Photon Unity Networking リファレンス

v1.21

日本語翻訳版

Chapter 1

1.メインページ

はじめに

Photon は、リアルタイム・マルチプレイヤーゲームのための柔軟性のある高速で効率的な開発フレームワークです。 Photon はサーバーと主要な各プラットフォーム向けのクライアント SDK 群で構成されています。

**Photon Unity Network **（「PUN」といいます） は、Unity に組み込まれているネットワーク機能とさらに拡張したネットワーク機能の実装を目的としたクライアント向け専用フレームワークです。この中で、Photonの機能を利用して通信やプレイヤーの対戦の組み合わせをします。

PhotonNetwork API は Unity に組み込まれているものと非常に良く似ていますので、すでにUnity でネットワークゲームの開発経験がある方々は、すぐに馴染むことができるでしょう。また、自動変換を利用すると、既存のマルチプレイヤープロジェクトを、効率的にPhoton向けプロジェクトとして移植することができます。

すべてのソースコードが提供されますので、必要に応じてあらゆるタイプのマルチプレイヤーゲームに対応するようにこのプラグインを拡張することができます。
このプラグインは、皆様のために Photon Server を稼働させているクラウドサービスの Photon Cloud [1]と互換性があります。このPhoton Cloud サービスには、セットアップ画面から、１分もかからずに登録（無料）することができます。

最も重要な機能／特徴として、
- とても簡単なAPI
- クラウドサービスによりサーバーが利用可能（開発目的には無償）
- Unity Networking から PUN への自動変換機能（一部コードは自動変換に非対応）
- オフラインモード（マルチプレイヤーのコードをシングルプレイヤーゲームで再利用）
- Photon Server の卓越した性能
- 負荷分散ワークフローによるサーバーの効率的な使用（追加作業は不要）
- 直接的な P2P や NATパンチスルーは不要

ファーストステップ

あなたが Unity のネットワーク機能の使い方を知っている場合は、PUN にも馴染みやすいことでしょう。すぐにでも変換機能（ALT+P でウィザードを開始します）を起動し、あなたのプロジェクトを変換して、実際にコードを見てみてはどうでしょう。
PUNについてよく理解していただくため、このドキュメントは基本説明（General Documentation）とパブリック API リファレンス（Public API reference）にわかれています。

また、別途 PUN のソースコードも提供されています。

---

[1] 開発元公式サイト（英文）http://doc.exitgames.com/photon-cloud/
日本公式サイト（原文翻訳）http://doc.exitgames.com/photon-cloud-jp/

Chapter 2
2.基本説明

Photon

Unity に組み込まれているネットワークとは違い、PUN は常に 1 つの専用サーバーに接続します。その専用サーバーでは、各Room／対戦の管理、マッチメイキング機能、Room 内でのプレイヤー間のコミュニケーション機能などを提供する特定のゲームロジックが実行されています。実際、内部的には PUN はいくつかのサーバーを使用しています。具体的には、複数の「ゲームサーバー」が実際の 各Room（各対戦）を実行している一方で、1 つの「マスターサーバー」が、各Room（各対戦）の準備や整理、マッチメイキングなどを行っています。

サーバー側の選択肢は二種類あります。

Exit Games Cloud

Exit Games Cloud は、 Exit Games社がすべて運用管理する、負荷分散されたPhoton Server の機能を一般に提供しているクラウドサービスです。無料トライアルもご利用できますし、また商用利用のためのサブスクリプションも比較的少ない費用で利用することができます。

このサービスは決められたあるゲームロジックの下に稼働していますので、ユーザー独自のゲームロジックをサーバーサイドで実装することができません。ユーザー独自のゲームロジックを実装するには、そのかわりにクライアント側に権限を持たせるようにする必要があります。
クライアントは、ゲームタイトルとゲームバージョン（game version）に関連付けられた「アプリケーション ID(AppID/ApplicationID）」により区別されます。これにより、プレイヤーは他の開発者のゲームや古いバージョンのゲームのプレイヤーと競合することはありません。

