# Common-cpp

## Table of data types

© Exit Games ®, all rights reserved.

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

<table>
<thead>
<tr>
<th>Client (C++)</th>
<th>Server (C#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nByte</td>
<td>Byte / byte</td>
</tr>
<tr>
<td>short</td>
<td>Int16 / short</td>
</tr>
<tr>
<td>int</td>
<td>Int32 / int</td>
</tr>
<tr>
<td>int64</td>
<td>Int64 / long</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean / bool</td>
</tr>
<tr>
<td>float</td>
<td>Single / float</td>
</tr>
<tr>
<td>double</td>
<td>Double / double</td>
</tr>
<tr>
<td>JString</td>
<td>String / string</td>
</tr>
<tr>
<td>Hashtable</td>
<td>Hashtable</td>
</tr>
<tr>
<td>Dictionary</td>
<td>Dictionary</td>
</tr>
</tbody>
</table>

We also support arrays for all the above types:

<table>
<thead>
<tr>
<th>Client (C++)</th>
<th>Server (C#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type*</td>
<td>type[]</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Client (C++)</th>
<th>Server (C#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object*</td>
<td>Object[]</td>
</tr>
</tbody>
</table>

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-built-in 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:

<table>
<thead>
<tr>
<th>nByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>int64</td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td>double</td>
</tr>
<tr>
<td>JString</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
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
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
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


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"
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:
   -lCommon-cpp_$(CONFIGURATION)_$(PLATFORM_NAME)
   -lPhoton-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"
Marmalade

For Marmelade 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 your project's .mkb-file

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


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


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"`
Photon C++
Client API  4.1.12.2

Blackberry NDK

Photon supports Blackberry NDK version 10 and newer. Photons 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"
```
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"
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 \texttt{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
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.
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.
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.
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.
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:
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.
# Photon C++ Client API

## 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:

```c
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.

```c
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`. 
### 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).
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>
```
Photon C++
Client API 4.1.12.2

Overview

LoadBalancing-cpp
Chat-cpp
# Namespace List

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

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
<td></td>
</tr>
<tr>
<td>Chat</td>
<td></td>
</tr>
<tr>
<td>ClientState</td>
<td></td>
</tr>
<tr>
<td>CustomAuthenticationType</td>
<td></td>
</tr>
<tr>
<td>DisconnectCause</td>
<td></td>
</tr>
<tr>
<td>ErrorCode</td>
<td></td>
</tr>
<tr>
<td>UserStatus</td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td></td>
</tr>
<tr>
<td>DebugLevel</td>
<td></td>
</tr>
<tr>
<td>MemoryManagement</td>
<td></td>
</tr>
<tr>
<td>TypeCode</td>
<td></td>
</tr>
<tr>
<td>Lite</td>
<td></td>
</tr>
<tr>
<td>EventCache</td>
<td></td>
</tr>
<tr>
<td>EventCode</td>
<td></td>
</tr>
<tr>
<td>EventKey</td>
<td></td>
</tr>
<tr>
<td>OperationCode</td>
<td></td>
</tr>
<tr>
<td>ParameterCode</td>
<td></td>
</tr>
<tr>
<td>ReceiverGroup</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing</td>
<td></td>
</tr>
<tr>
<td>CustomAuthenticationType</td>
<td></td>
</tr>
<tr>
<td>DirectMode</td>
<td></td>
</tr>
<tr>
<td>DisconnectCause</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>Lobby Type</td>
<td></td>
</tr>
<tr>
<td>Matchmaking Mode</td>
<td></td>
</tr>
<tr>
<td>Peer States</td>
<td></td>
</tr>
<tr>
<td>▼ Photon</td>
<td></td>
</tr>
<tr>
<td>Connection Protocol</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td></td>
</tr>
<tr>
<td>Network Port</td>
<td></td>
</tr>
<tr>
<td>Peer State</td>
<td></td>
</tr>
<tr>
<td>Punchthrough</td>
<td></td>
</tr>
<tr>
<td>Status Code</td>
<td></td>
</tr>
</tbody>
</table>
## ExitGames Namespace Reference

Namespaces
Namespaces

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lite</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoadBalancing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ExitGames::Chat
Namespace Reference
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientState</td>
</tr>
<tr>
<td>CustomAuthenticationType</td>
</tr>
<tr>
<td>DisconnectCause</td>
</tr>
<tr>
<td>ErrorCode</td>
</tr>
<tr>
<td>UserStatus</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>class</td>
</tr>
</tbody>
</table>
Variables

const EG_CHAR *const REGION
| ExitGames | Chat | ClientState | Variables |

ExitGames::Chat::ClientState Namespace Reference
<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int <strong>Uninitialized Peer</strong></td>
<td>is created but not used yet.</td>
</tr>
<tr>
<td>static const int <strong>ConnectingToNameServer</strong></td>
<td>Connecting to Name Server (includes connect authenticate and joining the lobby)</td>
</tr>
<tr>
<td>static const int <strong>ConnectedToNameServer</strong></td>
<td>Connected to Name Server.</td>
</tr>
<tr>
<td>static const int <strong>Authenticating</strong></td>
<td>Authenticating.</td>
</tr>
<tr>
<td>static const int <strong>Authenticated</strong></td>
<td>Authenticated.</td>
</tr>
<tr>
<td>static const int <strong>DisconnectingFromNameServer</strong></td>
<td>Transition from Name to Chat Server.</td>
</tr>
<tr>
<td>static const int <strong>ConnectingToFrontEnd</strong></td>
<td>Transition to Chat Server.</td>
</tr>
<tr>
<td>static const int <strong>ConnectedToFrontEnd</strong></td>
<td>Connected to Chat Server. Subscribe to channels and chat here.</td>
</tr>
<tr>
<td>static const int <strong>Disconnecting</strong></td>
<td>The client disconnects (from any server).</td>
</tr>
<tr>
<td>static const int <strong>Disconnected</strong></td>
<td>The client is no longer connected (to any server). Connect to Name Server to go on.</td>
</tr>
</tbody>
</table>
Detailed Description

Possible states for a Client.

ClientState
ExitGames::Chat::CustomAuthenticationType
Namespace Reference
## Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOM</td>
<td>Use a custom authentication service.</td>
<td></td>
</tr>
<tr>
<td>STEAM</td>
<td>Authenticates users by their Steam Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>FACEBOOK</td>
<td>Authenticates users by their Facebook Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>OCULUS</td>
<td>Authenticates users by their Oculus Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>PLAYSTATION</td>
<td>Authenticates users by their PSN Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>XBOX</td>
<td>Authenticates users by their XBox Network Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td>Disables custom authentication.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Options for optional "Custom Authentication" services used with Photon. Used when the client sends an authentication request to the server.
<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Chat</th>
<th>DisconnectCause</th>
<th>Variables</th>
</tr>
</thead>
</table>

**ExitGames::Chat::DisconnectCause**

Namespace Reference
## Variables

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int NONE</td>
<td>No error was tracked.</td>
</tr>
<tr>
<td>static const int DISCONNECT_BY_SERVER_USER_LIMIT</td>
<td>OnStatusChanged: The CCUs count of your Photon Server License is exhausted (temporarily).</td>
</tr>
<tr>
<td>static const int EXCEPTION_ON_CONNECT</td>
<td>OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.</td>
</tr>
<tr>
<td>static const int DISCONNECT_BY_SERVER</td>
<td>OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).</td>
</tr>
<tr>
<td>static const int DISCONNECT_BY_SERVER_LOGIC</td>
<td>OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).</td>
</tr>
<tr>
<td>static const int TIMEOUT_DISCONNECT</td>
<td>OnStatusChanged: This client detected that the server’s responses are not received in due time. Maybe you send / receive too much?</td>
</tr>
<tr>
<td>static const int EXCEPTION</td>
<td>OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.</td>
</tr>
<tr>
<td>static const int INVALID_AUTHENTICATION</td>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.</td>
</tr>
<tr>
<td>Static constant</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>INVALID_REGION</td>
<td>OnOperationResponse: Authenticate when the app’s Photon Cloud subscription is locked to some (other) region(s). Update your subscription or master server address.</td>
</tr>
<tr>
<td>OPERATION_NOT_ALLOWED_IN_CURRENT_STATE</td>
<td>OnOperationResponse: Operation that's (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.</td>
</tr>
<tr>
<td>CUSTOM_AUTHENTICATION_FAILED</td>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Chat</th>
<th>ErrorCode</th>
<th>Variables</th>
</tr>
</thead>
</table>

**ExitGames::Chat::ErrorCode Namespace Reference**
<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int OPERATION_DENIED</td>
</tr>
<tr>
<td>static const int OPERATION_INVALID</td>
</tr>
<tr>
<td>static const int INTERNAL_SERVER_ERROR</td>
</tr>
<tr>
<td>static const int OK</td>
</tr>
<tr>
<td>static const int INVALID_AUTHENTICATION</td>
</tr>
<tr>
<td>static const int GAME_ID_ALREADY_EXISTS</td>
</tr>
<tr>
<td>static const int GAME_FULL</td>
</tr>
<tr>
<td>static const int GAME_CLOSED</td>
</tr>
<tr>
<td>static const int ALREADY_MATCHED</td>
</tr>
<tr>
<td>static const int SERVER_FULL</td>
</tr>
<tr>
<td>static const int USER_BLOCKED</td>
</tr>
<tr>
<td>static const int NO_MATCH_FOUND</td>
</tr>
<tr>
<td>static const int GAME_DOES_NOT_EXIST</td>
</tr>
<tr>
<td>static const int MAX_CCU_REACHED</td>
</tr>
<tr>
<td>static const int INVALID_REGION</td>
</tr>
<tr>
<td>static const int CUSTOM_AUTHENTICATION_FAILED</td>
</tr>
</tbody>
</table>
ExitGames::Chat::UserStatus Namespace Reference
<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int</td>
<td><strong>OFFLINE</strong></td>
</tr>
<tr>
<td></td>
<td>Offline.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>INVISIBLE</strong></td>
</tr>
<tr>
<td></td>
<td>Be invisible to everyone. Sends no message.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ONLINE</strong></td>
</tr>
<tr>
<td></td>
<td>Online and available.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>AWAY</strong></td>
</tr>
<tr>
<td></td>
<td>Online but not available.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>DND</strong></td>
</tr>
<tr>
<td></td>
<td>Do not disturb.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>LFG</strong></td>
</tr>
<tr>
<td></td>
<td>Looking For Game/Group. Could be used when you want to be invited or do matchmaking.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>PLAYING</strong></td>
</tr>
<tr>
<td></td>
<td>Could be used when in a room, playing.</td>
</tr>
</tbody>
</table>
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
ExitGames::Common Namespace Reference
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebugLevel</td>
</tr>
<tr>
<td>MemoryManagement</td>
</tr>
<tr>
<td>TypeCode</td>
</tr>
</tbody>
</table>
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>ANSIString</td>
</tr>
<tr>
<td>class</td>
<td>Base</td>
</tr>
<tr>
<td>class</td>
<td>BaseCharString</td>
</tr>
<tr>
<td>class</td>
<td>BaseListener</td>
</tr>
<tr>
<td>class</td>
<td>CustomType</td>
</tr>
<tr>
<td>class</td>
<td>CustomTypeBase</td>
</tr>
<tr>
<td>class</td>
<td>CustomTypeFactory</td>
</tr>
<tr>
<td>class</td>
<td>DeSerializer</td>
</tr>
<tr>
<td>class</td>
<td>Dictionary</td>
</tr>
<tr>
<td>class</td>
<td>DictionaryBase</td>
</tr>
<tr>
<td>class</td>
<td>EGTime</td>
</tr>
<tr>
<td>class</td>
<td>Hashtable</td>
</tr>
<tr>
<td>class</td>
<td>JString</td>
</tr>
<tr>
<td>class</td>
<td>JVector</td>
</tr>
<tr>
<td>class</td>
<td>KeyObject</td>
</tr>
<tr>
<td>class</td>
<td>LogFormatOptions</td>
</tr>
<tr>
<td>class</td>
<td>Logger</td>
</tr>
<tr>
<td>class</td>
<td>Object</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>class</td>
<td>Serializer</td>
</tr>
<tr>
<td>class</td>
<td>ToString</td>
</tr>
<tr>
<td>class</td>
<td>UTF8String</td>
</tr>
<tr>
<td>class</td>
<td>ValueObject</td>
</tr>
</tbody>
</table>
Detailed Description

Common

© Exit Games ®, all rights reserved.
## ExitGames::Common::DebugLevel Namespace Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Common</th>
<th>DebugLevel</th>
<th>Variables</th>
</tr>
</thead>
</table>

### Photon C++ Client API 4.1.12.2
## Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int OFF</td>
<td>No debug output.</td>
<td></td>
</tr>
<tr>
<td>static const int ERRORS</td>
<td>Only error descriptions.</td>
<td></td>
</tr>
<tr>
<td>static const int WARNINGS</td>
<td>Warnings and errors.</td>
<td></td>
</tr>
<tr>
<td>static const int INFO</td>
<td>Information about internal workflows, warnings and errors.</td>
<td></td>
</tr>
<tr>
<td>static const int ALL</td>
<td>Most complete workflow description (but lots of debug output), info, warnings and errors.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

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

DebugLevel
### ExitGames::Common::MemoryManagement Namespace Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Common</th>
<th>MemoryManagement</th>
<th>Classes</th>
<th>Functions</th>
</tr>
</thead>
</table>

**Photon C++ Client API 4.1.12.2**
<table>
<thead>
<tr>
<th>class</th>
<th>AllocatorInterface</th>
</tr>
</thead>
</table>

Classes
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 objects that they allocate and deallocate.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setMaxAllocSize (size_t maxAllocSize)</td>
<td>Set the maximum allocated size</td>
</tr>
<tr>
<td>void setMaxSizeForAllocatorUsage (size_t maxSizeForAllocatorUsage)</td>
<td>Set the maximum size for allocator usage</td>
</tr>
<tr>
<td>void setAllocator (ExitGames::Common::MemoryManagement::AllocatorInterface &amp;allocator)</td>
<td>Set the allocator</td>
</tr>
<tr>
<td>void setAllocatorToDefault (void)</td>
<td>Set the allocator to default</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template&lt;typename Ftype &gt; Ftype * allocate (void)</td>
<td>Allocate memory</td>
</tr>
<tr>
<td>template&lt;typename Ftype &gt; Ftype * allocateArray (size_t count)</td>
<td>Allocate an array</td>
</tr>
<tr>
<td>template&lt;typename Ftype &gt; Ftype * reallocateArray (Ftype *p, size_t count)</td>
<td>Reallocation of an array</td>
</tr>
<tr>
<td>template&lt;typename Ftype &gt; void deallocate (const Ftype *p)</td>
<td>Deallocate memory</td>
</tr>
<tr>
<td>template&lt;typename Ftype &gt; void deallocateArray (const Ftype *p)</td>
<td>Deallocate an array</td>
</tr>
</tbody>
</table>
Detailed Description

MemoryManagement
Function Documentation
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 a memory manager might choose to request bigger amounts of memory at a time and give out smaller chunks of them to the application code. This way it can reduce the amount of unnecessary memory requests to the OS. Depending on the memory requests that come in from the application code, a memory manager might scale up its own requests to the OS.

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

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

Now let's imagine that there is a pool that serves requests between 65 and 128 bytes in size each and for this purpose keeps a bunch of 128 byte blocks around to give out to requesters. In the beginning it might just keep very few such blocks, as the memory manager does not know, how many blocks of this size an app might need to use in parallel. When an app requests lots of these blocks, the pool will scale accordingly and to not need to do a request to the OS too often, it might increase the size of its own requests. i.e. at first it could have just 4 blocks, then when it resizes, it would allocate memory for another 4 blocks, then another 8, then for another, 16, then 32, 64, 128, 256, 512, 1024 blocks more, and so on.

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

Accordingly with the same 8192 bytes limit in place a pool that holds 1024 byte blocks might present a different resize pattern depending on the memory requests that come in. For instance, it might have 16 blocks around. Then when it resizes, it might request another 16, then another 32, then 64, 128, 256, 512, 1024, 2048, 4096, 8192 bytes more, and so on.

§ setMaxAllocSize()
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 memory manager decides so (for example a pool based memory manager might serve differently sized requests from different pools that resize independently from each other).

**Remarks**
This function forwards the passed in value to the currently set allocator (see `setAllocator()` ) and does not store it itself. For this reason a call to this function only affects the settings of the currently set allocator and not those of a future 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 has 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 with a size above the limit get redirected 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 request that memory from the memory manager instead, which requests bigger amounts of memory from the OS, splits them up into smaller blocks 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 bigger amounts of memory.

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

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 currently set one.

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

See also
setAllocator(), AllocatorInterface
### setAllocator()

```cpp
void ExitGames::Common::MemoryManagement::setAllocator ( ExitGames::Common::MemoryManagement::AllocatorInterface allocator );
```

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

All dynamic memory allocation requests by the Photon Client libraries go through one of the **Low Level Memory Management macros**. The application code can also use these functions and macros for its own memory requests if its developer chooses so.

Each request for an amount of memory that does not exceed the limit set by the allocator. Photon provides a default general-purpose allocator that uses pool-based memory management and that works well for most applications.

However you can set your own allocator through this function and Photon will use that allocator for any memory requests that happen afterwards.


**Remarks**

Photons memory management stores the address of the allocator that served a specific memory request and forwards a request to free memory to the same allocator that allocated that memory.

This means a) that you can set a different allocator as often as you like once set allocator available even when it is no longer set as the current allocator, memory that once got requested from it, got returned to it and non of it is still in use.

If you want to already set an initial custom allocator before any global or file-level static instances of Photon classes get constructed, then you need to replace `AllocatorInterface::get()`.

**Parameters**

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

**See also**

- `setMaxSizeForAllocatorUsage()`, `AllocatorInterface`, `AllocatorInterface::get()`
§ setAllocatorToDefault()

```c
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 count )

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 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()

```cpp
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()

```cpp
void ExitGames::Common::MemoryManagement::deallocateArray ( const Ftype * p )
```

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 adress has not previously been returned by a call to `allocateArray()` or `reallocateArray()` and also isn't NULL, then the behavior is undefined.

**Parameters**

- `p` the address of the array, that should be deallocated.
<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Common</th>
<th>TypeCode</th>
<th>Variables</th>
</tr>
</thead>
</table>

**ExitGames::Common::TypeCode Namespace Reference**
<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static const nByte BYTE</strong></td>
</tr>
<tr>
<td>nByte</td>
</tr>
<tr>
<td><strong>static const nByte SHORT</strong></td>
</tr>
<tr>
<td>short</td>
</tr>
<tr>
<td><strong>static const nByte INTEGER</strong></td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td><strong>static const nByte LONG</strong></td>
</tr>
<tr>
<td>int64</td>
</tr>
<tr>
<td><strong>static const nByte FLOAT</strong></td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td><strong>static const nByte DOUBLE</strong></td>
</tr>
<tr>
<td>double</td>
</tr>
<tr>
<td><strong>static const nByte BOOLEAN</strong></td>
</tr>
<tr>
<td>bool</td>
</tr>
<tr>
<td><strong>static const nByte STRING</strong></td>
</tr>
<tr>
<td>JString</td>
</tr>
<tr>
<td><strong>static const nByte HASHTABLE</strong></td>
</tr>
<tr>
<td>Hashtable</td>
</tr>
<tr>
<td><strong>static const nByte DICTIONARY</strong></td>
</tr>
<tr>
<td>Dictionary</td>
</tr>
<tr>
<td><strong>static const nByte OBJECT</strong></td>
</tr>
<tr>
<td>Object, only allowed for arrays!</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>ARRAY</td>
</tr>
<tr>
<td>BYTEARRAY</td>
</tr>
<tr>
<td>PHOTON_COMMAND</td>
</tr>
<tr>
<td>EG_NULL</td>
</tr>
<tr>
<td>CUSTOM</td>
</tr>
<tr>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
Detailed Description

TypeCode

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
ExitGames::Lite
Namespace Reference
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventCache</td>
</tr>
<tr>
<td>EventCode</td>
</tr>
<tr>
<td>EventKey</td>
</tr>
<tr>
<td>OperationCode</td>
</tr>
<tr>
<td>ParameterCode</td>
</tr>
<tr>
<td>ReceiverGroup</td>
</tr>
</tbody>
</table>
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>LitePeer</th>
</tr>
</thead>
</table>

Detailed Description

Lite

© Exit Games®, all rights reserved.
ExitGames::Lite::EventCache Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>DO_NOT_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>MERGE_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REPLACE_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REMOVE_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ADD_TO_ROOM_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ADD_TO_ROOM_CACHE_GLOBAL</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REMOVE_FROM_ROOM_CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REMOVE_FROM_ROOM_CACHE_FOR_ACTORS</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SLICE_INC_INDEX</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SLICE_SET_INDEX</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SLICE_PURGE_INDEX</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SLICE_PURGE_UP_TO_INDEX</td>
</tr>
</tbody>
</table>
Detailed Description

EventCache
ExitGames::Lite::EventCode Namespace Reference
## Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td><strong>JOIN</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>LEAVE</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>PROPERTIES_CHANGED</strong></td>
</tr>
</tbody>
</table>
Detailed Description

EventCode
ExitGames::Lite::EventKey Namespace Reference
## Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const</td>
<td><code>ACTORNR</code></td>
</tr>
<tr>
<td>nByte</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>TARGET_ACTORNR</code></td>
</tr>
<tr>
<td></td>
<td><code>ACTORLIST</code></td>
</tr>
<tr>
<td></td>
<td><code>PROPERTIES</code></td>
</tr>
<tr>
<td></td>
<td><code>ACTORPROPERTIES</code></td>
</tr>
<tr>
<td></td>
<td><code>GAMEPROPERTIES</code></td>
</tr>
<tr>
<td></td>
<td><code>DATA</code></td>
</tr>
</tbody>
</table>
Detailed Description

EventKey

© Exit Games ®, all rights reserved.
ExitGames::Lite::OperationCode Namespace Reference
<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td><strong>JOIN</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>LEAVE</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>RAISE_EV</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>SETPROPERTIES</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>GETPROPERTIES</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>CHANGE_GROUPS</strong></td>
</tr>
</tbody>
</table>
Detailed Description

**OperationCode**
ExitGames::Lite::ParameterCode Namespace Reference
Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>GAMEID</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>TARGET_ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTOR_LIST</td>
</tr>
<tr>
<td>static const nByte</td>
<td>PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>BROADCAST</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTOR_PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GAME_PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>RECEIVER_GROUP</td>
</tr>
<tr>
<td>static const nByte</td>
<td>DATA</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CODE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GROUP</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REMOVE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ADD</td>
</tr>
</tbody>
</table>
Detailed Description

ParameterCode
ExitGames::Lite::ReceiverGroup Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>OTHERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>ALL</td>
</tr>
<tr>
<td>static const nByte</td>
<td>MASTER_CLIENT</td>
</tr>
</tbody>
</table>
Detailed Description

ReceiverGroup
ExitGames::LoadBalancing Namespace Reference
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomAuthenticationType</td>
</tr>
<tr>
<td>DirectMode</td>
</tr>
<tr>
<td>DisconnectCause</td>
</tr>
<tr>
<td>ErrorCode</td>
</tr>
<tr>
<td>LobbyType</td>
</tr>
<tr>
<td>MatchmakingMode</td>
</tr>
<tr>
<td>PeerStates</td>
</tr>
<tr>
<td>Class</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>AuthenticationValues</td>
</tr>
<tr>
<td>Client</td>
</tr>
<tr>
<td>FriendInfo</td>
</tr>
<tr>
<td>Listener</td>
</tr>
<tr>
<td>LobbyStatsRequest</td>
</tr>
<tr>
<td>LobbyStatsResponse</td>
</tr>
<tr>
<td>MutablePlayer</td>
</tr>
<tr>
<td>MutableRoom</td>
</tr>
<tr>
<td>Peer</td>
</tr>
<tr>
<td>Player</td>
</tr>
<tr>
<td>RaiseEventOptions</td>
</tr>
<tr>
<td>Room</td>
</tr>
<tr>
<td>RoomOptions</td>
</tr>
<tr>
<td>WebFlags</td>
</tr>
</tbody>
</table>
Detailed Description

LoadBalancing

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
**Namespace Reference**

**Namespace:** ExitGames::LoadBalancing::CustomAuthenticationType

**Variables**
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>CUSTOM</td>
<td>Use a custom authentication service.</td>
</tr>
<tr>
<td>static const nByte</td>
<td>STEAM</td>
<td>Authenticates users by their Steam Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>FACEBOOK</td>
<td>Authenticates users by their Facebook Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>OCULUS</td>
<td>Authenticates users by their Oculus Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>PLAYSTATION</td>
<td>Authenticates users by their PSN Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>XBOX</td>
<td>Authenticates users by their XBox Network Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>NONE</td>
<td>Disables custom authentication.</td>
</tr>
</tbody>
</table>
Detailed Description

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

CustomAuthenticationType
### Namespace Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>LoadBalancing</th>
<th>DirectMode</th>
</tr>
</thead>
</table>

**ExitGames::LoadBalancing::DirectMode**

Variables
### Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td><strong>NONE</strong></td>
<td>Do not create any 2p2 connections between the clients. This is the default.</td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>ALL_TO_ALL</strong></td>
<td>Each client establishes a direct connection with every other client inside the room.</td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>MASTER_TO_ALL</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Detailed Description

Options for optional client to client direct connections - set in RoomOptions during room creation.
### ExitGames::LoadBalancing::DisconnectCause

<table>
<thead>
<tr>
<th>Namespace Reference</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing::DisconnectCause</td>
<td></td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NONE</strong></td>
<td>No error was tracked.</td>
</tr>
<tr>
<td><strong>DISCONNECT_BY_SERVER_USER_LIMIT</strong></td>
<td>OnStatusChanged: The CCUs count of your Photon Server License is exausted (temporarily).</td>
</tr>
<tr>
<td><strong>EXCEPTION_ON_CONNECT</strong></td>
<td>OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.</td>
</tr>
<tr>
<td><strong>DISCONNECT_BY_SERVER</strong></td>
<td>OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).</td>
</tr>
<tr>
<td><strong>DISCONNECT_BY_SERVER_LOGIC</strong></td>
<td>OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).</td>
</tr>
<tr>
<td><strong>TIMEOUT_DISCONNECT</strong></td>
<td>OnStatusChanged: This client detected that the server’s responses are not received in due time. Maybe you send / receive too much?</td>
</tr>
<tr>
<td><strong>EXCEPTION</strong></td>
<td>OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.</td>
</tr>
<tr>
<td><strong>INVALID_AUTHENTICATION</strong></td>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.</td>
</tr>
<tr>
<td>Static Const Int</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>INVALID_REGION</strong></td>
<td>OnOperationResponse: Authenticate when the app’s Photon Cloud subscription is locked to some (other) region(s). Update your subscription or master server address.</td>
</tr>
<tr>
<td><strong>OPERATION_NOT_ALLOWED_IN_CURRENT_STATE</strong></td>
<td>OnOperationResponse: Operation that's (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.</td>
</tr>
<tr>
<td><strong>CUSTOM_AUTHENTICATION_FAILED</strong></td>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.</td>
</tr>
</tbody>
</table>
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
## Photon C++

### Client API 4.1.12.2

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>LoadBalancing</th>
<th>ErrorCode</th>
<th>Variables</th>
</tr>
</thead>
</table>

**Namespace Reference**

`ExitGames::LoadBalancing::ErrorCode`
### Variables

<table>
<thead>
<tr>
<th>Static Const Int</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION_DENIED</td>
<td></td>
</tr>
<tr>
<td>OPERATION_INVALID</td>
<td></td>
</tr>
<tr>
<td>INTERNAL_SERVER_ERROR</td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>INVALID_AUTHENTICATION</td>
<td></td>
</tr>
<tr>
<td>GAME_ID_ALREADY_EXISTS</td>
<td></td>
</tr>
<tr>
<td>GAME_FULL</td>
<td></td>
</tr>
<tr>
<td>GAME_CLOSED</td>
<td></td>
</tr>
<tr>
<td>ALREADY_MATCHED</td>
<td></td>
</tr>
<tr>
<td>SERVER_FULL</td>
<td></td>
</tr>
<tr>
<td>USER_BLOCKED</td>
<td></td>
</tr>
<tr>
<td>NO_MATCH_FOUND</td>
<td></td>
</tr>
<tr>
<td>GAME.Does Not Exist</td>
<td></td>
</tr>
<tr>
<td>MAX_CCU_REACHED</td>
<td></td>
</tr>
<tr>
<td>INVALID_REGION</td>
<td></td>
</tr>
<tr>
<td>CUSTOM_AUTHENTICATION_FAILED</td>
<td></td>
</tr>
<tr>
<td>AUTHENTICATION_TOKEN_EXPIRED</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Plugin reported error</td>
<td>PLUGIN_REPORTED_ERROR</td>
</tr>
<tr>
<td>Plugin mismatch</td>
<td>PLUGIN_MISMATCH</td>
</tr>
<tr>
<td>Join failed peer already joined</td>
<td>JOIN_FAILED_PEER_ALREADY_JOINED</td>
</tr>
<tr>
<td>Join failed found inactive joiner</td>
<td>JOIN_FAILED_FOUND_INACTIVE_JOINER</td>
</tr>
<tr>
<td>Join failed with rejoiner not found</td>
<td>JOIN_FAILED_WITH_REJOINER_NOT_FOUND</td>
</tr>
<tr>
<td>Join failed found excluded user id</td>
<td>JOIN_FAILED_FOUND_EXCLUDED_USER_ID</td>
</tr>
<tr>
<td>Join failed found active joiner</td>
<td>JOIN_FAILED_FOUND_ACTIVE_JOINER</td>
</tr>
<tr>
<td>HTTP limit reached</td>
<td>HTTP_LIMIT_REACHED</td>
</tr>
<tr>
<td>External HTTP call failed</td>
<td>EXTERNAL_HTTP_CALL_FAILED</td>
</tr>
<tr>
<td>Slot error</td>
<td>SLOT_ERROR</td>
</tr>
<tr>
<td>Invalid encryption parameters</td>
<td>INVALID_ENCRYPTION_PARAMETERS</td>
</tr>
</tbody>
</table>
Detailed Description

ErrorCode
ExitGames::LoadBalancing::LobbyType
Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>DEFAULT</th>
<th>This lobby type is used unless another lobby type is specified. <strong>Room</strong> lists will be sent and <strong>Client::opJoinRandomRoom()</strong> can filter by matching properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>SQL_LOBBY</td>
<td>This lobby type lists rooms like type DEFAULT but SQL-like &quot;where&quot; clauses for filtering can be used with <strong>Client::opJoinRandomRoom()</strong>. This allows 'bigger', 'less', 'or' and 'and' combinations.</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ASYNC_RANDOM_LOBBY</td>
<td>This lobby does not send room lists. It is only used for <strong>Client::opJoinRandomRoom()</strong>. It keeps rooms available for matchmaking for a while even when there are only inactive users left.</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>ExitGames</th>
<th>LoadBalancing</th>
<th>MatchmakingMode</th>
<th>Variables</th>
</tr>
</thead>
</table>

ExitGames::LoadBalancing::MatchmakingMode
Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL_ROOM</td>
<td>Fills up rooms (oldest first) to get players together as fast as possible. Default. Makes most sense with MaxPlayers &gt; 0 and games that can only start with more players.</td>
</tr>
<tr>
<td>SERIAL_MATCHING</td>
<td>Distributes players across available rooms sequentially but takes filters into account. Without filters, rooms get players evenly distributed.</td>
</tr>
<tr>
<td>RANDOM_MATCHING</td>
<td>Joins a (fully) random room. Expected properties must match, but aside from this, any available room might be selected.</td>
</tr>
</tbody>
</table>
Detailed Description


**MatchmakingMode**
ExitGames::LoadBalancing::PeerStates
Namespace Reference
## Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int Uninitialized</td>
<td></td>
</tr>
<tr>
<td>static const int PeerCreated</td>
<td></td>
</tr>
<tr>
<td>static const int ConnectingToNameserver</td>
<td></td>
</tr>
<tr>
<td>static const int ConnectedToNameserver</td>
<td></td>
</tr>
<tr>
<td>static const int DisconnectingFromNameserver</td>
<td></td>
</tr>
<tr>
<td>static const int Connecting</td>
<td></td>
</tr>
<tr>
<td>static const int Connected</td>
<td></td>
</tr>
<tr>
<td>static const int WaitingForCustomAuthenticationNextStepCall</td>
<td></td>
</tr>
<tr>
<td>static const int Authenticated</td>
<td></td>
</tr>
<tr>
<td>static const int JoinedLobby</td>
<td></td>
</tr>
<tr>
<td>static const int DisconnectingFromMasterserver</td>
<td></td>
</tr>
<tr>
<td>static const int ConnectingToGameserver</td>
<td></td>
</tr>
<tr>
<td>static const int ConnectedToGameserver</td>
<td></td>
</tr>
<tr>
<td>static const int AuthenticatedOnGameServer</td>
<td></td>
</tr>
<tr>
<td>static const int Joining</td>
<td></td>
</tr>
<tr>
<td>static const int Joined</td>
<td></td>
</tr>
<tr>
<td>static const int Leaving</td>
<td></td>
</tr>
<tr>
<td>Static const int</td>
<td>Value</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Left</td>
<td></td>
</tr>
<tr>
<td>DisconnectingFromGameserver</td>
<td></td>
</tr>
<tr>
<td>ConnectingToMasterserver</td>
<td></td>
</tr>
<tr>
<td>ConnectedComingFromGameserver</td>
<td></td>
</tr>
<tr>
<td>AuthenticatedComingFromGameserver</td>
<td></td>
</tr>
<tr>
<td>Disconnecting</td>
<td></td>
</tr>
<tr>
<td>Disconnected</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

PeerStates

© Exit Games®, all rights reserved.
Photon Documentation | Contact | Terms
ExitGames::Photon
Namespace Reference
### Namespaces

<table>
<thead>
<tr>
<th>ConnectionProtocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorCode</td>
</tr>
<tr>
<td>NetworkPort</td>
</tr>
<tr>
<td>PeerState</td>
</tr>
<tr>
<td>Punchthrough</td>
</tr>
<tr>
<td>StatusCode</td>
</tr>
</tbody>
</table>
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventData</td>
<td></td>
</tr>
<tr>
<td>OperationRequest</td>
<td></td>
</tr>
<tr>
<td>OperationResponse</td>
<td></td>
</tr>
<tr>
<td>PhotonListener</td>
<td></td>
</tr>
<tr>
<td>PhotonPeer</td>
<td></td>
</tr>
<tr>
<td>TrafficStats</td>
<td></td>
</tr>
<tr>
<td>TrafficStatsGameLevel</td>
<td></td>
</tr>
</tbody>
</table>
typedef Common::Dictionary< nByte, Common::Object > OperationRequestParameters
Detailed Description

Photon

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
ExitGames::Photon::ConnectionProtocol
Namespace Reference
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool getIsUDP</code></td>
<td>(nByte connectionProtocol)</td>
</tr>
<tr>
<td><code>bool getIsTCP</code></td>
<td>(nByte connectionProtocol)</td>
</tr>
<tr>
<td><code>bool getIsWebSocket</code></td>
<td>(nByte connectionProtocol)</td>
</tr>
<tr>
<td><code>bool getIsSecure</code></td>
<td>(nByte connectionProtocol)</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP</strong></td>
<td>Use UDP to connect to Photon, which allows you to send operations reliable or unreliable on demand.</td>
<td></td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td>Use TCP to connect to Photon.</td>
<td></td>
</tr>
<tr>
<td><strong>WS</strong></td>
<td>Use websockets to connect to Photon.</td>
<td></td>
</tr>
<tr>
<td><strong>WSS</strong></td>
<td>Use secure websockets to connect to Photon.</td>
<td></td>
</tr>
<tr>
<td><strong>DEFAULT</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

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

ConnectionProtocol
Function Documentation
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.
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.
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.
ExitGames::Photon::ErrorCode Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>No error.</td>
<td><strong>SUCCESS</strong></td>
</tr>
<tr>
<td>General failure.</td>
<td><strong>EFAILED</strong></td>
</tr>
<tr>
<td>Out of memory.</td>
<td><strong>ENOMEMORY</strong></td>
</tr>
<tr>
<td>NULL class object.</td>
<td><strong>EBADCLASS</strong></td>
</tr>
<tr>
<td>Invalid parameter.</td>
<td><strong>EBADPARM</strong></td>
</tr>
<tr>
<td>Context (system, interface, etc.) is busy.</td>
<td><strong>EITEMBUSY</strong></td>
</tr>
<tr>
<td>No network error, successful operation.</td>
<td><strong>NET_SUCCESS</strong></td>
</tr>
<tr>
<td>Unsuccessful operation.</td>
<td><strong>NET_ERROR</strong></td>
</tr>
<tr>
<td>Network subsystem unavailable.</td>
<td><strong>NET_ENETNONET</strong></td>
</tr>
<tr>
<td>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.</td>
<td><strong>NET_MSGSIZE</strong></td>
</tr>
<tr>
<td>static const int NET_ENOTCONN</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

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

**errorCode**
ExitGames::Photon::NetworkPort Namespace Reference
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>TCP</td>
</tr>
<tr>
<td>struct</td>
<td>UDP</td>
</tr>
<tr>
<td>struct</td>
<td>UDPAIternative</td>
</tr>
<tr>
<td>struct</td>
<td>WS</td>
</tr>
<tr>
<td>struct</td>
<td>WSS</td>
</tr>
</tbody>
</table>
Detailed Description

NetworkPort
### ExitGames::Photon::PeerState Namespace Reference

The table below provides an overview of the variables available in the ExitGames::Photon::PeerState namespace.
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int</td>
<td>DISCONNECTED</td>
<td>The peer is disconnected and can't call Operations. Call PhotonPeer_connect().</td>
</tr>
<tr>
<td>static const int</td>
<td>CONNECTING</td>
<td>The peer is establishing the connection: opening a socket, exchanging packages with Photon.</td>
</tr>
<tr>
<td>static const int</td>
<td>INITIALIZING_APPLICATION</td>
<td>The connection is established and now sends the application name to Photon. You set the &quot;application name&quot; by calling PhotonPeer_connect().</td>
</tr>
<tr>
<td>static const int</td>
<td>CONNECTED</td>
<td>The peer is connected and initialized (selected an application). You can now use operations.</td>
</tr>
<tr>
<td>static const int</td>
<td>DISCONNECTING</td>
<td>The peer is disconnecting. It sent a disconnect to the server, which will acknowledge closing the connection.</td>
</tr>
</tbody>
</table>
Detailed Description

PeerState
ExitGames::Photon::Punchthrough
Namespace Reference
## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class <strong>Puncher</strong></td>
<td></td>
</tr>
<tr>
<td>class <strong>PunchListener</strong></td>
<td></td>
</tr>
<tr>
<td>class <strong>RelayClient</strong></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Punchthrough
ExitGames::Photon::StatusCode Namespace Reference
### Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EXCEPTION_ON_CONNECT</code></td>
<td>the PhotonPeer encountered an exception while opening the incoming connection to the server. The server could be down / not running.</td>
</tr>
<tr>
<td><code>CONNECT</code></td>
<td>the PhotonPeer is connected.</td>
</tr>
<tr>
<td><code>DISCONNECT</code></td>
<td>the PhotonPeer just disconnected.</td>
</tr>
<tr>
<td><code>EXCEPTION</code></td>
<td>the PhotonPeer encountered an exception and will disconnect, too.</td>
</tr>
<tr>
<td><code>QUEUE_OUTGOING_RELIABLE_WARNING</code></td>
<td>PhotonPeer outgoing queue is filling up. Send more often.</td>
</tr>
<tr>
<td><code>QUEUE_OUTGOING_UNRELIABLE_WARNING</code></td>
<td>PhotonPeer outgoing queue is filling up. Send more often.</td>
</tr>
<tr>
<td><code>SEND_ERROR</code></td>
<td>Sending command failed. Either not connected, or the requested channel is bigger than the number of initialized channels.</td>
</tr>
<tr>
<td><code>QUEUE_OUTGOING_ACKS_WARNING</code></td>
<td>PhotonPeer outgoing queue is filling up. Send more often.</td>
</tr>
<tr>
<td><code>QUEUE_INCOMING_RELIABLE_WARNING</code></td>
<td>PhotonPeer incoming reliable queue is filling up.</td>
</tr>
<tr>
<td>Static Constant</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>QUEUE_INCOMING_UNRELIABLE_WARNING</code></td>
<td>PhotonPeer incoming unreliable queue is filling up. Dispatch more often.</td>
</tr>
<tr>
<td><code>QUEUE_SENT_WARNING</code></td>
<td>PhotonPeer sent queue is filling up. Check, why the server does not acknowledge your sent reliable commands.</td>
</tr>
<tr>
<td><code>INTERNAL_RECEIVE_EXCEPTION</code></td>
<td>Exception, if a server cannot be connected. Most likely, the server is not responding. Ask the user to try again later.</td>
</tr>
<tr>
<td><code>TIMEOUT_DISCONNECT</code></td>
<td>Disconnection due to a timeout (client did no longer receive ACKs from server).</td>
</tr>
<tr>
<td><code>DISCONNECT_BY_SERVER</code></td>
<td>Disconnect by server due to timeout (received a disconnect command, cause server misses ACKs of client).</td>
</tr>
<tr>
<td><code>DISCONNECT_BY_SERVER_USER_LIMIT</code></td>
<td>Disconnect by server due to concurrent user limit reached (received a disconnect command).</td>
</tr>
<tr>
<td><code>DISCONNECT_BY_SERVER_LOGIC</code></td>
<td>Disconnect by server due to server's logic (received a disconnect command).</td>
</tr>
<tr>
<td><code>ENCRYPTION_ESTABLISHED</code></td>
<td>The encryption-setup for secure communication finished successfully.</td>
</tr>
</tbody>
</table>
static const int ENCRYPTION_FAILED_TO_ESTABLISH
The encryption-setup failed for some reason. Check debug logs.
Detailed Description

StatusCode
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`
- 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
- EBADPARM : 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
- f -

- g -

- h -

- i -
ExitGames::LoadBalancing::DisconnectCause
- I -
  - 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

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

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

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

- XBOX : ExitGames::Chat::CustomAuthenticationType, ExitGames::LoadBalancing::CustomAuthenticationType
- `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`
Photon C++
Client API 4.1.12.2

- a -
  - ALL : ExitGames::Common::DebugLevel
  - ALL_TO_ALL : ExitGames::LoadBalancing::DirectMode
  - ARRAY : ExitGames::Common::TypeCode
  - ASYNC_RANKMOM_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
- EBADPARM : 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
- 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

- FACEBOOK : ExitGames::Chat::CustomAuthenticationType,  
  ExitGames::LoadBalancing::CustomAuthenticationType

- FILL_ROOM : ExitGames::LoadBalancing::MatchmakingMode

- FLOAT : ExitGames::Common::TypeCode

- HASHTABLE : ExitGames::Common::TypeCode

- 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

- LFG : ExitGames::Chat::UserStatus

- LONG : ExitGames::Common::TypeCode

- 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
- 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`
## Class List

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
<td>AuthenticationValues</td>
</tr>
<tr>
<td></td>
<td>Channel</td>
</tr>
<tr>
<td></td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>Listener</td>
</tr>
<tr>
<td></td>
<td>Peer</td>
</tr>
<tr>
<td>Common</td>
<td>MemoryManagement</td>
</tr>
<tr>
<td></td>
<td>AllocatorInterface</td>
</tr>
<tr>
<td></td>
<td>ANSIString</td>
</tr>
<tr>
<td></td>
<td>Base</td>
</tr>
<tr>
<td></td>
<td>BaseCharString</td>
</tr>
<tr>
<td></td>
<td>BaseListener</td>
</tr>
<tr>
<td></td>
<td>CustomType</td>
</tr>
<tr>
<td></td>
<td>CustomTypeBase</td>
</tr>
<tr>
<td></td>
<td>CustomTypeFactory</td>
</tr>
<tr>
<td></td>
<td>DeSerializer</td>
</tr>
<tr>
<td></td>
<td>Dictionary</td>
</tr>
<tr>
<td></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td></td>
<td>EGTIme</td>
</tr>
<tr>
<td>Package</td>
<td>Class/Interface</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Hashtable</td>
<td></td>
</tr>
<tr>
<td>JString</td>
<td></td>
</tr>
<tr>
<td>JVector</td>
<td></td>
</tr>
<tr>
<td>KeyObject</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions</td>
<td></td>
</tr>
<tr>
<td>Logger</td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td></td>
</tr>
<tr>
<td>Serializer</td>
<td></td>
</tr>
<tr>
<td>ToString</td>
<td></td>
</tr>
<tr>
<td>UTF8String</td>
<td></td>
</tr>
<tr>
<td>ValueObject</td>
<td></td>
</tr>
<tr>
<td>Lite</td>
<td>LitePeer</td>
</tr>
<tr>
<td>LoadBalancing</td>
<td>AuthenticationValues</td>
</tr>
<tr>
<td></td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>FriendInfo</td>
</tr>
<tr>
<td></td>
<td>Listener</td>
</tr>
<tr>
<td></td>
<td>LobbyStatsRequest</td>
</tr>
<tr>
<td></td>
<td>LobbyStatsResponse</td>
</tr>
<tr>
<td></td>
<td>MutablePlayer</td>
</tr>
<tr>
<td></td>
<td>MutableRoom</td>
</tr>
<tr>
<td></td>
<td>Peer</td>
</tr>
<tr>
<td></td>
<td>Player</td>
</tr>
<tr>
<td></td>
<td>RaiseEventOptions</td>
</tr>
<tr>
<td></td>
<td>Room</td>
</tr>
<tr>
<td></td>
<td>RoomOptions</td>
</tr>
<tr>
<td></td>
<td>WebFlags</td>
</tr>
<tr>
<td>Photon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NetworkPort</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
</tr>
<tr>
<td>UDPAlternative</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td></td>
</tr>
<tr>
<td>WSS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Punchthrough</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puncher</td>
</tr>
<tr>
<td>PunchListener</td>
</tr>
<tr>
<td>RelayClient</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EventData</th>
</tr>
</thead>
<tbody>
<tr>
<td>OperationRequest</td>
</tr>
<tr>
<td>OperationResponse</td>
</tr>
<tr>
<td>PhotonListener</td>
</tr>
<tr>
<td>PhotonPeer</td>
</tr>
<tr>
<td>TrafficStats</td>
</tr>
<tr>
<td>TrafficStatsGameLevel</td>
</tr>
</tbody>
</table>
## AuthenticationValues

### Class Reference

Inheritance diagram for AuthenticationValues:

```
    ToString
       ↓
      Base
       ↓
AuthenticationValues
```

Collaboration diagram for AuthenticationValues:

```
    ToString
       ↓
      Base
       ↓
AuthenticationValues
```

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AuthenticationValues (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>nByte getType (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>AuthenticationValues &amp; setType (nByte type)</code></td>
<td></td>
</tr>
<tr>
<td><code>const Common::JString &amp; getParameters (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>AuthenticationValues &amp; setParameters (const Common::JString &amp;parameters)</code></td>
<td></td>
</tr>
<tr>
<td><code>AuthenticationValues &amp; setParametersWithUsernameAndToken (const Common::JString &amp;username, const Common::JString &amp;token)</code></td>
<td></td>
</tr>
<tr>
<td><code>const Common::JVector&lt;nByte&gt; &amp; getData (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>AuthenticationValues &amp; setData (const Common::JVector&lt;nByte&gt; &amp;data)</code></td>
<td></td>
</tr>
<tr>
<td><code>const Common::JString &amp; getSecret (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>const Common::JString &amp; getUserID (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>AuthenticationValues &amp; setUserID (const Common::JString &amp;userID)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual Common::JString &amp; toString (Common::JString &amp;ret, withTypes=false) const</code></td>
<td></td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static void</strong> setListener (const <strong>BaseListener</strong> <em>baseListener)</em></td>
</tr>
<tr>
<td><strong>static int</strong> getDebugOutputLevel (void)</td>
</tr>
<tr>
<td><strong>static bool</strong> setDebugOutputLevel (int debugLevel)</td>
</tr>
<tr>
<td><strong>static const LogFormatOptions &amp;</strong> getLogFormatOptions (void)</td>
</tr>
<tr>
<td><strong>static void</strong> setLogFormatOptions (const LogFormatOptions &amp;options)</td>
</tr>
</tbody>
</table>
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.

