Chat-cpp:



Directories

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

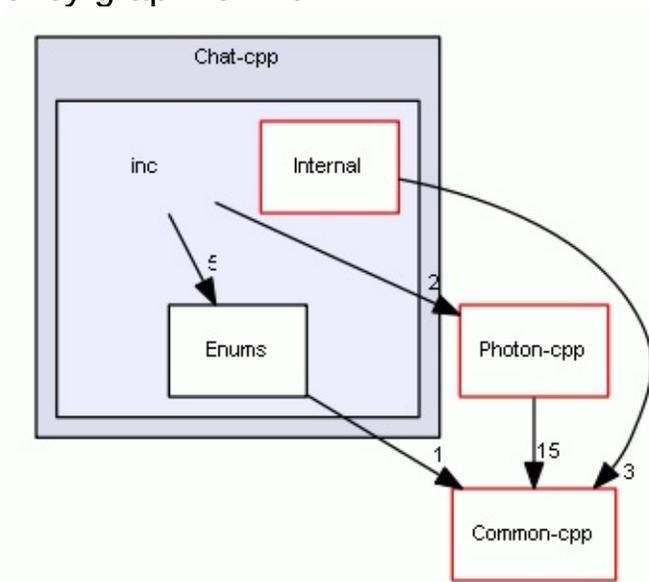
Photon C++

Client API 4.1.12.2

Chat-cpp > inc

inc Directory Reference

Directory dependency graph for inc:



Directories

Files

file [Chat-cpp/inc/AuthenticationValues.h](#)

file [Channel.h](#)

file [Chat-cpp/inc/Client.h](#)

file [Chat-cpp/inc/Listener.h](#)

file [Chat-cpp/inc/Peer.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

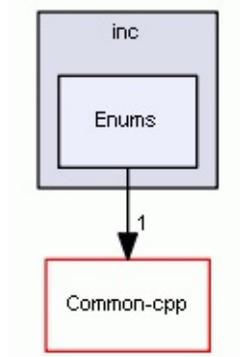
Photon C++

Client API 4.1.12.2

Chat-cpp > inc > Enums

Enums Directory Reference

Directory dependency graph for Enums:



Files

file [ClientState.h](#)

file [Chat-cpp/inc/Enums/CustomAuthenticationType.h](#)

file [Chat-cpp/inc/Enums/DisconnectCause.h](#)

file [Chat-cpp/inc/Enums/ErrorCode.h](#)

file [UserStatus.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

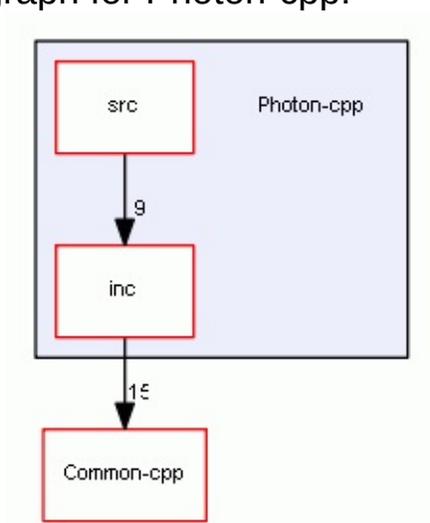
Photon C++

Client API 4.1.12.2

Photon-cpp

Photon-cpp Directory Reference

Directory dependency graph for Photon-cpp:



Directories

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

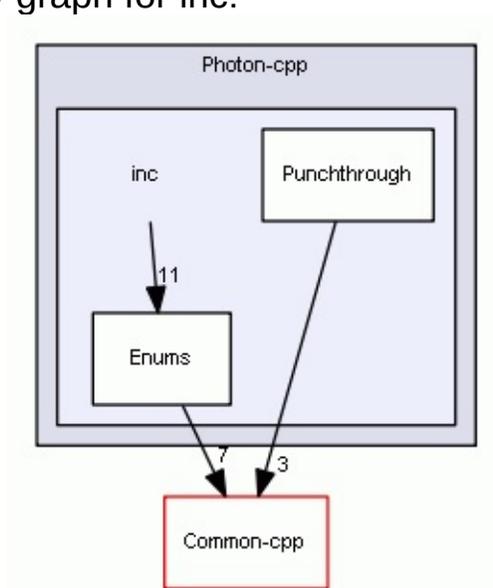
Photon C++

Client API 4.1.12.2

Photon-cpp > inc

inc Directory Reference

Directory dependency graph for inc:



Directories

Files

file [EventData.h](#)

file [LitePeer.h](#)

file [OperationRequest.h](#)

file [OperationResponse.h](#)

file [PhotonListener.h](#)

file [PhotonPeer.h](#)

file [TrafficStats.h](#)

file [TrafficStatsGameLevel.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

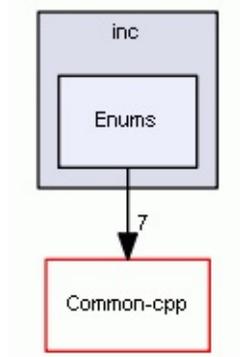
Photon C++

Client API 4.1.12.2

Photon-cpp > inc > Enums

Enums Directory Reference

Directory dependency graph for Enums:



Files

file [ConnectionProtocol.h](#)

file [Photon-cpp/inc/Enums/ErrorCode.h](#)

file [EventCache.h](#)

file [Photon-cpp/inc/Enums/EventCode.h](#)

file [EventKey.h](#)

file [NetworkPort.h](#)

file [Photon-cpp/inc/Enums/OperationCode.h](#)

file [Photon-cpp/inc/Enums/ParameterCode.h](#)

file [PeerState.h](#)

file [ReceiverGroup.h](#)

file [StatusCode.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

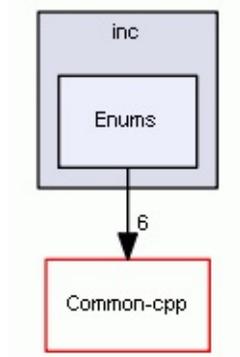
Photon C++

Client API 4.1.12.2

LoadBalancing-cpp > inc > Enums

Enums Directory Reference

Directory dependency graph for Enums:



Files

file [LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h](#)

file [DirectMode.h](#)

file [LoadBalancing-cpp/inc/Enums/DisconnectCause.h](#)

file [LoadBalancing-cpp/inc/Enums/ErrorCode.h](#)

file [LobbyType.h](#)

file [MatchmakingMode.h](#)

file [PeerStates.h](#)

file [RegionSelectionMode.h](#)

file [ServerType.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

 Photon C++
Client API 4.1.12.2

Common-cpp > inc > Enums >

Enums Directory Reference

Files

file [DebugLevel.h](#)

file [TypeCode.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

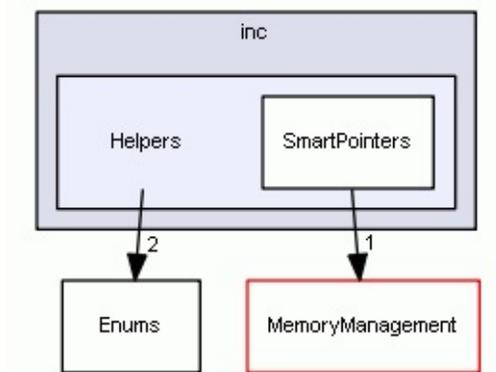
Photon C++

Client API 4.1.12.2

Common-cpp > inc > Helpers

Helpers Directory Reference

Directory dependency graph for Helpers:



Directories

Files

file [IsPrimitiveType.h](#)

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)

 Photon C++
Client API 4.1.12.2

Common-cpp > inc > platform_definitions >

platform_definitions Directory Reference

Directories

© Exit Games ®, all rights reserved.

[Photon Documentation](#) | [Contact](#) | [Terms](#)