Asset Storeで購入されたサブスクリプションについて[2]

Photon Cloud のサブスクリプションを Asset Storeで購入された場合は、次の手順に従ってください：

- Photon Cloud アカウントを登録：cloud-jp.exitgames.com
- ダッシュボードから自分のアプリケーションID（AppID）を取得
- developer@photoncloud.jp 宛にメールを送る
- 以下の項目をメールに含める：
  - お名前および会社名（法人の場合）
  - Asset Storeでの請求／購入ID
  - Photon Cloud のアプリケーションID

Photon Server SDK

Photon Cloud サービスと別のもう一つの選択肢としては、独自のサーバーを運用すること、そして提供されるC#ベースの ロードバランサー（負荷分散）ソリューション上に独自のサーバーサイドロジックを開発する方法があります。この方法により、サーバーロジックを完全に制御することができます。

[2] 今後、手順が修正される可能性があります。

Photon v3.2 SDK はここからダウンロードできます：http://www.exitgames.com/Download/Photon
サーバーを起動する：http://doc.exitgames.com/photon-server/PhotonIn5Min

ファーストステップ

PUN をインポートすると「ウィザード」ウインドウがポップアップ表示されます。ここで Photon Cloud に登録するメールアドレスを入力するか、もしくはこのステップをスキップし既に持っているアカウントのアプリケーションID を入力するか、または独自サーバーのアドレスを入力するため self hosted（自前のサーバー）の Photon に切り替えるか、のいずれかを行います。
これにより、（クラウドサービスの）Photon Cloud用または自前の Photon Server 用のいずれかのサーバー構成の設定がプロジェクト内で作られます： PhotonServerSettings
PUN は非常に多くのファイルで構成されていますが、本当に重要なのは１つだけ、 PhotonNetwork です。このクラスには必要とされるすべての関数と変数が含まれています。もしも実装しなければいけない特別な要件などがある場合には、いつでもソースファイルを修正することができます。極論すれば、このPUNのプラグインは Photonの実装のサンプルの１つでしかない、ということです。
Unityscriptを使用している場合には、Photon Unity Networkingフォルダをプロジェクトのルートディレクトリに移動させる必要があります。
このAPIがどのように機能するか、早速いくつかの例を使い説明しましょう。

マスターサーバーとロビー

PhotonNetwork は、１つのマスターサーバーと１つまたは複数のゲームサーバーを使用します。マスターサーバーは、各ゲームサーバーで進行中のゲームの管理だけを行います。

その時点で利用されているRoomのリストを取得するためには、クライアントはマスターサーバー上のロビー（Lobby）に参加（join）する必要があります。ロビーは、参加したクライアントに対しRoomリストの送信および更新を自動的に行います。これは、トラフィックに負担をかけない頻度で行われます。ちなみに、（混雑時の問題を避けるため）ロビーではプレイヤー同士は互いの存在に気づいたり、データを送りあったりすることはできません。
PUN は、接続時にデフォルトでロビーに入ります。しかし、これは必ずそうしなければならないというわけではありません。マスターサーバー上にいる間は、いつでもRoomを作ったりRoomに参加したりすることができます。JoinRandomRoom を使用すると、プレイヤーを各対戦（各Room）に迅速かつ簡単に割り振ることができます。
プレイヤーがRoomに参加するまたはRoomを作る時、このオペレーションはマスターサーバーに送られます。そして、マスターサーバーはクライアントを実際のゲームが行われるゲームサーバーの１つに転送します。
各サーバーは、サーバー専用のマシンで実行されます。つまり、「プレイヤーがホストするサーバー」というものはありません。とはいえ、APIが内部的にすべて管理しているので、開発者はサーバーの構成などを意識する必要はありません。