Therefor 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()
§ setTimeType()

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

gType(), CustomAuthenticationType
§ getParameters()

```cpp
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()

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&param2=value2&param3=value3)

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

See also
getParameter(), 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()

```cpp
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()

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

**Returns**
the unique user ID

**See also**
`setUserID()`
§ setUserID()

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clearMessages</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>getName</code></td>
<td><code>const Common::JString &amp;</code></td>
</tr>
<tr>
<td><code>getMessageCount</code></td>
<td><code>const unsigned int</code></td>
</tr>
<tr>
<td><code>getSenders</code></td>
<td><code>const Common::JVector&lt; Common::JString &gt; &amp;</code></td>
</tr>
<tr>
<td><code>getMessages</code></td>
<td><code>const Common::JVector&lt; Common::Object &gt; &amp;</code></td>
</tr>
<tr>
<td><code>getIsPrivate</code></td>
<td><code>const bool</code></td>
</tr>
<tr>
<td><code>toString</code></td>
<td><code>virtual Common::JString &amp;</code></td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from Base

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~Base</code></td>
<td><code>virtual</code></td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from ToString

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~ToString</code></td>
<td><code>virtual</code></td>
</tr>
<tr>
<td><code>typeToString</code></td>
<td><code>virtual JString</code></td>
</tr>
<tr>
<td><code>toString</code></td>
<td><code>JString</code></td>
</tr>
</tbody>
</table>

- `toString(bool`
withTypes=false)
const
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from Base</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static void</strong> <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td><strong>static int</strong> <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td><strong>static bool</strong> <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td><strong>static const <code>LogFormatOptions</code> &amp;</strong> <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td><strong>static void</strong> <code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
Detailed Description

Represents channel or private chat (channel with 2 users)
Member Function Documentation
### toString()

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

#### 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**.
Client Class Reference

Inheritance diagram for Client:

Collaboration diagram for Client:
<table>
<thead>
<tr>
<th>Public Member Functions</th>
<th>Client (Listener &amp;listener, const &amp;applicationID, const Common::JString connectionProtocol=Photon::ConnectionProtocol::DEFAULT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Client (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool connect (const AuthenticationValues &amp;authenticationValues=AuthenticationValues, Common::JString &amp;nameServerAddress=M_NAME_SERVER)</td>
<td></td>
</tr>
<tr>
<td>virtual void disconnect (void)</td>
<td></td>
</tr>
<tr>
<td>virtual void service (bool dispatchIncomingCommands)</td>
<td></td>
</tr>
<tr>
<td>virtual void serviceBasic (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool sendOutgoingCommands ()</td>
<td></td>
</tr>
<tr>
<td>virtual bool sendAcksOnly (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool dispatchIncomingCommands ()</td>
<td></td>
</tr>
<tr>
<td>virtual void fetchServerTimestamp (void)</td>
<td></td>
</tr>
<tr>
<td>virtual void resetTrafficStats (void)</td>
<td></td>
</tr>
<tr>
<td>virtual void resetTrafficStatsMaximumCounters</td>
<td></td>
</tr>
<tr>
<td>virtual Common::JString vitalStatsToString (bool all)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opSubscribe (const Common::JString &amp;channels, int messagesFromHistory=0)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Template Type</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>opUnsubscribe</code></td>
<td></td>
</tr>
<tr>
<td><code>opPublishMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opPublishMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opPublishMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opSendPrivateMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opSendPrivateMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opSendPrivateMessage</code></td>
<td>template&lt;typename Ftype &gt;</td>
</tr>
<tr>
<td><code>opSetOnlineStatus</code></td>
<td></td>
</tr>
<tr>
<td><code>opSetOnlineStatus</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>template&lt;typename Ftype &gt; bool opSetOnlineStatus (int status, const Ftype pMessageArray, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</code></td>
<td>Set online status of the client.</td>
</tr>
<tr>
<td><code>template&lt;typename Ftype &gt; bool opSetOnlineStatus (int status, const Ftype pMessageArray, const short *pArrSizes)</code></td>
<td>Set online status of the client using an array of sizes.</td>
</tr>
<tr>
<td><code>virtual bool opAddFriends (const Common::JVector Common::JString &amp;userIDs)</code></td>
<td>Add friends to the client.</td>
</tr>
<tr>
<td><code>virtual bool opRemoveFriends (const Common::JVector Common::JString &amp;userIDs)</code></td>
<td>Remove friends from the client.</td>
</tr>
<tr>
<td><code>int getServerTimeOffset (void)</code></td>
<td>Get the offset of the server time.</td>
</tr>
<tr>
<td><code>int getServerTime (void) const</code></td>
<td>Get the current server time.</td>
</tr>
<tr>
<td><code>int getBytesOut (void) const</code></td>
<td>Get the number of bytes sent out.</td>
</tr>
<tr>
<td><code>int getBytesIn (void) const</code></td>
<td>Get the number of bytes received.</td>
</tr>
<tr>
<td><code>int getByteCountCurrentDispatch</code></td>
<td>Get the byte count of the current dispatch.</td>
</tr>
<tr>
<td><code>int getByteCountLastOperation</code></td>
<td>Get the byte count of the last operation.</td>
</tr>
<tr>
<td><code>int getSentCountAllowance (void)</code></td>
<td>Get the sent count allowance.</td>
</tr>
<tr>
<td><code>void setSentCountAllowance (int sentCountAllowance)</code></td>
<td>Set the sent count allowance.</td>
</tr>
<tr>
<td><code>int getTimePingInterval (void)</code></td>
<td>Get the time ping interval.</td>
</tr>
<tr>
<td><code>void setTimePingInterval (int timePingInterval)</code></td>
<td>Set the time ping interval.</td>
</tr>
<tr>
<td><code>int getRoundTripTime (void)</code></td>
<td>Get the round trip time.</td>
</tr>
<tr>
<td><code>int getRoundTripTimeVariance</code></td>
<td>Get the round trip time variance.</td>
</tr>
</tbody>
</table>
int getTimestampOfLastSocketReceive
int getDebugOutputLevel (void)
bool setDebugOutputLevel (int debugLevel)
const Common::LogFormatOptions & getLogFormatOptions (void)
void setLogFormatOptions (const Common::LogFormatOptions &)
int getIncomingReliableCommandsCount
short getPeerID (void) const
int getDisconnectTimeout (void)
void setDisconnectTimeout (int disconnectTimeout)
int getQueuedIncomingCommands
int getQueuedOutgoingCommands
bool getIsPayloadEncryptionAvailable
int getResentReliableCommands
int getLimitOfUnreliableCommands
void setLimitOfUnreliableCommands
bool getCRCEnabled (void) const
void setCRCEnabled (bool crcEnabled)
int getPacketLossByCRC (void)
bool   getTrafficStatsEnabled (void)
void   setTrafficStatsEnabled (bool)
int   getTrafficStatsElapsedMs (void)
const Photon::TrafficStats & getTrafficStatsIncoming (void)
const Photon::TrafficStats & getTrafficStatsOutgoing (void)
const Photon::TrafficStatsGameLevel & getTrafficStatsGameLevel (void)
nByte   getQuickResendAttempts (void)
void   setQuickResendAttempts (nByte)
nByte   getChannelCountUserChannels (void)
const Common::JString & getUserID (void) const
int   getState (void) const
int   getDisconnectedCause (void)
const Common::JString & getRegion (void) const
void   setRegion (const Common::JString &)
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 &
const
<table>
<thead>
<tr>
<th>Static Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short getPeerCount (void)</td>
</tr>
</tbody>
</table>
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::ConnectionProtocol::DEFAULT )

Constructor.

Parameters

- **listener**  pointer to the application's implementation of the callback interface.
- **applicationID**  Photon Chat application id
- **appVersion**  Photon Chat application version
- **connectionProtocol**  connection protocol
§ ~Client()

~Client ( void ) virtual

Destructor.
Member Function Documentation
§ connect()

```cpp
bool connect ( const AuthenticationValues & authenticationValues = AuthenticationValues()
               const Common::JString & nameServerAddress = M_NAMESERVER )
```

Initiates a connection to the Photon name server. After a successful connection, 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 Custom Authentication with Photon.
- `nameServerAddress` used to specify a name server address different from the Photon Cloud name server.
§ disconnect()

```cpp
void disconnect ( void )
```

Disconnects from servers.
§ service()

```cpp
void service ( bool dispatchIncomingCommands = false ) 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 within a loop until all incoming commands have been dispatched.)
3. `sendOutgoingCommands()` (called within 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 = \texttt{dispatchIncomingCommands()}
won't be called, default is true
serviceBasic()

```cpp
void serviceBasic ( void )
```

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()

```c
bool sendOutgoingCommands ( void )
```

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/or 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()

```cpp
bool sendAcksOnly ( void )
```

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 explictly, when you use `sendAcksOnly()` and `dispatchIncomingCommands()` instead.
§ dispatchIncomingCommands()

bool dispatchIncomingCommands ( void )

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()

```c
void fetchServerTimestamp ( void )
```

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 roundtrip time is low and the timestamp + roundtrip time / 2 is close to the original value.
§ resetTrafficStats()

```c
void resetTrafficStats ( void )
```

Creates new instances of TrafficStats and starts a new timer for those.
§ resetTrafficStatsMaximumCounters()

void resetTrafficStatsMaximumCounters ( void )

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()

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

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

**Parameters**
- `channels`: list of channels to subscribe to.
- `messagesFromHistory`: 0: no history, 1 and higher: number of messages in history.

**Returns**
- true if request sent
§ opUnsubscribe()

bool opUnsubscribe ( const Common::JVector< Common::JString > & channels )

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
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
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
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
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)
§ opSendPrivateMessage() [2/2]

```cpp
template< typename Ftype >
bool opSendPrivateMessage ( 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]

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

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
template<typename Ftype>
bool opSetOnlineStatus ( int const Ftype,
  typename Common::Helpers::ArrayLengthType< Ftype >::type )

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
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()

```cpp
bool opRemoveFriends ( const Common::JVector< Common::JString > & userIDs )
```

Removes users from 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
§ `getServerTimeOffset()`

```c
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()

```c
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()

```c
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()

```c
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**

```c
setSentCountAllowance() getDisconnectTimeout()
setDisconnectTimeout()
```
§ setSentCountAllowance()

```c
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()

```cpp
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()  

```c
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()

```c
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()

```c
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()

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

Returns
the LogFormatOptions that are used by this instance.

See also
setFormatOptions()
void setLogFormatOptions ( const Common::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()`
§ getIncomingReliableCommandsCount()

```c
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()

```cpp
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()

```java
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()

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

**Returns**
the number of queued outgoing commands in all channels or -1 if not connected
§ getIsPayloadEncryptionAvailable()

```cpp
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()

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

**Returns**

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

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

**Returns**
the limit for the queue of received unreliable commands.

**See also**
`setLimitOfUnreliableCommands()`
§ setLimitOfUnreliableCommands()

```java
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()

```c
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()

```java
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
RoundTripTime+4*RoundTripTimeVariance).
§ setQuickResendAttempts()

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

**Returns**

the number of resend attempts for a reliable command that are done in quick succession (after RoundTripTime+4*RoundTripTimeVariance).
§ `getChannelCountUserChannels()`

```cpp
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 )

Returns
the count of peers, which have been initialized since the start of the application. Interesting mainly for debugging purposes.
<table>
<thead>
<tr>
<th><code>getUserID()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const JString &amp; getUserID ( void ) const</code></td>
</tr>
</tbody>
</table>

Returns the unique user id.

**Returns**
- the user id
§ getState()

int getState ( void ) const

Returns client state
§ getDisconnectedCause()

```c
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()

```cpp
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()

```cpp
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
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
Listener Class Reference

Inheritance diagram for Listener:

Collaboration diagram for Listener:
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>debugReturn</code></td>
<td>virtual void int debugLevel, const <code>Common::JString</code> &amp;string = 0</td>
</tr>
<tr>
<td><code>onStateChange</code></td>
<td>virtual void int state = 0</td>
</tr>
<tr>
<td><code>connectionErrorReturn</code></td>
<td>virtual void int errorCode = 0</td>
</tr>
<tr>
<td><code>clientErrorReturn</code></td>
<td>virtual void int errorCode = 0</td>
</tr>
<tr>
<td><code>warningReturn</code></td>
<td>virtual void int warningCode = 0</td>
</tr>
<tr>
<td><code>serverErrorReturn</code></td>
<td>virtual void int errorCode = 0</td>
</tr>
<tr>
<td><code>connectReturn</code></td>
<td>virtual void int errorCode, const <code>Common::JString</code> &amp;errorString = 0</td>
</tr>
<tr>
<td><code>disconnectReturn</code></td>
<td>virtual void void = 0</td>
</tr>
<tr>
<td><code>subscribeReturn</code></td>
<td>virtual void const <code>Common::JVector&lt;Common::JString&gt;</code> &amp;channels, const <code>Common::JVector&lt;bool&gt;</code> &amp;results = 0</td>
</tr>
<tr>
<td><code>unsubscribeReturn</code></td>
<td>virtual void const <code>Common::JVector&lt;Common::JString&gt;</code> &amp;channels = 0</td>
</tr>
<tr>
<td><code>onStatusUpdate</code></td>
<td>virtual void const <code>Common::JString</code> &amp;user, int status, bool gotMessage, const <code>Common::Object</code> &amp;message = 0</td>
</tr>
<tr>
<td><code>onGetMessages</code></td>
<td>virtual void const <code>Common::JString</code> &amp;channelName, const <code>Common::JVector&lt;Common::JString&gt;</code> &amp;senders, const <code>Common::JVector&lt;Common::Object&gt;</code> &amp;messages = 0</td>
</tr>
<tr>
<td><code>onPrivateMessage</code></td>
<td>virtual void const <code>Common::JString</code> &amp;sender,</td>
</tr>
</tbody>
</table>
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()

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

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 )

Notifies app that client state changed.

Parameters
state new client state

See also
ClientState::ClientState
§ connectReturn()

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

**Client** is connected now.
§ disconnectReturn()

virtual void disconnectReturn ( void )

Disconnection happened.
§ subscribeReturn()

```cpp
virtual void subscribeReturn ( const Common::JVector< Common::JString > & channels,
                               const Common::JVector< bool > & results )
```

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

**Parameters**
- `channels` channel names
- `results` per channel result: true if subscribed
virtual void unsubscribeReturn ( const Common::JVector< Common::JString > & channels )

Result of unsubscribe operation. Returns per channel name if the channel is now subscribed.

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 )

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
                        , const Common::JVector< Common::JString > & senders
                        , const Common::JVector< Common::Object > & messages )

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

Parameters
    channelName    channel from where messages came
    senders        list of users who sent messages
    messages       list of messages it self
§ onPrivateMessage()

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

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)
Peer Class Reference

Inheritance diagram for Peer:

Collaboration diagram for Peer:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peer</strong> (Photon::PhotonListener) connectionProtocol=Photon::ConnectionProtocol::DEFAULT)</td>
<td>virtual bool opAuthenticateOnNameServer(&amp;appID, const Common::JString &amp;region, const &amp;authenticationValues=AuthenticationValues)</td>
</tr>
<tr>
<td></td>
<td>virtual bool opAuthenticateOnFrontEnd(&amp;secret)</td>
</tr>
<tr>
<td></td>
<td>virtual bool opSubscribe(const Common::JVector(Common::JString) &amp;channels, int messagesFromHistory)</td>
</tr>
<tr>
<td></td>
<td>virtual bool opUnsubscribe(const Common::JVector(Common::JString) &amp;channels)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool opPublishMessage(const Common::JString &amp;channelName, const Ftype &amp;message)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool opPublishMessage(const Common::JString &amp;channelName, const Ftype &amp;messageArray, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool opPublishMessage(const Common::JString &amp;channelName, const Ftype &amp;messageArray, const short *pArrSizes)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool opSendPrivateMessage(const &amp;userName, const Ftype &amp;message, bool encrypt=false)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool <strong>opSendPrivateMessage</strong> (const &amp;userName, const Ftype pMessage, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize, bool encrypt=false)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool <strong>opSendPrivateMessage</strong> (const &amp;userName, const Ftype pMessageArray, const short *pArrSizes, bool encrypt=false)</td>
</tr>
<tr>
<td>virtual bool <strong>opSetOnlineStatus</strong> (int status)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool <strong>opSetOnlineStatus</strong> (int status, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool <strong>opSetOnlineStatus</strong> (int status, const short *pArrSizes)</td>
</tr>
<tr>
<td>virtual bool <strong>opAddFriends</strong> (const Common::JVector<a href="">Common::JString</a> &amp;userIDs)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <strong>opRemoveFriends</strong> (const Common::JVector<a href="">Common::JString</a> &amp;userIDs)</td>
<td></td>
</tr>
</tbody>
</table>

- **Public Member Functions inherited from PhotonPeer**

<p>| PhotonPeer (PhotonListener connectionProtocol=ConnectionProtocol::DEFAULT) |
| virtual ~PhotonPeer (void) |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual bool connect (const Common::JString &amp;appID=Common::JString)</td>
<td>Connect with specified application ID.</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt; bool connect (const Common::JString &amp;appID,</td>
<td>Connect template version with specified application ID.</td>
</tr>
<tr>
<td>const Ftype &amp;customData)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt; bool connect (const Common::JString &amp;appID,</td>
<td>Connect template version with specified application ID and custom data.</td>
</tr>
<tr>
<td>const Ftype customDataArray, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</td>
<td></td>
</tr>
<tr>
<td>virtual void disconnect (void)</td>
<td>Disconnect from the connection.</td>
</tr>
<tr>
<td>virtual void service (bool dispatchIncomingCommands)</td>
<td>Service method, dispatch incoming commands if specified.</td>
</tr>
<tr>
<td>virtual void serviceBasic (void)</td>
<td>Service basic method.</td>
</tr>
<tr>
<td>virtual bool opCustom (const OperationRequest &amp;sendReliable, nByte channelID=0, bool encrypt=false)</td>
<td>Custom operation method with specified channel ID and encryption.</td>
</tr>
<tr>
<td>virtual bool sendOutgoingCommands (void)</td>
<td>Send outgoing commands.</td>
</tr>
<tr>
<td>virtual bool sendAcksOnly (void)</td>
<td>Send only acknowledgments.</td>
</tr>
<tr>
<td>virtual bool dispatchIncomingCommands (void)</td>
<td>Dispatch incoming commands.</td>
</tr>
<tr>
<td>virtual bool establishEncryption (void)</td>
<td>Establish encryption connection.</td>
</tr>
<tr>
<td>virtual void fetchServerTimestamp (void)</td>
<td>Fetch server timestamp.</td>
</tr>
<tr>
<td>virtual void resetTrafficStats (void)</td>
<td>Reset traffic statistics.</td>
</tr>
</tbody>
</table>
virtual void resetTrafficStatsMaximumCounters

virtual Common::JString vitalStatsToString (bool all) const

virtual void pingServer (const Common::JString &secret)

virtual void initUserDataEncryption (const Common::JVector &encryptSecret, const Common::JVector &HMACSecret)

virtual void initUDPEncryption (const Common::JVector &secret)

PhotonListener * getListener (void)

int getServerTimeOffset (void) const

int getServerTime (void) const

int getBytesOut (void) const

int getBytesIn (void) const

int getByteCountCurrentDispatch

int getByteCountLastOperation

int getPeerState (void) const

int getSentCountAllowance (void)

void setSentCountAllowance (int sentCountAllowance)

int getTimePingInterval (void) const
void setPingInterval (int timePingInterval)

int getRoundTripTime (void) const

int getRoundTripTimeVariance (void)

int getTimestampOfLastSocketReceive

int getDebugOutputLevel (void) const

bool setDebugOutputLevel (int debugLevel)

const Common::LogFormatOptions & getLogFormatOptions (void) const

void setLogFormatOptions (const Common::LogFormatOptions &)

int getIncomingReliableCommandsCount

short getPeerID (void) const

int getDisconnectTimeout (void) const

void setDisconnectTimeout (int disconnectTimeout)

int getQueuedIncomingCommands

int getQueuedOutgoingCommands

Common::JString getServerAddress (void) const

bool getIsPayloadEncryptionAvailable

bool getIsEncryptionAvailable (void) const
int getResentReliableCommands

int getLimitOfUnreliableCommands

void setLimitOfUnreliableCommands

bool getCRCEnabled (void) const

void setCRCEnabled (bool crcEnabled)

int getPacketLossByCRC (void) const

bool getTrafficStatsEnabled (void)

void setTrafficStatsEnabled (bool trafficStasEnabled)

int getTrafficStatsElapsedMs (void) const

const TrafficStats & getTrafficStatsIncoming (void)

const TrafficStats & getTrafficStatsOutgoing (void)

const TrafficStatsGameLevel & getTrafficStatsGameLevel (void)

nByte getQuickResendAttempts (void)

void setQuickResendAttempts (nByte quickResendAttempts)

nByte getConnectionProtocol (void)

void setConnectionProtocol (nByte connectionProtocol)

nByte getChannelCountUserChannels
Additional Inherited Members

- Static Public Member Functions inherited from `PhotonPeer`
  - `static short getPeerCount (void)`
  - `static unsigned int getMaxAppIDLength (void)`
## AllocatorInterface

**Class Reference**

Abstract

### Inheritance diagram for AllocatorInterface:

```
  AllocatorInterface
    `-- DirectAllocator
    `-- MemoryPoolManager
```

[legend]
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~AllocatorInterface (void)</td>
<td>Constructor/Destructor method for the Allocator interface.</td>
</tr>
<tr>
<td>virtual void setMaxAllocSize (size_t maxAllocSize)=0</td>
<td>Sets the maximum allocation size.</td>
</tr>
<tr>
<td>virtual void * alloc (size_t size)=0</td>
<td>Allocates memory of the specified size.</td>
</tr>
<tr>
<td>virtual void dealloc (void *p)=0</td>
<td>Deallocates memory pointed to by p.</td>
</tr>
<tr>
<td>virtual void * resize (void *p, size_t size)=0</td>
<td>Resizes memory pointed to by p.</td>
</tr>
</tbody>
</table>
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 )

Destructor.
§ setMaxAllocSize()

virtual void setMaxAllocSize ( size_t maxAllocSize )

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 )

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 )

This function gets called by EG_FREE and an implementation is required to behave as explained in the documentation of that macro.
virtual void* resize ( void * p, size_t size )

This function gets called by \texttt{EG_REALLOC} and an implementation is required to behave as explained in the documentation of that macro.
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 sets 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() and in case those globals are not POD-types, then they might allocate memory upon creation and in case of classes that are provided by one of the Photon libs, such allocations will happen through Photon's memory management system. Hence such 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 replace the default implementation of this function by your own implementation. This works in the same way like replacing the platforms default implementations of the global new and delete operators with your own implementations: Photon provides a default implementation of this function that gets used when you don't 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 implemen

Usage example:

```cpp
class Allocator : public ExitGames::Common::MemoryManagement::AllocatorInterface
{
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)
{  
  // attention:  
  // The returned pointer must already point to a valid  
  // instance at the point at which it is returned, which  
  // due to the static initialization order fiasco is  
  // necessarily the case for a global variable.  
  // A dynamically allocated variable however will leak  
  // there is no way to deallocate it, because it needs  
  // to stay valid until after the last global or file  
  // level static object that uses it has been destructed  
  // and to the static initialization order fiasco there  
  // is no way to ensure that that deallocation happens  
  // after those destructions.  
  // The solution is to return the address of a function-  
  // level static variable.  
  static Allocator allocator;  
  return &allocator;  
}  
}  
}  

As you can see, that example implementation of a primitive custom allocator makes operator new private. The reason for this is that once it had been set through setAllocator() an allocator MUST stay valid until you can guarantee that all memory that was given out by it, has been returned to it. Deleting an unused allocator prematurely, even after a different allocator has been set as the current allocator, is undefined behavior and will most likely lead to an access violation crash. A static local variable of AllocatorInterface::get() is guaranteed by the C++ standard to be constructed before that function returns and hence also guaranteed by the standard to get destructed after everything that uses that allocator.

If however you want to manage the lifetime of an allocator instance dynamically through new and delete, then you need to keep track if some of its memory still be in use, before you can safely delete such an allocator. A simple approach to do this is reference counting:

```cpp
class ReferenceCountedAllocator : public
```
ExitGames::Common::MemoryManagement::AllocatorInterface
{
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(*pAllocator);
    ExitGames::Common::JString string = L"samplestring";
    ExitGames::Common::MemoryManagement::setAllocatorToDefault();
    pAllocator->release();
    // some more code
} // only at this point, when the local JString variable '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 deleted
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 return it in AllocatorInterface::get() (at least without leaking it). So there are usage scenarios for the approaches of both example custom allocators: use the approach of class Allocator for a custom allocator that should be returned in 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 would not be the case for ReferenceCountedAllocator, if it would not protect mRefCount 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 it is completely up to you where the memory that your provide to Photon is coming 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:

```cpp
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 on a static byte-array and does not reuse any memory that is returned to it (which is perfectly fine for memory that gets allocated in the constructor and deallocated in 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 avoid hard to track down crashes in unrelated code.

Be aware that this primitive variant that does not reuse any memory only sense when you set a different allocator through `setAllocator()` as early possible because the longer you wait the bigger the static array will need to serve all requests without running out of memory.
ANSIString Class Reference

Inheritance diagram for ANSIString:

Collaboration diagram for ANSIString:
<table>
<thead>
<tr>
<th>Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANSIString</strong> (void)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> (const <strong>ANSIString</strong> &amp;str)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> (const <strong>JString</strong> &amp;str)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> (const char *str)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> (const EG_CHAR *str)</td>
</tr>
<tr>
<td><strong>~ANSIString</strong> (void)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> &amp; operator= (const <strong>ANSIString</strong> &amp;Rhs)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> &amp; operator= (const <strong>JString</strong> &amp;Rhs)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> &amp; operator= (const char *Rhs)</td>
</tr>
<tr>
<td><strong>ANSIString</strong> &amp; operator= (const EG_CHAR *Rhs)</td>
</tr>
<tr>
<td><strong>operator const char</strong> * (void) const</td>
</tr>
<tr>
<td><strong>operator JString</strong> (void) const</td>
</tr>
<tr>
<td><strong>JString</strong> <strong>JStringRepresentation</strong> (void) const</td>
</tr>
<tr>
<td><strong>unsigned int</strong> size (void) const</td>
</tr>
</tbody>
</table>

Public Member Functions inherited from **BaseCharString**

**BaseCharString** ()

virtual **~BaseCharString** (void)

const char * **cstr** (void) const
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int length (void) const</td>
<td></td>
</tr>
<tr>
<td>JString &amp; toString (JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
<tr>
<td>Public Member Functions inherited from Base</td>
<td>virtual ~Base (void)</td>
</tr>
<tr>
<td>Public Member Functions inherited from ToString</td>
<td>virtual ~ToString (void)</td>
</tr>
<tr>
<td>virtual JString typeToString (void) const</td>
<td></td>
</tr>
<tr>
<td>JString toString (bool withTypes=false) const</td>
<td></td>
</tr>
</tbody>
</table>
## Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <code>Base</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td>static int <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td>static bool <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td>static const <code>LogFormatOptions</code> &amp; <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td>static void <code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
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 (void)

Constructor: Creates an empty ANSIString.
§ ANSIString() [2/5]

**ANSIString ( const ANSIString & str )**

Copy-Constructor: Creates a new **ANSIString** from a deep copy of the argument string.

**Parameters**

- **str** The **ANSIString** string to copy.
ANSIString ( const JString & wstr )

Copy-Constructor: Creates a new ANSIString 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=()

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=().

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.
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 *()

This overwrites old data in the left operand.

Implements BaseCharString.
§ operator JString()

operator JString ( void ) const

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.
### Photon C++ Client API 4.1.12.2

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Common</th>
<th>Base</th>
<th>Public Member Functions</th>
<th>Static Public Member Functions</th>
<th>List of all members</th>
</tr>
</thead>
</table>

**Base Class Reference**

Inheritance diagram for Base:
Collaboration diagram for Base:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual <code>~Base</code> (void)</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from `ToString`

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual <code>~ToString</code> (void)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual <code>JString typeToString</code> (void) const</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual <code>JString &amp; toString</code> (<code>JString</code> &amp;retStr, bool withTypes=false) const =0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JString toString</code> (bool withTypes=false) const</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------------</td>
</tr>
<tr>
<td>static void <code>setListener</code></td>
</tr>
<tr>
<td>static int <code>getDebugOutputLevel</code></td>
</tr>
<tr>
<td>static bool <code>setDebugOutputLevel</code></td>
</tr>
<tr>
<td>static const <code>LogFormatOptions</code> &amp; <code>getLogFormatOptions</code></td>
</tr>
<tr>
<td>static void <code>setLogFormatOptions</code></td>
</tr>
</tbody>
</table>
Detailed Description

This is the base-class of all Utility-classes except of \texttt{JString}.

This class provides a common callback interface for transmitting debug messages from all utility classes to your application. Please refer to \texttt{setListener()} for more information.

See also
\begin{itemize}
\item \texttt{BaseListener}, \texttt{setListener()}
\end{itemize}
§ ~Base()

~Base ( void ) virtual

Destructor.
Member Function Documentation
§ setListener()

```cpp
void setListener ( const BaseListener * baseListener )
```

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 events. Has to be a pointer to a class derived from BaseListener.

**See also**

BaseListener
§ getDebugOutputLevel()

```c
int getDebugOutputLevel ( void )
```

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

**Returns**

one of the values in `DebugLevel`

**See also**

`setDebugOutputLevel()`
```cpp
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()

```cpp
const LogFormatOptions & getLogFormatOptions() 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()
BaseCharString Class Reference

Inheritance diagram for BaseCharString:

Collaboration diagram for BaseCharString:
### Public Member Functions

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseCharString</td>
<td>()</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>virtual</td>
<td>~BaseCharString(void)</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>virtual</td>
<td>operator const char * (void) const =0</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>virtual</td>
<td>operator JString(void) const =0</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>const char *</td>
<td>cstr(void) const</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>virtual</td>
<td>JString JStringRepresentation(void) const =0</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>unsigned int</td>
<td>length(void) const</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>virtual unsigned int</td>
<td>size(void) const =0</td>
<td>BaseCharString()</td>
</tr>
<tr>
<td>JString &amp;</td>
<td>toString(JString &amp;retStr, bool withTypes=false) const</td>
<td>BaseCharString()</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from Base

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td>~Base(void)</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from ToString

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td>~ToString(void)</td>
</tr>
<tr>
<td>virtual</td>
<td>typeToString(void) const</td>
</tr>
<tr>
<td>JString</td>
<td>toString(bool withTypes=false) const</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>setListener</strong> (const <strong>BaseListener</strong> *baseListener)</td>
</tr>
</tbody>
</table>

| 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()

Destructor.
Member Function Documentation
§ operator const char *()

operator const char * ( void ) const

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  

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

Returns

a JString representation of the string.

Implemented in ANSIString, and UTF8String.
§ length()

```c
unsigned int length ( void ) const
```

**Returns**
the length of the string in characters
§ size()

unsigned int size ( void ) const

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**.
Inheritance diagram for BaseListener:
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()
debugReturn()

```cpp
debugReturn ( int debugLevel, 
        const JString & string )
```

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>constructClass</strong> (const <strong>CustomTypeFactory</strong> &amp;factory)</td>
<td></td>
</tr>
<tr>
<td>static void <strong>deconstructClass</strong> (void)</td>
<td></td>
</tr>
<tr>
<td>static void <strong>setListener</strong> (const <strong>BaseListener</strong> *baseListener)</td>
<td></td>
</tr>
<tr>
<td>static int <strong>getDebugOutputLevel</strong> (void)</td>
<td></td>
</tr>
<tr>
<td>static bool <strong>setDebugOutputLevel</strong> (int debugLevel)</td>
<td></td>
</tr>
<tr>
<td>static const <strong>LogFormatOptions &amp;</strong> <strong>getLogFormatOptions</strong> (void)</td>
<td></td>
</tr>
<tr>
<td>static void <strong>setLogFormatOptions</strong> (const <strong>LogFormatOptions</strong> &amp;options)</td>
<td></td>
</tr>
</tbody>
</table>
### Static Public Attributes

| static const nByte  | TypeCode |
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Public Member Functions inherited from <strong>CustomTypeBase</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void cleanup (void)=0</td>
</tr>
<tr>
<td>virtual bool compare (const <strong>CustomTypeBase</strong> &amp;other) const =0</td>
</tr>
<tr>
<td>virtual void duplicate (<strong>CustomTypeBase</strong> *pRetVal) const =0</td>
</tr>
<tr>
<td>virtual void deserialize (const nByte *pData, short length)=0</td>
</tr>
<tr>
<td>virtual short serialize (nByte *pRetVal) const =0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~<strong>Base</strong> (void)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public Member Functions inherited from <strong>ToString</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~<strong>ToString</strong> (void)</td>
</tr>
<tr>
<td>virtual <strong>JString</strong> typeToString (void) const</td>
</tr>
<tr>
<td>virtual <strong>JString</strong> &amp; toString (<strong>JString</strong> &amp;retStr, bool withTypes=false) const =0</td>
</tr>
<tr>
<td><strong>JString</strong> toString (bool withTypes=false) const</td>
</tr>
</tbody>
</table>
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()

```cpp
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()

```c
void deconstructClass ( void )
```

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

Check this public constant to find out the typecode of a custom type at runtime. This should normally not be of any interest.
Inheritance diagram for CustomTypeBase:

```
CustomTypeBase
  └── CustomType<typeCode>

  Base
```

Collaboration diagram for CustomTypeBase:

```
CustomTypeBase
  └── CustomType<typeCode>

  Base
```

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void cleanup</td>
<td></td>
<td>virtual void cleanup (void) = 0</td>
</tr>
<tr>
<td>virtual bool compare</td>
<td>(const CustomTypeBase &amp;other) const = 0</td>
<td>virtual bool compare (const CustomTypeBase &amp;other) const = 0</td>
</tr>
<tr>
<td>virtual void duplicate</td>
<td>(CustomTypeBase *pRetVal) const = 0</td>
<td>virtual void duplicate (CustomTypeBase *pRetVal) const = 0</td>
</tr>
<tr>
<td>virtual void deserialize</td>
<td>(const nByte *pData, short length) = 0</td>
<td>virtual void deserialize (const nByte *pData, short length) = 0</td>
</tr>
<tr>
<td>virtual short serialize</td>
<td>(nByte *pRetVal) const = 0</td>
<td>virtual short serialize (nByte *pRetVal) const = 0</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from Base

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Base</td>
<td></td>
<td>virtual ~Base (void)</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from ToString

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~ToString</td>
<td></td>
<td>virtual ~ToString (void)</td>
</tr>
<tr>
<td>virtual JString typeToString</td>
<td></td>
<td>virtual JString typeToString (void) const</td>
</tr>
<tr>
<td>virtual JString &amp; toString</td>
<td>(JString &amp;retStr, bool withTypes=false) const = 0</td>
<td>virtual JString &amp; toString (JString &amp;retStr, bool withTypes=false) const = 0</td>
</tr>
<tr>
<td>JString toString</td>
<td>(bool withTypes=false) const</td>
<td>JString toString (bool withTypes=false) const</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void setListener(const BaseListener* baseListener)</code></td>
<td>Set listener</td>
</tr>
<tr>
<td><code>static int getDebugOutputLevel(void)</code></td>
<td>Get debug output level</td>
</tr>
<tr>
<td><code>static bool setDebugOutputLevel(int debugLevel)</code></td>
<td>Set debug output level</td>
</tr>
<tr>
<td><code>static const LogFormatOptions&amp; getLogFormatOptions(void)</code></td>
<td>Get log format options</td>
</tr>
<tr>
<td><code>static void setLogFormatOptions(const LogFormatOptions&amp; options)</code></td>
<td>Set log format options</td>
</tr>
</tbody>
</table>
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()

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 therefore the constructor and destructor of the object instance to create/destroy are unknown.
§ compare()

```cpp
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 wrapper class around an integer it could just be implemented like this:

```cpp
bool Foo::compare ( const CustomTypeBase & other ) const
{
    return typeid(*this) == typeid(other) && mInt == ((Foo&其它).mInt;
}
```

**Parameters**

- **other** the object to compare the instance with

**Returns**

true, if both objects are equal, false otherwise
§ duplicate()

duplicate ( CustomTypeBase * pRetVal ) const

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()

```cpp
deserialize ( const nByte * pData,
            short       length
            )
```

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()

```cpp
serialize ( nByte * pRetVal ) const
```

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
Inheritance diagram for CustomTypeFactory< typeCode >:

```
    ToString
     |
     v
    Base
     |
     v
CustomTypeFactory< typeCode >
```

Collaboration diagram for CustomTypeFactory< typeCode >:

```
    ToString
     |
     v
    Base
     |
     v
CustomTypeFactory< typeCode >
```
Public Member Functions

<table>
<thead>
<tr>
<th>Virtual Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~CustomTypeFactory (void)</td>
<td></td>
</tr>
<tr>
<td>virtual CustomTypeFactory&lt; typeCode &gt; * copyFactory (void) const =0</td>
<td></td>
</tr>
<tr>
<td>virtual void destroyFactory (void)=0</td>
<td></td>
</tr>
<tr>
<td>virtual CustomType&lt; typeCode &gt; * create (short amount) const =0</td>
<td></td>
</tr>
<tr>
<td>virtual CustomType&lt; typeCode &gt; * copy (const CustomType&lt; typeCode &gt; *pToCopy, short amount) const =0</td>
<td></td>
</tr>
<tr>
<td>virtual void destroy (const CustomType&lt; typeCode &gt; *pToDestroy) const =0</td>
<td></td>
</tr>
<tr>
<td>virtual unsigned int sizeOf (void) const =0</td>
<td></td>
</tr>
<tr>
<td>virtual JString &amp; toString (JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
</tbody>
</table>

Public Member Functions inherited from Base

<table>
<thead>
<tr>
<th>Virtual Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Base (void)</td>
<td></td>
</tr>
</tbody>
</table>

Public Member Functions inherited from ToString

<table>
<thead>
<tr>
<th>Virtual Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~ToString (void)</td>
<td></td>
</tr>
<tr>
<td>virtual JString typeToString (void) const</td>
<td></td>
</tr>
<tr>
<td>JString</td>
<td>toString</td>
</tr>
</tbody>
</table>
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Function Description</th>
<th>Method</th>
<th>Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set listener</td>
<td><code>setListener</code></td>
<td>static void</td>
<td><code>BaseListener *baseListener</code></td>
</tr>
<tr>
<td>Get debug output level</td>
<td><code>getDebugOutputLevel</code></td>
<td>static int</td>
<td>void</td>
</tr>
<tr>
<td>Set debug output level</td>
<td><code>setDebugOutputLevel</code></td>
<td>static bool</td>
<td><code>int debugLevel</code></td>
</tr>
<tr>
<td>Get log format options</td>
<td><code>getLogFormatOptions</code></td>
<td>static const <code>LogFormatOptions &amp;</code></td>
<td>void</td>
</tr>
<tr>
<td>Set log format options</td>
<td><code>setLogFormatOptions</code></td>
<td>static void</td>
<td><code>const LogFormatOptions &amp;options</code></td>
</tr>
</tbody>
</table>
Detailed Description

\[
\text{template}<\text{nByte typeCode}>
\text{class ExitGames::Common::CustomTypeFactory<}
\text{typeCode >}
\]

This is the factory interface class template for the \textbf{CustomType} interface class template and offers an interface to create, copy and delete instances of \textbf{CustomType} subclasses, without the caller needing to know the names of the subclasses.

For every specialization of the \textbf{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**

\textbf{CustomType, CustomTypeBase}
Constructor & Destructor Documentation
~CustomTypeFactory()

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 )

This function shall deallocate the instance, on which it has been called on.
§ create()

create ( short amount ) const

text

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()

```cpp
copy( const CustomType< typeCode > * pToCopy, short amount )
```

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()

```cpp
destroy ( const CustomType< typeCode > * pToDestroy ) const
```

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  

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
```cpp
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 disproportionally 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**.
DeSerializer Class Reference

Inheritance diagram for DeSerializer:

Collaboration diagram for DeSerializer:
## Public Member Functions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DeSerializer</code></td>
<td>(const nByte *data, int size)</td>
</tr>
<tr>
<td><code>bool pop (Object &amp;object)</code></td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; toString (JString &amp;retStr, bool withTypes=false)</code></td>
<td>const</td>
</tr>
<tr>
<td>Public Member Functions inherited from Base</td>
<td></td>
</tr>
<tr>
<td><code>virtual ~Base (void)</code></td>
<td></td>
</tr>
<tr>
<td>Public Member Functions inherited from ToString</td>
<td></td>
</tr>
<tr>
<td><code>virtual ~ToString (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual JString typeToString (void)</code></td>
<td>const</td>
</tr>
<tr>
<td><code>JString toString (bool withTypes=false)</code></td>
<td>const</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <code>Base</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void  <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td>static int  <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td>static bool  <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td>static const <code>LogFormatOptions</code> &amp;  <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td>static void  <code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
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()

```cpp
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()

```cpp
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

**Dictionary**< **EKeyType**, **EValueType** > **Class Template Reference**

Inheritance diagram for **Dictionary**< **EKeyType**, **EValueType** >:

Collaboration diagram for **Dictionary**< **EKeyType**, **EValueType** >:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary (void)</td>
</tr>
<tr>
<td>~Dictionary (void)</td>
</tr>
<tr>
<td>Dictionary (const Dictionary&lt;const EKeyType, EValueType&gt; &amp;toCopy)</td>
</tr>
<tr>
<td>Dictionary &amp; operator= (const Dictionary&lt;const EKeyType, EValueType&gt; &amp;toCopy)</td>
</tr>
<tr>
<td>bool operator== (const Dictionary&lt;const EKeyType, EValueType&gt; &amp;toCompare) const</td>
</tr>
<tr>
<td>bool operator!= (const Dictionary&lt;const EKeyType, EValueType&gt; &amp;toCompare) const</td>
</tr>
<tr>
<td>const EValueType &amp; operator[] (unsigned int index) const</td>
</tr>
<tr>
<td>EValueType &amp; operator[] (unsigned int index)</td>
</tr>
<tr>
<td>const nByte * getKeyTypes (void) const</td>
</tr>
<tr>
<td>const nByte * getValueTypes (void) const</td>
</tr>
<tr>
<td>const unsigned int * getValueDimensions (void) const</td>
</tr>
<tr>
<td>void put (const Dictionary&lt;const EKeyType, EValueType&gt; &amp;src)</td>
</tr>
<tr>
<td>void put (const EKeyType &amp;key, const EValueType &amp;val)</td>
</tr>
<tr>
<td>void put (const EKeyType &amp;key)</td>
</tr>
</tbody>
</table>
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** **typeToString** (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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>setListener</strong> (const <strong>BaseListener</strong> *baseListener)</td>
</tr>
<tr>
<td>static int <strong>getDebugOutputLevel</strong> (void)</td>
</tr>
<tr>
<td>static bool <strong>setDebugOutputLevel</strong> (int debugLevel)</td>
</tr>
<tr>
<td>static const <strong>LogFormatOptions</strong> &amp; <strong>getLogFormatOptions</strong> (void)</td>
</tr>
<tr>
<td>static void <strong>setLogFormatOptions</strong> (const <strong>LogFormatOptions</strong> &amp;options)</td>
</tr>
</tbody>
</table>
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. Therefor 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]

Constructor: Creates an empty instance.
§ ~Dictionary()

~Dictionary (void )

Destructor.
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==()

```cpp
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 `Object` 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]

```cpp
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()

Returns

an array, holding the type code for the key type of the \texttt{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 \texttt{getValueTypes()} at the previous index in the following way: Only when \texttt{getValueTypes()[i] == TypeCode::DICTIONARY}, then \texttt{getKeyTypes()[i+1]} will be valid.

Type information for nested Dictionaries will be stored like in the following example: \texttt{Dictionary<int, Dictionary<short, float**>>} This is a \texttt{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 \texttt{TypeCode::INTEGER} at index 0 and \texttt{TypeCode::SHORT} at index 1.

The codes returned by this function match the ones, that are stored in member variable "typename" of class template \texttt{Helpers::ConfirmAllowedKey}'s specializations. Only the types, for which specializations of that template exist, are valid \texttt{Dictionary} keys.

Reimplemented from \texttt{DictionaryBase}.
§ getValueTypes()

```
const nByte * getValueTypes ( void ) const
```

**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

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.
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]

```cpp
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]

```c
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.
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 `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]

```c
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()

```cpp
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

**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()`

```cpp
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`. 
DictionaryBase Class Reference

Inheritance diagram for DictionaryBase:

Collaboration diagram for DictionaryBase:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~DictionaryBase (void)</td>
<td></td>
</tr>
<tr>
<td>DictionaryBase (const DictionaryBase &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>DictionaryBase &amp; operator= (const DictionaryBase &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>bool operator== (const DictionaryBase &amp;toCompare) const</td>
<td></td>
</tr>
<tr>
<td>bool operator!= (const DictionaryBase &amp;toCompare) const</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename FKeyType&gt; void remove (const FKeyType &amp;key)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename FKeyType&gt; bool contains (const FKeyType &amp;key) const</td>
<td></td>
</tr>
<tr>
<td>void removeAllElements (void)</td>
<td></td>
</tr>
<tr>
<td>JString typeToString (void) const</td>
<td></td>
</tr>
<tr>
<td>JString &amp; toString (JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
<tr>
<td>const Hashtable &amp; getHashtable (void) const</td>
<td></td>
</tr>
<tr>
<td>unsigned int getSize (void) const</td>
<td></td>
</tr>
<tr>
<td>virtual const nByte * getKeyTypes (void) const</td>
<td></td>
</tr>
<tr>
<td>virtual const nByte * getValueTypes (void) const</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Function</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>template&lt;typename FKeyType&gt;</td>
<td>const short * getValuesizes (const FKeyType &amp;key) const</td>
</tr>
<tr>
<td></td>
<td>virtual const unsigned int * getValueDimensions (void) const</td>
</tr>
<tr>
<td>template&lt;typename FKeyType, typename FValueType&gt;</td>
<td>const FValueType * getValue (const FKeyType &amp;key, const FValueType *) const</td>
</tr>
<tr>
<td></td>
<td>const Object * getValue (const FKeyType &amp;key, const Object *) const</td>
</tr>
<tr>
<td></td>
<td>JVector&lt; FKeyType &gt; getKeys (const FKeyType *) const</td>
</tr>
<tr>
<td></td>
<td>JVector&lt; Object &gt; getKeys (const Object *) const</td>
</tr>
<tr>
<td></td>
<td>virtual ~Base (void)</td>
</tr>
<tr>
<td></td>
<td>virtual ~ToString (void)</td>
</tr>
<tr>
<td></td>
<td>JString toString (bool withTypes=false) const</td>
</tr>
</tbody>
</table>

- `getValuesizes`: Returns the size of the value for a given key.
- `getValueDimensions`: Returns the dimensions of the value.
- `getValue`: Returns the value for a given key.
- `getKeys`: Returns a vector of keys.
- `toString`: Converts the object to a string representation.
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <code>Base</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td>static int <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td>static bool <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td>static const <code>LogFormatOptions</code> &amp; <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td>static void <code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
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.
§ 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()

```cpp
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()

```cpp
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()`

```java
void removeAllElements ( void )
```

Clears the **Hashtable**, which means deleting all its content.

**Returns**

Nothing.

**See also**

`remove()`
§ typeToString()

**JString** typeToString ( void ) const  

### 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 >**.
$\text{toString()}$

**JString & toString ( JString & retStr, bool withTypes = false )**

**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()

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()

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()

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 function's 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 >`. 
>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()
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 HasTable contains the specified key, NULL otherwise.

**See also**
put()
§ getKeys() [1/2]

**Returns**

A `JVector` holding all keys contained in the **Hashtable**.
JVector< Object > getKeys ( const Object * ) const

Returns

a JVector holding all keys contained in the Hashtable.
### EGTime Class Reference

Inheritance diagram for EGTime:

```
  ToSring
     |
     V
  Beee
     |
     V
  EGTime
  [legend]
```

Collaboration diagram for EGTime:

```
  ToSring
     |
     V
  Beee
     |
     V
  EGTime
  [legend]
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGTime (int time)</td>
<td></td>
</tr>
<tr>
<td>~EGTime (void)</td>
<td></td>
</tr>
<tr>
<td>EGTime (const EGTime &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>EGTime &amp; operator= (const EGTime &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>EGTime &amp; operator= (const int &amp;time)</td>
<td></td>
</tr>
<tr>
<td>const EGTime &amp; operator+= (const EGTime &amp;time)</td>
<td></td>
</tr>
<tr>
<td>const EGTime &amp; operator-= (const EGTime &amp;time)</td>
<td></td>
</tr>
<tr>
<td>EGTime operator+ (const EGTime &amp;time)</td>
<td></td>
</tr>
<tr>
<td>EGTime operator- (const EGTime &amp;time)</td>
<td></td>
</tr>
<tr>
<td>bool operator&lt; (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool operator&gt; (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool operator&lt;= (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool operator&gt;= (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool operator== (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool operator!= (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>bool overflowed (const EGTime &amp;time) const</td>
<td></td>
</tr>
<tr>
<td>JString &amp; toString (JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Public Member Functions</td>
<td>inherited from Base</td>
</tr>
<tr>
<td>virtual ~Base (void)</td>
<td></td>
</tr>
<tr>
<td>Public Member Functions</td>
<td>inherited from ToString</td>
</tr>
<tr>
<td>virtual ~ToString (void)</td>
<td></td>
</tr>
<tr>
<td>virtual JString typeToString (void) const</td>
<td></td>
</tr>
<tr>
<td>JString toString (bool withTypes=false) const</td>
<td></td>
</tr>
</tbody>
</table>
# Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <code>Base</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static void</strong> <a href="#">setListener</a> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td><strong>static int</strong> <a href="#">getDebugOutputLevel</a> (void)</td>
</tr>
<tr>
<td><strong>static bool</strong> <a href="#">setDebugOutputLevel</a> (int debugLevel)</td>
</tr>
<tr>
<td><strong>static const <code>LogFormatOptions</code> &amp;</strong> <a href="#">getLogFormatOptions</a> (void)</td>
</tr>
<tr>
<td><strong>static void</strong> <a href="#">setLogFormatOptions</a> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
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.
§ 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]

```cpp
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+()

```cpp
EGTime operator+ ( const EGTime & time )
```

operator+.

Adds the right time to the left time and returns the result as a new `EGTime` instance.
§ operator-()

\textbf{EGTime} \texttt{operator- ( const EGTime & time )}

\texttt{operator-=}.

Subtracts the right time from the left time and returns the result as a new \textbf{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\leq() 

```cpp
bool operator\leq( \text{ const } \textit{EGTime} \ & \ \textit{time} ) \text{ const}
```

operator\leq.

**Remarks**

An \textit{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>=( )

```cpp
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==()

```cpp
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.
bool overflowed(const EGTime & time) const

Returns
true, if the values of both instances differ by more than 24 hours,
false otherwise.
§ toString()

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

**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`.

---

© Exit Games ©, all rights reserved.

Photon Documentation | Contact | Terms
Hashtable Class Reference

Inheritance diagram for Hashtable:

```
  class Base
  +--- class Hashtable
  
  class ToString

[legend]
```

Collaboration diagram for Hashtable:

```
  class Base
  +--- class Hashtable
  
  class ToString

[legend]
```
Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Hashtable (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>~Hashtable (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>Hashtable (const Hashtable &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>Hashtable &amp; operator= (const Hashtable &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>bool operator== (const Hashtable &amp;toCompare)</code> const</td>
<td></td>
</tr>
<tr>
<td><code>bool operator!= (const Hashtable &amp;toCompare)</code> const</td>
<td></td>
</tr>
<tr>
<td><code>const Object &amp; operator[](unsigned int index)</code> const</td>
<td></td>
</tr>
<tr>
<td><code>Object &amp; operator[](unsigned int index)</code></td>
<td></td>
</tr>
<tr>
<td><code>void put (const Hashtable &amp;src)</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType, typename FValueType &gt; void put (const FKeyType &amp;key, const FValueType &amp;val)</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType &gt; void put (const FKeyType &amp;key)</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType, typename FValueType &gt; void put (const FKeyType &amp;key, const FValueType pVal, typename Common::Helpers::ArrayLengthType&lt;FValueType&gt;::type size)</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType, typename FValueType &gt; void put (const FKeyType &amp;key, const FValueType &amp;key, const FValueType &amp;key, const FValueType &amp;key)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>FValueType pVal, const short *sizes)</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType &gt; const Object * getValue (const FKeyType &amp;key)</code></td>
<td>gets a value by key</td>
</tr>
<tr>
<td><code>unsigned int getSize (void) const</code></td>
<td>gets the size of the object</td>
</tr>
<tr>
<td><code>const JVector&lt; Object &gt; &amp; getKeys (void)</code></td>
<td>gets the keys of the object</td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType &gt; void remove (const FKeyType &amp;key)</code></td>
<td>removes an element by key</td>
</tr>
<tr>
<td><code>template&lt;typename FKeyType &gt; bool contains (const FKeyType &amp;key)</code></td>
<td>checks if an element exists by key</td>
</tr>
<tr>
<td><code>void removeAllElements (void)</code></td>
<td>removes all elements</td>
</tr>
<tr>
<td><code>JString &amp; toString (JString &amp;retStr, bool withTypes=false)</code></td>
<td>converts the object to a string, with or without types</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from Base</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static void</strong> <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td><strong>static int</strong> <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td><strong>static bool</strong> <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td><strong>static const LogFormatOptions &amp;</strong> <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td><strong>static void</strong> <code>setLogFormatOptions</code> (const LogFormatOptions &amp;options)</td>
</tr>
</tbody>
</table>
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. Therefor 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
Constructor: Creates an empty instance.
§ ~Hashtable()

~Hashtable ( void )

Destructor.
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==()

```cpp
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]

```cpp
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]

```c
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]

```c
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.
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 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]

```c
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()

```cpp
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 HasTable.

Parameters

  key Pointer to the key of the key/value-pair to remove.

Returns

  nothing.

See also

  removeAllElements()
§ contains()

```cpp
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()

```java
void removeAllElements ( void )
```

Clears the **Hashtable**, which means deleting all its content.

**Returns**
nothing.

**See also**
remove()
**toString()**

```cpp
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:

```
 _ToString
 /     \
|       |
JString
```

Collaboration diagram for JString:

```
  _ToString
     |
     |    
JString
```

[legend]
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JString</code> (unsigned int bufferlen=0)</td>
<td></td>
</tr>
<tr>
<td><code>JString</code> (const char *Value)</td>
<td></td>
</tr>
<tr>
<td><code>JString</code> (const EG_CHAR *Value)</td>
<td></td>
</tr>
<tr>
<td><code>JString</code> (const <code>JString</code> &amp;Value)</td>
<td></td>
</tr>
<tr>
<td><code>JString</code> (const UTF8String &amp;Value)</td>
<td></td>
</tr>
<tr>
<td><code>JString</code> (const ANSIString &amp;Value)</td>
<td></td>
</tr>
<tr>
<td><code>~JString</code> (void)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (const <code>JString</code> &amp;Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (const char *Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (const EG_CHAR *Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (const UTF8String &amp;Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (const ANSIString &amp;Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (char Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (signed char Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (unsigned char Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (EG_CHAR Rhs)</td>
<td></td>
</tr>
<tr>
<td><code>JString &amp; operator=</code> (short aNum)</td>
<td></td>
</tr>
</tbody>
</table>
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

EGCHAR operator[](unsigned int Index) const

EGCHAR & operator[](unsigned int Index)

unsigned int capacity(void) const

EGCHAR charAt(unsigned int index) const

int compareTo(const JString &anotherString) const

const JString & concat(const JString &str)

const EGCHAR * 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(EGCHAR ch) const

int indexOf(EGCHAR ch, unsigned int fromIndex) const
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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>std::basic_ostream&lt;_Elem, Traits&gt; &amp; operator&lt;&lt; (::std::basic_ostream&lt;_Elem, Traits&gt; &amp;stream, const JString &amp;string)</code></td>
<td>Stream insertion operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JString operator+ (const JString &amp;Lsh, const Etype &amp;Rsh)</code></td>
<td>String concatenation operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>JString operator+ (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
<td>String concatenation operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator== (const JString &amp;Lsh, const Etype &amp;Rsh)</code></td>
<td>Equality comparison operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator== (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
<td>Equality comparison operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator!= (const JString &amp;Lsh, const Etype &amp;Rsh)</code></td>
<td>Inequality comparison operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator!= (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
<td>Inequality comparison operator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator&lt; (const JString</code></td>
<td>Less than comparison operator</td>
</tr>
<tr>
<td>Template</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&lt; (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&gt; (const JString &amp;Lsh, const Etype &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&gt;= (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&lt;= (const JString &amp;Lsh, const Etype &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&gt;= (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Etype&gt;</code></td>
<td><code>bool operator&gt; (const Etype &amp;Lsh, const JString &amp;Rsh)</code></td>
</tr>
<tr>
<td><code>JString operator+ (const JString &amp;Lsh, const JString &amp;Rsh)</code></td>
<td></td>
</tr>
</tbody>
</table>
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: 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.
Copy-Constructor: Creates a new `JString` from a deep copy of the argument string.

**Parameters**
- `Value` The UTF16 string to copy.
| 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.
Copy-Constructor: Creates a new `JString` from a deep copy of the argument string.

**Parameters**

- **Value**: The `ANSIString` to copy.
§ ~JString()

~JString ( void )

Destructor.
§ 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]

```cpp
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]

\textbf{JString} \& operator= ( long double \textit{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]

\begin{verbatim}
JString & operator+= ( const JString & Rhs )
\end{verbatim}

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==( )

```cpp
bool operator==( const JString & Rhs ) const
```

operator==.

**Returns**
true, if both operands are equal, false otherwise.
\section*{operator!=()}

```cpp
bool operator!= ( const JString & Rhs ) const
```

\begin{itemize}
\item \textbf{operator!=.}
\end{itemize}

\textbf{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>()

```cpp
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>=()

```cpp
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.
<table>
<thead>
<tr>
<th>§ operator<a href=""></a></th>
<th>[1/2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EG_CHAR operator[]( unsigned int index ) const</td>
<td></td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th>EG_CHAR &amp; operator[]( unsigned int index )</th>
</tr>
</thead>
</table>

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()

```c
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()

```cpp
text
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()

```c
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()

```cpp
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

ture, if both strings are equal to each other, false otherwise.
§ indexOf() [1/6]

```c
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]

```c
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()`

```c
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]

```c
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]

```cpp
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]

```cpp
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]

```c
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]

```c
int lastIndexOf ( EG_CHAR ch,
                unsigned int fromIndex
             )
```

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]

```cpp
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]

```cpp
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]

```cpp
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]

```cpp
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]

```cpp
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
bool startsWith ( const JString & prefix, 
    unsigned int offset )

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
§ subroutine() [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]

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()`

```c
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()

```cpp
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`. 
§ 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 )

operator+.

Adds its right operand to its left operand and returns the result as a new JString.
operator+.

Adds its right operand to its left operand and returns the result as a new JString.
§ operator==() [1/2]

```cpp
bool operator==(const JString & Lsh, const Etype & Rsh)
```

operator==.

**Returns**

true, if both operands are equal, false otherwise.
§ operator==( ) [2/2]

bool operator==( const Etype & Lsh,
                const JString & Rsh
         )

operator==.

**Returns**
true, if both operands are equal, false otherwise.
§ operator!=( ) [1/2]

```c++
bool operator!=( const JString & Lsh, const Etype & Rsh )
```

operator!=.

**Returns**

false, if both operands are equal, true otherwise.
§ operator!=( ) [2/2]

bool operator!=( const Etype & Lsh, const JString & Rsh )

operator!=.

**Returns**
false, if both operands are equal, true otherwise.
§ operator<() [1/2]

bool operator< ( const JString & Lsh,
                const Etype & Rsh )

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]

```cpp
bool operator< ( const Etype & Lsh,
                const JString & Rsh )
```

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 )

operator>. The return value indicates the lexicographic relation between the operands.

Returns
true, if left operand is greater than right operand, false otherwise.
bool operator> ( const Etype & Lsh, 
    const JString & Rsh )

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
 )

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\(\leq\)() [2/2]

```cpp
bool operator\(\leq\) ( const Etype & Lsh,  
    const JString & Rsh  
  )
```

operator\(\leq\). 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]

```cpp
bool operator>= ( const JString & Lsh,
                    const Etype & Rsh
               )
```

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.
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+

operator+.

Adds its right operand to its left operand and returns the result as a new JString.
Collaboration diagram for JVector< Etype >:

[legend]
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JVector</strong> (unsigned int initialCapacity=0, unsigned int capacityIncrement=1)</td>
<td></td>
</tr>
<tr>
<td><strong>JVector</strong> (const Etype *carray, unsigned int elementCount, unsigned int initialCapacity=0, unsigned int capacityIncrement=1)</td>
<td></td>
</tr>
<tr>
<td>virtual ~JVector (void)</td>
<td></td>
</tr>
<tr>
<td><strong>JVector</strong> (const JVector&lt;Etype&gt; &amp;rhv)</td>
<td></td>
</tr>
<tr>
<td><strong>JVector</strong> &amp; operator= (const JVector&lt;Etype&gt; &amp;rhv)</td>
<td></td>
</tr>
<tr>
<td>bool operator==(const JVector&lt;Etype&gt; &amp;toCompare) const</td>
<td></td>
</tr>
<tr>
<td>bool operator!=(const JVector&lt;Etype&gt; &amp;toCompare) const</td>
<td></td>
</tr>
<tr>
<td>const Etype &amp; operator[] (unsigned int index) const</td>
<td></td>
</tr>
<tr>
<td>Etype &amp; operator[] (unsigned int index)</td>
<td></td>
</tr>
<tr>
<td>unsigned int getCapacity (void) const</td>
<td></td>
</tr>
<tr>
<td>bool contains (const Etype &amp;elem) const</td>
<td></td>
</tr>
<tr>
<td>const Etype &amp; getFirstElement (void) const</td>
<td></td>
</tr>
<tr>
<td>int getIndexOf (const Etype &amp;elem) const</td>
<td></td>
</tr>
<tr>
<td>bool isEmpty (void) const</td>
<td></td>
</tr>
<tr>
<td>const Etype &amp; getLastElement (void) const</td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getLastIndexOf</td>
<td>Returns the index of the last occurrence of the specified element.</td>
</tr>
<tr>
<td>getSize</td>
<td>Returns the current size of the collection.</td>
</tr>
<tr>
<td>getArray</td>
<td>Returns a pointer to the internal array.</td>
</tr>
<tr>
<td>copyInto</td>
<td>Copies all elements from one array to another.</td>
</tr>
<tr>
<td>addElement</td>
<td>Adds a single element to the collection.</td>
</tr>
<tr>
<td>addElements</td>
<td>Adds all elements from another array.</td>
</tr>
<tr>
<td>trimToSize</td>
<td>Trims the collection to its current size.</td>
</tr>
<tr>
<td>ensureCapacity</td>
<td>Ensures that there is enough capacity in the collection.</td>
</tr>
<tr>
<td>removeAllElements</td>
<td>Removes all elements from the collection.</td>
</tr>
<tr>
<td>removeElement</td>
<td>Removes a single element from the collection.</td>
</tr>
<tr>
<td>getElementTypeAt</td>
<td>Returns the element at the specified index.</td>
</tr>
<tr>
<td>insertElementAt</td>
<td>Inserts a single element at the specified index.</td>
</tr>
<tr>
<td>removeElementAt</td>
<td>Removes an element at the specified index.</td>
</tr>
<tr>
<td>setElementAt</td>
<td>Sets the element at the specified index.</td>
</tr>
<tr>
<td>toString</td>
<td>Converts the collection to a string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public Member Functions inherited from Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Base</td>
</tr>
<tr>
<td>Public Member Functions inherited from <strong>ToString</strong></td>
</tr>
<tr>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>virtual <strong>~ToString</strong> (void)</td>
</tr>
<tr>
<td>virtual <strong>JString</strong> <strong>typeToString</strong> (void) const</td>
</tr>
<tr>
<td><strong>JString</strong> <strong>toString</strong> (bool withTypes=false) const</td>
</tr>
</tbody>
</table>
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static void</strong> <em>setListener</em> (const <em>BaseListener</em> <em>baseListener</em>)</td>
</tr>
<tr>
<td><strong>static int</strong> <em>getDebugOutputLevel</em> (void)</td>
</tr>
<tr>
<td><strong>static bool</strong> <em>setDebugOutputLevel</em> (int <em>debugLevel</em>)</td>
</tr>
<tr>
<td><strong>static const LogFormatOptions &amp;</strong> <em>getLogFormatOptions</em> (void)</td>
</tr>
<tr>
<td><strong>static void</strong> <em>setLogFormatOptions</em> (const LogFormatOptions &amp;<em>options</em>)</td>
</tr>
</tbody>
</table>
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.
Constructor.

Creates a `JVector`, initialized with the passed `carray` the template parameter.

**Parameters**

- **array**
  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 )

Destructor.
JVector() 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=()

\texttt{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[]. Wraps the function `getElementAt()`, so you have the same syntax like for arrays.
<table>
<thead>
<tr>
<th>§ getCapacity()</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int getCapacity ( void ) const</td>
</tr>
</tbody>
</table>

Returns the current capacity of the **JVector**.

**Returns**
the current capacity.
§ contains()

```cpp
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()

```cpp
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()

```c
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()

```cpp
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.
§ getCArry()

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()`

```c
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 )`

Add 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]

```cpp
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.getCArrary() and vector.getSize() as parameters.

**Parameters**

- `vector` the vector from which to copy the elements
§ addElements()  [2/2]

```c
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()

```c
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.
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()

```cpp
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()

```c
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()

```c
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()

```c
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**.

---

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
**KeyObject< Etype >**

Class Template Reference

Inheritance diagram for KeyObject< Etype >:

```
    | ToSring
    |     |
    |     | Base
    |     |     |
    |     | Object
    |     |     |
    | KeyObject< Etype >
```

[legend]

Collaboration diagram for KeyObject< Etype >:

```
    | ToSring
    |     |
    |     | Base
    |     |     |
    |     | Object
    |     |     |
    | KeyObject< Etype >
```

[legend]
## Public Member Functions

<table>
<thead>
<tr>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KeyObject</strong> (const <strong>KeyObject</strong>&lt; Etype &gt; &amp;toCopy)</td>
</tr>
<tr>
<td><strong>KeyObject</strong> (const <strong>Object</strong> &amp;obj)</td>
</tr>
<tr>
<td><strong>KeyObject</strong> (const <strong>Object</strong> *obj)</td>
</tr>
<tr>
<td><strong>KeyObject</strong> (const typename Helpers::ConfirmAllowedKey&lt; Etype &gt;::type &amp;data)</td>
</tr>
<tr>
<td>virtual <strong>~KeyObject</strong> (void)</td>
</tr>
<tr>
<td>virtual <strong>KeyObject</strong>&lt; Etype &gt; &amp; <strong>operator</strong>= (const <strong>KeyObject</strong>&lt; Etype &gt; &amp;toCopy)</td>
</tr>
<tr>
<td>virtual <strong>KeyObject</strong>&lt; Etype &gt; &amp; <strong>operator</strong>= (const <strong>Object</strong> &amp;toCopy)</td>
</tr>
<tr>
<td><strong>Etype</strong> <strong>getDataCopy</strong> (void) const</td>
</tr>
<tr>
<td><strong>Etype</strong> * <strong>getDataAddress</strong> (void) const</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from **Object**

<table>
<thead>
<tr>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong> (void)</td>
</tr>
<tr>
<td>virtual <strong>~Object</strong> (void)</td>
</tr>
<tr>
<td><strong>Object</strong> (const <strong>Object</strong> &amp;toCopy)</td>
</tr>
<tr>
<td>bool <strong>operator</strong>== (const <strong>Object</strong> &amp;toCompare) const</td>
</tr>
<tr>
<td>bool <strong>operator</strong>!= (const <strong>Object</strong> &amp;toCompare) const</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>nByte getType (void)</td>
</tr>
<tr>
<td>nByte getCustomType (void)</td>
</tr>
<tr>
<td>const short * getSizes (void) const</td>
</tr>
<tr>
<td>unsigned int getDimensions (void) const</td>
</tr>
<tr>
<td>JString &amp; toString (JString &amp;retStr, bool withTypes=false) const</td>
</tr>
</tbody>
</table>

- **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(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 >::type &

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.
§ `operator=()` [1/2]

```cpp
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=(): 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 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 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:

[Diagram of inheritance]

Collaboration diagram for LogFormatOptions:

[Diagram of collaboration]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool getAddDateTime (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setAddDateTime (bool addTime)</td>
<td></td>
</tr>
<tr>
<td>bool getAddLevel (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setAddLevel (bool addLevel)</td>
<td></td>
</tr>
<tr>
<td>bool getAddFile (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setAddFile (bool addFile)</td>
<td></td>
</tr>
<tr>
<td>bool getAddFunction (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setAddFunction (bool addFunction)</td>
<td></td>
</tr>
<tr>
<td>unsigned int getMaxNumberOfNamespaces (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setMaxNumberOfNamespaces (unsigned int maxNumberOfNamespaces)</td>
<td></td>
</tr>
<tr>
<td>bool getAddLine (void) const</td>
<td></td>
</tr>
<tr>
<td>LogFormatOptions &amp; setAddLine (bool addLine)</td>
<td></td>
</tr>
<tr>
<td>virtual Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
</tbody>
</table>

Public Member Functions inherited from **ToString**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~ToString (void)</td>
<td></td>
</tr>
<tr>
<td>virtual JString typeToString (void) const</td>
<td></td>
</tr>
</tbody>
</table>
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

gAddDateTime()
§ getAddLevel()

```cpp
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

gAddAddLevel()
§ getAddFile()

```c
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()

```c
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()

Sets the maximum number of namespaces that are included in the name of the 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 inner 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()

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

### 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logger</strong></td>
<td>(int debugLevel=\texttt{DebugLevel::WARNINGS})</td>
</tr>
<tr>
<td><strong>log</strong></td>
<td>(int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool printBrackets, unsigned int line, const EG_CHAR *dbgMsg,...) const</td>
</tr>
<tr>
<td><strong>vlog</strong></td>
<td>(int debugLevel, const EG_CHAR *file, const EG_CHAR *function, bool printBrackets, unsigned int line, const EG_CHAR *dbgMsg, va_list args) const</td>
</tr>
<tr>
<td><strong>getDebugOutputLevel</strong></td>
<td>(void) const</td>
</tr>
<tr>
<td><strong>setDebugOutputLevel</strong></td>
<td>(int debugLevel)</td>
</tr>
<tr>
<td><strong>setListener</strong></td>
<td>(const BaseListener &amp;listener)</td>
</tr>
<tr>
<td><strong>getFormatOptions</strong></td>
<td>(void) const</td>
</tr>
<tr>
<td><strong>setFormatOptions</strong></td>
<td>(const LogFormatOptions &amp;formatOptions)</td>
</tr>
<tr>
<td><strong>toString</strong></td>
<td>(Common::JString &amp;retStr, bool withTypes=false) const</td>
</tr>
<tr>
<td><strong>~ToString</strong></td>
<td>(void)</td>
</tr>
<tr>
<td><strong>typeToString</strong></td>
<td>(void) const</td>
</tr>
<tr>
<td><strong>toString</strong></td>
<td>(bool withTypes=false) const</td>
</tr>
</tbody>
</table>
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 = DebugLevel::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()

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void log</td>
<td>(int debugLevel, const EG_CHAR * file, const EG_CHAR * function, bool printBrackets, unsigned int line, const EG_CHAR * dbgMsg, ...)</td>
</tr>
</tbody>
</table>

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 )

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()

```cpp
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()

```cpp
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()

```cpp
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
```

**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

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td>(void)</td>
</tr>
<tr>
<td>virtual <strong>~Object</strong></td>
<td>(void)</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td>(const <strong>Object</strong> &amp;toCopy)</td>
</tr>
<tr>
<td>virtual <strong>Object</strong> &amp;  <strong>operator=</strong></td>
<td>(const <strong>Object</strong> &amp;toCopy)</td>
</tr>
<tr>
<td><strong>bool</strong></td>
<td><strong>operator==</strong> (const <strong>Object</strong> &amp;toCompare) const</td>
</tr>
<tr>
<td><strong>bool</strong></td>
<td><strong>operator!=</strong> (const <strong>Object</strong> &amp;toCompare) const</td>
</tr>
<tr>
<td><strong>nByte</strong></td>
<td><strong>getType</strong> (void) const</td>
</tr>
<tr>
<td><strong>nByte</strong></td>
<td><strong>getCustomType</strong> (void) const</td>
</tr>
<tr>
<td><strong>const short</strong> *</td>
<td><strong>getSizes</strong> (void) const</td>
</tr>
<tr>
<td><strong>unsigned int</strong></td>
<td><strong>getDimensions</strong> (void) const</td>
</tr>
<tr>
<td><strong>JString</strong> &amp;  <strong>toString</strong></td>
<td>(JString &amp;retStr, bool withTypes=false) const</td>
</tr>
<tr>
<td>Public Member Functions inherited from <strong>Base</strong></td>
<td></td>
</tr>
<tr>
<td>virtual <strong>~Base</strong></td>
<td>(void)</td>
</tr>
<tr>
<td>Public Member Functions inherited from <strong>ToString</strong></td>
<td></td>
</tr>
<tr>
<td>virtual <strong>~ToString</strong></td>
<td>(void)</td>
</tr>
<tr>
<td><strong>virtual JString</strong></td>
<td><strong>typeToString</strong> (void) const</td>
</tr>
<tr>
<td><strong>JString</strong> &amp;  <strong>toString</strong></td>
<td>(bool withTypes=false) const</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void setListener (const BaseListener *baseListener)</code></td>
<td>Static function inherited from Base</td>
</tr>
<tr>
<td><code>static int getDebugOutputLevel (void)</code></td>
<td>Static function</td>
</tr>
<tr>
<td><code>static bool setDebugOutputLevel (int debugLevel)</code></td>
<td>Static function</td>
</tr>
<tr>
<td><code>static const LogFormatOptions &amp; getLogFormatOptions (void)</code></td>
<td>Static function</td>
</tr>
<tr>
<td><code>static void setLogFormatOptions (const LogFormatOptions &amp;options)</code></td>
<td>Static function</td>
</tr>
</tbody>
</table>
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 )

Destructor.
Copy-Constructor: Creates an **Object** containing a deep copy of the argument passed.

**Parameters**
- **toCopy** The object to copy.
Member Function Documentation
§ operator=()

\[ \text{Object} & \ \text{operator=} \ ( \text{const Object} \ & \ \text{toCopy} ) \]

operator= : Makes a deep copy of its right operand into its left operand. This overwrites old data in the left operand.

Reimplemented in \text{ValueObject}< \text{Etype} >, and \text{KeyObject}< \text{Etype} >.
§ operator==()

```cpp
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()`

```c
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()

```c
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.
JNIEXPORT void JNICALL Java_com_example_PrototypeTest_JString_toString(JNIEnv *env, jobject thisObj, jstring retStr, jboolean withTypes)
{
    const char *str = stringToCString(thisObj); // Assuming a helper function to convert Java string to C string
    StringBuffer sb;
    sb.append(str);
    sb.append(withTypes ? " : " : "\n"); // Append type information if withTypes is true
    jni_log(sb.str()); // Log the generated string
    sb.clear();
}

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**.
Serializer Class Reference

Inheritance diagram for Serializer:

Collaboration diagram for Serializer:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const nByte * getData(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>int getSize(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>template&lt;typename T &gt; bool push(const T &amp;data)</code></td>
<td>bool <code>push</code> (const T &amp;data)</td>
</tr>
<tr>
<td><code>template&lt;typename T &gt; bool push(const T pData, typename Helpers::ArrayLengthType&lt;T&gt;::type arraySize)</code></td>
<td>bool <code>push</code> (const T pData, typename Helpers::ArrayLengthType&lt;T&gt;::type arraySize)</td>
</tr>
<tr>
<td><code>template&lt;typename T &gt; bool push(const T pData, const short *arraySizes)</code></td>
<td>bool <code>push</code> (const T pData, const short *arraySizes)</td>
</tr>
</tbody>
</table>

`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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void setListener (const BaseListener *baseListener)</td>
</tr>
<tr>
<td>static int getDebugOutputLevel (void)</td>
</tr>
<tr>
<td>static bool setDebugOutputLevel (int debugLevel)</td>
</tr>
<tr>
<td>static const LogFormatOptions &amp; getLogFormatOptions (void)</td>
</tr>
<tr>
<td>static void setLogFormatOptions (const LogFormatOptions &amp;options)</td>
</tr>
</tbody>
</table>
**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 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 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]

```cpp
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]

```cpp
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
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**.
Inheritance diagram for ToString:
[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~ToString</td>
<td>(void)</td>
</tr>
<tr>
<td>virtual JString</td>
<td>typeToString (void) const</td>
</tr>
<tr>
<td>virtual JString &amp;</td>
<td>toString (JString &amp;retStr, bool withTypes=false) const =0</td>
</tr>
<tr>
<td></td>
<td>JString toString (bool withTypes=false) const</td>
</tr>
</tbody>
</table>


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
§ typeToString()

**JString** typeToString ( void ) const  

**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**.
$\mathsf{toString()}$ [1/2]

```cpp
toString ( JObject & retStr,
    bool withTypes = false
  ) const
```

### 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 JObject representation of the instance and its contents for debugging purposes.

Implemented in JObject, JVector< Etype >, JVector< Common::ExitGames::Common::JObject >, 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.
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

Inheritance diagram for UTF8String:

Collaboration diagram for UTF8String:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>UTF8String (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String (const UTF8String &amp;str)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String (const JString &amp;str)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String (const char *str)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String (const EG_CHAR *str)</code></td>
<td></td>
</tr>
<tr>
<td><code>~UTF8String (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String &amp; operator=(const UTF8String &amp;Rhs)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String &amp; operator=(const JString &amp;Rhs)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String &amp; operator=(const char *Rhs)</code></td>
<td></td>
</tr>
<tr>
<td><code>UTF8String &amp; operator=(const EG_CHAR *Rhs)</code></td>
<td></td>
</tr>
<tr>
<td><code>operator const char * (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>operator JString (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>JString JStringRepresentation (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>unsigned int size (void) const</code></td>
<td></td>
</tr>
</tbody>
</table>

Public Member Functions inherited from **BaseCharString**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseCharString ()</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual ~BaseCharString (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>const char * cstr (void) const</code></td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>unsigned int length</td>
<td>(void) const</td>
</tr>
<tr>
<td>JString &amp; toString</td>
<td>(JString &amp;retStr, bool withTypes=false) const</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static unsigned int <strong>size</strong> (const <strong>JString</strong> &amp;str)</td>
<td>Static function to calculate the size of a string</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>setListener</strong> (const <strong>BaseListener</strong> *baseListener)</td>
<td>Function to set a listener object</td>
</tr>
<tr>
<td>static int <strong>getDebugOutputLevel</strong> (void)</td>
<td>Function to get the debug output level</td>
</tr>
<tr>
<td>static bool <strong>setDebugOutputLevel</strong> (int debugLevel)</td>
<td>Function to set the debug output level</td>
</tr>
<tr>
<td>static const <strong>LogFormatOptions</strong> &amp; <strong>getLogFormatOptions</strong> (void)</td>
<td>Function to get the log format options</td>
</tr>
<tr>
<td>static void <strong>setLogFormatOptions</strong> (const <strong>LogFormatOptions</strong> &amp;options)</td>
<td>Function to set the log format options</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>UTF8String (void)</th>
</tr>
</thead>
</table>

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 ( const JString & wstr )

Copy-Constructor: Creates a new UTF8String from a deep copy of the argument string.

Parameters
  wstr The JString to copy.
Copy-Constructor: Creates a new `UTF8String` from a deep copy of the argument string.

**Parameters**
- `str` The UTF8 string to copy.
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()

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.
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.
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

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  

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`. 
unsigned int size ( const JString & str )

Parameters
str a JString instance

Returns
the size in bytes that an UTF8 representation of the provided JString instance would have
ValueObject< Etype >
Class Template Reference

Inheritance diagram for ValueObject< Etype >:

Collaboration diagram for ValueObject< Etype >:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueObject</td>
<td>(const ValueObject&lt; Etype &gt; &amp;toCopy)</td>
</tr>
<tr>
<td>ValueObject</td>
<td>(const Object &amp;obj)</td>
</tr>
<tr>
<td>ValueObject</td>
<td>(const Object *obj)</td>
</tr>
<tr>
<td>ValueObject</td>
<td>(const typename Helpers::ConfirmAllowed&lt; Etype &gt;::type &amp;data)</td>
</tr>
<tr>
<td>ValueObject</td>
<td>(const typename Helpers::ConfirmAllowed&lt; Etype &gt;::type pData,</td>
</tr>
<tr>
<td></td>
<td>typename Helpers::ArrayLengthType&lt; Etype &gt;::type size)</td>
</tr>
<tr>
<td>ValueObject</td>
<td>(const typename Helpers::ConfirmAllowed&lt; Etype &gt;::type pData, const short *sizes)</td>
</tr>
<tr>
<td>virtual ~ValueObject</td>
<td>(void)</td>
</tr>
<tr>
<td>virtual ValueObject&lt; Etype &gt; &amp; operator=</td>
<td>(const ValueObject&lt; Etype &gt; &amp;toCopy)</td>
</tr>
<tr>
<td>virtual ValueObject&lt; Etype &gt; &amp; operator=</td>
<td>(const Object &amp;obj)</td>
</tr>
</tbody>
</table>
```cpp
const Helpers::ArrayLengthType< Etype >::type * getSizes (void) const

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**

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>~Base</td>
<td>virtual (void)</td>
</tr>
</tbody>
</table>

### Public Member Functions inherited from **ToString**

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ToString</td>
<td>virtual (void)</td>
</tr>
<tr>
<td>typeToString</td>
<td>virtual JString (void)</td>
</tr>
<tr>
<td>toString</td>
<td>JString (bool withTypes=false) const</td>
</tr>
</tbody>
</table>
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void setListener (const BaseListener *baseListener)</td>
</tr>
<tr>
<td>static int getDebugOutputLevel (void)</td>
</tr>
<tr>
<td>static bool setDebugOutputLevel (int debugLevel)</td>
</tr>
<tr>
<td>static const LogFormatOptions &amp; getLogFormatOptions (void)</td>
</tr>
<tr>
<td>static void setLogFormatOptions (const LogFormatOptions &amp;options)</td>
</tr>
</tbody>
</table>
Detailed Description

```cpp
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 ( 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 **getAddress()** return 0.