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

このコードは、PhotonNetwork の機能を利用するためには必ず必要となります。これにより、クライアントのゲームのバージョンが設定され、セットアップウィザードでの設定（PhotonServerSettings に保存されている）を使用することになります。セットアップウィザードですが、Photon を独自サーバーでホストする場合にも利用できます。また、この代わりに Connect() とした場合は、PhotonServerSettings ファイルの設定を無視することができます。

バージョンによる管理

Photon の負荷分散ロジックでは、特定のゲームのプレイヤーと他のゲームのプレイヤーを区別するためアプリケーションID を利用しています。同じように、ゲームバージョンを利用して、クライアントの新旧バージョンによりプレイヤーを区別することができます。また、PUNの異なるバージョン間での互換性は保証されていないため、ゲームバージョンに加え PUN のバージョンも付加されてサーバーに送信されます。（PUN v1.7 以降）

ゲームを始める（Roomに参加、Roomを作る）

次に、Roomに参加、もしくは Roomを新しく作りたいと思います。以下のコードでいくつか必要な機能をお見せします：

```
//Roomに参加
PhotonNetwork.JoinRoom(roomName);
```

```
//Roomを作成
PhotonNetwork.CreateRoom(roomName);
//既に存在する場合は失敗し、これをコール：: OnPhotonCreateGameFailed

//ランダムにどのRoomにでも参加する
PhotonNetwork.JoinRandomRoom();
//マッチするゲームが無い場合、失敗、そしてこれをコール：: OnPhotonRandomJoinFailed
```

マスターサーバーのロビーにより、進行中のゲーム（Room／対戦）のリストが提供されます。ロビーは他の各Roomと同じように入ることができますが、そこでは各Room（対戦）のリストだけが提供／更新されるだけです。PhotonNetwork プラグインでは、接続後に自動的にロビーに入ります。いずれかの Roomに参加すると、Roomリストは更新されなくなります。

（ロビー内で）Roomリストを表示するためには次のようにします：

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

その他の方法としては、ランダムマッチメイキングを利用することもできます。すなわち、どのRoomにでもランダムに参加するようにしますが、もしどのRoomも対戦相手に空きがない場合は失敗となります。その場合には、名前のないRoomを作り、他のプレイヤーがランダムにそのRoomに参加してくるまで待つようにしましょう。

高度なマッチメイキングとRoomプロパティ

基本の単純なランダムなマッチメイキングでは、プレイヤーが必ずしも楽しめるわけではありません。特定のマップをプレイしたい、２対２での対戦をしたい、などの要望があることでしょう。
Photon Cloud および Photon Server のロードバランサー（負荷分散機能）では、任意のRoomプロパティを設定し、JoinRandomにおいてそのプロパティをフィルター条件とすることができます。

Roomプロパティとロビー

Roomプロパティは、そのRoomにいるすべてのプレイヤーで共有／同期され、進行中のマップ、ラウンド、開始時間などの記録に役に立ちます。これは文字列をキーとしたハッシュテーブルとして扱われます。キーは、短いほうが望ましいです。
選択したRoomプロパティをロビーに転送することもできます。そのプロパティによりリスト化したり、ランダムなマッチメイキングにプロパティを利用したりすることが、これにより可能となります。
Roomプロパティのすべてがロビーで役立つわけではありませんので、Roomを作成する時に、ロビー転送用のRoomプロパティのリストを定義することになります。

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

ここで “ai” はまだ値がないことに注意してください。Room.SetCustomProperties() により、そのゲーム内で設定されるまでは、これがロビーで表示されることはありません。また、"map" や "ai"の値を変更した時は、それらの値はロビー内ですぐに更新されます。
リストの読み込みによりクライアントのパフォーマンスが低下することが無いように、このリストは短くするようにしてください。

Join RandomでのRoomプロパティによるフィルター機能

JoinRandomでは、想定されるRoomのカスタムプロパティ(expectedCustomRoomProperties)と最大プレイヤー（maxPlayer）の値をハッシュテーブルで渡すことができます。サーバーが適合するRoomを選択する際に、これらの値はフィルター条件として機能します。