**Parameters**

- **obj** The **Object** to copy.
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.
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.
Constructor.

Creates an object out of a deep copy of the passed multi-dimensional E\(\text{type}\) array.

**Parameters**

- **pData** The array to copy.
- **sizes** The array of element counts for the different dimensions of data.
§ ~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=(): 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:

```
[legend]
```

Collaboration diagram for LitePeer:

```
[legend]
```
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LitePeer</strong> (Photon::PhotonList connectionProtocol=Photon::ConnectionProtocol::DEFAULT)</td>
<td>Virtual destructor</td>
</tr>
<tr>
<td>virtual <strong>~LitePeer</strong> (void)</td>
<td></td>
</tr>
</tbody>
</table>

**Template Function: opRaiseEvent**

```cpp
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)
```

**Template Function: opRaiseEvent**

```cpp
template<typename Ftype>
bool opRaiseEvent (bool reliable, Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, 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)
```

**Template Function: opRaiseEvent**

```cpp
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)
```

**Virtual Function: opJoin**

```cpp
virtual bool opJoin (const Common::JString Common::Hashtable &gameProperties=Common::Hashtable(), Common::Hashtable &actorProperties=Common::Hashtable(), bool broadcastActorProperties=false)
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual bool opLeave(void)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opChangeGroups(const Common::JVector*pGroupsToRemove, const Common::JVector*pGroupsToAdd)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opSetPropertiesOfActor(int actorNr, const Common::Hashtable&amp;properties, bool broadcast, nByte channelID=0)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opSetPropertiesOfGame(const Common::Hashtable&amp;properties, bool broadcast, nByte channelID=0)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opGetProperties(nByte channelID=0)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opGetPropertiesOfActor(const nByte*properties, short numProperties, const int*actorNrList=NULL, short numActors=0, nByte channelID=0)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opGetPropertiesOfGame(const nByte*properties, short numProperties)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual bool opGetPropertiesOfGame(const nByte*properties, short numProperties, nByte channelID=0)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Public Member Functions inherited from PhotonPeer**

PhotonPeer (PhotonListener
connectionProtocol=ConnectionProtocol::DEFAULT)

`virtual ~PhotonPeer(void)`

`virtual bool connect(const Common::JString)`
<table>
<thead>
<tr>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Common::JString &amp;appID=Common::JString</code></td>
</tr>
<tr>
<td><code>template&lt;typename Ftype &gt; bool connect (const Common::JString &amp;appID, const Ftype &amp;customData)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Ftype &gt; bool connect (const Common::JString &amp;appID, const Ftype &amp;customData, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</code></td>
</tr>
<tr>
<td><code>template&lt;typename Ftype &gt; bool connect (const Common::JString &amp;appID, const Ftype &amp;customDataArray, const short *pArrSizes)</code></td>
</tr>
<tr>
<td><code>virtual void disconnect (void)</code></td>
</tr>
<tr>
<td><code>virtual void service (bool dispatchIncomingCommands)</code></td>
</tr>
<tr>
<td><code>virtual void serviceBasic (void)</code></td>
</tr>
<tr>
<td><code>virtual bool opCustom (const OperationRequest &amp;request, bool sendReliable, nByte channelId)</code></td>
</tr>
<tr>
<td><code>virtual bool sendOutgoingCommands (void)</code></td>
</tr>
<tr>
<td><code>virtual bool sendAcksOnly (void)</code></td>
</tr>
<tr>
<td><code>virtual bool dispatchIncomingCommands (void)</code></td>
</tr>
<tr>
<td><code>virtual bool establishEncryption (void)</code></td>
</tr>
<tr>
<td><code>virtual void fetchServerTimestamp (void)</code></td>
</tr>
<tr>
<td><code>virtual void resetTrafficStats (void)</code></td>
</tr>
</tbody>
</table>
virtual void resetTrafficStatsMaximumCounters

virtual Common::JString vitalStatsToString (bool all) const

virtual void pingServer (const Common::JString & pingAttempts)

virtual void initUserDataEncryption (const > &secret)

virtual void initUDPEncryption (const Common::JVector & encryptSecret, const Common::JVector & HMACSecret)

PhotonListener * getListener (void)

int getServerTimeOffset (void) const

int getServerTime (void) const

int getBytesOut (void) const

int getBytesIn (void) const

int getByteCountCurrentDispatch

int getByteCountLastOperation (void)

int getPeerState (void) const

int getSentCountAllowance (void)

void setSentCountAllowance (int s)

int getTimePingInterval (void) const
void setTimePingInterval (int timePingInterval)

int getRoundTripTime (void) const

int getRoundTripTimeVariance (void)

int getTimestampOfLastSocketReceive

int getDebugOutputLevel (void) const

bool setDebugOutputLevel (int debugLevel)

const Common::LogFormatOptions & getLogFormatOptions (void) const

void setLogFormatOptions (const Common::LogFormatOptions &)

int getIncomingReliableCommandsCount

short getPeerID (void) const

int getDisconnectTimeout (void)

void setDisconnectTimeout (int disconnectTimeout)

int getQueuedIncomingCommands

int getQueuedOutgoingCommands

Common::JString getServerAddress (void) const

bool getIsPayloadEncryptionAvailable

bool getIsEncryptionAvailable (void) const

int getResentReliableCommands
int getLimitOfUnreliableCommands

void setLimitOfUnreliableCommands

bool getCRCEnabled (void) const

void setCRCEnabled (bool crcEnabled)

int getPacketLossByCRC (void) const

bool getTrafficStatsEnabled (void)

void setTrafficStatsEnabled (bool trafficStasEnabled)

int getTrafficStatsElapsedMs (void) const

const TrafficStats & getTrafficStatsIncoming (void)

const TrafficStats & getTrafficStatsOutgoing (void)

const TrafficStatsGameLevel & getTrafficStatsGameLevel (void)

nByte getQuickResendAttempts (void) const

void setQuickResendAttempts (nByte quickResendAttempts)

nByte getConnectionProtocol (void)

void setConnectionProtocol (nByte connectionProtocol)

nByte getChannelCountUserChannels
## 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()

Constructor

Parameters

- **listener**
  Pointer to the application’s implementation of the Listener interface. Has to be valid for at least the lifetime of the instance, which is created by this constructor.

- **connectionProtocol**
  Protocol to use to connect to Photon. One of the constants specified in Photon::ConnectionProtocol.

See also

- PhotonListener
- Photon::ConnectionProtocol
§ ~LitePeer()

~LitePeer (void )

Destructor.
Member Function Documentation
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 \texttt{opRaiseEvent()} operation, the local application will be notified by a call to the **PhotonListener::onOperationResponse()** callback with the parameter \texttt{opcCode} being set to \texttt{OperationCode::RAISE_EV}.

```cpp
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 \texttt{EventCache::DO\_NOT\_CACHE}

- **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 \texttt{ReceiverGroup::OTHERS}
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]

```cpp
template<
typename Ftype>
bool opRaiseEvent ( bool Ftype,
    typename Common::Helpers::ArrayLengthType< Ftype > nByte,
    nByte nByte nByte const int * short nByte nByte )
```

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 one of the supported data types, specified at [Table of Datatypes](#)

- **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 and Channels](#)

- **eventCaching**
  - has to be one of the constants specified in [EventCache](#)::DO_NOT_CACHE

- **targetPlayers**
  - the actorNrs of the clients, which should receive actors in the room

- **numTargetPlayers**
  - the number of actorNrs passed (array size)
receiverGroup has to be one of the constants specified in the `ReceiverGroup` definition. `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()** [3/3]

```cpp
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
<table>
<thead>
<tr>
<th><strong>eventCaching</strong></th>
<th>Information. Has to be one of the constants specified in <code>EventCache</code>, default is <code>EventCache::DO_NOT_CACHE</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>targetPlayers</strong></td>
<td>The actorNrs of the clients, which should receive the event, set to NULL, to send the event to all actors in the room</td>
</tr>
<tr>
<td><strong>numTargetPlayers</strong></td>
<td>The number of actorNrs passed (array size)</td>
</tr>
<tr>
<td><strong>receiverGroup</strong></td>
<td>Has to be one of the constants specified in <code>ReceiverGroup</code>, default is <code>ReceiverGroup::OTHERS</code></td>
</tr>
<tr>
<td><strong>interestGroup</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
§ opJoin()

```cpp
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 is not in use yet.

Rooms (or games) are simply identified by name. Lite assumes that users want to get into a room - no matter if it existed before or not, so it might be a new one. If you want to make sure a room is created (new, empty), the client 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 select a distinct one.

Each actor (a.k.a. player) in a room will get events that are raised for the room player (if he is contained in the receiver list).

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

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

Joining a room will 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 convention: only used if game is new/created
- **actorProperties**: optional, set of actor properties
- **broadcastActorProperties**: true to broadcast actor properties in join/leave, false to not broadcast them, default is false

Returns

- true, if successful, false otherwise

See also

- PhotonListener::onEvent(), PhotonListener::onOperationResponse(), opLeave()
§ 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()

```cpp
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 won’t add/remove any groups. &JVector<nByte>() will add/remove all (existing) groups. 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. A &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()

```c
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 to 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()

```cpp
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 to 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, opGetPropertiesOfGame()
§ \text{opGetProperties()}

\begin{verbatim}
bool opGetProperties ( nByte channelID = 0 )
\end{verbatim}

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 \textit{Photon Properties}

\textbf{Parameters}

- \texttt{channelID} the channel index. See \textit{Fragmentation and Channels}

\textbf{Returns}

- true, if successful, false otherwise

\textbf{See also}

- \textit{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 actor numbers
- **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 Channel

Returns

true, if successful, false otherwise

See also

Photon Properties, opSetPropertiesOfActor()
§ opGetPropertiesOfActor() [2/2]

```c
bool opGetPropertiesOfActor ( const nByte * properties,
    short numProperties,
    const int * actorNrList = NULL,
    short numActors = 0,
    nByte channelID = 0 )
```

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](#)
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()
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](#)
AuthenticationValues Class Reference

Inheritance diagram for AuthenticationValues:

```
AuthenticationValues
   `-- Base
         `-- ToString
```

[legend]

Collaboration diagram for AuthenticationValues:

```
AuthenticationValues
   `-- Base
         `-- ToString
```

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthenticationValues (void)</td>
</tr>
<tr>
<td>nByte getType (void) const</td>
</tr>
<tr>
<td>AuthenticationValues &amp; setType (nByte type)</td>
</tr>
<tr>
<td>const Common::JString &amp; getParameters (void) const</td>
</tr>
<tr>
<td>AuthenticationValues &amp; setParameters (const Common::JString &amp;parameters)</td>
</tr>
<tr>
<td>AuthenticationValues &amp; setParametersWithUsernameAndToken (const Common::JString &amp;username, const Common::JString &amp;token)</td>
</tr>
<tr>
<td>const Common::JVector&lt; nByte &gt; &amp; getData (void) const</td>
</tr>
<tr>
<td>AuthenticationValues &amp; setData (const Common::JVector&lt; nByte &gt; &amp;data)</td>
</tr>
<tr>
<td>const Common::JString &amp; getSecret (void) const</td>
</tr>
<tr>
<td>const Common::JString &amp; getUserID (void) const</td>
</tr>
<tr>
<td>AuthenticationValues &amp; setUserID (const Common::JString &amp;userID)</td>
</tr>
<tr>
<td>virtual Common::JString &amp; toString (Common::JString &amp;ret withTypes=false) const</td>
</tr>
</tbody>
</table>

- **Public Member Functions inherited from Base**
  - virtual ~Base (void)

- **Public Member Functions inherited from ToString**
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~ToString (void)</td>
<td></td>
</tr>
<tr>
<td>virtual JString typeToString (void) const</td>
<td></td>
</tr>
<tr>
<td>JString toString (bool withTypes=false) const</td>
<td></td>
</tr>
</tbody>
</table>
# 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.

Therefor 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()**

```cpp
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()

```cpp
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&param2=value2&param3=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()`
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 chainable 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**
getParameter(), setParameters(),
setParametersWithUsernameAndToken(), setData()
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()

```cpp
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()**

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

**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 disproportionally 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

Client Class Reference

Inheritance diagram for Client:

Collaboration diagram for Client:
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Member Functions</td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td>(LoadBalancing::Listener &amp;applicationID, const Common::JString &amp;appVersion,</td>
</tr>
<tr>
<td></td>
<td>nByte connectionProtocol=Photon::ConnectionProtocol::DEFAULT, bool autoLobbyStats</td>
</tr>
<tr>
<td></td>
<td>regionSelectionMode=RegionSelectionMode::DEFAULT, bool useAlternativePorts)</td>
</tr>
<tr>
<td>virtual ~Client (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool connect</td>
<td>(const AuthenticationValues &amp;authenticationValues=Common::JString &amp;username=L&quot;&quot;,</td>
</tr>
<tr>
<td>virtual void disconnect (void)</td>
<td></td>
</tr>
<tr>
<td>virtual void service (bool dispatchIncomingCommands)</td>
<td></td>
</tr>
<tr>
<td>virtual void serviceBasic (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opCustom</td>
<td>(const Photon::OperationRequest &amp;operationRequest, bool sendReliable, nByte channelID=0, bool encrypt=false)</td>
</tr>
<tr>
<td>virtual bool sendOutgoingCommands</td>
<td></td>
</tr>
<tr>
<td>virtual bool sendAcksOnly (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool dispatchIncomingCommands</td>
<td></td>
</tr>
<tr>
<td>virtual void fetchServerTimestamp</td>
<td></td>
</tr>
<tr>
<td>virtual void resetTrafficStats</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Parameters</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>virtual void resetTrafficStatsMaximumCounters</td>
<td></td>
</tr>
<tr>
<td>virtual Common::JString vitalStatsToString (bool all) const</td>
<td></td>
</tr>
<tr>
<td>virtual bool opJoinLobby (const &amp;lobbyName=Common::JString,</td>
<td></td>
</tr>
<tr>
<td>lobbyType=LobbyType::DEFAULT)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opLeaveLobby (void)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opCreateRoom (const RoomOptions &amp;options=Common::JVector&lt;</td>
<td></td>
</tr>
<tr>
<td>&amp;expectedUsers=Common::JVector())</td>
<td></td>
</tr>
<tr>
<td>virtual bool opJoinOrCreateRoom (&amp;gameID, const RoomOptions &amp;options=</td>
<td></td>
</tr>
<tr>
<td>Common::JVector&lt;&amp;expectedUsers=Common::JString&gt;)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opJoinRoom (const rejoin=false, int cacheSliceIndex=0,</td>
<td></td>
</tr>
<tr>
<td>const Common::JVector&lt;&amp;expectedUsers=Common::JVector&gt;)</td>
<td></td>
</tr>
<tr>
<td>virtual bool opJoinRandomRoom (&amp;customRoomProperties=maxPlayers=0,</td>
<td></td>
</tr>
<tr>
<td>nBytes=0, matchmakingMode=const Common::JString &amp;lobbyName=Common::JString &amp;lobbyType=LobbyType::DEFAULT &amp;sqlLobbyFilter=const Common::JVector&lt;&amp;expectedUsers=Common::JVector&gt;)</td>
<td></td>
</tr>
</tbody>
</table>
virtual bool opLeaveRoom (bool willComeBack=false, bool sendAuthCookie=false)

template<typename Ftype >
bool opRaiseEvent (bool reliable, const Ftype &parameters, nByte eventCode, const RaiseEventOptions &options=RaiseEventOptions)

template<typename Ftype >
bool opRaiseEvent (bool reliable, const Ftype &ParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, nByte eventCode, const RaiseEventOptions &options=RaiseEventOptions)

virtual bool opFindFriends (const short numFriendsToFind)

virtual bool opLobbyStats (const LoadBalancing::LobbyStatsRequest &lobbiesToQuery=Common::JVector<LoadBalancing::LobbyStatsRequest> untouch)

virtual bool opChangeGroups (const *pGroupsToRemove, const *pGroupsToAdd)

virtual bool opCustomAuthenticationSendNextStepData (AuthenticationValues)

virtual bool opWebRpc (const Common::JString C)
| template<typename Ftype > | bool | opWebRpc (const C Ftype &parameters, | \begin{itemize}
  \item \textbf{template<typename Ftype >}
  \item \textbf{virtual bool selectRegion (const)}
  \item \textbf{virtual bool reconnectAndRejoin}
  \item \textbf{template<typename Ftype >}
  \item bool \textbf{sendDirect (const F pParameterArray, bool sendAuthCookie=false)}
  \item \textbf{template<typename Ftype >}
  \item bool \textbf{sendDirect (const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, int targetPlayer, bool fallbackRelay=false)}
  \item \textbf{template<typename Ftype >}
  \item bool \textbf{sendDirect (const Ftype pParameterArray, const short*pArrSizes, int targetPlayer, bool fallbackRelay=false)}
  \item \textbf{template<typename Ftype >}
  \item int \textbf{sendDirect (const Ftype &parameters, int targetPlayers=Common::JVector< &targetPlayers=Common::JVector<, bool fallbackRelay=false)}}
\end{itemize} |
template<typename Ftype>

int sendDirect (const Ftype &ParameterArray,
    typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize,
    const Common::JVector &targetPlayers,
    const fallbackRelay=false)

int sendDirect (const Ftype &ParameterArray,
    const short *pArrSizes,
    const Common::JVector &targetPlayers,
    const fallbackRelay=false)

int getServerTimeOffset

int getServerTime (void)

int getBytesOut (void)

int getBytesIn (void)

int getByteCountCurrentDispatch

int getByteCountLastOperation

int getSentCountAllowance

void setSentCountAllowance

int getTimePingInterval

void setTimePingInterval

int getRoundTripTime

int getRoundTripTimeVariance

int getTimestampOfLastSocketReceive
int getDebugOutputLevel

bool setDebugOutputLevel

const Common::LogFormatOptions & getLogFormatOptions

void setLogFormatOptions

Common::LogFormatOptions & getLogFormatOptions

int getIncomingReliableCommandsCount

short getPeerID (void) const

int getDisconnectTimeout

void setDisconnectTimeout

int getQueuedIncomingCommands

int getQueuedOutgoingCommands

bool getIsPayloadEncryptionAvailable

bool getIsEncryptionAvailable

int getResentReliableCommands

int getLimitOfUnreliableCommands

void setLimitOfUnreliableCommands

bool getCRCEnabled (void) const

void setCRCEnabled (bool crcEnabled)

int getPacketLossByCRC
bool getTrafficStatsEnabled

void setTrafficStatsEnabled

int getTrafficStatsElapsedMs

const Photon::TrafficStats & getTrafficStatsIncoming

const Photon::TrafficStats & getTrafficStatsOutgoing

const Photon::TrafficStatsGameLevel & getTrafficStatsGameLevel

nByte getQuickResendAttempts

void setQuickResendAttempts

nByte getChannelCountUserChannels

int getState (void) const

const Common::JString & getMasterserverAddress

int getCountPlayersIngame

int getCountGamesRunning

int getCountPlayersOnline

MutableRoom & getCurrentlyJoinedRoom

const Common::JVector< Room * > & getRoomList (void)

const Common::JVector< Common::JString > & getRoomNameList

bool isInRoom (void)
bool getIsInGameRoom

bool getIsInLobby (void)

bool getAutoJoinLobby

void setAutoJoinLobby

MutablePlayer & getLocalPlayer (void)

cost Common::JVector< FriendInfo > & getFriendList (void)

int getFriendListAge (void)

int getDisconnectedCause

const Common::JString & getUserID (void)

const Common::JString & getRegionWithBestPing
<table>
<thead>
<tr>
<th>Static Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static short</strong> getPeerCount (void)</td>
</tr>
</tbody>
</table>
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:createElement() 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()

```cpp
Client ( LoadBalancing::Listener & listener,
    const Common::JString & applicationID,
    const Common::JString & appVersion,
    nByte connectionProtocol = Photon::ConnectionProtocol::DEFAULT,
    bool autoLobbyStats = false,
    nByte regionSelectionMode = RegionSelectionMode::DEFAULT,
    bool useAlternativePorts = false
)
```

Constructor.

**Parameters**

- **listener**
  Reference to the application's implementatio 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 your dashboard for Photon Cloud. This parameter gets ignored by Photon Server.

- **appVersion**
  Only clients that use the exact same appVersion can see each other or to even see each other, i.e. incompatible game or public, closed-beta, QA, staging and dev clients. This parameter gets ignored by Photon Server.

- **connectionProtocol**
  The protocol to use to connect to the Photon Server. Must match one of the constants specified in ConnectionProtocol.

- **autoLobbyStats**
  Pass true, if you want to automatically receive lobby stats, false otherwise. Call `opLobbyStats()` to explicitly request a lobby stats update.

- **regionSelectionMode**
  Determines how the Photon Cloud Region to which the connect should be selected. Must match one of the constants specified in RegionSelectionMode. This parameter gets ignored when connecting to Photon Server.
**useAlternativePorts**

Determines if the standard or the alternative port range should be used. This parameter currently is only relevant when `ConnectionProtocol::UDP` is passed for parameter `connectionProtocol` and gets ignored otherwise. A router or firewall might block connections that use one port range but don’t block connections that use the other, so when connecting with one range fails, then you may want to try with the other.

**See also**

- [Listener](#), [ConnectionProtocol](#), [RegionSelectionMode](#), [NetworkPort](#)
§ ~Client()

~Client ( void )

Destructor.
Member Function Documentation
§ connect()

```cpp
bool connect ( const AuthenticationValues & authenticationValues = AuthenticationValues(),
              const Common::JString & username = L"",
              const Common::JString & serverAddress = M_NAMESERVER,
              nByte serverType = ServerType::NAME_SERVER )
```

This function starts establishing a connection to a Photon server. The server's response will arrive in `Listener::connectReturn()`.

The connection is successfully established when the Photon client receives a valid response from the server. The connect-attempt fails when a network error occurs or when the server is not responding. A call to this function starts an asynchronous operation. The result of this 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, `Listener::connectionErrorReturn()` or `Listener::connectReturn()` will get called. The operation was successful, when `Listener::connectReturn()` got called with `errorCode==0`.

**Parameters**

- **authenticationValues** An instance of class `AuthenticationValues`
- **username** The users display name as shown to other users - not to be confused with the users unique ID for identification and authentication 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 require square brackets to indicate where the address and port begins. If no port is given, then the default protocol and server type will be used.
- **serverType** One of the values in `ServerType`. Must match...
Photon server that is reachable at the given address and port. Should be ServerType::NAME_SERVER for Photon Cloud and ServerType::MASTER_SERVER for self-hosted Photon Server instances. You should NOT directly pass the address of a master server with Photon Cloud, but always connect to the name server.

**Returns**

true, if it could successfully start establishing a connection (the result will be passed in a callback function in this case) or false, if an error occurred and the connection could not be established (no callback function will be called then).

**See also**

`disconnect()`, `NetworkPort`
§ disconnect()

```c
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()

```cpp
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 within a loop until all incoming commands have been dispatched.)
3. `sendOutgoingCommands()` (called within 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 = 
\texttt{dispatchIncomingCommands()}
won't be called, default is true
§ serviceBasic()

```c
void serviceBasic ( void )
```

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()

```cpp
bool opCustom ( const Photon::OperationRequest & operationRequest,
               bool sendReliable,
               nByte channelID = 0,
               bool encrypt = false )
```

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 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 )  

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/or 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 )

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 )

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

ture 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()

```c
void fetchServerTimestamp ( void )
```

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 roundtrip time is low and the timestamp + roundtrip time / 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 )

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()

```cpp
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 true, 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, the 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 `connect()` for `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 )

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()

```cpp
bool opCreateRoom ( const Common::JString & gameID,
const RoomOptions & options,
const Common::JVector< Common::JString > & expectedUsers )
```

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 assign a roomName (a GUID as string). Room names are unique.

A room will be attached to the lobby that you have specified in the passed in options. 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 this function sends a request to create a room with. Must be unique and not in use or the room can't be created. If this is an empty string, then the server will assign a GUID as name.

If a room with the specified name does already exist, then the operation will fail and Listener::createRoomReturn() will get called with an error code.

Parameters

- **gameID** The name to create a room with. Must be unique and not in use.
- **options** An instance of RoomOptions, that can be used to specify various options for room creation.
- **expectedUsers** Sets a list of user IDs for which the server should reserve slots.

Returns

true, if the request could successfully be queued for sending to the server; false otherwise.
See also

opJoinOrCreateRoom(), opJoinRoom(), opJoinRandomRoom(),
Listener::createRoomReturn()
§ opJoinOrCreateRoom()

```cpp
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 if it does not exist yet.

This function sends a request to the server to join the specified game room. Listener::joinOrCreateRoomReturn() gets called when the operation has finished.

Unlike opJoinRoom(), this operation does not fail if the room does not exist. It can be useful when sending invitations to a room before actually creating it: Any invited player (whoever is first) can call this and on demand, the room gets created implicitly.

This operation does not allow you to re-join a game. To return to a room, use previously.

**Remarks**

A Client instance can only be inside one room at a time. Therefore, another game room. Any lobby the client currently resides in will implicitly be left when entering a game room.

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 various options for room creation. These options will be ignored when the room already exists.

- **cacheSliceIndex**  
  Allows to request a specific cache slice - all events published to the client after joining the room - see Lite::EventCache.

- **expectedUsers**  
  Sets a list of user IDs for which the server should reserve slots. Those slots can't be taken by other players. If the room already exists, then this list will be merged with any previously set list of expected users for this room.

**Returns**

...
true, if the request could successfully be queued for sending to the server.

See also
opCreateRoom(), opJoinRoom(), opJoinRandomRoom(), opLeaveRoom(), Listener::joinOrCreateRoomReturn()
§ opJoinRoom()

```cpp
bool opJoinRoom ( const Common::JString & gameID,
    bool          rejoin,
    int           cacheSliceIndex,
    const Common::JVector< Common::JString > & expectedUsers )
```

Joins the specified game room.

This function sends a request to the server to join the specified game room. It gets called when the operation has been finished.

This function is useful when you are using a lobby to list rooms and know their names (per 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 does not matter if the room is already set up.

**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 implicitly be left when entering a game room.

If a room with the specified name does not exist or if the room is full or closed, then this operation will fail and `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. It needs to be true if this is a rejoin.
- **cacheSliceIndex**
  Allows to request a specific cache slice - all events published to the client after joining the room - see cached events.
- **expectedUsers**
  Sets a list of user IDs for which the server should reserve slots. Those slots can't be taken by other players. This list will be merged with any previously set list of expected users for this room.

**Returns**
true, if the request could successfully be queued for sending to the server.

See also

`opCreateRoom()`, `opJoinOrCreateRoom()`, `opJoinRandomRoom`
§ opJoinRandomRoom()

```c++
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 gets called when the operation has been finished.

Remarks

A **Client** instance can only be inside one room at a time. Therefore the client cannot be inside another game room. Any lobby the client currently resides in will be left when entering a game room.

If no rooms are fitting or available (all full, closed or not visible), then **Listener::joinRandomRoomReturn()** will get called with an error code.

Parameters

- **customRoomProperties** Used as a filter for matchmaking. The server will only consider rooms that match the specified filters. Note that only those custom room properties that have been specified for listing in the lobby will be used for matchmaking, so a room with a custom property that was not specified in the list of properties to show in the lobby will not be considered.

- **maxPlayers** Must match the value of a room's maxPlayers property for that room to be considered for matchmaking.

- **matchmakingMode** Needs to be one of the values in **MatchMakingMode**.

- **lobbyName** The name of the lobby in which matchmaking should take place. Only rooms that are listed in that lobby will be considered for matchmaking.

- **lobbyType** The type of the lobby in which matchmaking should take place. Needs to be one of the values in **LobbyType**. Note that a lobby with the same name, but a different type, cannot be considered for matchmaking, as a lobby name only needs to be unique among lobbies of the same type.
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 |

Returns
ture, if the request could successfully be queued for sending to the s

See also
opCreateRoom(), opJoinOrCreateRoom(), opJoinRandomRoom
Listener::joinRoomReturn(), Matchmaking and Lobby
§ opLeaveRoom()

```cpp
bool opLeaveRoom ( bool willComeBack = false,
                   bool sendAuthCookie = false )
```

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]

```cpp
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 a
supported for values at **serializable datatypes**.

This function provides the option to raise events reliably or unreliably. With
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
ture, if the request could successfully be queued for sending to the server, false otherwise.

See also
Listenern::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 so on).
  options    see RaiseEventOptions
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 from the above function only in what argument(s) it accepts.

**Parameters**

- **reliable**
  true = operation will be sent reliably; false = no resend is required, 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 Table of Datatypes

- **arrSize**
  the number of elements in pParameterArray

- **eventCode**
  number for arbitrary classification of the type of e.g., for chat messages, and so on).

- **options**
  see 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 )

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 >& lobbiesToQuery )

Sends the specified list of LobbyStatsRequest objects to the server. This is useful for retrieving statistics for various lobbies.

This function can be called when the caller does not currently reside in a game room.

Remarks

This operation will fail and return false if the client does currently reside inside a game room.

Note

Pass 'true' for the 'autoLobbyStats' parameter of Client() to automatically receive regular stats updates for all lobbies in Listener::onLobbyStatsUpdate().

Parameters

lobbiesToQuery A Common::JVector containing a LobbyStatsRequest.

Returns

true, if the request could successfully be queued for sending to the server.

See also

Client(), Listener::onLobbyStatsResponse(), Listener::onLobbyStatsUpdate()
opChangeGroups()

```cpp
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 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 operation (approximately getRoundTripTime() / 2 milliseconds after the Client sent it).

Remarks
This operation will fail and return false if the client does not currently inside a game room.

Parameters
- **pGroupsToRemove** Groups to remove from interest. NULL will not any. An empty instance will remove all.
- **pGroupsToAdd** Groups to add to interest. NULL will not add any empty instance will add all existing groups.
Returns
true, if the request could successfully be queued for sending to the server, false otherwise.

See also
opRaiseEvent(), RaiseEventOptions::setInterestGroups(), Interestgroups
§ \textbf{opCustomAuthenticationSendNextStepData()}

\begin{verbatim}
bool opCustomAuthenticationSendNextStepData ( const AuthenticationValues &authenticationValues ) {
    // Implementation
}
\end{verbatim}

Used in conjunction with \code{Listener::onCustomAuthenticationIntermediateStep()} to implement multi-leg custom authentication.

While normally custom authentication is single-legged, occasionally a certain service may require multi-leg authentication. This means that the client sends some authentication data to the server when calling \code{connect()} and the server does not respond with a final result (successful connect or failed connect attempt due to an authentication error), but with some intermediate data that gets passed to your \code{Listener::onCustomAuthenticationIntermediateStep()} implementation. These intermediate results are needed by your application to acquire the authentication data for the next step of the authentication process. You can then pass that next step data to this function to continue the authentication process after the \code{connect()} call.

\section*{Remarks}
This operation will fail and return false if the client is not currently expecting it to be called. A call by you is only expected after you have received a call to \code{Listener::onCustomAuthenticationIntermediateStep()} beforehand and only expected after each received call to \code{Listener::onCustomAuthenticationIntermediateStep()}. If a call is ever expected if the custom authentication that you have set up is single-legged (which is far more common) or if you have not set up any custom authentication at all, this function will always fail in these scenarios.

\section*{Parameters}
\begin{itemize}
    \item \textbf{authenticationValues} An instance of class \textbf{AuthenticationValues}
\end{itemize}

\section*{Returns}
\begin{itemize}
    \item true, if the request could successfully be queued for sending to the server
\end{itemize}

\section*{See also}
\begin{itemize}
    \item \code{connect()}, \code{Listener::onCustomAuthenticationIntermediateStep()}, \textbf{AuthenticationValues}
\end{itemize}
bool opWebRpc ( const Common::JString & uriPath )  

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". TheBaseUrl 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
```cpp
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uriPath</td>
<td>the URL path to call, relative to the baseUrl configured on Photon's server-side</td>
</tr>
<tr>
<td>parameters</td>
<td>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</td>
</tr>
<tr>
<td>sendAuthCookie</td>
<td>defines if the authentication cookie gets sent to a WebHook (if setup)</td>
</tr>
</tbody>
</table>
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 configured on 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
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()

```cpp
bool selectRegion ( const Common::JString & selectedRegion )
```

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", 


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()

```cpp
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()`
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.
送Direct() [2/6]

```cpp
template<typename Ftype>
bool sendDirect(
    const Ftype
    typename Common::Helpers::ArrayLengthType<Ftype>::type
    int
    bool
)
```

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

**Parameters**

- **pParameterArray** the data to send - has to be provided in the form of the supported data types, specified at Table of Datatypes
- **arrSize** the number of elements in pParameterArray
- **targetPlayer** the player number of the intended receiver of the number of another active player inside the same room
- **fallbackRelay** true if the Photon game server that hosts the room should be used as a fallback relay (by an automatic call to opRaiseEvent()), receivers to which no direct connection exists, false otherwise.

**Returns**

true, if the request could successfully be sent (this does not guarantee receipt), false otherwise.
### sendDirect() \[3/6\]

```cpp
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 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::JVector
                                           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, less than 1200 bytes

- **targetPlayers**
  the player numbers of the intended receivers of the message - must be the numbers of other active players inside 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()` specified receivers to which no direct connection exists, otherwise

**Returns**

the number of target players, for which the request could successfully be sent (this does not guarantee that it will be received).
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 supported data types, specified at Table of Datatypes - must be less than 1200 bytes
- **arrSize** the number of elements in pParameterArray
- **targetPlayers** the player numbers of the intended receivers of the message - must be the numbers of other active players 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**

the number of target players, for which the request could successfully be sent (this does not guarantee that it will be received).
template< typename Ftype >
bool sendDirect(
    const Ftype pParameterArray,
    const short * pArrSizes,
    const Common::JVector< int > & targetPlayers = Common::JVector<bool>(),
    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 `Tab 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 message must be the numbers of other active players 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

- the number of target players, for which the request could successfully be sent (this does not guarantee that it will be received).
§ getServerTimeOffset()

```c
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()`

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

**Returns**
the Photon Server's system time in ms.

see `getServerTimeOffset()`
§ getBytesOut()

```c
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()

```c
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()

```cpp
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()

```c
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()

```c
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()

```c
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()

```c
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()

```c
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()

```cpp
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()

```cpp
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()

```cpp
void setLogFormatOptions ( const Common::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()`
§ 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()

```c
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()

```java
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()

```c
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()

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

**Returns**

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

```c
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()

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

**Returns**
true if CRC enabled

**See also**
setCRCEnabled
§ setCRCEnabled()

```c
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()

```c
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()

```cpp
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()

**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()

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

**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
RoundTripTime+4*RoundTripTimeVariance).
§ setQuickResendAttempts()

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

Sets the number of resend attempts for a reliable command can be done in quick succession (after RoundTripTime+4*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 ) |

**Returns**

the count of peers, which have been initialized since the start of
the application. Interesting mainly for debugging purposes.
§ `getState()`

```cpp
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()

c) 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()

```c
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()**

```c
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()**

```c
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. Therefore 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()

```c
bool getIsInGameRoom ( void ) const
```

**Returns**
true if this client instance currently resides within a game room, false otherwise.

**See also**
getIsInRoom(), getIsInLobby()
§ getIsInLobby()

```cpp
bool getIsInLobby ( void ) const
```

**Returns**
true if this client instance currently resides within a lobby, false otherwise.

**See also**
getIsInRoom(), getIsInGameRoom()
§ `getAutoJoinLobby()`

```c
bool getAutoJoinLobby ( void ) const
```

**Returns**
the current value of the autJoinLobby 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()

```java
void setAutoJoinLobby ( bool autoJoinLobby )
```

Sets the value of the autJoinLobby 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()`
§ getLocalPlayer()

MutablePlayer & getLocalPlayer ( void )

Returns

a non-const reference to the MutablePlayer instance that is representing the local player.
§ `getFriendList()`

```cpp
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 server's 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()

```c
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 adresses 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 therefor to keep the time short that is needed for establishing a connection. For the same reason it is recommend that you acquire the result of the ping-procedure through this function and store it in local persistant 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.
FriendInfo Class Reference

Inheritance diagram for FriendInfo:

Collaboration diagram for FriendInfo:
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Common::JString getUserID (void) const</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>bool getIsOnline (void) const</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Common::JString getRoom (void) const</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>bool getIsInRoom (void) const</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>virtual Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <strong>Base</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <strong>setListener</strong> (const <strong>BaseListener</strong> *baseListener)</td>
</tr>
<tr>
<td>static int <strong>getDebugOutputLevel</strong> (void)</td>
</tr>
<tr>
<td>static bool <strong>setDebugOutputLevel</strong> (int debugLevel)</td>
</tr>
<tr>
<td>static const <strong>LogFormatOptions</strong> &amp; <strong>getLogFormatOptions</strong> (void)</td>
</tr>
<tr>
<td>static void <strong>setLogFormatOptions</strong> (const <strong>LogFormatOptions</strong> &amp;options)</td>
</tr>
</tbody>
</table>
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()`

```c
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()

```cpp
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**.
Inheritance diagram for Listener:

[Diagram of inheritance structure]

Collaboration diagram for Listener:

[Diagram of collaboration structure]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void debugReturn (int debugLevel, const Common::JString &amp;string) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void connectionErrorReturn (int errorCode) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void clientErrorReturn (int errorCode) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void warningReturn (int warningCode) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void serverErrorReturn (int errorCode) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void joinRoomEventAction (int playerNr, const Common::JVector&lt; int &gt; &amp;playernrs, const Player &amp;player) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void leaveRoomEventAction (int playerNr, bool isInactive) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void customEventAction (int playerNr, nByte eventCode, const Common::Object &amp;eventContent) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void connectReturn (int errorCode, const Common::JString &amp;errorString, const Common::JString &amp;region, const Common::JString &amp;cluster) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void disconnectReturn (void) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void createRoomReturn (int localPlayerNr, const Common::Hashtable &amp;roomProperties, const Common::Hashtable &amp;playerProperties, int errorCode, const Common::JString &amp;errorString) = 0</td>
<td></td>
</tr>
<tr>
<td>virtual void joinOrCreateRoomReturn (int localPlayerNr, const Common::Hashtable &amp;roomProperties, const Common::Hashtable &amp;playerProperties, int errorCode, const Common::JString &amp;errorString) = 0</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Signature</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>joinRoomReturn</td>
<td>virtual void joinRoomReturn (int localPlayerNr, const Common::Hashtable &amp;roomProperties, const Common::Hashtable &amp;playerProperties, int errorCode, const Common::JString &amp;errorString)=0</td>
</tr>
<tr>
<td>joinRandomRoomReturn</td>
<td>virtual void joinRandomRoomReturn (int localPlayerNr, const Common::Hashtable &amp;roomProperties, const Common::Hashtable &amp;playerProperties, int errorCode, const Common::JString &amp;errorString)=0</td>
</tr>
<tr>
<td>leaveRoomReturn</td>
<td>virtual void leaveRoomReturn (int errorCode, const Common::JString &amp;errorString)=0</td>
</tr>
<tr>
<td>joinLobbyReturn</td>
<td>virtual void joinLobbyReturn (void)=0</td>
</tr>
<tr>
<td>leaveLobbyReturn</td>
<td>virtual void leaveLobbyReturn (void)=0</td>
</tr>
<tr>
<td>onFindFriendsResponse</td>
<td>virtual void onFindFriendsResponse (void)</td>
</tr>
<tr>
<td>onLobbyStatsResponse</td>
<td>virtual void onLobbyStatsResponse (const Common::JVector&lt;LobbyStatsResponse&gt; &amp;)&amp;</td>
</tr>
<tr>
<td>webRpcReturn</td>
<td>virtual void webRpcReturn (int, const Common::JString &amp;, const Common::JString &amp;, int, const Common::Dictionary&lt;Common::Object, Common::Object&gt;&amp;)&amp;</td>
</tr>
<tr>
<td>onRoomListUpdate</td>
<td>virtual void onRoomListUpdate (void)</td>
</tr>
<tr>
<td>onRoomPropertiesChange</td>
<td>virtual void onRoomPropertiesChange (const Common::Hashtable &amp;)&amp;</td>
</tr>
<tr>
<td>onPlayerPropertiesChange</td>
<td>virtual void onPlayerPropertiesChange (int, const Common::Hashtable &amp;)&amp;</td>
</tr>
<tr>
<td>onAppStatsUpdate</td>
<td>virtual void onAppStatsUpdate (void)</td>
</tr>
<tr>
<td>onLobbyStatsUpdate</td>
<td>virtual void onLobbyStatsUpdate (const Common::JVector&lt; &amp;)</td>
</tr>
<tr>
<td>Function Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>virtual void onCacheSliceChanged</td>
<td>(int)</td>
</tr>
<tr>
<td>virtual void onMasterClientChanged</td>
<td>(int, int)</td>
</tr>
<tr>
<td>virtual void onCustomAuthenticationIntermediateStep</td>
<td>(const Common::Dictionary&lt; Common::JString, Common::Object &gt; &amp;)</td>
</tr>
<tr>
<td>virtual void onAvailableRegions</td>
<td>(const Common::JVector&lt; Common::JString &gt; &amp;, const Common::JVector&lt; Common::JString &gt; &amp;)</td>
</tr>
<tr>
<td>virtual void onSecretReceival</td>
<td>(const Common::JString &amp;)</td>
</tr>
<tr>
<td>virtual void onDirectMessage</td>
<td>(const Common::Object &amp;, int, bool)</td>
</tr>
<tr>
<td>virtual void onCustomOperationResponse</td>
<td>(const Photon::OperationResponse &amp;operationResponse)</td>
</tr>
</tbody>
</table>
Member Function Documentation
§ debugReturn()  

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

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.
LobbyStatsRequest Class Reference

Inheritance diagram for LobbyStatsRequest:

Collaboration diagram for LobbyStatsRequest:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LobbyStatsRequest</strong></td>
<td>(const Common::JString &amp;name=Common::JString(), nByte type=LobbyType::DEFAULT)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const Common::JString &amp;</td>
<td><strong>getName</strong> (void) const</td>
</tr>
<tr>
<td>nByte</td>
<td><strong>getType</strong> (void) const</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual Common::JString &amp;</td>
<td><strong>toString</strong> (Common::JString &amp;retStr, bool withTypes=false) const</td>
</tr>
</tbody>
</table>

#### Public Member Functions inherited from **Base**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td>~<strong>Base</strong> (void)</td>
</tr>
</tbody>
</table>

#### Public Member Functions inherited from **ToString**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual</td>
<td>~<strong>ToString</strong> (void)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual Common::JString &amp;</td>
<td><strong>typeToString</strong> (void) const</td>
</tr>
<tr>
<td>JString</td>
<td><strong>toString</strong> (bool withTypes=false) const</td>
</tr>
</tbody>
</table>
### Additional Inherited Members

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void setListener (const BaseListener *baseListener)</td>
<td>Inherited from <code>Base</code></td>
</tr>
<tr>
<td>static int getDebugOutputLevel (void)</td>
<td></td>
</tr>
<tr>
<td>static bool setDebugOutputLevel (int debugLevel)</td>
<td></td>
</tr>
<tr>
<td>static const LogFormatOptions &amp; getLogFormatOptions (void)</td>
<td></td>
</tr>
<tr>
<td>static void setLogFormatOptions (const LogFormatOptions &amp;options)</td>
<td></td>
</tr>
</tbody>
</table>
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::JString nByte, type = LobbyType::DEFAULT )

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 LobbyType.
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.
LobbyStatsResponse Class Reference

Inheritance diagram for LobbyStatsResponse:

```
|ToString|
|        |
|        |
|Base   |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|LobbyStatsResponse|
```

[legend]

Collaboration diagram for LobbyStatsResponse:

```
|ToString|
|        |
|        |
|Base   |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|LobbyStatsResponse|
```

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const Common::JString &amp; getName (void) const</code></td>
<td>Getter for name</td>
</tr>
<tr>
<td><code>nByte getType (void) const</code></td>
<td>Getter for type</td>
</tr>
<tr>
<td><code>int getPeerCount (void) const</code></td>
<td>Getter for peer count</td>
</tr>
<tr>
<td><code>int getRoomCount (void) const</code></td>
<td>Getter for room count</td>
</tr>
<tr>
<td><code>virtual Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Stringification method</td>
</tr>
</tbody>
</table>

#### Public Member Functions inherited from **Base**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual ~Base (void)</code></td>
<td>Destructor</td>
</tr>
</tbody>
</table>

#### Public Member Functions inherited from **ToString**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual ~ToString (void)</code></td>
<td>Destructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual JString typeToString (void) const</code></td>
<td>Type stringification</td>
</tr>
<tr>
<td><code>JString toString (bool withTypes=false) const</code></td>
<td>Stringification method with type indication</td>
</tr>
</tbody>
</table>

## 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()

```cpp
int getPeerCount ( void ) const
```

**Returns**

the number of clients that currently reside in this specific lobby
§ getRoomCount()

```cpp
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)

**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:

[Diagram of inheritance]

Collaboration diagram for MutablePlayer:

[Diagram of collaboration]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mutablePlayer</strong> (const <strong>mutablePlayer</strong> &amp;toCopy)</td>
<td>Virtual function that allows for the assignment of another <code>mutablePlayer</code> object.</td>
</tr>
<tr>
<td>virtual <strong>mutablePlayer</strong> &amp; operator= (const <strong>player</strong> &amp;toCopy)</td>
<td>Virtual function that allows for the assignment of another <code>Player</code> object.</td>
</tr>
<tr>
<td>virtual <strong>mutablePlayer</strong> &amp; operator= (const <strong>mutablePlayer</strong> &amp;toCopy)</td>
<td>Virtual function that allows for the assignment of another <code>mutablePlayer</code> object.</td>
</tr>
<tr>
<td>void setName (const <strong>Common::JString</strong> &amp;name, const <strong>WebFlags</strong> &amp;webflags=<strong>WebFlags</strong>( ))</td>
<td>Function to set the name of the player, with optional web flags.</td>
</tr>
<tr>
<td>void mergeCustomProperties (const <strong>Common::Hashtable</strong> &amp;customProperties, const <strong>WebFlags</strong> &amp;webflags=<strong>WebFlags</strong>( ))</td>
<td>Function to merge custom properties, with optional web flags.</td>
</tr>
<tr>
<td>template&lt;typename ktype , typename vtype &gt;</td>
<td>Function to add a custom property of a specific key and value type.</td>
</tr>
<tr>
<td>void addCustomProperty (const ktype &amp;key, const vtype &amp;value, const <strong>WebFlags</strong> &amp;webflags=<strong>WebFlags</strong>( ))</td>
<td>Adds a single custom property.</td>
</tr>
<tr>
<td>template&lt;typename ktype , typename vtype &gt;</td>
<td>Function to add multiple custom properties of a specific key and value type.</td>
</tr>
<tr>
<td>void addCustomProperty (ktype &amp;key, const vtype pValueArray, typename Common::Helpers::ArrayLengthType&lt;vtype&gt;::type arrSize, const <strong>WebFlags</strong> &amp;webflags=<strong>WebFlags</strong>( ))</td>
<td>Adds multiple custom properties at once.</td>
</tr>
<tr>
<td>template&lt;typename ktype , typename vtype &gt;</td>
<td>Function to add multiple custom properties with an array of sizes.</td>
</tr>
<tr>
<td>void addCustomProperty (ktype &amp;key, const vtype pValueArray, short *pArrSizes, const <strong>WebFlags</strong> &amp;webflags=<strong>WebFlags</strong>( ))</td>
<td>Adds multiple custom properties with an array of sizes.</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Common::JString toString (bool withTypes, bool withCustomProperties) const</code></td>
<td>Public Member Functions inherited from <code>Base</code></td>
</tr>
<tr>
<td><code>~Base (void)</code></td>
<td>Public Member Functions inherited from <code>Base</code></td>
</tr>
<tr>
<td><code>~ToString (void)</code></td>
<td>Public Member Functions inherited from <code>ToString</code></td>
</tr>
<tr>
<td><code>JString typeToString (void) const</code></td>
<td>virtual <code>JString typeToString (void) const</code></td>
</tr>
<tr>
<td><code>toString (bool withTypes=false) const</code></td>
<td>virtual <code>JString toString (bool withTypes=false) const</code></td>
</tr>
</tbody>
</table>
### 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 )

operator=.

Makes a deep copy of its right operand into its left operand.

This overwrites old data in the left operand.

Reimplemented from Player.
MutableRoom Class Reference

Inheritance diagram forMutableRoom:

[legend]

Collaboration diagram forMutableRoom:

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MutableRoom (const</strong> <strong>&amp;operator=</strong></td>
<td>virtual <strong>MutableRoom &amp;</strong> operator= (const <strong>Rc</strong> <strong>&amp;)</strong></td>
</tr>
<tr>
<td><strong>mutable Room &amp; operator= (const</strong> <strong>M</strong></td>
<td>virtual <strong>MutableRoom &amp;</strong> operator= (const <strong>Mu</strong> <strong>&amp;)</strong></td>
</tr>
<tr>
<td>nByte <strong>getPlayerCount (void)</strong></td>
<td>void <strong>setMaxPlayers (nB)</strong></td>
</tr>
<tr>
<td><strong>void setMaxPlayers (nB)</strong></td>
<td>void <strong>getMaxPlayers (nB)</strong></td>
</tr>
<tr>
<td><strong>void setIsOpen (bool isOpen, const</strong></td>
<td>void <strong>setIsOpen (bool isOpen, const</strong> <strong>WebFlags</strong></td>
</tr>
<tr>
<td><strong>&amp;webflags=WebFlags</strong></td>
<td><strong>void setIsOpen (bool isOpen, const</strong> <strong>WebFlags</strong></td>
</tr>
<tr>
<td><strong>bool getIsVisible (void)</strong></td>
<td><strong>void setIsVisible (bool isVisible, const</strong> <strong>WebFlags</strong></td>
</tr>
<tr>
<td><strong>void setIsVisible (bool isVisible,</strong></td>
<td><strong>void setIsVisible (bool isVisible, const</strong> <strong>WebFlags</strong></td>
</tr>
<tr>
<td><strong>&amp;webflags=WebFlags</strong></td>
<td><strong>const</strong> <strong>Common::JVector&lt; Player * &gt; &amp;getPlayers</strong></td>
</tr>
<tr>
<td><strong>const Common::JVector&lt; Player * &gt; &amp;</strong></td>
<td><strong>const</strong> <strong>Common::JVector&lt; Common::JString &gt; &amp;getPropsListedInLobby</strong></td>
</tr>
<tr>
<td><strong>getPlayerForNumber (int)</strong></td>
<td><strong>void setPropsListedInLobby (Common::JString, Const</strong></td>
</tr>
<tr>
<td><strong>void setPropsListedInLobby (Common::JString, Const</strong></td>
<td><strong>int getPlayerTtl (void)</strong></td>
</tr>
<tr>
<td><strong>void setPropsListedInLobby (Common::JString, Const</strong></td>
<td><strong>int getPlayerTtl (void)</strong></td>
</tr>
</tbody>
</table>
int getEmptyRoomTtl();

bool getSuppressRoom();

custom Common::JVector< Common::JString > * getPlugins (void) const;

bool getPublishUserID (void) const;

custom Common::JVector< Common::JString > & getExpectedUsers;

void setExpectedUsers (const Common::JString > &expectedUsers, WebFlags &webflags);

void mergeCustomProperties (Common::Hashtable, Common::Hashtable &expectedCustomProperties, const WebFlags &webflags);

template<typename ktype, typename vtype>
void addCustomProperty (pValueArray, const Common::Hashtable &expectedCustomProperties, const WebFlags &webflags);

template<typename ktype, typename vtype>
void addCustomProperty (pValueArray, typename Common::Helpers::ArrayLengthType<vtype>::type arrSize, const Common::Hashtable &expectedCustomProperties, const WebFlags &webflags);

template<typename ktype, typename vtype>
void addCustomProperty (pValueArray, const short *pArrSizes, const WebFlags &webflags);
Common::Hashtable

void addCustomProperties
&customProperties, 
&expectedCustomProperties=
const WebFlags &webflags=

void removeCustomProperty
Common::Hashtable
&expectedCustomProperties=
const WebFlags &webflags=

void removeCustomProperties
unsigned int count, 
&expectedCustomProperties=
const WebFlags &webflags=

template<typename ktype >

virtual Common::JString toString(bool withTypes=false, bool withCustomProperties=false, bool withPlayers=false) const

Public Member Functions inherited from Room
virtual ~Room (void)

Room (const Room &)

const Common::JString & getName (void) const

nByte getMaxPlayers (void) const

bool isOpen (void) const

nByte getDirectMode (void) const
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const Common::Hashtable &amp; getCustomProperties</code></td>
<td></td>
</tr>
<tr>
<td><code>bool operator== (const Room &amp; R)</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual Common::JString &amp; toString (Common::JString withTypes=false)</code> const</td>
<td></td>
</tr>
<tr>
<td><code>virtual Common::JString toString (bool withTypes, bool withCustomProperties)</code> const</td>
<td></td>
</tr>
</tbody>
</table>

- **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 withTypes)`
## 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 )

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  

Returns
the count of players that are currently inside this room

Reimplemented from Room.
 photon C++
Client API  4.1.12.2

Peer Class Reference

Inheritance diagram for Peer:

Collaboration diagram for Peer:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void disconnect()</td>
<td>Disconnects from the peer, typically when the user leaves the game.</td>
</tr>
<tr>
<td>bool opJoinLobby(const Common::JString&amp; name, LobbyType type=DEFAULT)</td>
<td>Joins a lobby with the specified name and type.</td>
</tr>
<tr>
<td>bool opLeaveLobby()</td>
<td>Leaves the current lobby.</td>
</tr>
<tr>
<td>bool opCreateRoom(const RoomOptions&amp; options, Common::Hashtable&amp; properties)</td>
<td>Creates a new room with the specified options and properties.</td>
</tr>
<tr>
<td>bool opJoinRoom(const RoomOptions&amp; options, Common::Hashtable&amp; properties, bool createIfNotExists=false, bool rejoin=false, int cacheSliceIndex=0, const Common::JVector&lt; Common::JString &gt;&amp; expectedUsers=Common::JVector&lt;&gt;()</td>
<td>Joins an existing room with the specified options and properties.</td>
</tr>
<tr>
<td>bool opJoinRandomRoom(const Common::Hashtable&amp; roomProperties, int maxPlayers=0, nByte matchmakingMode=MatchmakingMode::FILL_ROOM, const Common::JString&amp; lobbyName=Common::JString, LobbyType type=DEFAULT, Common::JString&amp; sqlLobbyFilter=Common::JString )</td>
<td>Joins a random room with the specified room properties and matchmaking options.</td>
</tr>
</tbody>
</table>
const Common::JVector< Common::JString >& expectedUsers = Common::JVector<>()

virtual bool opLeaveRoom (bool willComeBack = false, bool sendAuthCookie = false)

template<typename T>
bool opRaiseEvent (bool reliable = true, const T& parameters, nByte eventCode, const RaiseEventOptions& options = RaiseEventOptions())

template<typename T>
bool opRaiseEvent (bool reliable = true, const T& parameters, typename Common::Helpers::ArrayLengthType<T>::type arrSize, nByte eventCode, const RaiseEventOptions& options = RaiseEventOptions())

template<typename T>
bool opRaiseEvent (bool reliable = true, const T& parameters, const short *pArrSizes, nByte eventCode, const RaiseEventOptions& options = RaiseEventOptions())

virtual bool opAuthenticate (const Common::JString& appVersion, bool encrypted = true, const AuthenticationValues& authenticationValues = AuthenticationValues(), lobbyStats = false, const Common::JString& regionCode = Common::JString())

virtual bool opAuthenticateOnce (const Common::JString& appVersion, const Common::JString& regionCode, nByte connectionProtocol, nByte encryptionMode, const AuthenticationValues& authenticationValues = AuthenticationValues(), lobbyStats = false, const Common::JString& regionCode = Common::JString())

virtual bool opFindFriends (const Common::JString& appVersion, const Common::JString& regionCode)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual bool <code>opLobbyStats</code> (const <code>Common::LoadBalancing::LobbyStatsRequest</code> &amp;lobbiesToQuery=<code>Common::J::LoadBalancing::LobbyStatsRequest</code>)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <code>opChangeGroups</code> (const <code>Common::JVector&lt;Common::JVector&lt;Common::LoadBalancing::LobbyStatsRequest&gt;&gt;</code> *pGroupsToRemove, const <code>Common::JVector&lt;Common::JVector&lt;Common::LoadBalancing::LobbyStatsRequest&gt;&gt;</code> *pGroupsToAdd)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <code>opWebRpc</code> (const <code>Common::JString</code> &amp;parameters, bool sendAuthCookie=false)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt; <code>opWebRpc</code> (const <code>Common::JString</code> &amp;parameters, bool sendAuthCookie=false)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt; <code>opWebRpc</code> (const <code>Common::JString</code> pParameterArray, typename <code>Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type</code> arrSize, bool sendAuthCookie=false)</td>
<td></td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt; <code>opWebRpc</code> (const <code>Common::JString</code> pParameterArray, const short *pArrSizes, bool sendAuthCookie=false)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <code>opGetRegions</code> (bool encrypted, const <code>Common::Hashtable</code> &amp;appID)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <code>opSetPropertiesOfPlayer</code> (int playerNr, const <code>Common::Hashtable</code> &amp;properties, const <code>Common::Hashtable</code> &amp;expectedProperties=<code>Common::Hashtable()</code>)</td>
<td></td>
</tr>
<tr>
<td>virtual bool <code>opSetPropertiesOfRoom</code> (const <code>Common::Hashtable</code> &amp;properties, const <code>Common::Hashtable</code> webFlags=<code>WebFlags()</code>)</td>
<td></td>
</tr>
<tr>
<td>Public Member Functions inherited from PhotonPeer</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>PhotonPeer</strong> (PhotonListener connectionProtocol=ConnectionProtocol::DEFAULT)</td>
<td></td>
</tr>
<tr>
<td>virtual ~PhotonPeer (void)</td>
<td></td>
</tr>
</tbody>
</table>
| ```cpp
template<typename Ftype >
bool connect (const Common::JString &appID, const Ftype &customData) |
``` |
| ```cpp
template<typename Ftype >
bool connect (const Common::JString &appID, const Ftype &customDataArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize) |
``` |
| ```cpp
template<typename Ftype >
bool connect (const Common::JString &appID, const short *pArrSizes) |
``` |
| virtual void service (bool dispatchIncoming) |
| virtual void serviceBasic (void) |
| virtual bool opCustom (const OperationRequest &request, bool sendReliable, nByte channelID) |
| 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 resetTrafficStatsMaximumCounters

virtual Common::JString vitalStatsToString (bool all) const

virtual void pingServer (const Common::JString &pingAttempts)

virtual void initUserDataEncryption (const &secret)

virtual void initUDPEncryption (const Common::JVector &encryptSecret, const Common::JVector &HMACSecret)

PhotonListener * getListener (void)

int getServerTimeOffset (void) const

int getServerTime (void) const

int getBytesOut (void) const

int getBytesIn (void) const

int getByteCountCurrentDispatch

int getByteCountLastOperation (void)

int getPeerState (void) const
int getSentCountAllowance (void)

void setSentCountAllowance (int sentCountAllowance)

int getTimePingInterval (void) const

void setTimePingInterval (int timePingInterval)

int getRoundTripTime (void) const

int getRoundTripTimeVariance (void)

int getTimestampOfLastSocketReceive

int getDebugOutputLevel (void) const

bool setDebugOutputLevel (int debugLevel)

const Common::LogFormatOptions & getLogFormatOptions (void) const

void setLogFormatOptions (const Common::LogFormatOptions &)

int getIncomingReliableCommandsCount

short getPeerID (void) const

int getDisconnectTimeout (void) const

void setDisconnectTimeout (int disconnectTimeout)

int getQueuedIncomingCommands

int getQueuedOutgoingCommands
Common::JString getServerAddress (void) const

bool getIsPayloadEncryptionAvailable (void)

bool getIsEncryptionAvailable (void)

int getResentReliableCommands

int getLimitOfUnreliableCommands

void setLimitOfUnreliableCommands

bool getCRCEncabled (void) const

void setCRCEncabled (bool crcEnabled)

int getPacketLossByCRC (void) const

bool getTrafficStatsEnabled (void)

void setTrafficStatsEnabled (bool trafficStatsEnabled)

int getTrafficStatsElapsedMs (void)

const TrafficStats & getTrafficStatsIncoming (void)

const TrafficStats & getTrafficStatsOutgoing (void)

const TrafficStatsGameLevel & getTrafficStatsGameLevel (void)

nByte getQuickResendAttempts (void)

void setQuickResendAttempts (nByte quickResendAttempts)

nByte getConnectionProtocol (void)
void setConnectionProtocol (nByte)
nByte getChannelCountUserChannels
### 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 StatusCode::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`.

---

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
Photon C++
Client API 4.1.12.2

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

<table>
<thead>
<tr>
<th>Static Public Member Functions inherited from <code>Base</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>static void <code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
</tr>
<tr>
<td>static int <code>getDebugOutputLevel</code> (void)</td>
</tr>
<tr>
<td>static bool <code>setDebugOutputLevel</code> (int debugLevel)</td>
</tr>
<tr>
<td>static const <code>LogFormatOptions</code> &amp; <code>getLogFormatOptions</code> (void)</td>
</tr>
<tr>
<td>static void <code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
</tr>
</tbody>
</table>
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 )

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 )

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()

```cpp
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 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]**

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

**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 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
RaiseEventOptions
Class Reference

Inheritance diagram for RaiseEventOptions:

![Inheritance diagram]

[legend]

Collaboration diagram for RaiseEventOptions:

![Collaboration diagram]

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RaiseEventOptions</strong> (nByte channelID=0, nByte eventCaching=Lite::EventCache::DO_NOT_CACHE, const int *targetPlayers=NULL, short numTargetPlayers=0, nByte receiverGroup=Lite::ReceiverGroup::OTHERS, nByte interestGroup=0, const WebFlags &amp;webFlags=WebFlags(), int cacheSliceIndex=0)</td>
<td>Raise an event with specified options.</td>
</tr>
<tr>
<td>~RaiseEventOptions (void)</td>
<td>Destructor function.</td>
</tr>
<tr>
<td><strong>RaiseEventOptions</strong> (const RaiseEventOptions &amp;toCopy)</td>
<td>Copy constructor for RaiseEventOptions.</td>
</tr>
<tr>
<td>RaiseEventOptions &amp; operator= (const RaiseEventOptions &amp;toCopy)</td>
<td>Assignment operator for RaiseEventOptions.</td>
</tr>
<tr>
<td>nByte getChannelID (void) const</td>
<td>Get the channel ID.</td>
</tr>
<tr>
<td><strong>RaiseEventOptions</strong> &amp; setChannelID (nByte channelID)</td>
<td>Set the channel ID.</td>
</tr>
<tr>
<td>nByte getEventCaching (void) const</td>
<td>Get the event caching.</td>
</tr>
<tr>
<td><strong>RaiseEventOptions</strong> &amp; setEventCaching (nByte eventCaching)</td>
<td>Set the event caching.</td>
</tr>
<tr>
<td>const int * getTargetPlayers (void) const</td>
<td>Get the target players.</td>
</tr>
<tr>
<td>short getNumTargetPlayers (void) const</td>
<td>Get the number of target players.</td>
</tr>
<tr>
<td><strong>RaiseEventOptions</strong> &amp; setTargetPlayers (const int *targetPlayers, short numTargetPlayers)</td>
<td>Set the target players.</td>
</tr>
<tr>
<td>nByte getReceiverGroup (void) const</td>
<td>Get the receiver group.</td>
</tr>
<tr>
<td><strong>RaiseEventOptions</strong> &amp; setReceiverGroup (nByte receiverGroup)</td>
<td>Set the receiver group.</td>
</tr>
<tr>
<td>nByte getInterestGroup (void) const</td>
<td>Get the interest group.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>RaiseEventOptions &amp; setInterestGroup</td>
<td>(nByte interestGroup)</td>
</tr>
<tr>
<td>const WebFlags &amp; getWebFlags</td>
<td>(void) const</td>
</tr>
<tr>
<td>RaiseEventOptions &amp; setWebFlags</td>
<td>(const WebFlags &amp;webFlags)</td>
</tr>
<tr>
<td>int getCacheSliceIndex</td>
<td>(void) const</td>
</tr>
<tr>
<td>RaiseEventOptions &amp; setCacheSliceIndex</td>
<td>(int cacheSliceIndex)</td>
</tr>
<tr>
<td>virtual Common::JString &amp; toString</td>
<td>(Common::JString &amp;retStr, bool withTypes=false) const</td>
</tr>
</tbody>
</table>

- **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 | channelID = 0, channelID = 0,          |
|                   | eventCaching = Lite::EventCache::DO_NOT_CACHE, |
| nByte             | targetPlayers = NULL,                  |
| nByte             | numTargetPlayers = 0,                  |
| const int *       | receiverGroup = Lite::ReceiverGroup::OTHERS, |
| short             | interestGroup = 0,                     |
| nByte             | webFlags = WebFlags(),                |
| nByte             | cacheSliceIndex = 0                   |
| const WebFlags &  |                                       |
| int               |                                       |

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 `NULL`.

- **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 constructed `WebFlags` instance.

- **cacheSliceIndex**
  - see `setCacheSliceIndex()` - optional, defaults to 0.
~RaiseEventOptions()

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
getPlayerPositions()

See also
getPlayerPositions(), setPlayerPositions()
§ 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()

```c++
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:

[Diagram]

Collaboration diagram for Room:

[Diagram]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~Room (void)</td>
<td></td>
</tr>
<tr>
<td>Room (const Room &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>virtual Room &amp; operator= (const Room &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td>const Common::JString &amp; getName (void) const</td>
<td></td>
</tr>
<tr>
<td>virtual nByte getPlayerCount (void) const</td>
<td></td>
</tr>
<tr>
<td>nByte getMaxPlayers (void) const</td>
<td></td>
</tr>
<tr>
<td>bool getIsOpen (void) const</td>
<td></td>
</tr>
<tr>
<td>nByte getDirectMode (void) const</td>
<td></td>
</tr>
<tr>
<td>const Common::Hashtable &amp; getCustomProperties (void) const</td>
<td></td>
</tr>
<tr>
<td>bool operator==(const Room &amp;room) const</td>
<td></td>
</tr>
<tr>
<td>virtual Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</td>
<td></td>
</tr>
<tr>
<td>virtual Common::JString toString (bool withTypes, bool withCustomProperties) const</td>
<td></td>
</tr>
</tbody>
</table>

- **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

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Public Member Functions</td>
<td>static void setListener (const BaseListener* baseListener)</td>
<td>inherited from Base</td>
</tr>
<tr>
<td></td>
<td>static int getDebugOutputLevel (void)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>static bool setDebugOutputLevel (int debugLevel)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>static const LogFormatOptions &amp; getLogFormatOptions (void)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>static void setLogFormatOptions (const LogFormatOptions &amp;options)</td>
<td></td>
</tr>
</tbody>
</table>
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()`
§ ~Room()

~Room ( void ) virtual

Destructor.
§ Room()

<table>
<thead>
<tr>
<th><strong>Room</strong> ( const <strong>Room</strong> &amp; <strong>toCopy</strong> )</th>
</tr>
</thead>
</table>

Copy-Constructor: Creates a new instance that is a deep copy of the argument instance.

**Parameters**

**toCopy** The instance to copy.
operator=()

Room & operator= ( const Room & toCopy )

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 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 |

**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()

```c
bool getIsOpen ( void ) const
```

**Returns**

'false' if the room can be joined, 'true' 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==( )

```cpp
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.
$$\texttt{toString()} [1/2]$$

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

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`.
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
RoomOptions Class Reference

Inheritance diagram for RoomOptions:

```
ToString
  |
  v
Base
  |
  v
RoomOptions
```

[legend]

Collaboration diagram for RoomOptions:

```
ToString
  |
  v
Base
  |
  v
RoomOptions
```

[legend]
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoomOptions</td>
<td>(bool isVisible=true, bool isOpen=true, nByte maxPlayers=0, const Common::Hashtable &amp;customRoomProperties=, const Common::JVector &amp;propsListedInLobby=, const Common::JString &amp;lobbyName=, LobbyType lobbyType=DEFAULT, int playerTtl=0, int empt suppressRoomEvents=false, const Common::JVector&lt; *pPlugins=NULL, bool publishUserID=false, nByte directMode=DirectMode::NONE)</td>
</tr>
<tr>
<td>~RoomOptions</td>
<td>(void)</td>
</tr>
<tr>
<td>RoomOptions</td>
<td>(const RoomOptions &amp;)</td>
</tr>
<tr>
<td>RoomOptions &amp; operator=</td>
<td>(const RoomOptions &amp;)</td>
</tr>
<tr>
<td>bool isVisible</td>
<td>(void) const</td>
</tr>
<tr>
<td>RoomOptions &amp; setVisible</td>
<td>(bool isVisible)</td>
</tr>
<tr>
<td>bool isOpen</td>
<td>(void) const</td>
</tr>
<tr>
<td>RoomOptions &amp; setOpen</td>
<td>(bool isOpen)</td>
</tr>
<tr>
<td>nByte getMaxPlayers</td>
<td>(void) const</td>
</tr>
<tr>
<td>RoomOptions &amp; setMaxPlayers</td>
<td>(nByte) const</td>
</tr>
<tr>
<td>const Common::Hashtable &amp; getCustomRoomProps</td>
<td>(const Common::Hashtable &amp;)</td>
</tr>
</tbody>
</table>
RoomOptions & setCustomRoomProperties

const Common::JVector< Common::JString > & getPropsListedInLobby

RoomOptions & setPropsListedInLobby

const Common::JString & getLobbyName (void)

RoomOptions & setLobbyName (const &lobbyName)

nByte getLobbyType (void)

RoomOptions & setLobbyType (nByte lobbyType)

int getPlayerTtl (void)

RoomOptions & setPlayerTtl (int playerTtl)

int getEmptyRoomTtl ()

RoomOptions & setEmptyRoomTtl (

bool getSuppressRoomEvents

RoomOptions & setSuppressRoomEvents suppressionRoomEvent

const Common::JVector< Common::JString > * getPlugins (void) const

RoomOptions & setPlugins (const Common::JString >

bool getPublishUserID (
RoomOptions & setPublishUserID (l)

nByte getDirectMode (void)

RoomOptions & setDirectMode (nBy)

virtual Common::JString & toString (Common::JString 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)

JString toString (bool withTypes=false)
## 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 isVisible, bool isOpen, nByte maxPlayers, const Common::Hashtable & customRoomProperties, const Common::JVector< Common::JString > & propsListedInLobby, const Common::JString & lobbyName, nByte lobbyType, int playerTtl, int emptyRoomTtl, bool suppressRoomEvents, const Common::JVector< Common::JString > * pPlugins, bool publishUserID, nByte directMode )

Constructor: Creates a new instance with the specified parameters.

Parameters

isVisible see setIsVisible() - optional, defaults to true.
isOpen see setIsOpen() - optional, defaults to true.
maxPlayers see setMaxPlayers() - optional, defaults to 0.
customRoomProperties see setCustomRoomProperties() - optional, defaults to an empty Hashtable instance.
propsListedInLobby see setPropsListedInLobby() - optional, defaults to an empty JVector instance.
lobbyName see setLobbyName() - optional, defaults to an empty JString instance.
lobbyType see setLobbyType() - optional, defaults to 0.
playerTtl see setPlayerTtl() - optional, defaults to 0.
emptyRoomTtl see setEmptyRoomTtl() - optional, defaults to 0.
suppressRoomEvents see setSuppressRoomEvents() - optional, defaults to false.
pPlugins see setPlugins() - optional, defaults to NULL.
publishUserID see setPublishUserID() - optional, defaults to false.
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.
§ 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()

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()

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 Datatypes.

Remarks

This function sets the initial custom properties that are used for room creation. To change the custom properties of an already existing room, see
ModelAttribute::mergeCustomProperties(), Model::addCustomProperties(),
ModelAttribute::addCustomProperties(),
ModelAttribute::removeCustomProperty() and
ModelAttribute::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 chaini

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 in the lobby. List the keys of all the custom room properties that should be available to the lobby. Use with care. Unless a custom property is essential for matchmaking, it should not be sent to the lobby, which causes traffic and delays for clients. 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 lot of data is quickly added up.

Remarks
This function sets the initial list of property keys. To change which properties are shown to 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()**

```c
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()

```c
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()

```c
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 (1 minute) 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()`

```cpp
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 > * pPlugins)

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 setting 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()**

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

**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**.
WebFlags Class Reference

Inheritance diagram for WebFlags:

Collaboration diagram for WebFlags:
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WebFlags (nByte webFlags=0)</code></td>
<td></td>
</tr>
<tr>
<td><code>nByte getFlags (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>WebFlags &amp; setFlags (nByte webFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>bool getHttpForward (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>WebFlags &amp; setHttpForward (bool httpWebForward)</code></td>
<td></td>
</tr>
<tr>
<td><code>bool getSendAuthCookie (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>WebFlags &amp; setSendAuthCookie (bool sendAuthCookie)</code></td>
<td></td>
</tr>
<tr>
<td><code>bool getSendSync (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>WebFlags &amp; setSendSync (bool sendSync)</code></td>
<td></td>
</tr>
<tr>
<td><code>bool getSendState (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>WebFlags &amp; setSendState (bool sendState)</code></td>
<td></td>
</tr>
<tr>
<td>virtual <code>Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td></td>
</tr>
</tbody>
</table>

- **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`
§ WebFlags()

WebFlags ( nByte  webFlags = 0 )

Constructor: Creates a new instance with the specified parameters.

Parameters
  webFlags see setFlags() - optional, defaults to 0.
§ 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()

```c
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()](#)
§ getTimeSync()

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()

```c++
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**.
Inheritance diagram for Protocol:

Collaboration diagram for Protocol:
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>const unsigned short</td>
<td>MASTER</td>
</tr>
<tr>
<td>const unsigned short</td>
<td>GAME</td>
</tr>
<tr>
<td>const unsigned short</td>
<td>NAME</td>
</tr>
</tbody>
</table>
TCP Struct Reference

Inheritance diagram for TCP:

Collaboration diagram for TCP:
**Static Public Attributes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const unsigned short</td>
<td>MASTER</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>GAME</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>NAME</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

- **Public Attributes inherited from Protocol**
  - `const unsigned short MASTER`
  - `const unsigned short GAME`
  - `const unsigned short NAME`
UDP Struct Reference

Inheritance diagram for UDP:

Collaboration diagram for UDP:
### Static Public Attributes

<table>
<thead>
<tr>
<th>static const unsigned short</th>
<th>MASTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const unsigned short</td>
<td>GAME</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>NAME</td>
</tr>
</tbody>
</table>
Additional Inherited Members

- Public Attributes inherited from Protocol
  - const unsigned short MASTER
  - const unsigned short GAME
  - const unsigned short NAME
### UDPAlternative Struct Reference

#### Inheritance diagram for UDPAlternative:

```
Protocol
    
UDPAlternative
```

#### Collaboration diagram for UDPAlternative:

```
short

MASTER GAME NAME

Protocol

UDPAlternative
```

[legend]
### Static Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const unsigned short</td>
<td>NAME</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>MASTER</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>GAME</td>
</tr>
</tbody>
</table>
## Additional Inherited Members

- **Public Attributes inherited from Protocol**
  - `const unsigned short MASTER`
  - `const unsigned short GAME`
  - `const unsigned short NAME`
WS Struct Reference

Inheritance diagram for WS:

Collaboration diagram for WS:
### Static Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const unsigned short</td>
<td>MASTER</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>GAME</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>NAME</td>
</tr>
</tbody>
</table>
Additional Inherited Members

- Public Attributes inherited from **Protocol**
  - const unsigned short  **MASTER**
  - const unsigned short  **GAME**
  - const unsigned short  **NAME**
Photon C++
Client API 4.1.12.2

WSS Struct Reference

Inheritance diagram for WSS:

Collaboration diagram for WSS:
## Static Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const unsigned short</td>
<td>MASTER</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>GAME</td>
</tr>
<tr>
<td>static const unsigned short</td>
<td>NAME</td>
</tr>
</tbody>
</table>
Additional Inherited Members

- Public Attributes inherited from Protocol
  - const unsigned short MASTER
  - const unsigned short GAME
  - const unsigned short NAME
Puncher Class Reference
**Public Member Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Puncher</strong></td>
<td>(RelayClient *pRelayClient, const Common::Logger &amp;logger)</td>
</tr>
<tr>
<td>bool <strong>init</strong></td>
<td>(PunchListener *pPunchListener)</td>
</tr>
<tr>
<td>void <strong>clear</strong></td>
<td>(void)</td>
</tr>
<tr>
<td>bool <strong>startPunch</strong></td>
<td>(int remoteID)</td>
</tr>
<tr>
<td>bool <strong>sendDirect</strong></td>
<td>(const Common::JVector&lt;nByte&gt; &amp;buffer, int targetID, bool fallbackRelay)</td>
</tr>
<tr>
<td>int <strong>sendDirect</strong></td>
<td>(const Common::JVector&lt;nByte&gt; &amp;buffer, const Common::JVector&lt;int&gt; &amp;targetIDs, bool fallbackRelay)</td>
</tr>
<tr>
<td>bool <strong>processPackage</strong></td>
<td>(const Common::JVector&lt;nByte&gt; &amp;packet, bool relay, int relayRemoteID)</td>
</tr>
<tr>
<td>void <strong>service</strong></td>
<td>(void)</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
Inheritance diagram for PunchListener:
### Public Member Functions

```cpp
virtual void onReceiveDirect (const Common::JVector< nByte > &inBuf, int remoteID, bool relay)=0
```
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
## EventData Class Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Photon</th>
<th>EventData</th>
</tr>
</thead>
</table>

Public Member Functions | List of all members
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~EventData</td>
<td>Destructor</td>
</tr>
<tr>
<td>EventData</td>
<td>Constructor</td>
</tr>
<tr>
<td>EventData &amp; operator= EventData</td>
<td>Assignment operator</td>
</tr>
<tr>
<td>const Common::Object &amp; operator[]</td>
<td>Access operator</td>
</tr>
<tr>
<td>Common::JString toString</td>
<td>Converts object to a string</td>
</tr>
<tr>
<td>Common::Object getPassword</td>
<td>Gets parameter by code</td>
</tr>
<tr>
<td>nByte getCode</td>
<td>Gets code</td>
</tr>
<tr>
<td>const Common::Dictionary&lt; nByte, Common::Object &gt; &amp; getPassword</td>
<td>Gets parameter dictionary by code</td>
</tr>
</tbody>
</table>
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[]( )

```cpp
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()`

```csharp
JString toString ( bool withParameters = false,
                  bool withParameterTypes = false )
```

**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.
## OperationRequest

Class Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Photon</th>
<th>OperationRequest</th>
</tr>
</thead>
</table>

Public Member Functions | List of all members
## Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OperationRequest (nByte operationCode, const OperationRequestParameters &amp;parameters=OperationRequestParameters)</code></td>
<td></td>
</tr>
<tr>
<td><code>~OperationRequest (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationRequest (const OperationRequest &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationRequest &amp; operator= (const OperationRequest &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>const Common::Object &amp; operator[] (unsigned int index)</code></td>
<td></td>
</tr>
<tr>
<td><code>Common::JString toString (bool withParameters=false, bool withParameterTypes=false)</code></td>
<td></td>
</tr>
<tr>
<td><code>Common::Object getParameterForCode (nByte const)</code></td>
<td></td>
</tr>
<tr>
<td><code>nByte getOperationCode (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>const OperationRequestParameters &amp; getParameters (void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationRequestParameters &amp; getParameters (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>void setParameters (const OperationRequestParameters)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

This is a container for an Operation request, which consists of a code and parameters.
Constructor & Destructor Documentation
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.
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()

```cpp
void setParameters ( const OperationRequestParameters & parameters )
```

**Parameters**

`parameters` Sets the payload of the operation.
### OperationResponse Class Reference

<table>
<thead>
<tr>
<th>ExitGames</th>
<th>Photon</th>
<th>OperationResponse</th>
<th>Public Member Functions</th>
<th>List of all members</th>
</tr>
</thead>
</table>

**Photon C++ Client API 4.1.12.2**
<table>
<thead>
<tr>
<th>Public Member Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>~OperationResponse (void)</td>
</tr>
<tr>
<td>OperationResponse (const OperationResponse &amp;toCopy)</td>
</tr>
<tr>
<td>OperationResponse &amp; operator=(const OperationResponse &amp;toCopy)</td>
</tr>
<tr>
<td>const Common::Object &amp; operator[](unsigned int index) const</td>
</tr>
<tr>
<td>Common::JString toString (bool withDebugMessage=false, bool withParameters=false, bool withParameterTypes=false) const</td>
</tr>
<tr>
<td>Common::Object getParameterForCode (nByte parameterCode) const</td>
</tr>
<tr>
<td>nByte getOperationCode const</td>
</tr>
<tr>
<td>short getReturnCode const</td>
</tr>
<tr>
<td>const Common::JString &amp; getDebugMessage const</td>
</tr>
</tbody>
</table>
const Common::Dictionary< nByte, Common::Object > & getParameter
const
Detailed Description

Contains the servers response for an OperationRequest sent by this client.
Constructor & Destructor Documentation
<table>
<thead>
<tr>
<th>~OperationResponse()</th>
</tr>
</thead>
</table>

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()

```cpp
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.
Inheritance diagram for PhotonListener:

```
BaseListener

PhotonListener

Client
```

[legend]

Collaboration diagram for PhotonListener:

```
BaseListener

PhotonListener
```

[legend]
## Public Member Functions

<table>
<thead>
<tr>
<th>virtual</th>
<th><code>~PhotonListener</code> (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual void</td>
<td><code>onOperationResponse</code> (const OperationResponse &amp;operationResponse)=0</td>
</tr>
<tr>
<td>virtual void</td>
<td><code>onStatusChanged</code> (int statusCode)=0</td>
</tr>
<tr>
<td>virtual void</td>
<td><code>onEvent</code> (const EventData &amp;eventData)=0</td>
</tr>
<tr>
<td>virtual void</td>
<td><code>onPingResponse</code> (const Common::JString &amp;address, unsigned int pingResult)</td>
</tr>
</tbody>
</table>

- **Public Member Functions inherited from** `BaseListener`
  | virtual void | `debugReturn` (int debugLevel, const JString &string)=0 |
Constructor & Destructor Documentation
§ ~PhotonListener()

virtual ~PhotonListener ( void )

Destructor.
Member Function Documentation
§ onOperationResponse()

virtual void
onOperationResponse ( const OperationResponse & operationResponse )

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 an operation is identified by an operation code.

An operation's response is summarized by the return code: an int typed code: OK or some error code defined by the application, which is defining the operation itself. The opCode defines the type of operation called on Photon and in turn also the return values. They are provided as a Hashtable which contains the complete response of Photon, including keys for operation code and return code. Each operation returns its opCode and returnCode but anything else can be defined serverside.

Parameters

operationResponse the OperationResponse
§ onStatusChanged()

```cpp
virtual void onStatusChanged ( int statusCode )
```

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 `getStatusCode.h` for a list.

**Parameters**

`statusCode` see description
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 )

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

PhotonPeer Class Reference

Inheritance diagram for PhotonPeer:

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhotonPeer</td>
<td>(PhotonListener connectionProtocol=ConnectionProtocol::DEFAULT)</td>
</tr>
<tr>
<td>virtual</td>
<td>~PhotonPeer (void)</td>
</tr>
<tr>
<td>virtual bool</td>
<td>connect (const Common::JString &amp;appID=Common::JString)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool connect (const Common::JString &amp;appID, const Ftype &amp;customData)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool connect (const Common::JString &amp;appID, const Ftype pCustomDataArray, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</td>
</tr>
<tr>
<td>template&lt;typename Ftype&gt;</td>
<td>bool connect (const Common::JString &amp;appID, const Ftype pCustomDataArray, const short *pArrSizes)</td>
</tr>
<tr>
<td>virtual void</td>
<td>disconnect (void)</td>
</tr>
<tr>
<td>virtual void</td>
<td>service (bool dispatchIncoming)</td>
</tr>
<tr>
<td>virtual void</td>
<td>serviceBasic (void)</td>
</tr>
<tr>
<td>virtual bool</td>
<td>opCustom (const OperationRequest &amp;operationRequest, bool sendReliable, nByte channelId=0, bool encrypt=false)</td>
</tr>
<tr>
<td>Function Calling</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>virtual bool</td>
<td>sendOutgoingCommands (void)</td>
</tr>
<tr>
<td>virtual bool</td>
<td>sendAcksOnly (void)</td>
</tr>
<tr>
<td>virtual bool</td>
<td>dispatchIncomingCommands</td>
</tr>
<tr>
<td>virtual bool</td>
<td>establishEncryption (void)</td>
</tr>
<tr>
<td>virtual void</td>
<td>fetchServerTimestamp (void)</td>
</tr>
<tr>
<td>virtual void</td>
<td>resetTrafficStats (void)</td>
</tr>
<tr>
<td>virtual void</td>
<td>resetTrafficStatsMaximumCounters</td>
</tr>
<tr>
<td>virtual Common::JString</td>
<td>vitalStatsToString (bool all) const</td>
</tr>
<tr>
<td>virtual void</td>
<td>pingServer (const Common::JString &amp; pingAttempts)</td>
</tr>
<tr>
<td>virtual void</td>
<td>initUserDataEncryption (const Common::JVector&lt;nByte&gt; &amp; secret)</td>
</tr>
<tr>
<td>virtual void</td>
<td>initUDPEncryption (const Common::JVector&lt;nByte&gt; &amp; encryptSecret, const Common::JVector&lt;nByte&gt; &amp; HMACSecret)</td>
</tr>
<tr>
<td>PhotonListener *</td>
<td>getListener (void)</td>
</tr>
<tr>
<td>int</td>
<td>getServerTimeOffset (void) const</td>
</tr>
<tr>
<td>int</td>
<td>getServerTime (void) const</td>
</tr>
<tr>
<td>int</td>
<td>getBytesOut (void) const</td>
</tr>
<tr>
<td>int</td>
<td>getBytesIn (void) const</td>
</tr>
<tr>
<td>int</td>
<td>getByteCountCurrentDispatch</td>
</tr>
</tbody>
</table>
int getByteCountLastOperation ( )

int getPeerState ( void ) const

int getSentCountAllowance ( void )

void setSentCountAllowance ( int sentCountAllowance )

int getTimePingInterval ( void ) const

void setTimePingInterval ( int timePingInterval )

int getRoundTripTime ( void ) const

int getRoundTripTimeVariance ( )

int getTimestampOfLastSocketReceive

int getDebugOutputLevel ( void ) const

bool setDebugOutputLevel ( int debugLevel )

const Common::LogFormatOptions & getLogFormatOptions ( void ) const

void setLogFormatOptions ( const Common::LogFormatOptions & )

int getIncomingReliableCommandsCount

const

short getPeerID ( void ) const

int getDisconnectTimeout ( void )

void setDisconnectTimeout ( int disconnectTimeout )
int getQueuedIncomingCommands
int getQueuedOutgoingCommands

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 trafficStatsEnabled)

int getTrafficStatsElapsedMs (void)

const TrafficStats & getTrafficStatsIncoming (void)
const TrafficStats & getTrafficStatsOutgoing (void)
const TrafficStatsGameLevel & getTrafficStatsGameLevel (void)

nByte getQuickResendAttempts (void)
void setQuickResendAttempts (nB)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quickResendAttempts()</td>
<td></td>
</tr>
<tr>
<td>nByte getConnectionProtocol (void)</td>
<td></td>
</tr>
<tr>
<td>void setConnectionProtocol (nByte connectionProtocol)</td>
<td></td>
</tr>
<tr>
<td>nByte getChannelCountUserChannels</td>
<td></td>
</tr>
</tbody>
</table>
## Static Public Member Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Name</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>static short</td>
<td><code>getPeerCount</code></td>
<td>(void)</td>
</tr>
<tr>
<td>static unsigned int</td>
<td><code>getMaxAppIDLength</code></td>
<td>(void)</td>
</tr>
</tbody>
</table>
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()

Constructor.