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

フィルター条件とするプロパティをより多く渡すと、条件にあう Room がある可能性が低くなります。条件とするプロパティは限定したほうがいいでしょう。また、ロビーが認識していないプロパティをフィルター条件とすることがないように注意してください（前述を参照してください）。

MonoBehaviour コールバック

PhotonNetwork は、「接続済み(connected）」や「ゲームに参加中（joined a game）」のような状態の変化をゲームに知らせるために、何種類ものコールバックを実装しています。コールバックとして使用されているメソッドのそれぞれは、PhotonNetworkingMessage enum型（列挙型）に含まれているものです。列挙されている定数毎に、それぞれ使い方が説明されています。（入力時にツールティップを確認してください。例　PhotonNetworkingMessage.OnConnectedToPhoton）これらのメソッドは、MonoBehaviour インスタンスがいくつあろうとも、各インスタンスに追加することができ、それぞれのシチュエーションでコールされます。すべてのコールバックのリストは、プラグインのレファレンスにあります。
ここまでが、ゲームRoomのセットアップの基本です。次にゲーム内での実際のコミュニケーションについて説明します。

ゲームRoomでメッセージを送る

Room内では、接続している他のプレイヤーにネットワークメッセージを送ることができます。さらに、バッファーリングされたメッセージを、その後に接続してくるプレイヤーに対し送ることもできます。（例えば、自分のプレイヤーをスポーンさせる場合など。）
メッセージの送信は２つの方法により行うことができます。RPCで行うか、PhotonView プロパティの OnSerializePhotonView を利用するか、のいずれかです。実際には、もっと多くのネットワークのやりとりがあります。特定のネットワークイベントのコールバック（例えば、OnPhotonInstantiateや OnPhotonPlayerConnected）を監視して、何らかのイベント（例えば、PhotonNetwork.Instantiate）を発生させることができます。最後の説明がわからなくても心配しないでください。次から、それぞれのトピックを説明します。

PUNでのグループの利用

PhotonView 上でグループが変更されると、それらは共有／同期されません。開発者の決定次第ですが、必要であれば、すべてのクライアント上で各PhotonView を同じグループ内に保持することができます。別々のクライアント上で同じ PhotonView にそれぞれ異なるグループ番号を使用することは、一貫性のない動作を引き起こすことになります。一部のネットワークメッセージは、受信者側でのみ、その受信対象のグループの確認がされます。具体的にいうと、シングルプレイヤー（または、MasterClient）向けのRPC、バッファーリング（AllBuffered/OthersBuffered）されたRPCがそうなります。PhotonNetwork.Instantiate も、バッファーリングされるので、この中に含まれます。
技術的な理由： Photon Server は、特定のアクター（プレイヤー）向けでなく、かつ、キャッシュされていないメッセージに対するインタレストグループだけをサポートしています。将来的に、これは変更されるかもしれません。

PhotonView

PhotonView は、メッセージ（RPC または OnSerializePhotonView）を送信するために使用されるスクリプトコンポーネントです。PhotonView をゲームの各ゲームオブジェクト（gameobject）に付ける必要があります。ちなみに、PhotonView はUnityのNetworkViewにとても似ています。
ゲームの中でメッセージを送信するため、そして任意に他の PhotonView をインスタンス化したり、割り当てたりするためには、常に、少なくとも１つの PhotonView がゲーム内に必要です。
PhotonView をゲームオブジェクトに追加するためには、シンプルにゲームオブジェクトを選び、"Components/Miscellaneous/PhotonView" を使ってください。

トランスフォームの監視

PhotonViewの Observe プロパティとして Transform をつける場合、位置（Position）、回転（Rotation）、スケール（Scale）のそれぞれまたはそれらの組合せを、各プレイヤー間で同期するように選択することができます。これは、プロトタイプや小規模のゲームを開発する時に大変に役立ちます。注意：Observeプロパティ（監視するプロパティ）の値がどれか一つでも変更があると、その変更された値だけではなく、すべてのObserveプロパティの値が送信されます。また、更新データは、平準化（スムーズ化）されたり、補完（インターポーレート）されたりしません。

MonoBehaviourの監視

PhotonView は、MonoBehaviourを監視するように設定することもできます。その場合、スクリプトの OnPhotonSerializeView メソッドがコールされることになります。スクリプトがローカルプレイヤーにコントロールされているかどうかにもよりますが、このメソッドはオブジェクトの状態を書き出すそして読み込むためにコールされます。
以下のシンプルなコードは、ほんの数行のコードの追加により、キャラクターの状態を同期させる方法を例示しています：

```
void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo
     info)
{
   if (stream.isWriting)
   {
   // 我々がこのプレイヤーを所有：他のプレイヤーにこちらのデータを送る
   stream.SendNext((int)controllerScript._characterState);
   stream.SendNext(transform.position);
   stream.SendNext(transform.rotation);
   }
   else
   {
   // ネットワークプレイヤー、受け取るデータ
      controllerScript._characterState = (CharacterState)(int)stream.
      ReceiveNext();
      correctPlayerPos = (Vector3)stream.ReceiveNext();
      correctPlayerRot = (Quaternion)stream.ReceiveNext();
   }
}
```

ReliableDeltaCompressedモードで送信する場合は、常に同じ順番でストリームにデータを書き込むように注意してください。PhotonStream に何も書き込まないと、更新情報は送られません。これはポーズの時に便利です。さあ、それではもう一つのコミュニケーションの方法、RPC を見てみましょう。

Remote Procedure Calls（リモートプロシージャーコール）

Remote Procedure Calls (RPCs) は、まさにその名前が表通り、同じRoomのリモートクライアント上でコールされるメソッドです。MonoBehaviourのメソッドを RPCとしてリモートでコールするには、RPC属性 [RPC] を適用する必要があります。マーキングされた関数をコールするためには、同じGameObjectに PhotonView のインスタンスが必要です。

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

スクリプトからメソッドをコールするには、PhotonView オブジェクトにアクセスする必要があります。スクリプトが Photon.MonoBehaviour クラスからのものであれば、photonView フィールドがあります。標準のMonoBehaviour および GameObject では、PhotonView.Get(this) を使用して、PhotonView コンポーネントにアクセスして、その上で RPC をコールします。

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

このように対象メソッドを直接コールする代わりに、PhotonView 上でRPC()をコールします。コールするメソッドの名前、メソッドをコールするべきプレイヤー、そしてパラメーターリストを記述します。
注意：RPC() 内のパラメーターリストは、想定されるパラメーターの数と合致していなければなりません。受信するクライアント側で合致するメソッドが見つからない場合は、エラーが記録されます。このルールには例外が一つだけあり、RPCメソッドの最後のパラメーターは、各コールの基本情報を提供するPhotonMessageInfo のタイプにすることができます。

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

RPCのタイミングとレベルのロード

RPC は特定の PhotonView でコールされ、リモートクライアント上の一致するPhotonViewを常に対象とします。もしリモートクライアントで一致する PhotonView が不明な場合、RPCは迷子となり、実行されません。
RPCが実行されないケースがよく起こるのは、クライアントがレベルをロードし、セットアップしている時です。つまり、クライアントの一つが、Roomの他のクライアントより早いとか長く滞在していて、他のクライアントがまだロードしていないオブジェクトに対する大事なRPCを送信した時です。RPCがバッファーリングされた時も同じようなことが起こります。
解決法としては、シーンのセットアップの間は、メッセージキューをポーズしておくことです：

```
private IEnumerator MoveToGameScene()
{
   // これ以降のネットワークメッセージの処理を一時的に停止する
   PhotonNetwork.isMessageQueueRunning =
   false;
   Application.LoadLevel(levelName);
}
```