Parameters

- **listener**  
  Reference to the application's implementation of the PhotonPeer instance, which is valid for the lifetime of the PhotonPeer instance, which is created by this constructor.

- **connectionProtocol**  
  The protocol to use to connect to Photon. Must be one of the constants specified in ConnectionProtocol.

See also  
PhotonListener, ConnectionProtocol
~PhotonPeer()

Destructor.
§ connect() [1/4]

```cpp
bool connect ( const Common::JString & ipAddr,
               const Common::JString & appID = Common::JString() )
```

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]

```cpp
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]

```cpp
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 the above function only in what argument(s) it accepts.

This overload accepts singledimensional arrays and NULL-pointers passed for 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 format of "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 - has to be provided in the form of a 1D array of one of the supported data types, specified at [Table of Datatypes](#).

- **arrSize**
  - the element count of the customData array
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()

```c
void disconnect ( void )
```

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 status code of `StatusCode::DISCONNECT`.

**See also**

`connect()`

Reimplemented in `Peer`. 
§ service()

```cpp
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 =
  ```cpp
dispatchIncomingCommands()
  ```
will be called; false =
\texttt{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()

```cpp
bool opCustom ( const OperationRequest & operationRequest,
    bool sendReliable,
    nByte channelID = 0,
    bool encrypt = false )
```

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 )

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/or 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 )

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 explictly , when you use `sendAcksOnly()` and `dispatchIncomingCommands()` instead.
§ dispatchIncomingCommands()

```cpp
bool dispatchIncomingCommands ( void )
```

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 )

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()

```cpp
void fetchServerTimestamp ( void )
```

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()

```c
void resetTrafficStats ( void )
```

Creates new instances of `TrafficStats` and starts a new timer for those.
§ resetTrafficStatsMaximumCounters()

void resetTrafficStatsMaximumCounters ( void )

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 )

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()

```c
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 rece through an unsecured connection.

**Remarks**

If you don't already have generated a key that you can access secur 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()

```c
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 or 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()

```c
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()

```c
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()

```cpp
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()**

```c
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()

```c
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()

```c
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()

```cpp
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()

```cpp
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()
void setLogFormatOptions ( const Common::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()*
§ 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()`

```c
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()

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

**Returns**

the number of queued outgoing commands in all channels or -1 if not connected
§ getServerAddress()

`JString getServerAddress ( void ) const`

**Returns**
the IP or url of the server, to which the peer is connected to
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()

```cpp
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()

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

**Returns**
true if CRC enabled

**See also**
setCRCEnabled
§ setCRCEnabled()

```c
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()

```c
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()

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

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

```c
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()

```cpp
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
RoundTripTime+4*RoundTripTimeVariance).
§ setQuickResendAttempts()

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

Sets the number of resend attempts for a reliable command can be done in quick succession (after RoundTripTime+4*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.
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()

```c
short getPeerCount ( void )
```

**Returns**
the count of peers, which have been initialized since the start of
the application. Interesting mainly for debugging purposes.
§ getMaxAppIDLength()

```c
unsigned int getMaxAppIDLength ( void )
```

**Returns**

the maximum allowed length for the appID that gets passed to `connect()` in characters
TrafficStats Class Reference

Inheritance diagram for TrafficStats:

Collaboration diagram for TrafficStats:
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~TrafficStats (void)</td>
</tr>
<tr>
<td>int getPackageHeaderSize (void) const</td>
</tr>
<tr>
<td>int getReliableCommandCount (void) const</td>
</tr>
<tr>
<td>int getUnreliableCommandCount (void) const</td>
</tr>
<tr>
<td>int getFragmentCommandCount (void) const</td>
</tr>
<tr>
<td>int getControlCommandCount (void) const</td>
</tr>
<tr>
<td>int getTotalPacketCount (void) const</td>
</tr>
<tr>
<td>int getTotalCommandsInPackets (void) const</td>
</tr>
<tr>
<td>int getReliableCommandBytes (void) const</td>
</tr>
<tr>
<td>int getUnreliableCommandBytes (void) const</td>
</tr>
<tr>
<td>int getFragmentCommandBytes (void) const</td>
</tr>
<tr>
<td>int getControlCommandBytes (void) const</td>
</tr>
<tr>
<td>int getTotalCommandCount (void) const</td>
</tr>
<tr>
<td>int getTotalCommandBytes (void) const</td>
</tr>
<tr>
<td>int getTotalPacketBytes (void) const</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td><code>int getTimestampOfLastAck (void) const</code></td>
</tr>
<tr>
<td><code>int getTimestampOfLastReliableCommand (void) const</code></td>
</tr>
<tr>
<td><code>virtual Common::JString &amp; toString (Common::JString &amp;retStr, bool withTypes=false) const</code></td>
</tr>
</tbody>
</table>

- **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 )

Destructor.
Member Function Documentation
§ getPackageHeaderSize()

```c
int getPackageHeaderSize ( void ) const
```

**Returns**
the byte-size of per-package headers.
§ getReliableCommandCount()

```c
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()

```cpp
int getUnreliableCommandCount ( void ) const
```

**Returns**

the unreliable commands that are created/received by this client.
§ getFragmentCommandCount()

```c
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 fragments for fragmented commands that are created/received by this client.
§ getControlCommandCount()

```c
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()

```c
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()

```c
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()

```c
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()`

```c
int getTotalCommandBytes ( void ) const
```

**Returns**

the sum of the return values of `getReliableCommandBytes()`, `getUnreliableCommandBytes()`, `getFragmentCommandBytes()` and `getControlCommandBytes()`
§ getTotalPacketBytes()

```c
int getTotalPacketBytes ( void ) const
```

**Returns**

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

```cpp
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()

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

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:

[legend]

Collaboration diagram for TrafficStatsGameLevel:

[legend]
### Public Member Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual ~TrafficStatsGameLevel</td>
<td>(void)</td>
</tr>
<tr>
<td>int getOperationByteCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getOperationCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getResultByteCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getResultCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getEventByteCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getEventCount</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getLongestOpResponseCallback</td>
<td>(void) const</td>
</tr>
<tr>
<td>nByte getLongestOpResponseCallbackOpCode</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getLongestEventCallback</td>
<td>(void) const</td>
</tr>
<tr>
<td>nByte getLongestEventCallbackCode</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getLongestDeltaBetweenDispatching</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getLongestDeltaBetweenSending</td>
<td>(void) const</td>
</tr>
<tr>
<td>int getDispatchIncomingCommandsCalls</td>
<td>(void) const</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static void setListener (const BaseListener *baseListener)</code></td>
<td>Inherited from Base</td>
</tr>
<tr>
<td><code>static int getDebugOutputLevel (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>static bool setDebugOutputLevel (int debugLevel)</code></td>
<td></td>
</tr>
<tr>
<td><code>static const LogFormatOptions &amp; getLogFormatOptions (void)</code></td>
<td></td>
</tr>
<tr>
<td><code>static void setLogFormatOptions (const LogFormatOptions &amp;options)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

This class provides game level traffic statistics.

See also

 PhotonPeer::getTrafficStatsEnabled(), PhotonPeer::setTrafficStatsEnabled(), PhotonPeer::getTrafficStatsElapsedMs(), PhotonPeer::getTrafficStatsGameLevel()
§ ~TrafficStatsGameLevel()

~TrafficStatsGameLevel ( void )

destructor.
Member Function Documentation
§ **getOperationByteCount()**

```c
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()

```c
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.
§昃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()

```c
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()**

```cpp
int getSendOutgoingCommandsCalls ( void ) const
```

**Returns**

the number of calls of `PhotonPeer::sendOutgoingCommands()`.
§ getHeight()

int getHeight ( void ) const

**Returns**

the sum of byte-cost of all "logic level" messages.
§ getTotalMessageCount()

```c
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()

```cpp
int getTotalOutgoingByteCount ( void ) const
```

**Returns**

the sum of byte-cost of all outgoing "logic level" messages (= OperationByteCount).
§ `getTotalOutgoingMessageCount()`

```c
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()

```cpp
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

**Returns**

a JString representation of the vital stats for debugging purposes.
Photon C++
Client API 4.1.12.2

Class Index

| a | b | c | d | e | f | h | j | k | l | m | o | p | r | s | t | u | v | w |
| AllocatorInterface (ExitGames::Common::MemoryManagement) | ANSIString (ExitGames::Common) | AuthenticationValues (ExitGames::LoadBalancing) | AuthenticationValues (ExitGames::Chat) |
| Base (ExitGames::Common) | BaseCharString (ExitGames::Common) | BaseListener (ExitGames::Common) |
| Channel (ExitGames::Chat) | Client (ExitGames::LoadBalancing) | Client (ExitGames::Chat) |
| CustomType (ExitGames::Common) | CustomTypeBase (ExitGames::Common) | CustomTypeFactory (ExitGames::Common) |
| DeSerializer (ExitGames::Common) | | | | |
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:

- AllocatorInterface
- BaseListener
  - Listener
  - Listener
- PhotonListener
  - Client
- Client
  - Client
- EventData
- OperationRequest
- OperationResponse
- PhotonPeer
  - Peer
  - LitePeer
  - Peer
- Protocol
  - TCP
  - UDP
  - UDPAlternative
  - WS
  - WSS
MutablePlayer
RaiseEventOptions
Room
MutableRoom
RoomOptions
WebFlags
TrafficStats
TrafficStatsGameLevel
JVector< Common::ExitGames::Common::JString >
JVector< Common::ExitGames::Common::JVector< unsigned int > >
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
Here is a list of all documented class members with links to the class documentation for each member:

- a -

  - `addElement() : JVector< Etype >`
  - `addElement() : JVector< Etype >`
  - `alloc() : AllocatorInterface`
  - `ANSIRepresentation() : JString`
  - `ANSIString() : ANSIString`
  - `AuthenticationValues() : AuthenticationValues`
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
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
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**
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**
Here is a list of all documented class members with links to the class documentation for each member:

- f -

- `fetchServerTimestamp()` : [Client](#), [PhotonPeer](#)
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
- getElementType() : 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
- 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`
- `getResultReturnCode()` : `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`
- `getTrafficStatsElapsedTime()`: `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`
Here is a list of all documented class members with links to the class documentation for each member:

- h -

- Hashtable() : [Hashtable](#)
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 >`
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 >](#)
Here is a list of all documented class members with links to the class documentation for each member:

- k -

- KeyObject() : `KeyObject< Etype >`
Here is a list of all documented class members with links to the class documentation for each member:

- I -

- lastIndexOf() : JString
- length() : BaseCharString, JString
- LitePeer() : LitePeer
- LobbyStatsRequest() : LobbyStatsRequest
- log() : Logger
- Logger() : Logger
Here is a list of all documented class members with links to the class documentation for each member:

- o -

- 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<=() : EGTime, JString
• operator=() : ANSIString, Dictionary< EKeyType, EValue_Type >, DictionaryBase, EGTime, Hashtable, JString, JVector< Etype >, KeyObject< Etype >, Object, UTF8String, ValueObject< Etype >, MutablePlayer, MutableRoom, Player, RaiseEventOptions, Room, RoomOptions, EventData, OperationRequest, OperationResponse
• operator==() : Dictionary< EKeyType, EValue_Type >, DictionaryBase, EGTime, Hashtable, JString, JVector< Etype >, Object, Player, Room
• operator>() : EGTime, JString
• operator>=() : EGTime, JString
• operator[]() : Dictionary< EKeyType, EValue_Type >, 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
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
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
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`
- `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`
Here is a list of all documented class members with links to the class documentation for each member:

- t -

- toInt() : `JString`
- toLowerCase() : `JString`
- toStringVitalStats() : `TrafficStatsGameLevel`
- toUpperCase() : `JString`
- trim() : `JString`
- trimToSize() : `JVector< Etype >`
- TypeCode : `CustomType< typeCode >`
- typeToString() : `Dictionary< EKeyType, EValueType >, DictionaryBase, ToString`
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
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](#)
Here is a list of all documented class members with links to the class documentation for each member:

- w -

  - WebFlags() : [WebFlags](#)
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 >
- a -

- addElement() : `JVector< Etype >`
- addElements() : `JVector< Etype >`
- alloc() : `AllocatorInterface`
- ANSIRepresentation() : `JString`
- ANSIString() : `ANSIString`
- AuthenticationValues() : `AuthenticationValues`
- b -

- BaseCharString() : BaseCharString
- 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**
- 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
- e -

- EGT\text{ime}() : \text{{\texttt{EGTime}}}  
- \text{{\texttt{endsWith}}}() : \text{{\texttt{JString}}}  
- \text{{\texttt{ensureCapacity}}}() : \text{{\texttt{JString, JVector< Etype >}}}  
- \text{{\texttt{equals}}}() : \text{{\texttt{JString}}}  
- \text{{\texttt{equalsIgnoreCase}}}() : \text{{\texttt{JString}}}  
- \text{{\texttt{establishEncryption}}}() : \text{{\texttt{PhotonPeer}}}  
- \text{{\texttt{EventData}}}() : \text{{\texttt{EventData}}}
- f -

- fetchServerTimestamp() : Client, PhotonPeer
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 >
- getDataTypeCopy() : KeyObject< Etype >, ValueObject< Etype >
- getDebugMessage() : OperationResponse
- getDebugOutputLevel() : Client, Base, Logger, Client, PhotonPeer
- getDimensions() : Object
- getDirectMode() : Room, RoomOptions
- getDisconnectedCause() : Client
- getDisconnectTimeout() : Client, PhotonPeer
- getDispatchIncomingCommandsCalls() : TrafficStatsGameLevel
- getElementType() : 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
- getIndex() : JVector< Etype >
- getInterestGroup() : RaiseEventOptions
- getIsEmpty() : JVector< Etype >
- getIsEncryptionAvailable() : Client, PhotonPeer
- getIsInactive() : Player
- getIsGameRoom() : Client
- getIsLobby() : 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
- 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`
- `getTimePingInterval()` : `Client, PhotonPeer`
- `getTimestampOfLastAck()` : `TrafficStats`
getTimestampOfLastReliableCommand() : TrafficStats
getTimestampOfLastSocketReceive() : Client, PhotonPeer
getTotalByteCount() : TrafficStatsGameLevel
getTotalCommandBytes() : TrafficStats
getTotalCommandCount() : TrafficStats
getTotalCommandsInPackets() : TrafficStats
getTotalIncomingByteCount() : TrafficStatsGameLevel
getTotalIncomingMessageCount() : 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, Player
getValue() : Dictionary< EKeyType, EValueType >, DictionaryBase, Hashtable
getValueDimensions() : Dictionary< EKeyType, EValueType >, DictionaryBase
getValueSizes() : DictionaryBase
getValueTypes() : Dictionary< EKeyType, EValueType >, DictionaryBase
getWebFlags() : RaiseEventOptions
- h -

- Hashtable() : Hashtable
- i -

- indexOf() : `JString`
- initUDPEncryption() : `PhotonPeer`
- initUserDataEncryption() : `PhotonPeer`
- insertElementAt() : `JVector< Etype >`
- j -

- JString() : **JString**
- JStringRepresentation() : **ANSIString**, **BaseCharString**, **UTF8String**
- JVector() : **JVector< Etype >**
- k -

- KeyObject() : `KeyObject< Etype >`
Photon C++
Client API  4.1.12.2

- I -

- lastIndexOf() : JObject
- length() : BaseCharString, JObject
- LitePeer() : LitePeer
- LobbyStatsRequest() : LobbyStatsRequest
- log() : Logger
- Logger() : Logger

©  Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
- o -

- 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
- p -

- PhotonPeer() : PhotonPeer
- pingServer() : PhotonPeer
- Player() : Player
- pop() : DeSerializer
- push() : Serializer
- put() : Dictionary< EKeyType, EValueType > , Hashtable
- 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**
- 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
- toStringVitalStats() : TrafficStatsGameLevel
- toUpperCase() : JString
- trim() : JString
- trimToSize() : JVector<Etype>
- typeToString() : Dictionary<EKeyType, EValueType>, DictionaryBase, ToString
- u -

- unsubscribeReturn() : **Listener**
- UTF8Representation() : **JString**
- UTF8String() : **UTF8String**
- v -

- ValueObject() : `ValueObject< Etype >`
- vitalStatsToString() : `Client`, `PhotonPeer`
- vlog() : `Logger`
- w -

- WebFlags() : **WebFlags**
- \~\text{AllocatorInterface() : AllocatorInterface}
- \~\text{ANSIString() : ANSIString}
- \~\text{Base() : Base}
- \~\text{BaseCharString() : BaseCharString}
- \~\text{Client() : Client}
- \~\text{CustomTypeFactory() : CustomTypeFactory< typeCode >}
- \~\text{Dictionary() : Dictionary< EKeyType, EValueType >}
- \~\text{DictionaryBase() : DictionaryBase}
- \~\text{EGTime() : EGTime}
- \~\text{EventData() :EventData}
- \~\text{Hashtable() : Hashtable}
- \~\text{JString() : JString}
- \~\text{JVector() : JVector< Etype >}
- \~\text{KeyObject() : KeyObject< Etype >}
- \~\text{LitePeer() : LitePeer}
- \~\text{Object() : Object}
- \~\text{OperationRequest() : OperationRequest}
- \~\text{OperationResponse() : OperationResponse}
- \~\text{PhotonListener() : PhotonListener}
- \~\text{PhotonPeer() : PhotonPeer}
- \~\text{Player() : Player}
- \~\text{RaiseEventOptions() : RaiseEventOptions}
- \~\text{Room() : Room}
- \~\text{RoomOptions() : RoomOptions}
- \~\text{ToString() : ToString}
- \~\text{TrafficStats() : TrafficStats}
- \~\text{TrafficStatsGameLevel() : TrafficStatsGameLevel}
- \~\text{UTF8String() : UTF8String}
- \~\text{ValueObject() : ValueObject< Etype >}
- `TypeCode : CustomType< typeCode >`
## File List

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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate.h</td>
<td></td>
</tr>
<tr>
<td>AllocatorInterface.h</td>
<td></td>
</tr>
<tr>
<td>ANSIString.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/AuthenticationValues.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/AuthenticationValues.h</td>
<td></td>
</tr>
<tr>
<td>Base.h</td>
<td></td>
</tr>
<tr>
<td>BaseCharString.h</td>
<td></td>
</tr>
<tr>
<td>BaseListener.h</td>
<td></td>
</tr>
<tr>
<td>Channel.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Client.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Client.h</td>
<td></td>
</tr>
<tr>
<td>ClientState.h</td>
<td></td>
</tr>
<tr>
<td>Common.h</td>
<td></td>
</tr>
<tr>
<td>ConnectionProtocol.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Enums/CustomAuthenticationType.h</td>
<td></td>
</tr>
<tr>
<td>CustomType.h</td>
<td></td>
</tr>
<tr>
<td>CustomTypeBase.h</td>
<td></td>
</tr>
<tr>
<td>CustomTypeFactory.h</td>
<td></td>
</tr>
<tr>
<td>DebugLevel.h</td>
<td></td>
</tr>
<tr>
<td>DeSerializer.h</td>
<td></td>
</tr>
<tr>
<td>Dictionary.h</td>
<td></td>
</tr>
<tr>
<td>DictionaryBase.h</td>
<td></td>
</tr>
<tr>
<td>DirectMode.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Enums/DisconnectCause.h</td>
<td></td>
</tr>
<tr>
<td>File Name</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Enums/DisconnectCause.h</td>
<td></td>
</tr>
<tr>
<td>EGTime.h</td>
<td></td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/ErrorCode.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Enums/ErrorCode.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Enums/ErrorCode.h</td>
<td></td>
</tr>
<tr>
<td>EventCache.h</td>
<td></td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/EventCode.h</td>
<td></td>
</tr>
<tr>
<td>EventData.h</td>
<td></td>
</tr>
<tr>
<td>EventKey.h</td>
<td></td>
</tr>
<tr>
<td>FriendInfo.h</td>
<td></td>
</tr>
<tr>
<td>Hashtable.h</td>
<td></td>
</tr>
<tr>
<td>IsPrimitiveType.h</td>
<td></td>
</tr>
<tr>
<td>JString.h</td>
<td></td>
</tr>
<tr>
<td>JVector.h</td>
<td></td>
</tr>
<tr>
<td>KeyObject.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Listener.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Listener.h</td>
<td></td>
</tr>
<tr>
<td>LitePeer.h</td>
<td></td>
</tr>
<tr>
<td>LobbyStatsRequest.h</td>
<td></td>
</tr>
<tr>
<td>LobbyStatsResponse.h</td>
<td></td>
</tr>
<tr>
<td>LobbyType.h</td>
<td></td>
</tr>
<tr>
<td>Logger.h</td>
<td></td>
</tr>
<tr>
<td>MatchmakingMode.h</td>
<td></td>
</tr>
<tr>
<td>MutablePlayer.h</td>
<td></td>
</tr>
<tr>
<td>MutableRoom.h</td>
<td></td>
</tr>
<tr>
<td>NetworkPort.h</td>
<td></td>
</tr>
<tr>
<td>Object.h</td>
<td></td>
</tr>
<tr>
<td>Photon-cpp/inc/ Enums/OperationCode.h</td>
<td></td>
</tr>
<tr>
<td>OperationRequest.h</td>
<td></td>
</tr>
<tr>
<td>OperationResponse.h</td>
<td></td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/ParameterCode.h</td>
<td></td>
</tr>
<tr>
<td>LoadBalancing-cpp/inc/Peer.h</td>
<td></td>
</tr>
<tr>
<td>Chat-cpp/inc/Peer.h</td>
<td></td>
</tr>
</tbody>
</table>
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
Include dependency graph for Allocate.h:

This graph shows which files directly or indirectly include this file:
Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
<tr>
<td>ExitGames::Common::MemoryManagement</td>
</tr>
</tbody>
</table>
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[].

```plaintext
#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 objects that they allocate and deallocate.

```cpp
#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 maxSizeForAllocatorUsage)  
void setAllocator (ExitGames::Common::MemoryManagement::AllocatorInterface &allocator)  
void setAllocatorToDefault (void)  

template<
typename Ftype>
Ftype * allocate (void)

```
Ftype * **allocateArray** (size_t count)

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

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

```cpp
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**

```c
#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

```c
#define REALLOCATE_ARRAY ( type, p, count, ... )
```

This is the macro version of the `reallocatemArray()` 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
#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
Include dependency graph for AllocatorInterface.h:

This graph shows which files directly or indirectly include this file:
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>AllocatorInterface</th>
</tr>
</thead>
</table>

Photon C++
Client API 4.1.12.2

ANSIString.h File Reference

Include dependency graph for ANSIString.h:
This graph shows which files directly or indirectly include this file:
| class   | ANSIString |
# Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
LoadBalancing-cpp/inc/AuthenticationValues.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/AuthenticationValues.h:
This graph shows which files directly or indirectly include this file:
Classes

| class | AuthenticationValues |
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
Photon C++
Client API  4.1.12.2

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

<table>
<thead>
<tr>
<th>class</th>
<th>AuthenticationValues</th>
</tr>
</thead>
</table>
## Namespaces

- **ExitGames**
- **ExitGames::Chat**

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>Base</th>
</tr>
</thead>
</table>

Classes
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>
Include dependency graph for BaseCharString.h:

This graph shows which files directly or indirectly include this file:
## Classes

| class | BaseCharString |
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
Include dependency graph for BaseListener.h:

This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>BaseListener</th>
</tr>
</thead>
</table>

---
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>
Channel.h File Reference

Include dependency graph for Channel.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Channel</th>
</tr>
</thead>
</table>


### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Chat</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
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
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
ClientState.h File Reference

This graph shows which files directly or indirectly include this file:
# Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Chat</td>
</tr>
<tr>
<td>ExitGames::Chat::ClientState</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int</td>
<td><strong>Uninitialized</strong> Peer</td>
<td>is created but not used yet.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ConnectingToNameServer</strong></td>
<td>Connecting to Name Server (includes connect authenticate and joining the lobby)</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ConnectedToNameServer</strong></td>
<td>Connected to Name Server.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>Authenticating</strong></td>
<td>Authenticating.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>Authenticated</strong></td>
<td>Authenticated.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>DisconnectingFromNameServer</strong></td>
<td>Transition from Name to <strong>Chat</strong> Server.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ConnectingToFrontEnd</strong></td>
<td>Transition to <strong>Chat</strong> Server.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ConnectedToFrontEnd</strong></td>
<td>Connected to <strong>Chat</strong> Server. Subscribe to channels and chat here.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>Disconnecting</strong></td>
<td>The client disconnects (from any server).</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>Disconnected</strong></td>
<td>The client is no longer connected (to any server). Connect to Name Server to go on.</td>
</tr>
</tbody>
</table>
Common.h File Reference

Include dependency graph for Common.h:
This graph shows which files directly or indirectly include this file:
### ConnectionProtocol.h

File Reference

Include dependency graph for ConnectionProtocol.h:
This graph shows which files directly or indirectly include this file:
Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Photon</td>
</tr>
<tr>
<td>ExitGames::Photon::ConnectionProtocol</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>bool <code>getIsUDP</code></td>
</tr>
<tr>
<td>bool <code>getIsTCP</code></td>
</tr>
<tr>
<td>bool <code>getIsWebSocket</code></td>
</tr>
<tr>
<td>bool <code>getIsSecure</code></td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>static const nByte</td>
<td>Use UDP to connect to Photon, which allows you to send operations reliable or unreliable on demand.</td>
</tr>
<tr>
<td>TCP</td>
<td>static const nByte</td>
<td>Use TCP to connect to Photon.</td>
</tr>
<tr>
<td>WS</td>
<td>static const nByte</td>
<td>Use websockets to connect to Photon.</td>
</tr>
<tr>
<td>WSS</td>
<td>static const nByte</td>
<td>Use secure websockets to connect to Photon.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>static const nByte</td>
<td></td>
</tr>
</tbody>
</table>
Include dependency graph for `LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h`:

<table>
<thead>
<tr>
<th>LoadBalancing-cpp</th>
<th>inc</th>
<th>Enums</th>
<th>Namespaces</th>
<th>Variables</th>
</tr>
</thead>
</table>

LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h

File Reference
This graph shows which files directly or indirectly include this file:
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::CustomAuthenticationType</td>
</tr>
</tbody>
</table>
**Variables**

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOM</td>
<td>Use a custom authentication service.</td>
<td></td>
</tr>
<tr>
<td>STEAM</td>
<td>Authenticates users by their Steam Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>FACEBOOK</td>
<td>Authenticates users by their Facebook Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>OCULUS</td>
<td>Authenticates users by their Oculus Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>PLAYSTATION</td>
<td>Authenticates users by their PSN Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>XBOX</td>
<td>Authenticates users by their XBox Network Account. Set auth values accordingly!</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td>Disables custom authentication.</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Chat</td>
</tr>
<tr>
<td>ExitGames::Chat::CustomAuthenticationType</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOM</td>
<td>static const nByte</td>
<td>Use a custom authentication service.</td>
</tr>
<tr>
<td>STEAM</td>
<td>static const nByte</td>
<td>Authenticates users by their Steam Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>FACEBOOK</td>
<td>static const nByte</td>
<td>Authenticates users by their Facebook Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>OCULUS</td>
<td>static const nByte</td>
<td>Authenticates users by their Oculus Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>PLAYSTATION</td>
<td>static const nByte</td>
<td>Authenticates users by their PSN Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>XBOX</td>
<td>static const nByte</td>
<td>Authenticates users by their XBox Network Account. Set auth values accordingly!</td>
</tr>
<tr>
<td>NONE</td>
<td>static const nByte</td>
<td>Disables custom authentication.</td>
</tr>
</tbody>
</table>
Include dependency graph for CustomType.h:

This graph shows which files directly or indirectly include this file:
| class | CustomType< typeCode > |
Include dependency graph for CustomTypeBase.h:

This graph shows which files directly or indirectly include this file:
Classes

class CustomTypeBase
Include dependency graph for CustomTypeFactory.h:

This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>Class</th>
<th>CustomType&lt; typeCode &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>CustomTypeFactory&lt; typeCode &gt;</td>
</tr>
</tbody>
</table>
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>
## DebugLevel.h File Reference

This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
<tr>
<td>ExitGames::Common::DebugLevel</td>
</tr>
</tbody>
</table>
**Variables**

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int</td>
<td><strong>OFF</strong></td>
<td>No debug out.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ERRORS</strong></td>
<td>Only error descriptions.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>WARNINGS</strong></td>
<td>Warnings and errors.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>INFO</strong></td>
<td>Information about internal workflows, warnings and errors.</td>
</tr>
<tr>
<td>static const int</td>
<td><strong>ALL</strong></td>
<td>Most complete workflow description (but lots of debug output), info, warnings and errors.</td>
</tr>
</tbody>
</table>
Include dependency graph for DeSerializer.h:
This graph shows which files directly or indirectly include this file:
Classes

<table>
<thead>
<tr>
<th>class</th>
<th>DeSerializer</th>
</tr>
</thead>
</table>

Namespaces

ExitGames

ExitGames::Common
Include dependency graph for Dictionary.h:
This graph shows which files directly or indirectly include this file:
Classes

```java
class Dictionary<EKeyType, EValueType>
```
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
Include dependency graph for DictionaryBase.h:
This graph shows which files directly or indirectly include this file:
| class | DictionaryBase |
Namespaces

ExitGames

ExitGames::Common
DirectMode.h File Reference

Include dependency graph for DirectMode.h:
This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ExitGames</strong></td>
</tr>
<tr>
<td><strong>ExitGames::LoadBalancing</strong></td>
</tr>
<tr>
<td><strong>ExitGames::LoadBalancing::DirectMode</strong></td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not create any 2p2 connections between the clients. This is the default.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>ALL_TO_ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each client establishes a direct connection with every other client inside the room.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>MASTER_TO_ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
LoadBalancing-cpp/inc/Enums/DisconnectCause.h File Reference

This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::DisconnectCause</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Variable Description</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static const int</strong></td>
<td><strong>NONE</strong></td>
</tr>
<tr>
<td>No error was tracked.</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>DISCONNECT_BY_SERVER_USER_LIMIT</strong></td>
</tr>
<tr>
<td>OnStatusChanged: The CCUs count of your Photon Server License is exhausted (temporarily).</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>EXCEPTION_ON_CONNECT</strong></td>
</tr>
<tr>
<td>OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>DISCONNECT_BY_SERVER</strong></td>
</tr>
<tr>
<td>OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>DISCONNECT_BY_SERVER_LOGIC</strong></td>
</tr>
<tr>
<td>OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>TIMEOUT_DISCONNECT</strong></td>
</tr>
<tr>
<td>OnStatusChanged: This client detected that the server's responses are not received in due time. Maybe you send / receive too much?</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>EXCEPTION</strong></td>
</tr>
<tr>
<td>OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.</td>
<td></td>
</tr>
<tr>
<td><strong>static const int</strong></td>
<td><strong>INVALID_AUTHENTICATION</strong></td>
</tr>
<tr>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.</td>
<td></td>
</tr>
<tr>
<td>static const int</td>
<td><strong>MAX_CCU_REACHED</strong></td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>OnOperationResponse: Authenticate (temporarily) failed when using a Photon Cloud subscription without CCU Burst. Update your subscription.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th><strong>INVALID_REGION</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Authenticate when the app’s Photon Cloud subscription is locked to some (other) region(s). Update your subscription or master server address.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th><strong>OPERATION_NOT_ALLOWED_IN_CURRENT_STATE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Operation that’s (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th><strong>CUSTOM_AUTHENTICATION_FAILED</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.</td>
<td></td>
</tr>
</tbody>
</table>
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Chat</td>
</tr>
<tr>
<td>ExitGames::Chat::DisconnectCause</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>static const int</strong> NONE</td>
<td>No error was tracked.</td>
</tr>
<tr>
<td><strong>static const int</strong> DISCONNECT_BY_SERVER_USER_LIMIT</td>
<td>OnStatusChanged: The CCUs count of your Photon Server License is exhausted (temporarily).</td>
</tr>
<tr>
<td><strong>static const int</strong> EXCEPTION_ON_CONNECT</td>
<td>OnStatusChanged: The server is not available or the address is wrong. Make sure the port is provided and the server is up.</td>
</tr>
<tr>
<td><strong>static const int</strong> DISCONNECT_BY_SERVER</td>
<td>OnStatusChanged: The server disconnected this client. Most likely the server's send buffer is full (receiving too much from other clients).</td>
</tr>
<tr>
<td><strong>static const int</strong> DISCONNECT_BY_SERVER_LOGIC</td>
<td>OnStatusChanged: The server disconnected this client due to server's logic (received a disconnect command).</td>
</tr>
<tr>
<td><strong>static const int</strong> TIMEOUT_DISCONNECT</td>
<td>OnStatusChanged: This client detected that the server’s responses are not received in due time. Maybe you send / receive too much?</td>
</tr>
<tr>
<td><strong>static const int</strong> EXCEPTION</td>
<td>OnStatusChanged: Some internal exception caused the socket code to fail. Contact Exit Games.</td>
</tr>
<tr>
<td><strong>static const int</strong> INVALID_AUTHENTICATION</td>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid AppId. Update your subscription or contact Exit Games.</td>
</tr>
<tr>
<td>static const int</td>
<td>MAX_CCU_REACHED</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>OnOperationResponse: Authenticate (temporarily) failed when using a Photon Cloud subscription without CCU Burst. Update your subscription.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th>INVALID_REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Authenticate when the app’s Photon Cloud subscription is locked to some (other) region(s). Update your subscription or master server address.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th>OPERATION_NOT_ALLOWED_IN_CURRENT_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Operation that's (currently) not available for this client (not authorized usually). Only tracked for op Authenticate.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const int</th>
<th>CUSTOM_AUTHENTICATION_FAILED</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnOperationResponse: Authenticate in the Photon Cloud with invalid client values or custom authentication setup in Cloud Dashboard.</td>
<td></td>
</tr>
</tbody>
</table>
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>EGTime</th>
</tr>
</thead>
</table>

Classes
Namespaces

ExitGames

ExitGames::Common
 Photon C++
Client API 4.1.12.2

Photon-cpp/inc/Enums/ErrorCode.h File Reference

This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Photon</td>
</tr>
<tr>
<td>ExitGames::Photon::ErrorCode</td>
</tr>
</tbody>
</table>
**Variables**

<table>
<thead>
<tr>
<th>Static const int</th>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>Static const int</td>
<td>No error.</td>
</tr>
<tr>
<td>EFAILED</td>
<td>Static const int</td>
<td>General failure.</td>
</tr>
<tr>
<td>ENOMEMORY</td>
<td>Static const int</td>
<td>Out of memory.</td>
</tr>
<tr>
<td>EBADCLASS</td>
<td>Static const int</td>
<td>NULL class object.</td>
</tr>
<tr>
<td>EBADPARM</td>
<td>Static const int</td>
<td>Invalid parameter.</td>
</tr>
<tr>
<td>EITEMBUSY</td>
<td>Static const int</td>
<td>Context (system, interface, etc.) is busy.</td>
</tr>
<tr>
<td>NET_SUCCESS</td>
<td>Static const int</td>
<td>No network error, successful operation.</td>
</tr>
<tr>
<td>NET_ERROR</td>
<td>Static const int</td>
<td>Unsuccessful operation.</td>
</tr>
<tr>
<td>NET_ENETNONET</td>
<td>Static const int</td>
<td>Network subsystem unavailable.</td>
</tr>
<tr>
<td>NET_MSGSIZE</td>
<td>Static const int</td>
<td>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.</td>
</tr>
</tbody>
</table>
static const int NET_ENOTCONN
This graph shows which files directly or indirectly include this file:

```
includes:
- Client.h
- Photon::Client

indirect includes:
- Client.cpp
- PuncherClient.cpp
- MutatedPlayer.cpp
- MutatedRoom.cpp
```
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
<td></td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
<td></td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::ErrorCode</td>
<td></td>
</tr>
</tbody>
</table>
Variables

<table>
<thead>
<tr>
<th>static const int</th>
<th>OPERATION_DENIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int</td>
<td>OPERATION_INVALID</td>
</tr>
<tr>
<td>static const int</td>
<td>INTERNAL_SERVER_ERROR</td>
</tr>
<tr>
<td>static const int</td>
<td>OK</td>
</tr>
<tr>
<td>static const int</td>
<td>INVALID_AUTHENTICATION</td>
</tr>
<tr>
<td>static const int</td>
<td>GAME_ID_ALREADY_EXISTS</td>
</tr>
<tr>
<td>static const int</td>
<td>GAME_FULL</td>
</tr>
<tr>
<td>static const int</td>
<td>GAME_CLOSED</td>
</tr>
<tr>
<td>static const int</td>
<td>ALREADY_MATCHED</td>
</tr>
<tr>
<td>static const int</td>
<td>SERVER_FULL</td>
</tr>
<tr>
<td>static const int</td>
<td>USER_BLOCKED</td>
</tr>
<tr>
<td>static const int</td>
<td>NO_MATCH_FOUND</td>
</tr>
<tr>
<td>static const int</td>
<td>GAME_DOES_NOT_EXIST</td>
</tr>
<tr>
<td>static const int</td>
<td>MAX_CCU_REACHED</td>
</tr>
<tr>
<td>static const int</td>
<td>INVALID_REGION</td>
</tr>
<tr>
<td>static const int</td>
<td>CUSTOM_AUTHENTICATION_FAILED</td>
</tr>
<tr>
<td>static const int</td>
<td>AUTHENTICATION_TOKEN_EXPIRED</td>
</tr>
</tbody>
</table>
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
Chat-cpp/inc/Enums/ErrorCode.h File Reference

This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Chat</td>
</tr>
<tr>
<td>ExitGames::Chat::ErrorCode</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int OPERATION_DENIED</td>
</tr>
<tr>
<td>static const int OPERATION_INVALID</td>
</tr>
<tr>
<td>static const int INTERNAL_SERVER_ERROR</td>
</tr>
<tr>
<td>static const int OK</td>
</tr>
<tr>
<td>static const int INVALID_AUTHENTICATION</td>
</tr>
<tr>
<td>static const int GAME_ID_ALREADY_EXISTS</td>
</tr>
<tr>
<td>static const int GAME_FULL</td>
</tr>
<tr>
<td>static const int GAME_CLOSED</td>
</tr>
<tr>
<td>static const int ALREADY_MATCHED</td>
</tr>
<tr>
<td>static const int SERVER_FULL</td>
</tr>
<tr>
<td>static const int USER_BLOCKED</td>
</tr>
<tr>
<td>static const int NO_MATCH_FOUND</td>
</tr>
<tr>
<td>static const int GAME DOES NOT_EXIST</td>
</tr>
<tr>
<td>static const int MAX_CCU_REACHED</td>
</tr>
<tr>
<td>static const int INVALID_REGION</td>
</tr>
<tr>
<td>static const int CUSTOM_AUTHENTICATION_FAILED</td>
</tr>
</tbody>
</table>
Photon C++
Client API 4.1.12.2

EventCache.h File Reference

Include dependency graph for EventCache.h:
This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Lite</td>
</tr>
<tr>
<td>ExitGames::Lite::&lt;EventCache&gt;</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Static const nByte</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO_NOT_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>MERGE_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>REPLACE_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>REMOVE_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>ADD_TO_ROOM_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>ADD_TO_ROOM_CACHE_GLOBAL</td>
<td>static const nByte</td>
</tr>
<tr>
<td>REMOVE_FROM_ROOM_CACHE</td>
<td>static const nByte</td>
</tr>
<tr>
<td>REMOVE_FROM_ROOM_CACHE_FOR_ACTORS</td>
<td>static const nByte</td>
</tr>
<tr>
<td>SLICE_INC_INDEX</td>
<td>static const nByte</td>
</tr>
<tr>
<td>SLICE_SET_INDEX</td>
<td>static const nByte</td>
</tr>
<tr>
<td>SLICE_PURGE_INDEX</td>
<td>static const nByte</td>
</tr>
<tr>
<td>SLICE_PURGE_UP_TO_INDEX</td>
<td>static const nByte</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Lite</td>
</tr>
<tr>
<td>ExitGames::Lite::EventCode</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td><strong>JOIN</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>LEAVE</strong></td>
</tr>
<tr>
<td>static const nByte</td>
<td><strong>PROPERTIES_CHANGED</strong></td>
</tr>
</tbody>
</table>
Include dependency graph for EventData.h:
This graph shows which files directly or indirectly include this file:
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>EventData</th>
</tr>
</thead>
</table>


EventKey.h File Reference

Include dependency graph for EventKey.h:
This graph shows which files directly or indirectly include this file:

- EventKey.h
- LitePeer.h
- Client.cpp
- LitePeer.cpp
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Lite</td>
</tr>
<tr>
<td>ExitGames::Lite::EventKey</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>TARGET_ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTORLIST</td>
</tr>
<tr>
<td>static const nByte</td>
<td>PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTORPROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GAMEPROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>DATA</td>
</tr>
</tbody>
</table>
Include dependency graph for FriendInfo.h:
This graph shows which files directly or indirectly include this file:

```
FriendInfo.h

Client.h  FriendInfo.cpp

Client.cpp  PuncherClient.cpp  MutablePlayer.cpp  MutableRoom.cpp
```
| class | FriendInfo |
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
**Hashtable.h File Reference**

Include dependency graph for Hashtable.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>Hashtable</th>
</tr>
</thead>
</table>

Classes
### Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
This graph shows which files directly or indirectly include this file:
JString.h File Reference

Include dependency graph for JString.h:

This graph shows which files directly or indirectly include this file:
Classes

<table>
<thead>
<tr>
<th>class</th>
<th>JString</th>
</tr>
</thead>
</table>


## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
Include dependency graph for JVector.h:

This graph shows which files directly or indirectly include this file:
# 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

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

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
</table>

| ExitGames::Common |

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
Include dependency graph for LoadBalancing-cpp/inc/Listener.h:
This graph shows which files directly or indirectly include this file:

```
Listener.h
        |
        |
        Client.h
```

- Client.h
- PuncherClient.cpp
- MutablePlayer.cpp
- MutableRoom.cpp
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Listener</th>
</tr>
</thead>
</table>
## Namespaces

### ExitGames

### ExitGames::LoadBalancing
Chat-cpp/inc/Listener.h File Reference

Include dependency graph for Chat-cpp/inc/Listener.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>Listener</th>
</tr>
</thead>
</table>

Classes
Namespaces

ExitGames

ExitGames::Chat
LitePeer.h File Reference

Include dependency graph for LitePeer.h:
This graph shows which files directly or indirectly include this file:

LitePeer.h

LitePeer.cpp
# Classes

<table>
<thead>
<tr>
<th>class</th>
<th>LitePeer</th>
</tr>
</thead>
</table>

**Namespaces**

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Lite</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
Include dependency graph for LobbyStatsRequest.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>LobbyStatsRequest</th>
</tr>
</thead>
</table>
LobbyStatsResponse.h File Reference

Include dependency graph for LobbyStatsResponse.h:
This graph shows which files directly or indirectly include this file:
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>LobbyStatsResponse</th>
</tr>
</thead>
</table>
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
Include dependency graph for LobbyType.h:
This graph shows which files directly or indirectly include this file:
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::LobbyType</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static const nByte DEFAULT</code></td>
<td>This lobby type is used unless another lobby type is specified. <strong>Room</strong> lists will be sent and <code>Client::opJoinRandomRoom()</code> can filter by matching properties.</td>
</tr>
<tr>
<td><code>static const nByte SQL_LOBBY</code></td>
<td>This lobby type lists rooms like type DEFAULT but SQL-like &quot;where&quot; clauses for filtering can be used with <code>Client::opJoinRandomRoom()</code>. This allows 'bigger', 'less', 'or' and 'and' combinations.</td>
</tr>
<tr>
<td><code>static const nByte ASYNC_RANDOM_LOBBY</code></td>
<td>This lobby does not send room lists. It is only used for <code>Client::opJoinRandomRoom()</code>. It keeps rooms available for matchmaking for a while even when there are only inactive users left.</td>
</tr>
</tbody>
</table>
Logger.h File Reference

Include dependency graph for Logger.h:

This graph shows which files directly or indirectly include this file:
# Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Logger</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>
## Macros

```c
#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 therefore 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()`
Include dependency graph for MatchmakingMode.h:
This graph shows which files directly or indirectly include this file:
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::MatchmakingMode</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>FILL_ROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fills up rooms (oldest first) to get players together as fast as possible. Default. Makes most sense with MaxPlayers &gt; 0 and games that can only start with more players.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>SERIAL_MATCHING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distributes players across available rooms sequentially but takes filters into account. Without filters, rooms get players evenly distributed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static const nByte</th>
<th>RANDOM_MATCHING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Joins a (fully) random room. Expected properties must match, but aside from this, any available room might be selected.</td>
</tr>
</tbody>
</table>
MutablePlayer.h File

Reference

Include dependency graph for MutablePlayer.h:
This graph shows which files directly or indirectly include this file:

- **MutablePlayer.h**
- **Client.h**
- **MutablePlayerFactory.cpp**
- **Client.cpp**
- **PuncherClient.cpp**
- **MutablePlayer.cpp**
- **MutableRoom.cpp**
### Classes

<table>
<thead>
<tr>
<th>class</th>
<th>MutablePlayer</th>
</tr>
</thead>
</table>
Namespaces

ExitGames

ExitGames::LoadBalancing
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

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
Photon C++
Client API 4.1.12.2

NetworkPort.h File Reference

This graph shows which files directly or indirectly include this file:
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>UDP</td>
</tr>
<tr>
<td>struct</td>
<td>UDPAlternative</td>
</tr>
<tr>
<td>struct</td>
<td>TCP</td>
</tr>
<tr>
<td>struct</td>
<td>WS</td>
</tr>
<tr>
<td>struct</td>
<td>WSS</td>
</tr>
<tr>
<td>Namespaces</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>ExitGames</td>
<td></td>
</tr>
<tr>
<td>ExitGames::Photon</td>
<td></td>
</tr>
<tr>
<td>ExitGames::Photon::NetworkPort</td>
<td></td>
</tr>
</tbody>
</table>
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
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

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>JOIN</td>
</tr>
<tr>
<td>static const nByte</td>
<td>LEAVE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>RAISE_EV</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SETPROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GETPROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CHANGE_GROUPS</td>
</tr>
</tbody>
</table>
Photon C++
Client API 4.1.12.2

OperationRequest.h
File Reference

Include dependency graph for OperationRequest.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>OperationRequest</th>
</tr>
</thead>
</table>
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Photon</td>
</tr>
</tbody>
</table>

typedef Common::Dictionary< nByte, Common::Object > OperationRequestParameters
OperationResponse.h
File Reference

Include dependency graph for OperationResponse.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>OperationResponse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Photon</td>
</tr>
</tbody>
</table>
Photon-C++

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

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Lite</td>
</tr>
<tr>
<td>ExitGames::Lite::ParameterCode</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>GAMEID</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>TARGET_ACTORNR</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTOR_LIST</td>
</tr>
<tr>
<td>static const nByte</td>
<td>PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>BROADCAST</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ACTOR_PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GAME_PROPERTIES</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CACHE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>RECEIVER_GROUP</td>
</tr>
<tr>
<td>static const nByte</td>
<td>DATA</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CODE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>GROUP</td>
</tr>
<tr>
<td>static const nByte</td>
<td>REMOVE</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ADD</td>
</tr>
</tbody>
</table>
LoadBalancing-cpp/inc/Pin.h File Reference

Include dependency graph for LoadBalancing-cpp/inc/Pin.h:
This graph shows which files directly or indirectly include this file:
# Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Peer</th>
</tr>
</thead>
</table>

## Chat-cpp/inc/Peer.h

**File Reference**

Include dependency graph for Chat-cpp/inc/Peer.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Peer</th>
</tr>
</thead>
</table>

Namespaces

ExitGames

ExitGames::Chat
PeerState.h File Reference

This graph shows which files directly or indirectly include this file:
### Namespaces

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ExitGames</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ExitGames::Photon</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ExitGames::Photon::PeerState</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Timer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int <code>DISCONNECTED</code></td>
<td>The peer is disconnected and can't call Operations. Call PhotonPeer_connect().</td>
</tr>
<tr>
<td>static const int <code>CONNECTING</code></td>
<td>The peer is establishing the connection: opening a socket, exchanging packages with Photon.</td>
</tr>
<tr>
<td>static const int <code>INITIALIZING_APPLICATION</code></td>
<td>The connection is established and now sends the application name to Photon. You set the &quot;application name&quot; by calling PhotonPeer_connect().</td>
</tr>
<tr>
<td>static const int <code>CONNECTED</code></td>
<td>The peer is connected and initialized (selected an application). You can now use operations.</td>
</tr>
<tr>
<td>static const int <code>DISCONNECTING</code></td>
<td>The peer is disconnecting. It sent a disconnect to the server, which will acknowledge closing the connection.</td>
</tr>
</tbody>
</table>
### PeerStates.h File Reference

This graph shows which files directly or indirectly include this file:

```plaintext
[Graph showing dependencies]
```
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing::PeerStates</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int <strong>Uninitialized</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>PeerCreated</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>ConnectingToNameserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>ConnectedToNameserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>DisconnectingFromNameserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Connecting</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Connected</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>WaitingForCustomAuthenticationNextStepCall</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Authenticated</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>JoinedLobby</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>DisconnectingFromMasterserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>ConnectingToGameserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>ConnectedToGameserver</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>AuthenticatedOnGameServer</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Joining</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Joined</strong></td>
<td></td>
</tr>
<tr>
<td>static const int <strong>Leaving</strong></td>
<td></td>
</tr>
<tr>
<td>static const int</td>
<td>Left</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>static const int</td>
<td>DisconnectingFromGameserver</td>
</tr>
<tr>
<td>static const int</td>
<td>ConnectingToMasterserver</td>
</tr>
<tr>
<td>static const int</td>
<td>ConnectedComingFromGameserver</td>
</tr>
<tr>
<td>static const int</td>
<td>AuthenticatedComingFromGameserver</td>
</tr>
<tr>
<td>static const int</td>
<td>Disconnecting</td>
</tr>
<tr>
<td>static const int</td>
<td>Disconnected</td>
</tr>
</tbody>
</table>
PhotonListener.h File
Reference

Include dependency graph for PhotonListener.h:
This graph shows which files directly or indirectly include this file:
| class | PhotonListener |
# Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Photon</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.  
Photon Documentation | Contact | Terms
Photon C++
Client API 4.1.12.2

PhotonPeer.h File Reference

Include dependency graph for PhotonPeer.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>PhotonPeer</th>
</tr>
</thead>
</table>
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Photon</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Player.h File Reference

Include dependency graph for Player.h:
This graph shows which files directly or indirectly include this file:
| Class | Player |
Namespaces

ExitGames

ExitGames::LoadBalancing
# RaiseEventOptions.h

File Reference

Include dependency graph for RaiseEventOptions.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>RaiseEventOptions</th>
</tr>
</thead>
</table>

Classes
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon C++
Client API 4.1.12.2

ReceiverGroup.h File Reference

Include dependency graph for ReceiverGroup.h:
This graph shows which files directly or indirectly include this file:
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Lite</td>
</tr>
<tr>
<td>ExitGames::Lite::ReceiverGroup</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>OTHERS</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ALL</td>
</tr>
<tr>
<td>static const nByte</td>
<td>MASTER_CLIENT</td>
</tr>
</tbody>
</table>

© Exit Games®, all rights reserved.
Photon Documentation | Contact | Terms
Include dependency graph for RegionSelectionMode.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SELECT</td>
</tr>
<tr>
<td>static const nByte</td>
<td>BEST</td>
</tr>
</tbody>
</table>
Room.h File Reference

Include dependency graph for Room.h:
This graph shows which files directly or indirectly include this file:
| class | Room |
## Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
RoomOptions.h File Reference

Include dependency graph for RoomOptions.h:
This graph shows which files directly or indirectly include this file:
# Classes

| class | RoomOptions |
### Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
Include dependency graph for Serializer.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>Serializer</th>
</tr>
</thead>
</table>


Namespaces

ExitGames

ExitGames::Common
Include dependency graph for ServerType.h:
This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::LoadBalancing</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>NAME_SERVER</td>
</tr>
<tr>
<td>static const nByte</td>
<td>MASTER_SERVER</td>
</tr>
</tbody>
</table>
Photon C++
Client API 4.1.12.2

StatusCode.h File
Reference

This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Photon</td>
</tr>
<tr>
<td>ExitGames::Photon::StatusCode</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTION_ON_CONNECT</td>
<td>The PhotonPeer encountered an exception while opening the incoming connection to the server. The server could be down / not running.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>The PhotonPeer is connected.</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>The PhotonPeer just disconnected.</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>The PhotonPeer encountered an exception and will disconnect, too.</td>
</tr>
<tr>
<td>QUEUE_OUTGOING_RELIABLE_WARNING</td>
<td>PhotonPeer outgoing queue is filling up. send more often.</td>
</tr>
<tr>
<td>QUEUE_OUTGOING_UNRELIABLE_WARNING</td>
<td>PhotonPeer outgoing queue is filling up. send more often.</td>
</tr>
<tr>
<td>SEND_ERROR</td>
<td>Sending command failed. Either not connected, or the requested channel is bigger than the number of initialized channels.</td>
</tr>
<tr>
<td>QUEUE_OUTGOING_ACKS_WARNING</td>
<td>PhotonPeer outgoing queue is filling up. Send more often.</td>
</tr>
<tr>
<td>QUEUE_INCOMING_RELIABLE_WARNING</td>
<td>PhotonPeer incoming reliable queue is filling up.</td>
</tr>
<tr>
<td>static const int</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>QUEUE_INCOMING_UNRELIABLE_WARNING</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>PhotonPeer incoming unreliable queue is filling up. Dispatch more often.</td>
<td></td>
</tr>
<tr>
<td>QUEUE_SENT_WARNING</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>PhotonPeer sent queue is filling up. Check, why the server does not acknowledge your sent reliable commands.</td>
<td></td>
</tr>
<tr>
<td>INTERNAL_RECEIVE_EXCEPTION</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>Exception, if a server cannot be connected. Most likely, the server is not responding. Ask the user to try again later.</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT_DISCONNECT</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>Disconnection due to a timeout (client did no longer receive ACKs from server).</td>
<td></td>
</tr>
<tr>
<td>DISCONNECT_BY_SERVER</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>Disconnect by server due to timeout (received a disconnect command, cause server misses ACKs of client).</td>
<td></td>
</tr>
<tr>
<td>DISCONNECT_BY_SERVER_USER_LIMIT</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>Disconnect by server due to concurrent user limit reached (received a disconnect command).</td>
<td></td>
</tr>
<tr>
<td>DISCONNECT_BY_SERVER_LOGIC</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>Disconnect by server due to server's logic (received a disconnect command).</td>
<td></td>
</tr>
<tr>
<td>ENCRYPTION_ESTABLISHED</td>
<td>Dispatch more often.</td>
</tr>
<tr>
<td>The encryption-setup for secure communication finished successfully.</td>
<td></td>
</tr>
</tbody>
</table>
static const int ENCRYPTION_FAILED_TO_ESTABLISH
The encryption-setup failed for some reason. Check debug logs.
Include dependency graph for `ToString.h`:

This graph shows which files directly or indirectly include this file:
<table>
<thead>
<tr>
<th>class</th>
<th>ToString</th>
</tr>
</thead>
</table>

Classes
Namespaces

ExitGames

ExitGames::Common
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
TrafficStatsGameLevel.h File Reference

Include dependency graph for TrafficStatsGameLevel.h:
This graph shows which files directly or indirectly include this file:
| class | TrafficStatsGameLevel |
Namespaces

ExitGames

ExitGames::Photon
Include dependency graph for TypeCode.h:

This graph shows which files directly or indirectly include this file:
## Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
<tr>
<td>ExitGames::Common::TypeCode</td>
</tr>
</tbody>
</table>
## Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Java Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const nByte</td>
<td>BYTE</td>
<td>nByte</td>
</tr>
<tr>
<td>static const nByte</td>
<td>SHORT</td>
<td>short</td>
</tr>
<tr>
<td>static const nByte</td>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>static const nByte</td>
<td>LONG</td>
<td>int64</td>
</tr>
<tr>
<td>static const nByte</td>
<td>FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>static const nByte</td>
<td>DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>static const nByte</td>
<td>BOOLEAN</td>
<td>bool</td>
</tr>
<tr>
<td>static const nByte</td>
<td>STRING</td>
<td>JString.</td>
</tr>
<tr>
<td>static const nByte</td>
<td>HASHTABLE</td>
<td>Hashtable.</td>
</tr>
<tr>
<td>static const nByte</td>
<td>DICTIONARY</td>
<td>Dictionary.</td>
</tr>
<tr>
<td>static const nByte</td>
<td>OBJECT</td>
<td>Object, only allowed for arrays!</td>
</tr>
<tr>
<td>static const nByte</td>
<td>ARRAY</td>
<td>internal only</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>static const nByte</td>
<td>BYTEARRAY</td>
<td>internal only</td>
</tr>
<tr>
<td>static const nByte</td>
<td>PHOTON_COMMAND</td>
<td>internal only</td>
</tr>
<tr>
<td>static const nByte</td>
<td>EG_NULL</td>
<td>internal only</td>
</tr>
<tr>
<td>static const nByte</td>
<td>CUSTOM</td>
<td>internal only</td>
</tr>
<tr>
<td>static const nByte</td>
<td>UNKNOWN</td>
<td>internal only</td>
</tr>
</tbody>
</table>
UserStatus.h File Reference

This graph shows which files directly or indirectly include this file:
Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Chat</td>
</tr>
<tr>
<td>ExitGames::Chat::UserStatus</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Static const int</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OFFLINE</strong></td>
<td>Offline.</td>
</tr>
<tr>
<td><strong>INVISIBLE</strong></td>
<td>Be invisible to everyone. Sends no message.</td>
</tr>
<tr>
<td><strong>ONLINE</strong></td>
<td>Online and available.</td>
</tr>
<tr>
<td><strong>AWAY</strong></td>
<td>Online but not available.</td>
</tr>
<tr>
<td><strong>DND</strong></td>
<td>Do not disturb.</td>
</tr>
<tr>
<td><strong>LFG</strong></td>
<td>Looking For Game/Group. Could be used when you want to be invited or do matchmaking.</td>
</tr>
<tr>
<td><strong>PLAYING</strong></td>
<td>Could be used when in a room, playing.</td>
</tr>
</tbody>
</table>
UTF8String.h File Reference

Include dependency graph for UTF8String.h:
This graph shows which files directly or indirectly include this file:
| class | UTF8String |
# Namespaces

<table>
<thead>
<tr>
<th>ExitGames</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.

Photon Documentation | Contact | Terms
ValueObject.h File Reference

Include dependency graph for ValueObject.h:
This graph shows which files directly or indirectly include this file:
## Classes

<table>
<thead>
<tr>
<th>class</th>
<th>ValueObject&lt; Etype &gt;</th>
</tr>
</thead>
</table>


# Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExitGames</td>
</tr>
<tr>
<td>ExitGames::Common</td>
</tr>
</tbody>
</table>

© Exit Games®, all rights reserved.

Photon Documentation | Contact | Terms
WebFlags.h File Reference

Include dependency graph for WebFlags.h:
This graph shows which files directly or indirectly include this file:
| class | WebFlags |
Namespaces

ExitGames

ExitGames::LoadBalancing
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
• 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
This is the complete list of members for `AuthenticationValues`, including all inherited members.

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getData(void)</code></td>
<td>const</td>
</tr>
<tr>
<td><code>getSecret(void)</code></td>
<td>const</td>
</tr>
<tr>
<td><code>getType(void)</code></td>
<td>const</td>
</tr>
<tr>
<td><code>getUserID(const Common::JString &amp;userID)</code></td>
<td></td>
</tr>
<tr>
<td><code>setUserID(const Common::JString &amp;userID)</code></td>
<td></td>
</tr>
<tr>
<td><code>setDataType(nByte type)</code></td>
<td></td>
</tr>
<tr>
<td><code>setUserID(const Common::JString &amp;userID)</code></td>
<td></td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false)</code></td>
<td>const</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td></td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td></td>
</tr>
</tbody>
</table>
Graph Legend

This page explains how to interpret the graphs that are generated by doxygen.

Consider the following example:

```cpp
/*! 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.
Channel Member List

This is the complete list of members for Channel, including all inherited members.

- `clearMessages(void)` (defined in Channel)
- `getDebugOutputLevel(void)`
- `getIsPrivate(void) const` (defined in Channel)
- `getLogFormatOptions(void)`
- `getMessageCount(void) const` (defined in Channel)
- `getMessages(void) const` (defined in Channel)
- `getName(void) const` (defined in Channel)
- `getSenders(void) const` (defined in Channel)
- `setDebugOutputLevel(int debugLevel)`
- `setListener(const BaseListener *baseListener)`
- `setLogFormatOptions(const LogFormatOptions &options)`
- `toString(Common::JString &retStr, bool withTypes=false) const` (defined in Channel)
- `ExitGames::Common::Base::toString(bool withTypes=false) const` (defined in Channel)
- `typeToString(void)` const
- `~Base(void)`
- `~ToString(void)`
## Client Member List

This is the complete list of members for `Client`, including all inherited members.

```cpp
Client(Listener &listener, const Common::JString &applicationID, const Common::JString &appVersion,
       byte connectionProtocol=Photon::ConnectionProtocol::DEFAULT)
connect(const AuthenticationValues &authenticationValues=AuthenticationValues(),
        const Common::JString &nameServerAddress=M_NAMESERVER)
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 pingResult)
opAddFriends(const Common::JVector< Common::JString > &userIDs)
opPublishMessage(const Common::JString &channelName, const Ftype &message)
opPublishMessage(const Common::JString &channelName, const Ftype &MessageArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)
opPublishMessage(const Common::JString &channelName, const Ftype &MessageArray, const short *pArrSizes)
opRemoveFriends(const Common::JVector< Common::JString > &userIDs)
opSendPrivateMessage(const Common::JString &userName, const Ftype &message, bool encrypt=false)
opSendPrivateMessage(const Common::JString &userName, const Ftype &messageArray, const short *pArrSizes, bool encrypt=false)

opSetOnlineStatus(int status)

opSetOnlineStatus(int status, const Ftype &message) (defined in Client)

opSetOnlineStatus(int status, const Ftype pMessageArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)

opSetOnlineStatus(int status, const Ftype pMessageArray, const short *pArrSizes)

opSubscribe(const Common::JVector<Common::JString> &channels, int messagesFromHistory=0)

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 &formatOptions)

setQuickResendAttempts(nByte quickResendAttempts)

setRegion(const Common::JString &region)

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::JVector<Common::JString> &senders, const Common::JVector<Common::Object> &messages)=0`
- `onPrivateMessage(const Common::JString &sender, const Common::Object &message, const Common::JString &channelName)=0`
- `onStateChange(int state)=0`
- `onStatusUpdate(const Common::JString &user, int status, bool gotMessage, const Common::Object &message)=0`
- `serverErrorReturn(int errorCode)=0` (defined in Listener)
- `subscribeReturn(const Common::JVector<Common::JString> &channels, const Common::JVector<bool> &results)=0`
- `unsubscribeReturn(const Common::JVector<Common::JString> &channels)=0`
- `warningReturn(int warningCode)=0` (defined in Listener)
- `~BaseListener()` (defined in BaseListener)
- `~Listener(void)` (defined in Listener)
Peer Member List

This is the complete list of members for Peer, including all inherited members.

- `connect(const Common::JString &ipAddr, const Common::JString &appID=Common::JString())`
- `connect(const Common::JString &ipAddr, const Common::JString &appID, const Ftype &customData)`
- `connect(const Common::JString &ipAddr, const Common::JString &appID, const Ftype &customDataArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)`
- `connect(const Common::JString &ipAddr, const Common::JString &appID, const Ftype &customDataArray, const short *pArrSizes)`
- `disconnect(void)`
- `dispatchIncomingCommands(void)`
- `establishEncryption(void)`
- `fetchServerTimestamp(void)`
- `getBytesIn(void) const`
- `getBytesOut(void) const`
getListener(void)
getLogFormatOptions(void) const
getMaxApplIDLength(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, const Common::JVector<nByte> &HMACSecret)
initUserDataEncryption(const Common::JVector<nByte> &secret)
opAddFriends(const Common::JVector<Common::JString> &userIDs) (defined in
opAuthenticateOnFrontEnd(const Common::JString &secret) (defined in
opAuthenticateOnNameServer(const Common::JString &appID, const Common::JString &appVersion, const Common::JString &region, const AuthenticationValues &authenticationValues=AuthenticationValues()) (defined in
opCustom(const OperationRequest &operationRequest, bool sendReliable, nByte channelID=0, bool encrypt=false)
opPublishMessage(const Common::JString &channelName, const Ftype &message) (defined in
opPublishMessage(const Common::JString &channelName, const Ftype pMessageArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)

opPublishMessage(const Common::JString &channelName, const Ftype pMessageArray, const short *pArrSizes)

opRemoveFriends(const Common::JVector<Common::JString> &userIDs)

opSendPrivateMessage(const Common::JString &userName, const Ftype &message, bool encrypt=false)

opSendPrivateMessage(const Common::JString &userName, const Ftype pMessageArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, bool encrypt=false)

opSetOnlineStatus(int status)

opSetOnlineStatus(int status, const Ftype &message)

opSetOnlineStatus(int status, const Ftype pMessageArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)

opSetOnlineStatus(int status, const Ftype pMessageArray, const short *pArrSizes)

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

opUnsubscribe(const Common::JVector<Common::JString> &channels)

PhotonPeer(Photon::PhotonListener &listener, nByte connectionProtocol=Photon::ConnectionProtocol::DEFAULT)

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

~Peer(void) (defined in Peer)

~PhotonPeer(void)
## AllocatorInterface Member List

This is the complete list of members for `AllocatorInterface`, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alloc(size_t size)=0</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
<tr>
<td><code>dealloc(void *p)=0</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
<tr>
<td><code>get(void)</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
<tr>
<td><code>resize(void *p, size_t size)=0</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
<tr>
<td><code>setMaxAllocSize(size_t maxAllocSize)=0</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
<tr>
<td><code>~AllocatorInterface(void)</code></td>
<td><code>AllocatorInterface</code></td>
</tr>
</tbody>
</table>
**ANSIString Member List**

This is the complete list of members for `ANSIString`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSIString(void)</td>
<td>ANSIString, ANSIString const ANSIString &amp;str, ANSIString const JString &amp;str, ANSIString const char *str, ANSIString const EG_CHAR *str</td>
</tr>
<tr>
<td>BaseCharString()</td>
<td>BaseCharString, cstr(void) const</td>
</tr>
<tr>
<td>operator const char *(void) const</td>
<td>ANSIString, operator JString(void) const</td>
</tr>
<tr>
<td>operator=(const ANSIString &amp;Rhs)</td>
<td>ANSIString, operator=(const JString &amp;Rhs)</td>
</tr>
<tr>
<td>operator=(const char *Rhs)</td>
<td>ANSIString, operator=(const EG_CHAR *Rhs)</td>
</tr>
<tr>
<td>operator=(const EG_CHAR *Rhs)</td>
<td>ANSIString, operator=(const char *Rhs)</td>
</tr>
<tr>
<td>setDebugOutputLevel(int debugLevel)</td>
<td>BaseCharString, setListener(const BaseListener *baseListener)</td>
</tr>
<tr>
<td>setListener(const BaseListener *baseListener)</td>
<td>BaseCharString, setLogFormatOptions(const LogFormatOptions &amp;options)</td>
</tr>
<tr>
<td>setLogFormatOptions(const LogFormatOptions &amp;options)</td>
<td>BaseCharString, size(void) const</td>
</tr>
</tbody>
</table>
**toString**(JString &retStr, bool withTypes=false) const

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~ANSIString(void)

~Base(void)

~BaseCharString(void)

~ToString(void)
This is the complete list of members for **Base**, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const =0</code></td>
<td>ToString</td>
</tr>
<tr>
<td><code>toString(bool withTypes=false) const</code></td>
<td>ToString</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToString</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToString</td>
</tr>
</tbody>
</table>
# BaseCharString Member List

This is the complete list of members for `BaseCharString`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseCharString()</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>cstr(void) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>JStringRepresentation(void) const =0</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>length(void) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>operator const char *(void) const =0</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>operator JString(void) const =0</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>size(void) const =0</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~BaseCharString(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToS</td>
</tr>
</tbody>
</table>
BaseListener Member List

This is the complete list of members for `BaseListener`, including all inherited members.

- `debugReturn(int debugLevel, const JString &string)=0` [BaseListener]
- `~BaseListener()` (defined in `BaseListener`) [BaseListener]

© Exit Games®, all rights reserved.
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=false) const
- TypeCode
- toString(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=false)
- ExitGames::Common::Base::toString(bool withTypes=false) const
- typeToString(void) const
- ~Base(void)
- ~ToString(void)
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DeSerializer(const nByte *data, int size)</code></td>
<td><code>DeS</code></td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>pop(Object &amp;object)</code></td>
<td><code>DeS</code></td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
<td><code>DeS</code></td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td><code>ToS</code></td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td><code>ToS</code></td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td><code>Bas</code></td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td><code>ToS</code></td>
</tr>
</tbody>
</table>
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)
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)
ExitGames::Common::DictionaryBase::getValue(const FKeyType &key)
getValueDimensions(void) const
getValueSizes(const FKeyType &key) const
getValueTypes(void) const
operator!=(const Dictionary< EKeyType, EValueType > &toCompare) const
ExitGames::Common::DictionaryBase::operator!=(const DictionaryBase &toCompare) const
ExitGames::Common::DictionaryBase::operator=(const DictionaryBase &toCopy)
ExitGames::Common::DictionaryBase::operator==(const DictionaryBase &toCompare) 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::Helpers::ArrayLengthType< EValueType >::type size)
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::DictionaryBase::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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contains(const FKeyType &amp;key) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>DictionaryBase(const DictionaryBase &amp;toCopy)</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>getHashtable(void) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getKeys(const FKeyType *) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getKeys(const Object *) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getKeyTypes(void) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>getSize(void) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getValue(const FKeyType &amp;key, const FValueType *) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getValue(const FKeyType &amp;key, const Object *) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getValueDimensions(void) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getValueSizes(const FKeyType &amp;key) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>getValueTypes(void) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>operator!=(const DictionaryBase &amp;toCompare) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>operator=(const DictionaryBase &amp;toCopy)</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>operator==(const DictionaryBase &amp;toCompare) const</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>remove(const FKeyType &amp;key)</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>removeAllElements(void)</code></td>
<td>DictionaryBase</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Base</td>
</tr>
</tbody>
</table>
setLogFormatOptions(const LogFormatOptions &options)

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

ExitGames::Common::Base::toString(bool withTypes=false) const

typeToString(void) const

~Base(void)

~DictionaryBase(void)

~ToString(void)
This is the complete list of members for **EGTime**, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGTime(int time)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>EGTime(const EGTime &amp;toCopy)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>getDebugOutputLevel(void)</td>
<td>Base member</td>
</tr>
<tr>
<td>getLogFormatOptions(void)</td>
<td>Base member</td>
</tr>
<tr>
<td>operator!=(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator+(const EGTime &amp;time)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator+=(const EGTime &amp;time)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator-=(const EGTime &amp;time)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator&lt;=(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator&lt;=(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator=(const EGTime &amp;toCopy)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator=(const int &amp;time)</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator===(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator&gt;=(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>operator&gt;=(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>overflowed(const EGTime &amp;time) const</td>
<td>EGTime member</td>
</tr>
<tr>
<td>setDebugOutputLevel(int debugLevel)</td>
<td>Base member</td>
</tr>
<tr>
<td>setListener(const BaseListener *baseListener)</td>
<td>Base member</td>
</tr>
<tr>
<td>setLogFormatOptions(const LogFormatOptions &amp;options)</td>
<td>Base member</td>
</tr>
<tr>
<td>toString(JString &amp;retStr, bool withTypes=false) const</td>
<td>EGTime member</td>
</tr>
</tbody>
</table>
ExitGames::Common::Base::toString(bool withTypes=false) const ToS

typeToString(void) const ToS

~Base(void) Bas

~EGTime(void) EG1

~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::Helpers::ArrayLengthType<FValueType>::type size)`  
- `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)
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 ANSIString &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 JString &string)
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 ANSIString &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)
### 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
- **getCArrray**(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 initialCapacity=0, unsigned int capacityIncrement=1)

**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)
ExitGames  Common  KeyObject

**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) const`
- `getDimensions(void) const`
- `getLogFormatOptions(const LogFormatOptions &options)`
- `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 &data)`
- `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) const`
- `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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getAddDateTime(void) const</code></td>
<td>Return add date time option</td>
</tr>
<tr>
<td><code>getAddFile(void) const</code></td>
<td>Return add file option</td>
</tr>
<tr>
<td><code>getAddFunction(void) const</code></td>
<td>Return add function option</td>
</tr>
<tr>
<td><code>getAddLevel(void) const</code></td>
<td>Return add level option</td>
</tr>
<tr>
<td><code>getAddLine(void) const</code></td>
<td>Return add line option</td>
</tr>
<tr>
<td><code>getMaxNumberOfNamespaces(void) const</code></td>
<td>Return max number of namespaces option</td>
</tr>
<tr>
<td>LogFormatOptions(void) (defined in LogFormatOptions)</td>
<td>Constructor</td>
</tr>
<tr>
<td><code>setAddDateTime(bool addTime)</code></td>
<td>Set add date time option</td>
</tr>
<tr>
<td><code>setAddFile(bool addFile)</code></td>
<td>Set add file option</td>
</tr>
<tr>
<td><code>setAddFunction(bool addFunction)</code></td>
<td>Set add function option</td>
</tr>
<tr>
<td><code>setAddLevel(bool addLevel)</code></td>
<td>Set add level option</td>
</tr>
<tr>
<td><code>setAddLine(bool addLine)</code></td>
<td>Set add line option</td>
</tr>
<tr>
<td><code>setMaxNumberOfNamespaces(unsigned int maxNumberOfNamespaces)</code></td>
<td>Set max number of namespaces option</td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Convert to string</td>
</tr>
<tr>
<td><code>ExitGames::Common::ToString::toString(bool withTypes=false) const</code></td>
<td>Method for converting</td>
</tr>
<tr>
<td><code>typeToString(void)</code></td>
<td>Convert type to string</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>Constructor</td>
</tr>
</tbody>
</table>

© Exit Games®, all rights reserved.
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 printBrackets, unsigned int line, const EG_CHAR *dbgMsg,...) const`
- `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 printBrackets, unsigned int line, const EG_CHAR *dbgMsg, va_list args) const`
- `~Logger(void) (defined in Logger)`
- `~ToString(void)`
## Object Member List

This is the complete list of members for Object, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCustomType(void) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getDimensions(void) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getSizes(void) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>getType(void) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>Object(void)</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>Object(const Object &amp;toCopy)</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>operator!=(const Object &amp;toCompare) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>operator=(const Object &amp;toCopy)</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>operator==(const Object &amp;toCompare) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~Object(void)</code></td>
<td>Obj</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToS</td>
</tr>
</tbody>
</table>
Serializer Member List

This is the complete list of members for **Serializer**, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getData(void) const</code></td>
<td>Get data</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Get debug output level</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Get log format options</td>
</tr>
<tr>
<td><code>getSize(void) const</code></td>
<td>Get size</td>
</tr>
<tr>
<td><code>push(const T &amp;data)</code></td>
<td>Push data with type T</td>
</tr>
<tr>
<td><code>push(const T pData, typename Helpers::ArrayLengthType&lt; T &gt;::type arraySize)</code></td>
<td>Push data with type T and array size</td>
</tr>
<tr>
<td><code>push(const T pData, const short *arraySizes)</code></td>
<td>Push data with type T and custom array sizes</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Set debug output level</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Set listener</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Set log format options</td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Convert to string</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>Convert to string</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>Type string</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Destructor of Base class</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>Destructor of ToString class</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
ToString Member List

This is the complete list of members for `ToString`, including all inherited members.

- `toString(JString &retStr, bool withTypes=false) const =0` (pure virtual)
- `toString(bool withTypes=false) const` (virtual)
- `typeToString(void) const` (virtual)
- `~ToString(void)`
UTF8String Member List

This is the complete list of members for **UTF8String**, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BaseCharString()</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>cstr(void) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>JStringRepresentation(void) const</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>length(void) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>operator const char *(void) const</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>operator JString(void) const</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>operator=(const UTF8String &amp;Rhs)</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>operator=(const JString &amp;Rhs)</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>operator=(const char *Rhs)</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>operator=(const EG_CHAR *Rhs)</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>size(void) const</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>size(const JString &amp;str)</code></td>
<td>UTF</td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>UTF8String(void)</code></td>
<td>UTF</td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>UTF8String(const UTF8String &amp;str)</td>
<td>UTF8String</td>
</tr>
<tr>
<td>UTF8String(const JString &amp;str)</td>
<td>UTF8String</td>
</tr>
<tr>
<td>UTF8String(const char *str)</td>
<td>UTF8String</td>
</tr>
<tr>
<td>UTF8String(const EG_CHAR *str)</td>
<td>UTF8String</td>
</tr>
<tr>
<td>~Base(void)</td>
<td>Base</td>
</tr>
<tr>
<td>~BaseCharString(void)</td>
<td>BaseCharString</td>
</tr>
<tr>
<td>~ToString(void)</td>
<td>ToString</td>
</tr>
<tr>
<td>~UTF8String(void)</td>
<td>UTF8String</td>
</tr>
</tbody>
</table>
ValueObject< Etype > Member List

This is the complete list of members for ValueObject< Etype >, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCustomType(void) const</code></td>
</tr>
<tr>
<td><code>getDataAddress(void) const</code></td>
</tr>
<tr>
<td><code>getDataCopy(void) const</code></td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
</tr>
<tr>
<td><code>getDimensions(void) const</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
</tr>
<tr>
<td><code>getSizes(void) const</code></td>
</tr>
<tr>
<td><code>getType(void) const</code></td>
</tr>
<tr>
<td><code>Object(void)</code></td>
</tr>
<tr>
<td><code>Object(const Object &amp;toCopy)</code></td>
</tr>
<tr>
<td><code>operator!=(const Object &amp;toCompare) const</code></td>
</tr>
<tr>
<td><code>operator=(const ValueObject&lt; Etype &gt; &amp;toCopy)</code></td>
</tr>
<tr>
<td><code>operator=(const Object &amp;toCopy)</code></td>
</tr>
<tr>
<td><code>operator==(const Object &amp;toCompare) const</code></td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
</tr>
<tr>
<td><code>toString(JString &amp;retStr, bool withTypes=false) const</code></td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString</code></td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
</tr>
<tr>
<td><code>ValueObject(const ValueObject&lt; Etype &gt; &amp;toCopy)</code></td>
</tr>
</tbody>
</table>
ValueObject(const Object &obj)
ValueObject(const Object *obj)
ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type &data)
ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type pData,
typename Helpers::ArrayLengthType< Etype >::type size)
ValueObject(const typename Helpers::ConfirmAllowed< Etype >::type pData,
const short *sizes)
~Base(void)
~Object(void)
~ToString(void)
~ValueObject(void)
This is the complete list of members for LitePeer, including all inherited members.

```cpp
connect(const Common::JString &ipAddr, const Common::JString &appID=Common::JString())
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype &customData)
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype pCustomDataArray,
        typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype pCustomDataArray,
        const short *pArrSizes)
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
canCRCEnabled(void) const
canDebugOutputLevel(void) const
canDisconnectTimeOut(void) const
canIncomingReliableCommandsCount(void) const
canIsEncryptionAvailable(void) const
canIsPayloadEncryptionAvailable(void) const
canLimitOfUnreliableCommands(void) const
```
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getListener(void)</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions(void) const</code></td>
</tr>
<tr>
<td><code>getMaxApplIDLength(void)</code></td>
</tr>
<tr>
<td><code>getPacketLossByCRC(void) const</code></td>
</tr>
<tr>
<td><code>getPeerCount(void)</code></td>
</tr>
<tr>
<td><code>getPeerID(void) const</code></td>
</tr>
<tr>
<td><code>getPeerState(void) const</code></td>
</tr>
<tr>
<td><code>getQueuedIncomingCommands(void) const</code></td>
</tr>
<tr>
<td><code>getQueuedOutgoingCommands(void) const</code></td>
</tr>
<tr>
<td><code>getQuickResendAttempts(void) const</code></td>
</tr>
<tr>
<td><code>getResentReliableCommands(void) const</code></td>
</tr>
<tr>
<td><code>getRoundTripTime(void) const</code></td>
</tr>
<tr>
<td><code>getRoundTripTimeVariance(void) const</code></td>
</tr>
<tr>
<td><code>getSentCountAllowance(void) const</code></td>
</tr>
<tr>
<td><code>getServerAddress(void) const</code></td>
</tr>
<tr>
<td><code>getServerTime(void) const</code></td>
</tr>
<tr>
<td><code>getServerTimeOffset(void) const</code></td>
</tr>
<tr>
<td><code>getTimePingInterval(void) const</code></td>
</tr>
<tr>
<td><code>getTimestampOfLastSocketReceive(void) const</code></td>
</tr>
<tr>
<td><code>getTrafficStatsElapsedMs(void) const</code></td>
</tr>
<tr>
<td><code>getTrafficStatsEnabled(void) const</code></td>
</tr>
<tr>
<td><code>getTrafficStatsGameLevel(void) const</code></td>
</tr>
<tr>
<td><code>getTrafficStatsIncoming(void) const</code></td>
</tr>
<tr>
<td><code>getTrafficStatsOutgoing(void) const</code></td>
</tr>
<tr>
<td><code>initUDPEncryption(const Common::JVector&lt;nByte&gt; &amp;encryptSecret, const Common::JVector&lt;nByte&gt; &amp;HMACSecret)</code></td>
</tr>
<tr>
<td><code>initUserDataEncryption(const Common::JVector&lt;nByte&gt; &amp;secret)</code></td>
</tr>
<tr>
<td><code>LitePeer(Photon::PhotonListener &amp;listener, nByte connectionProtocol=Photon::ConnectionProtocol::DEFAULT)</code></td>
</tr>
<tr>
<td><code>opChangeGroups(const Common::JVector&lt;nByte&gt; *pGroupsToRemove, const Common::JVector&lt;nByte&gt; *pGroupsToAdd)</code></td>
</tr>
<tr>
<td><code>opCustom(const OperationRequest &amp;operationRequest, bool sendReliable, nByte channelID=0)</code></td>
</tr>
<tr>
<td><code>opGetProperties(nByte channelID=0)</code></td>
</tr>
<tr>
<td><code>opGetPropertiesOfActor(const Common::JString *properties, short num)</code></td>
</tr>
</tbody>
</table>
opGetPropertiesOfActor(const nByte *properties, short numProperties, const int *actorNrList=NULL, short numActors=0, nByte channelID=0)
opGetPropertiesOfGame(const Common::JString *properties, short numProperties, nByte channelID=0)
opGetPropertiesOfGame(const nByte *properties, short numProperties, nByte channelID=0)
opJoin(const Common::JString &gameId, const Common::Hashtable &gameProperties=Common::Hashtable(), const Common::Hashtable &actorProperties=Common::Hashtable(), bool broadcastActorProperties=false)
opLeave(void)
opRaiseEvent(bool reliable, Ftype parameters, nByte eventCode, nByte channelID=0, nByte eventCaching=EventCache::DO_NOT_CACHE, nByte *targetPlayers=NULL, short numTargetPlayers=0, nByte receiverGroup=ReceiverGroup::OTHERS, nByte interestGroup=0)
opRaiseEvent(bool reliable, Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, nByte eventCode, nByte channelID=0, nByte eventCaching=EventCache::DO_NOT_CACHE, nByte *targetPlayers=NULL, short numTargetPlayers=0, nByte receiverGroup=ReceiverGroup::OTHERS, nByte interestGroup=0)
opRaiseEvent(bool reliable, Ftype pParameterArray, const short *pArrSizes, nByte eventCode, nByte channelID=0, nByte eventCaching=EventCache::DO_NOT_CACHE, nByte *targetPlayers=NULL, short numTargetPlayers=0, nByte receiverGroup=ReceiverGroup::OTHERS, nByte interestGroup=0)
opSetPropertiesOfActor(int actorNr, const Common::Hashtable &properties, bool broadcast, nByte channelID=0)
opSetPropertiesOfGame(const Common::Hashtable &properties, bool broadcast, nByte channelID=0)
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=ConnectionProtocol::DEFAULT)
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 trafficStasEnabled)
vitalStatsToString(bool all) const
~LitePeer(void)
~PhotonPeer(void)
AuthenticationValues Member List

This is the complete list of members for AuthenticationValues, including all inherited members.

AuthenticationValues( void )
get( Data() 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 &parameters )
setParametersWithUsernameAndToken( const Common::JString &username, const Common::JString &token )
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 )
Photon C++
Client API 4.1.12.2

Client Member List

This is the complete list of members for Client, including all inherited members.