メッセージキューの停止は、キューが再開されるまで送受信するメッセージを遅延させることになります。当たり前ですが、続行する準備ができたらキューを再開させることがとても重要です。
前のシーンに属するRPC で、シーン変更後に遅れて着信したものは、すべて廃棄されます。逆にいえば、RPCによりシーン間の区切りを定義することができるということです。

その他のトピック

Unity Networkingとの違い

1．ホスティングモデル

- Unity Networking は、サーバークライアントベース（P2Pでない！）です。サーバーは、Unityクライアントの１台で（つまり、プレイヤーの１人により）稼働します。
- Photon も、同様にサーバークライアントベースですが、専用のサーバーがあります。サーバーをホスティングしているクライアントの離脱による接続の中断が起こりません。

2．接続性

- Unity Networking は、NAT パンチスルーを利用し接続性を向上させています。なぜなら、プレイヤーがネットワークサーバーをホストしているため、ファイヤーウォールやルーターにより接続が失敗することがあるからです。接続の成功率は低く、接続は保証されることはありません。
- Photon は、専用のサーバーがあります。NAT パンチスルーやその他の特別な方法は必要ありません。接続は１００％保証されています。まれに、接続が失敗することもあり得ますが、それはクライアント側の厳しい制約（例として、企業内VPNなど）による場合です。

3．パフォーマンス

- Photon は、パフォーマンス的に Unity Networking を凌駕しています。これを証明する数値はまだありませんが、これまで何年もにわたりライブラリーが最適化されてきています。しかも、Unityのサーバーはプレイヤーがホストしている、つまりプレイヤーが稼働させているサーバーへの接続に頼っているため、遅延時間／pingの結果は通常悪いものとなります。このような接続は、専用である Photon Server への接続より良いということは決してありません。

4．価格

- Unity Networking と同じように、Photon Unity Networking （PUN）のプラグインも無料です。自社ゲームのために、Photon Cloud ホスティングサービスをサブスクリプションし利用することができます。 もしくは、自前のサーバーを借り、そこで Photon を稼働させることもできます。無料ライセンスで同時接続 100プレイヤーまでが可能です。他のライセンスでは（ホスティングをする際に）一括料金が発生し、同時接続ユーザー数を増やすこととなります。

5．機能とメンテナンス

- Unity は、ネットワークの実装をそれほど重要視しているようではありません。機能の改良もまれで、バグフィックスもめったにされていません。Photon のソリューションは、活発にメンテナンスされており、多くがソースコードで提供されています。その上、組み込みロードバランサーやオフラインモードなどのように、Photon は既にUnityより多くの機能を提供しています。

6．マスターサーバー

- Photon でのマスターサーバーは、通常の Unity Networkingのマスターサーバーとは、少し違っています。我々の場合、いわゆる「ロビー」で進行しているゲーム（対戦）の Room（部屋）の名前をリストするのは Photon Server となります。そして、そのサーバーが Unity のマスターサーバーと同じように、実際のゲームが行われているゲームサーバーにクライアントを転送します。

ネットワークでのオブジェクトのインスタンス化

どのようなゲームにおいても、プレイヤーそれぞれに必ず１つ以上のプレイヤーオブジェクトをインスタンス化（生成）する必要があります。以下にリストするように、その方法はいくつかあります。

PhotonNetwork.Instantiate

PUN では、PhotonNetwork.Instantiateメソッドにスタートの位置と回転およびプレハブの名前を渡すことで、オブジェクトの生成を自動的に管理することができます。この条件として、実行時にロードされるようにプレハブが Resourcesフォルダの直下になければなりません。ウェブプレイヤーに注意してください：デフォルトでは Resourcesフォルダのすべてのものが最初のシーンでストリーミングされてしまいます。ウェブプレイヤーの設定で、「First streamed level」を使い、Resourcesフォルダのどのアセットを最初のレベルで使用するかを指定することができます。もし最初のゲームシーンでこれを設定し、Resourcesフォルダのアセットを使わないようにすると、プリローダーとメインメニューがスローダウンされることはありません。

```
void SpawnMyPlayerEverywhere()
{
   PhotonNetwork.Instantiate(“MyPrefabName”, new
   Vector3(0,0,0), Quaternion.identity, 0);
   // 最後の変数はオプションでグループ番号です。今は無視してください。
}
```