Client(LoadBalancing::Listener &listener, const Common::JString &applicationID, const Common::JString &appVersion, ... ,
connnect(const AuthenticationValues &authenticationValues=AuthenticationValues(),
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
g.getServerTime(void) const
g.getServerTimeOffset(void) const
getState(void) const
g.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, const Common::JVector< nByte > *pGroupsToAdd)

opCreateRoom (const Common::JString &gameID, const RoomOptions &options=RoomOptions(), const Common::JVector< Common::JString > &expectedUsers=Common::JVector< Common::JString >())

opCustom (const Photon::OperationRequest &operationRequest, bool sendReliable, nByte channelID=0, bool encrypt=false)

opCustomAuthenticationSendNextStepData (const AuthenticationValues &authenticationValues)

opFindFriends (const Common::JString *friendsToFind, short numFriendsToFind)

opJoinLobby (const Common::JString &lobbyName=Common::JString(), nByte lobbyType=LobbyType::DEFAULT)

opJoinOrCreateRoom (const Common::JString &gameID, const RoomOptions &options=RoomOptions(), int cacheSliceIndex=0, const Common::JVector< Common::JString > &expectedUsers=Common::JVector< Common::JString >())

opJoinRandomRoom (const Common::Hashtable &customRoomProperties=Common::Hashtable(), nByte maxPlayers=0, nByte matchmakingMode=Matchmaker::MATCHMAKING_MODE_DEFAULT, const Common::JString &gameID=Common::JString(), const Common::JVector< Common::JString > &expectedUsers=Common::JVector< Common::JString >())

opJoinRoom (const Common::JString &gameID, bool rejoin=false, int cacheSliceIndex=0, const Common::JVector< Common::JString > &expectedUsers=Common::JVector< Common::JString >())

opLeaveLobby (void)

opLeaveRoom (bool willComeBack=false, bool sendAuthCookie=false)

opLobbyStats (const Common::JVector< LoadBalancing::LobbyStatsRequest > &lobbiesToQuery=Common::JVector< LoadBalancing::LobbyStatsRequest >())

opRaiseEvent (bool reliable, const Ftype &parameters, nByte eventCode, const RaiseEventOptions &options=RaiseEventOptions())

opRaiseEvent (bool reliable, const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType< Ftype >::type arrSize, nByte eventCode, const RaiseEventOptions &options=RaiseEventOptions())

opRaiseEvent (bool reliable, const Ftype pParameterArray, const short *pArrSizes, nByte eventCode, const RaiseEventOptions &options=RaiseEventOptions())

opWebRpc (const Common::JString &uriPath)

opWebRpc (const Common::JString &uriPath, const Ftype &parameters, bool sendAuthCookie=false)

opWebRpc (const Common::JString &uriPath, const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType< Ftype >::type arrSize, bool sendAuthCookie=false)

opWebRpc (const Common::JString &uriPath, const Ftype pParameterArray, const short *pArrSizes, bool sendAuthCookie=false)

reconnectAndRejoin (void)

resetTrafficStats (void)

resetTrafficStatsMaximumCounters (void)

selectRegion (const Common::JString &selectedRegion)

sendAcksOnly (void)

sendDirect (const Ftype &parameters, int targetPlayer, bool fallbackRelay=false)

sendDirect (const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType< Ftype >::type arrSize, int targetPlayer, bool fallbackRelay=false)

sendDirect (const Ftype pParameterArray, const short *pArrSizes, int targetPlayer, bool fallbackRelay=false)

sendDirect (const Ftype &parameters, const Common::JVector< int > &targetPlayers, bool fallbackRelay=false)
sendDirect(const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize, const Common::JVector<int>& targetPlayers=Common::JVector<int>(), bool fallbackRelay=false)
sendDirect(const Ftype pParameterArray, const short*pArrSizes, const Common::JVector<int>& targetPlayers=Common::JVector<int>(), bool fallbackRelay=false)
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& formatOptions)
setQuickResendAttempts(nByte quickResendAttempts)
setSentCountAllowance(int sentCountAllowance)
setTimePingInterval(int timePingInterval)
setTrafficStatsEnabled(bool trafficStatsEnabled)
vitalStatsToString(bool all) const
~BaseListener() (defined in BaseListener)
~Client(void)
~PhotonListener(void)
FriendInfo Member List

This is the complete list of members for **FriendInfo**, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getIsInRoom(void) const</code></td>
<td>Frie</td>
</tr>
<tr>
<td><code>getIsOnline(void) const</code></td>
<td>Frie</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getRoom(void) const</code></td>
<td>Frie</td>
</tr>
<tr>
<td><code>getUserID(void) const</code></td>
<td>Frie</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Frie</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToS</td>
</tr>
</tbody>
</table>
### 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**)
- `createRoomReturn(int errorCode, const Common::JString &errorString, const Common::Hashtable &roomProperties)`: 0
- `customEventAction(int playerNr, nByte eventCode, const Common::Object &eventContent)=0`
- `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 &roomProperties, const Common::Hashtable &playerProperties, int errorCode, const Common::JString &errorString)=0`
- `joinRandomRoomReturn(int localPlayerNr, const Common::Hashtable &roomProperties, const Common::Hashtable &playerProperties, int errorCode, const Common::JString &errorString)=0`
- `joinRoomEventAction(int playerNr, const Common::JVector<int> &playernrs, const Player &player)=0`
- `joinRoomReturn(int localPlayerNr, const Common::Hashtable &roomProperties, const Common::Hashtable &playerProperties, int errorCode, const Common::JString &errorString)=0`
- `leaveLobbyReturn(void)=0` (defined in **Listener**)
- `leaveRoomEventAction(int playerNr, bool isInactive)=0` (defined in **Listener**)
- `leaveRoomReturn(int errorCode, const Common::JString &errorString)=0`
- `onAppStatsUpdate(void)` (defined in **Listener**)
- `onAvailableRegions(const Common::JVector<Common::JString>&, const Common::JVector<Common::JString>&)=0`
- `onCacheSliceChanged(int)` (defined in **Listener**)
- `onCustomAuthenticationIntermediateStep(const Common::Dictionary<Common::JString, Common::Object>&)=0`
- `onCustomOperationResponse(const Photon::OperationResponse &operationResponse)=0`
- `onDirectMessage(const Common::Object &, int, bool)` (defined in **Listener**)
- `onEnterRoomEventAction(int playerNr, const Common::Hashtable &playerProperties, int errorCode, const Common::JString &errorString)=0`
- `onEnterRoomReturn(int localPlayerNr, const Common::Hashtable &roomProperties, const Common::Hashtable &playerProperties, int errorCode, const Common::JString &errorString)=0`
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onFindFriendsResponse(void)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onLobbyStatsResponse(const Common::JVector&lt; LobbyStatsResponse &gt; &amp;)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onLobbyStatsUpdate(const Common::JVector&lt; LobbyStatsResponse &gt; &amp;)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onMasterClientChanged(int, int)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onPlayerPropertiesChange(int, const Common::Hashtable &amp;)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onRoomListUpdate(void)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onRoomPropertiesChange(const Common::Hashtable &amp;)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>onSecretReceival(const Common::JString &amp;)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>serverErrorReturn(int errorCode)=0</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>warningReturn(int warningCode)=0</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
<tr>
<td><code>webRpcReturn(int, const Common::JString &amp;, const Common::JString &amp;, int, const Common::Dictionary&lt; Common::Object, Common::Object &gt; &amp;)</code></td>
<td>(defined in <code>~BaseListener()</code> (defined in <code>BaseListener</code>))</td>
</tr>
<tr>
<td><code>~BaseListener()</code></td>
<td>(defined in <code>BaseListener</code>)</td>
</tr>
<tr>
<td><code>~Listener(void)</code></td>
<td>(defined in <code>Listener</code>)</td>
</tr>
</tbody>
</table>
## LobbyStatsRequest Member List

This is the complete list of members for `LobbyStatsRequest`, including all inherited members.

<table>
<thead>
<tr>
<th>Member Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
</tr>
<tr>
<td><code>getName(void) const</code></td>
</tr>
<tr>
<td><code>getType(void) const</code></td>
</tr>
<tr>
<td><code>LobbyStatsRequest(const Common::JString &amp;name=Common::JString())</code></td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
</tr>
</tbody>
</table>
LobbyStatsResponse Member List

This is the complete list of members for LobbyStatsResponse, including all inherited members.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDebugOutputLevel(void)</td>
<td>Base</td>
</tr>
<tr>
<td>getLogFormatOptions(void)</td>
<td>Base</td>
</tr>
<tr>
<td>getName(void) const</td>
<td>Lobby</td>
</tr>
<tr>
<td>getPeerCount(void) const</td>
<td>Lobby</td>
</tr>
<tr>
<td>getRoomCount(void) const</td>
<td>Lobby</td>
</tr>
<tr>
<td>getType(void) const</td>
<td>Lobby</td>
</tr>
<tr>
<td>setDebugOutputLevel(int debugLevel)</td>
<td>Base</td>
</tr>
<tr>
<td>setListener(const BaseListener *baseListener)</td>
<td>Base</td>
</tr>
<tr>
<td>setLogFormatOptions(const LogFormatOptions &amp;options)</td>
<td>Base</td>
</tr>
<tr>
<td>toString(Common::JString &amp;retStr, bool withTypes=false) const</td>
<td>Lobby</td>
</tr>
<tr>
<td>ExitGames::Common::Base::toString(bool withTypes=false) const</td>
<td>ToS</td>
</tr>
<tr>
<td>typeToString(void) const</td>
<td>ToS</td>
</tr>
<tr>
<td>~Base(void)</td>
<td>Bas</td>
</tr>
<tr>
<td>~ToString(void)</td>
<td>ToS</td>
</tr>
</tbody>
</table>
# MutablePlayer Member List

This is the complete list of members for `MutablePlayer`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addCustomProperties</code></td>
<td><code>addCustomProperties(const Common::Hashtable &amp;customProperties, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>addCustomProperty</code></td>
<td><code>addCustomProperty(const ktype &amp;key, const vtype &amp;value, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>addCustomProperty</code></td>
<td><code>addCustomProperty(const ktype &amp;key, const vtype &amp;valueArray, typename Common::Helpers::ArrayLengthType&lt;vtype&gt;::type arrSize, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>addCustomProperty</code></td>
<td><code>addCustomProperty(const ktype &amp;key, const vtype &amp;valueArray, const short *pArrSizes, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>getCustomProperties</code></td>
<td><code>getCustomProperties() const</code></td>
</tr>
<tr>
<td><code>getDebugOutputLevel</code></td>
<td><code>getDebugOutputLevel(void)</code></td>
</tr>
<tr>
<td><code>getIsInactive</code></td>
<td><code>getIsInactive(void) const</code></td>
</tr>
<tr>
<td><code>getIsMasterClient</code></td>
<td><code>getIsMasterClient(void) const</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions</code></td>
<td><code>getLogFormatOptions(void)</code></td>
</tr>
<tr>
<td><code>getName</code></td>
<td><code>getName() const</code></td>
</tr>
<tr>
<td><code>getNumber</code></td>
<td><code>getNumber(void) const</code></td>
</tr>
<tr>
<td><code>getUserID</code></td>
<td><code>getUserID() const</code></td>
</tr>
<tr>
<td><code>mergeCustomProperties</code></td>
<td><code>mergeCustomProperties(const Common::Hashtable &amp;customProperties, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>MutablePlayer</code></td>
<td><code>MutablePlayer(const MutablePlayer &amp;toCopy)</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>operator=</code></td>
<td><code>operator=(const Player &amp;toCopy)</code></td>
</tr>
<tr>
<td><code>operator=</code></td>
<td><code>operator=(const MutablePlayer &amp;toCopy)</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>operator==</code></td>
<td><code>operator==(const Player &amp;player) const</code></td>
</tr>
<tr>
<td><code>Player</code></td>
<td><code>Player(const Player &amp;toCopy)</code></td>
</tr>
<tr>
<td><code>removeCustomProperties</code></td>
<td><code>removeCustomProperties(const ktype *keys, unsigned int count, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>removeCustomProperty</code></td>
<td><code>removeCustomProperty(const ktype &amp;key, const WebFlags &amp;webflags=WebFlags())</code> (defined in <code>MutablePlayer</code>)</td>
</tr>
<tr>
<td><code>setDebugOutputLevel</code></td>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>setListener</code></td>
<td>Seta listener.</td>
</tr>
<tr>
<td><code>setLogFormatOptions</code></td>
<td>Set log format options.</td>
</tr>
<tr>
<td><code>setName</code></td>
<td>Set name.</td>
</tr>
<tr>
<td><code>toString</code></td>
<td>Get string representation.</td>
</tr>
<tr>
<td><code>~Base</code></td>
<td>Destructor.</td>
</tr>
<tr>
<td><code>~MutablePlayer</code></td>
<td>Destructor (defined in <code>MutablePlayer</code>).</td>
</tr>
<tr>
<td><code>~Player</code></td>
<td>Destructor.</td>
</tr>
<tr>
<td><code>~ToString</code></td>
<td>Destructor.</td>
</tr>
</tbody>
</table>
## MutableRoom Member List

This is the complete list of members for **MutableRoom**, including all inherited members.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>addCustomProperties</strong></td>
<td>(const Common::Hashtable &amp;customProperties, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td>(defined in <strong>MutableRoom</strong>)</td>
</tr>
<tr>
<td><strong>addCustomProperty</strong></td>
<td>(const ktype &amp;key, const vtype &amp;value, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td>(defined in <strong>MutableRoom</strong>)</td>
</tr>
<tr>
<td><strong>addCustomProperty</strong></td>
<td>(const ktype &amp;key, const vtype pValueArray, typename Common::Helpers::ArrayLengthType&lt;vtype&gt;::type arrSize, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td>(defined in <strong>MutableRoom</strong>)</td>
</tr>
<tr>
<td><strong>addCustomProperty</strong></td>
<td>(const ktype &amp;key, const vtype pValueArray, const short *pArrSizes, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td>(defined in <strong>MutableRoom</strong>)</td>
</tr>
<tr>
<td><strong>getCustomProperties</strong></td>
<td>(void) const</td>
<td>(defined in <strong>MutableRoom</strong>)</td>
</tr>
<tr>
<td><strong>getDebugOutputLevel</strong></td>
<td>(void)</td>
<td></td>
</tr>
<tr>
<td><strong>getDirectMode</strong></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><strong>getEmptyRoomTtl</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getExpectedUsers</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getIsOpen</strong></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><strong>getIsVisible</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getLogFormatOptions</strong></td>
<td>(void)</td>
<td></td>
</tr>
<tr>
<td><strong>getMasterClientID</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getMaxPlayers</strong></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><strong>getName</strong></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><strong>getPlayerCount</strong></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><strong>getPlayerForNumber</strong></td>
<td>(int playerNumber) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getPlayers</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getPlayerTtl</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getPlugins</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td><strong>getPropsListedInLobby</strong></td>
<td>(void) const (defined in <strong>MutableRoom</strong>)</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Defined In</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td><code>getPublishUserID()</code></td>
<td>(void) const (defined in <code>MutableRoom</code>)</td>
<td></td>
</tr>
<tr>
<td><code>getSuppressRoomEvents()</code></td>
<td>(void) const (defined in <code>MutableRoom</code>)</td>
<td></td>
</tr>
<tr>
<td><code>mergeCustomProperties()</code></td>
<td>(const Common::Hashtable &amp;customProperties)</td>
<td><code>MutableRoom</code></td>
</tr>
<tr>
<td><code>operator=(const Room &amp;toCopy)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>operator=(const MutableRoom &amp;toCopy)</code></td>
<td>(defined in <code>MutableRoom</code>)</td>
<td></td>
</tr>
<tr>
<td><code>operator==(const Room &amp;room)</code></td>
<td>const</td>
<td></td>
</tr>
<tr>
<td><code>removeCustomProperties()</code></td>
<td>(const ktype *keys, unsigned int count, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td></td>
</tr>
<tr>
<td><code>removeCustomProperty()</code></td>
<td>(const ktype &amp;key, const Common::Hashtable &amp;expectedCustomProperties=Common::Hashtable(), const WebFlags &amp;webflags=WebFlags())</td>
<td></td>
</tr>
<tr>
<td><code>Room()</code></td>
<td>(const Room &amp;toCopy)</td>
<td></td>
</tr>
<tr>
<td><code>setDebugOutputLevel()</code></td>
<td>(int debugLevel)</td>
<td></td>
</tr>
<tr>
<td><code>setExpectedUsers()</code></td>
<td>(const Common::JVector&lt; Common::JString &gt; &amp;expectedUsers, const WebFlags &amp;webflags=WebFlags())</td>
<td></td>
</tr>
<tr>
<td><code>setIsOpen()</code></td>
<td>(bool isOpen, const WebFlags &amp;webflags=WebFlags()) (defined)</td>
<td></td>
</tr>
<tr>
<td><code>setIsVisible()</code></td>
<td>(bool isVisible, const WebFlags &amp;webflags=WebFlags()) (defined)</td>
<td></td>
</tr>
<tr>
<td><code>setListener()</code></td>
<td>(const BaseListener *baseListener)</td>
<td></td>
</tr>
<tr>
<td><code>setLogFormatOptions()</code></td>
<td>(const LogFormatOptions &amp;options)</td>
<td></td>
</tr>
<tr>
<td><code>setMaxPlayers()</code></td>
<td>(nByte maxPlayers, const WebFlags &amp;webflags=WebFlag)</td>
<td></td>
</tr>
<tr>
<td><code>setPropsListedInLobby()</code></td>
<td>(const Common::JVector&lt; Common::JString &gt; &amp;propsListedInLobby, const Common::JVector&lt; Common::JString &gt; &amp;expectedList=Common::JVector&lt; Common::JString &gt;(), const WebFlags &amp;webflags=WebFlags())</td>
<td></td>
</tr>
<tr>
<td><code>toString()</code></td>
<td>(bool withTypes=false, bool withCustomProperties=false, bool withPlayers=false)</td>
<td></td>
</tr>
<tr>
<td><code>ExitGames::LoadBalancing::Room::toString()</code></td>
<td>(Common::JString &amp;retStr)</td>
<td></td>
</tr>
<tr>
<td><code>ExitGames::LoadBalancing::Room::toString()</code></td>
<td>(bool withTypes, bool withPlayers)</td>
<td></td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString()</code></td>
<td>(bool withTypes=false)</td>
<td></td>
</tr>
<tr>
<td><code>typeToString()</code></td>
<td>(void) const</td>
<td></td>
</tr>
<tr>
<td><code>~Base()</code></td>
<td>(void)</td>
<td></td>
</tr>
<tr>
<td><code>~MutableRoom()</code></td>
<td>(void) (defined in <code>MutableRoom</code>)</td>
<td></td>
</tr>
<tr>
<td><code>~Room()</code></td>
<td>(void)</td>
<td></td>
</tr>
<tr>
<td><code>~ToString()</code></td>
<td>(void)</td>
<td></td>
</tr>
</tbody>
</table>
Peer Member List

This is the complete list of members for **Peer**, including all inherited members.

```cpp
connect(const Common::JString &ipAddr, const Common::JString &appID=Common::JString())
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype &customData)
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype &customDataArray,
        typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize)
connect(const Common::JString &ipAddr, const Common::JString &appID,
        const Ftype &customDataArray,
        const short *pArrSizes)
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
getMaxApplIDLength(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, const Common::JVector< nByte > &HMACSecret)
initUserDataEncryption(const Common::JVector< nByte > &secret)
opAuthenticate(const Common::JString &appID, const Common::JString &appVersion, bool encrypted, const AuthenticationValues &au ... ues=AuthenticationValues(), bool lobbyStats=false, const Common::JString &regionCode=Common::JString()) (defined in
opAuthenticateOnce(const Common::JString &appID, const Common::JString &appVersion, nByte connectionProtocol, nByte encryptionMode, bool encrypted, const AuthenticationValues &au ... ues=AuthenticationValues(), bool lobbyStats=false, const Common::JString &regionCode=Common::JString()) (defined in
opChangeGroups(const Common::JVector< nByte > *pGroupsToRemove, const Common::JVector< nByte > *pGroupsToAdd)
opCreateRoom(const Common::JString &gameID, const RoomOptions &options=RoomOptions(), const Common::Hashtable &customLocalPlay ... able(), const Common::JVector< Common::JString > &expectedUsers=Common::JVector< Common::JString >()) (defined in
opCustom(const OperationRequest &operationRequest, bool sendReliable
opFindFriends(const Common::JString *friendsToFind, short numFriendsToFind)
opGetRegions(bool encrypted, const Common::JString &appId) (defined in opGetRegions)
opJoinLobby(const Common::JString &lobbyName=Common::JString(),
              nByte lobbyType=LobbyType::DEFAULT)
opJoinRandomRoom(const Common::Hashtable &customRoomProperties=Common::Hashtable(),
              nByte maxPlayers=0,
              nByte matchmakingMode=MatchmakingMode::DEFAULT,
              const Common::JVector<Common::JString> &expectedUsers=Common::JVector<Common::JString>()) (defined in opJoinRandomRoom)
opJoinRoom(const Common::JString &gameID,
            const RoomOptions &options=RoomOptions(),
            const Common::Hashtable &customLocalPlayProperties=Common::Hashtable(),
            nByte gameIndex=0,
            const Common::JVector<Common::JString> &expectedUsers=Common::JVector<Common::JString>()) (defined in opJoinRoom)
opLeaveLobby(void) (defined in Peer)
opLeaveRoom(bool willComeBack=false, bool sendAuthCookie=false) (defined in Peer)
opLobbyStats(const Common::JVector<LoadBalancing::LobbyStatsRequest> &lobbiesToQuery=Common::JVector<LoadBalancing::LobbyStatsRequest>()) (defined in Peer)
opRaiseEvent(bool reliable, const Ftype &parameters, nByte eventCode,
             const RaiseEventOptions &options=RaiseEventOptions()) (defined in Peer)
opRaiseEvent(bool reliable, const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize,
             nByte eventCode,
             const RaiseEventOptions &options=RaiseEventOptions()) (defined in Peer)
opRaiseEvent(bool reliable, const Ftype pParameterArray, const short *pArrSizes,
             nByte eventCode,
             const RaiseEventOptions &options=RaiseEventOptions()) (defined in Peer)
opSetPropertiesOfPlayer(int playerNr,
                          const Common::Hashtable &properties,
                          const Common::Hashtable &expectedProperties=Common::Hashtable(),
                          WebFlags webFlags=WebFlags()) (defined in Peer)
opSetPropertiesOfRoom(const Common::Hashtable &properties,
                       const Common::Hashtable &expectedProperties=Common::Hashtable(),
                       WebFlags webFlags=WebFlags()) (defined in Peer)
opWebRpc(const Common::JString &uriPath) (defined in Peer)
opWebRpc(const Common::JString &uriPath, const Ftype &parameters, bool sendAuthCookie=false) (defined in Peer)
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArray, typename Common::Helpers::ArrayLengthType<Ftype>::type arrSize,
               bool sendAuthCookie=false) (defined in Peer)
opWebRpc(const Common::JString &uriPath, const Ftype pParameterArray, const short *pArrSizes,
               bool sendAuthCookie=false) (defined in Peer)
PhotonPeer(Photon::PhotonListener &listener,
          nByte connectionProtocol=Photon::ConnectionProtocol::DEFAULT)
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 trafficStasEnabled)
vitalStatsToString(bool all) const
~Peer(void) (defined in Peer)
~PhotonPeer(void)
### Player Member List

This is the complete list of members for `Player`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCustomProperties()</code> const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>getDebugOutputLevel</code> (void)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>getIsInactive</code> (void) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>getIsMasterClient</code> (void) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>getLogFormatOptions</code> (void)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>getName</code> () const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>getNumber</code> (void) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>getUserID</code> () const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>operator=</code> (const <code>Player</code> &amp;toCopy)</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>operator==</code> (const <code>Player</code> &amp;player) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>Player</code> (const <code>Player</code> &amp;toCopy)</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>setDebugOutputLevel</code> (int debugLevel)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>setListener</code> (const <code>BaseListener</code> *baseListener)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>setLogFormatOptions</code> (const <code>LogFormatOptions</code> &amp;options)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>toString</code> (Common::JString &amp;retStr, bool withTypes=false) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>toString</code> (bool withTypes, bool withCustomProperties) const</td>
<td><code>Player</code></td>
</tr>
<tr>
<td>ExitGames::Common::Base::toString (bool withTypes=false) const</td>
<td><code>ToS</code></td>
</tr>
<tr>
<td><code>typeToString</code> (void) const</td>
<td><code>ToS</code></td>
</tr>
<tr>
<td><code>~Base</code> (void)</td>
<td><code>Base</code></td>
</tr>
<tr>
<td><code>~Player</code> (void)</td>
<td><code>Player</code></td>
</tr>
<tr>
<td><code>~ToString</code> (void)</td>
<td><code>ToS</code></td>
</tr>
</tbody>
</table>
RaiseEventOptions Member List

This is the complete list of members for RaiseEventOptions, including all inherited members.

**Get 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`

**Set Members**

- `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)`

**Operator**

- `operator=(const RaiseEventOptions &toCopy)`

**Constructor**

- `RaiseEventOptions(nByte channelID=0, nByte eventCaching=Lite::EventCache::DO_NOT_CACHE, const int *targetPlayers=NULL, short numTargetPlayers, int cacheSliceIndex=0)`
- `RaiseEventOptions(const RaiseEventOptions &toCopy)`
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)
### Room Member List

This is the complete list of members for `Room`, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCustomProperties(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getDirectMode(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>getIsOpen(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getMaxPlayers(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>getName(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>getPlayerCount(void) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>operator=(const Room &amp;toCopy)</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>operator==(const Room &amp;room) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>Room(const Room &amp;toCopy)</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>toString(bool withTypes, bool withCustomProperties) const</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString</code> (bool withTypes=false) const`</td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~Room(void)</code></td>
<td>Room</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToS</td>
</tr>
</tbody>
</table>
RoomOptions Member List

This is the complete list of members for RoomOptions, including all inherited members.

- `getCustomRoomProperties(void) const`
- `getDebugEnabled(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 &customRoomProperties)`
- `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 > &propsListedInLobby)
setPublishUserID(bool publishUserID)
setSuppressRoomEvents(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)
WebFlags Member List

This is the complete list of members for **WebFlags**, including all inherited members.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getFlags(void) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>getHttpForward(void) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>getSendAuthCookie(void) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>getSendState(void) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>getSendSync(void) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setFlags(nByte webFlags)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>setHttpForward(bool httpWebForward)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>setSendAuthCookie(bool sendAuthCookie)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>setSendState(bool sendState)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>setSendSync(bool sendSync)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>WebFlags(nByte webFlags=0)</code></td>
<td>Wel</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>Bas</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>ToS</td>
</tr>
</tbody>
</table>
### Protocol Member List

This is the complete list of members for `Protocol`, including all inherited members.

<table>
<thead>
<tr>
<th>GAME (defined in Protocol)</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER (defined in Protocol)</td>
<td>Protocol</td>
</tr>
<tr>
<td>NAME (defined in Protocol)</td>
<td>Protocol</td>
</tr>
</tbody>
</table>
TCP Member List

This is the complete list of members for TCP, including all inherited members.

<table>
<thead>
<tr>
<th>GAME</th>
<th>TCP</th>
<th>static</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASTER</td>
<td>TCP</td>
<td>static</td>
</tr>
<tr>
<td>NAME</td>
<td>TCP</td>
<td>static</td>
</tr>
<tr>
<td>TCP(void)</td>
<td>TCP</td>
<td></td>
</tr>
</tbody>
</table>
**UDP Member List**

This is the complete list of members for **UDP**, including all inherited members.

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GAME</code> (defined in <strong>UDP</strong>)</td>
<td><strong>UDP</strong></td>
</tr>
<tr>
<td><code>MASTER</code> (defined in <strong>UDP</strong>)</td>
<td><strong>UDP</strong></td>
</tr>
<tr>
<td><code>NAME</code> (defined in <strong>UDP</strong>)</td>
<td><strong>UDP</strong></td>
</tr>
<tr>
<td><code>UDP(void)</code> (defined in <strong>UDP</strong>)</td>
<td><strong>UDP</strong></td>
</tr>
</tbody>
</table>
UDPAlternative Member List

This is the complete list of members for UDPAlternative, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAME (defined in UDPAlternative)</td>
<td>UDPAlternative</td>
</tr>
<tr>
<td>MASTER (defined in UDPAlternative)</td>
<td>UDPAlternative</td>
</tr>
<tr>
<td>NAME (defined in UDPAlternative)</td>
<td>UDPAlternative</td>
</tr>
<tr>
<td>UDPAlternative(void) (defined in UDPAlternative)</td>
<td>UDPAlternative</td>
</tr>
</tbody>
</table>
This is the complete list of members for `WS`, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Definition Location</th>
<th>Type</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GAME</code></td>
<td>defined in <code>WS</code></td>
<td><code>WS</code></td>
<td>static</td>
</tr>
<tr>
<td><code>MASTER</code></td>
<td>defined in <code>WS</code></td>
<td><code>WS</code></td>
<td>static</td>
</tr>
<tr>
<td><code>NAME</code></td>
<td>defined in <code>WS</code></td>
<td><code>WS</code></td>
<td>static</td>
</tr>
<tr>
<td><code>WS(void)</code></td>
<td>defined in <code>WS</code></td>
<td><code>WS</code></td>
<td></td>
</tr>
</tbody>
</table>
This is the complete list of members for **WSS**, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAME (defined in <strong>WSS</strong>)</td>
<td>WSS static</td>
</tr>
<tr>
<td>MASTER (defined in <strong>WSS</strong>)</td>
<td>WSS static</td>
</tr>
<tr>
<td>NAME (defined in <strong>WSS</strong>)</td>
<td>WSS static</td>
</tr>
<tr>
<td>WSS(void) (defined in <strong>WSS</strong>)</td>
<td>WSS</td>
</tr>
</tbody>
</table>
This is the complete list of members for Puncher, including all inherited members.

<table>
<thead>
<tr>
<th>Method</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear(void)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>init(PunchListener *pPunchListener)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>processPackage(const Common::JVector&lt;nByte&gt; &amp;packet, bool relay, int relayRemoteID)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>Puncher(RelayClient *pRelayClient, const Common::Logger &amp;logger)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>sendDirect(const Common::JVector&lt;nByte&gt; &amp;buffer, int targetID, bool fallbackRelay)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>sendDirect(const Common::JVector&lt;nByte&gt; &amp;buffer, const Common::JVector&lt;int&gt; &amp;targetIDs, bool fallbackRelay)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>service(void)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>startPunch(int remoteID)</td>
<td>(defined in Puncher)</td>
</tr>
<tr>
<td>~Puncher(void)</td>
<td>(defined in Puncher)</td>
</tr>
</tbody>
</table>
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)
Photon C++
Client API  4.1.12.2

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::JVector<int> &targetIDs)=0 (defined in ~RelayClient)

~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 OperationRequestParameters &parameters)`
- `OperationRequest(const OperationRequest &toCopy)`
- `operator=(const OperationRequest &toCopy)`
- `operator[](unsigned int index) const`
- `setParameters(const OperationRequestParameters &parameters)`
- `toString(bool withParameters=false, bool withParameterTypes=false) const`
- `~OperationRequest(void)`

© Exit Games®, all rights reserved.
This is the complete list of members for `OperationResponse`, including all inherited members.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDebugMessage(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>getOperationCode(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>getParameterForCode(nByte parameterCode) const</code></td>
<td></td>
</tr>
<tr>
<td><code>getParameters(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>getReturnCode(void) const</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationResponse(const OperationResponse &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>operator=(const OperationResponse &amp;toCopy)</code></td>
<td></td>
</tr>
<tr>
<td><code>operator[](unsigned int index) const</code></td>
<td></td>
</tr>
<tr>
<td><code>toString(bool withDebugMessage=false, bool withParameters=false, bool withParameterTypes=false)</code></td>
<td></td>
</tr>
<tr>
<td><code>~OperationResponse(void)</code></td>
<td></td>
</tr>
</tbody>
</table>
Photon C++
Client API  4.1.12.2

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 pingResult)
onStatusChanged(int statusCode)=0
~BaseListener() (defined in BaseListener)
~PhotonListener(void)
## PhotonPeer Member List

This is the complete list of members for PhotonPeer, including all inherited members.

<table>
<thead>
<tr>
<th>Member Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connect</code> (const Common::JString &amp;ipAddr, const Common::JString &amp;appID=Common::JString())</td>
</tr>
<tr>
<td><code>connect</code> (const Common::JString &amp;ipAddr, const Common::JString &amp;appID, const Ftype &amp;customData)</td>
</tr>
<tr>
<td><code>connect</code> (const Common::JString &amp;ipAddr, const Common::JString &amp;appID, const Ftype&amp; customDataArray, typename Common::Helpers::ArrayLengthType&lt;Ftype&gt;::type arrSize)</td>
</tr>
<tr>
<td><code>connect</code> (const Common::JString &amp;ipAddr, const Common::JString &amp;appID, const Ftype&amp; customDataArray, const short *pArrSizes)</td>
</tr>
<tr>
<td><code>disconnect</code> (void)</td>
</tr>
<tr>
<td><code>dispatchIncomingCommands</code> (void)</td>
</tr>
<tr>
<td><code>establishEncryption</code> (void)</td>
</tr>
<tr>
<td><code>fetchServerTimestamp</code> (void)</td>
</tr>
<tr>
<td><code>getByteCountCurrentDispatch</code> (void) const</td>
</tr>
<tr>
<td><code>getByteCountLastOperation</code> (void) const</td>
</tr>
<tr>
<td><code>getBytesIn</code> (void) const</td>
</tr>
<tr>
<td><code>getBytesOut</code> (void) const</td>
</tr>
<tr>
<td><code>getChannelCountUserChannels</code> (void) const</td>
</tr>
<tr>
<td><code>getConnectionProtocol</code> (void) const</td>
</tr>
<tr>
<td><code>getCRCEnabled</code> (void) const</td>
</tr>
<tr>
<td><code>getDebugOutputLevel</code> (void) const</td>
</tr>
<tr>
<td><code>getDisconnectTimeout</code> (void) const</td>
</tr>
<tr>
<td><code>getIncomingReliableCommandsCount</code> (void) const</td>
</tr>
<tr>
<td><code>getIsEncryptionAvailable</code> (void) const</td>
</tr>
<tr>
<td><code>getIsPayloadEncryptionAvailable</code> (void) const</td>
</tr>
<tr>
<td><code>getLimitOfUnreliableCommands</code> (void) const</td>
</tr>
</tbody>
</table>
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, const Common::JVector<nByte> &HMACSecret)
initUserDataEncryption(const Common::JVector<nByte> &secret)
opCustom(const OperationRequest &operationRequest, bool sendReliable, nByte channelID=0, bool encrypt=false)
PhotonPeer(PhotonListener &listener, nByte connectionProtocol=ConnectionProtocol::DEFAULT)
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
~PhotonPeer(void)
## TrafficStats Member List

This is the complete list of members for **TrafficStats**, including all inherited members.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getControlCommandBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getControlCommandCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getDebugOutputLevel(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>getFragmentCommandBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getFragmentCommandCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getLogFormatOptions(void)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>getPackageHeaderSize(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getReliableCommandBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getReliableCommandCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTimestampOfLastAck(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTimestampOfLastReliableCommand(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTotalCommandBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTotalCommandCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTotalCommandsInPackets(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTotalPacketBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getTotalPacketCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getUnreliableCommandBytes(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>getUnreliableCommandCount(void) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>setDebugOutputLevel(int debugLevel)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setListener(const BaseListener *baseListener)</code></td>
<td>Base</td>
</tr>
<tr>
<td><code>setLogFormatOptions(const LogFormatOptions &amp;options)</code></td>
<td>Base</td>
</tr>
<tr>
<td>Function</td>
<td>Class</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><code>toString(Common::JString &amp;retStr, bool withTypes=false) const</code></td>
<td>TrafficStats</td>
</tr>
<tr>
<td><code>ExitGames::Common::Base::toString(bool withTypes=false) const</code></td>
<td>ToS</td>
</tr>
<tr>
<td><code>typeToString(void) const</code></td>
<td>ToString</td>
</tr>
<tr>
<td><code>~Base(void)</code></td>
<td>~Base</td>
</tr>
<tr>
<td><code>~ToString(void)</code></td>
<td>~ToS</td>
</tr>
<tr>
<td><code>~TrafficStats(void)</code></td>
<td>~TrafficStats</td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
TrafficStatsGameLevel Member List

This is the complete list of members for TrafficStatsGameLevel, including all inherited members.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDebugOutputLevel(void)</td>
<td>Base</td>
</tr>
<tr>
<td>getDispatchIncomingCommandsCalls(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getEventByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getEventCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLogFormatOptions(void)</td>
<td>Base</td>
</tr>
<tr>
<td>getLongestDeltaBetweenDispatching(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLongestDeltaBetweenSending(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLongestEventCallback(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLongestEventCallbackCode(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLongestOpResponseCallback(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getLongestOpResponseCallbackOpCode(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getOperationByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getOperationCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getResultByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getResultCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getSendOutgoingCommandsCalls(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getTotalByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getTotalIncomingByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getTotalIncomingMessageCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getTotalMessageCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
<tr>
<td>getTotalOutgoingByteCount(void) const</td>
<td>TrafficStatsGameLevel</td>
</tr>
</tbody>
</table>
getTotalOutgoingMessageCount(void) const
resetMaximumCounters(void)
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
toStringVitalStats(void) const
typeToString(void) const
~Base(void)
~ToString(void)
~TrafficStatsGameLevel(void)
Go to the textual class hierarchy
# Common-cpp Directory Reference

Directory dependency graph for Common-cpp:
inc Directory Reference

Directory dependency graph for inc:
Directories

<table>
<thead>
<tr>
<th>directory</th>
<th>platform_definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>ANSIString.h</td>
<td></td>
</tr>
<tr>
<td>Base.h</td>
<td></td>
</tr>
<tr>
<td>BaseCharString.h</td>
<td></td>
</tr>
<tr>
<td>BaseListener.h</td>
<td></td>
</tr>
<tr>
<td>Common.h</td>
<td></td>
</tr>
<tr>
<td>CustomType.h</td>
<td></td>
</tr>
<tr>
<td>CustomTypeBase.h</td>
<td></td>
</tr>
<tr>
<td>CustomTypeFactory.h</td>
<td></td>
</tr>
<tr>
<td>DeSerializer.h</td>
<td></td>
</tr>
<tr>
<td>Dictionary.h</td>
<td></td>
</tr>
<tr>
<td>DictionaryBase.h</td>
<td></td>
</tr>
<tr>
<td>EGTime.h</td>
<td></td>
</tr>
<tr>
<td>Hashtable.h</td>
<td></td>
</tr>
<tr>
<td>JString.h</td>
<td></td>
</tr>
<tr>
<td>JVector.h</td>
<td></td>
</tr>
<tr>
<td>KeyObject.h</td>
<td></td>
</tr>
<tr>
<td>Logger.h</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Object.h</td>
<td></td>
</tr>
<tr>
<td>Serializer.h</td>
<td></td>
</tr>
<tr>
<td>ToString.h</td>
<td></td>
</tr>
<tr>
<td>UTF8String.h</td>
<td></td>
</tr>
<tr>
<td>ValueObject.h</td>
<td></td>
</tr>
</tbody>
</table>
MemoryManagement Directory Reference

Directory dependency graph for MemoryManagement:
Directories
## Files

<table>
<thead>
<tr>
<th>file</th>
<th>Allocate.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>AllocatorInterface.h</td>
</tr>
</tbody>
</table>
LoadBalancing-cpp Directory Reference

Directory dependency graph for LoadBalancing-cpp:
inc Directory Reference

Directory dependency graph for inc:
Directories
<table>
<thead>
<tr>
<th>file</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthenticationValues.h</td>
<td>LoadBalancing-cpp/inc/AuthenticationValues.h</td>
</tr>
<tr>
<td>Client.h</td>
<td>LoadBalancing-cpp/inc/Client.h</td>
</tr>
<tr>
<td>FriendInfo.h</td>
<td>LoadBalancing-cpp/inc/FriendInfo.h</td>
</tr>
<tr>
<td>Listener.h</td>
<td>LoadBalancing-cpp/inc/Listener.h</td>
</tr>
<tr>
<td>LobbyStatsRequest.h</td>
<td>LoadBalancing-cpp/inc/LobbyStatsRequest.h</td>
</tr>
<tr>
<td>LobbyStatsResponse.h</td>
<td>LoadBalancing-cpp/inc/LobbyStatsResponse.h</td>
</tr>
<tr>
<td>MutablePlayer.h</td>
<td>LoadBalancing-cpp/inc/MutablePlayer.h</td>
</tr>
<tr>
<td>MutableRoom.h</td>
<td>LoadBalancing-cpp/inc/MutableRoom.h</td>
</tr>
<tr>
<td>Peer.h</td>
<td>LoadBalancing-cpp/inc/Peer.h</td>
</tr>
<tr>
<td>Player.h</td>
<td>Player.h</td>
</tr>
<tr>
<td>RaiseEventOptions.h</td>
<td>RaiseEventOptions.h</td>
</tr>
<tr>
<td>Room.h</td>
<td>Room.h</td>
</tr>
<tr>
<td>RoomOptions.h</td>
<td>RoomOptions.h</td>
</tr>
<tr>
<td>WebFlags.h</td>
<td>WebFlags.h</td>
</tr>
</tbody>
</table>
Directory dependency graph for Chat-cpp:
Directories
inc Directory Reference

Directory dependency graph for inc:
Directories
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Chat-cpp/inc/AuthenticationValues.h</td>
</tr>
<tr>
<td>file</td>
<td>Channel.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Client.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Listener.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Peer.h</td>
</tr>
</tbody>
</table>
Enums Directory Reference

Directory dependency graph for Enums:

![Diagram of directory dependency graph for Enums](image-url)
### Files

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>ClientState.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Enums/CustomAuthenticationType.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Enums/DisconnectCause.h</td>
</tr>
<tr>
<td>file</td>
<td>Chat-cpp/inc/Enums/ErrorCode.h</td>
</tr>
<tr>
<td>file</td>
<td>UserStatus.h</td>
</tr>
</tbody>
</table>
Photon-cpp Directory Reference

Directory dependency graph for Photon-cpp:
inc Directory Reference

Directory dependency graph for inc:
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EventData.h</code></td>
<td></td>
</tr>
<tr>
<td><code>LitePeer.h</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationRequest.h</code></td>
<td></td>
</tr>
<tr>
<td><code>OperationResponse.h</code></td>
<td></td>
</tr>
<tr>
<td><code>PhotonListener.h</code></td>
<td></td>
</tr>
<tr>
<td><code>PhotonPeer.h</code></td>
<td></td>
</tr>
<tr>
<td><code>TrafficStats.h</code></td>
<td></td>
</tr>
<tr>
<td><code>TrafficStatsGameLevel.h</code></td>
<td></td>
</tr>
</tbody>
</table>

© Exit Games ®, all rights reserved.
Photon Documentation | Contact | Terms
Enums Directory Reference

Directory dependency graph for Enums:

- inc
  - Enums
  - Common-cpp
## Files

<table>
<thead>
<tr>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionProtocol.h</td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/ErrorCode.h</td>
</tr>
<tr>
<td>EventCache.h</td>
</tr>
<tr>
<td>Photon-cpp/inc/ Enums/EventCode.h</td>
</tr>
<tr>
<td>EventKey.h</td>
</tr>
<tr>
<td>NetworkPort.h</td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/OperationCode.h</td>
</tr>
<tr>
<td>Photon-cpp/inc/Enums/ParameterCode.h</td>
</tr>
<tr>
<td>PeerState.h</td>
</tr>
<tr>
<td>ReceiverGroup.h</td>
</tr>
<tr>
<td>StatusCode.h</td>
</tr>
</tbody>
</table>
Enums Directory Reference

Directory dependency graph for Enums:
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>LoadBalancing-cpp/inc/Enums/CustomAuthenticationType.h</td>
</tr>
<tr>
<td>file</td>
<td>DirectMode.h</td>
</tr>
<tr>
<td>file</td>
<td>LoadBalancing-cpp/inc/Enums/DisconnectCause.h</td>
</tr>
<tr>
<td>file</td>
<td>LoadBalancing-cpp/inc/Enums/ErrorCode.h</td>
</tr>
<tr>
<td>file</td>
<td>LobbyType.h</td>
</tr>
<tr>
<td>file</td>
<td>MatchmakingMode.h</td>
</tr>
<tr>
<td>file</td>
<td>PeerStates.h</td>
</tr>
<tr>
<td>file</td>
<td>RegionSelectionMode.h</td>
</tr>
<tr>
<td>file</td>
<td>ServerType.h</td>
</tr>
</tbody>
</table>
Enums Directory Reference
Files

- file  DebugLevel.h
- file  TypeCode.h
Helpers Directory Reference

Directory dependency graph for Helpers:
Directories
<p>| file | IsPrimitiveType.h |</p>
<table>
<thead>
<tr>
<th>Common-cpp</th>
<th>inc</th>
<th>platform_definitions</th>
</tr>
</thead>
</table>

platform_definitions Directory Reference