より詳細にコントロール：マニュアルでのインスタンス化

ネットワーク上でのオブジェクトのインスタンス化において、Resources フォルダに依存したくない場合、このセクションの末尾の例にあるようにマニュアルでオブジェクトを Instantiate （インスタンス化）する必要があります。

マニュアルでインスタンス化する大きな理由は、ウェブプレイヤーのストリーミングのために、どのタイミングで何がダウンロードされるかをよりコントロールするためです。ストリーミングと UnityのResources フォルダの詳細については、ここに書かれています。

インスタンス化をマニュアルで行う場合は、PhotonViewID の割り当ても自分でやらなければいけません。viewIDは、正しいゲームオブジェクトやスクリプトへ、ネットワークメッセージをルーティングするキーとなっています。新しいオブジェクトを所有、生成したいユーザーは、PhotonNetwork.AllocateViewID(); を使って新しい viewID を割り当てる必要があります。そして割り当てられたその PhotonViewID は、既に設定済みの PhotonView（例えば、現在あるシーンのPhotonView）を利用して、他のすべてのプレイヤーに送られなければなりません。ここで忘れてならないのは、このRPCはバッファーリングされる必要があるということです。そうすることで、後から接続してくるクライアントがオブジェクト生成の命令を受け取れるようになります。また、このオブジェクトを生成するためのRPCメッセージは、目的とするプレハブを参照できる必要があり、その上で、Unity の GameObject.Instantiate を利用して、オブジェクトのインスタンス化を行います。最後に、すべてのPhotonView毎にそれぞれPhotonViewIDを割り当てることで、対象のプレハブに付けられた各 PhotonView の設定をする必要があります。

```
void SpawnMyPlayerEverywhere()
{
   //マニュアルでPhotonViewIDを割り当てる
   PhotonViewID id1 = PhotonNetwork.AllocateViewID(
   );

   photonView.RPC("SpawnOnNetwork", PhotonTargets.AllBuffered,
      transform.position,
      transform.rotation, id1, PhotonNetwork.player);
}
public Transform playerPrefab;  // インスペクターで設定

[RPC]
void SpawnOnNetwork(Vector3 pos, Quaternion rot, PhotonViewID id1, PhotonPlayer
     np)
{
   Transform newPlayer = Instantiate(playerPrefab, pos, rot) as Transform;
   // PhotonViewを設定
   PhotonView[] nViews = go.GetComponentsInChildren<PhotonView>();
   nViews[0].viewID = id1;
}
```

もしアセットバンドルを使い、そこからネットワークオブジェクトを一括してロードさせたい場合には、そのアセットバンドルをロードするコードを追加し、サンプルコードの “playerPrefab” をそのアセットバンドルのプレハブと置き換えるだけで可能です。

オフラインモード

オフラインモードとは、マルチプレイヤーのコードをシングルプレイヤーゲームでも、再利用することができる機能です。M2H社、Mike Hergaarden のコメント：通常、ゲームポータルではマルチプレイヤー機能を完全に無くすことが要求されるので、M2H社では、しばしば自分たちのゲームを作り直さなければなりませんでした。その必要が無くなるというだけでなく、その上、シングルプレイヤーとマルチプレイヤーで同じコードを使えるということ自体が、大きく作業量を軽減することになります。
シンブルプレイヤーモードでも使えるようにしたい機能として、最も良く知られているのは、RPCの送信と PhotonNetwork.Instantiate の利用です。オフラインモードの主なゴールは、接続されていない状況で PhotonNetwork 機能を使用しても、参照不能のエラーやその他のエラーが起こらないようにすることです。もちろん、シングルプレイヤーモードでゲームを実行している状態であることを把握し、ゲームのセットアップなどを含めて設定しておく必要があります。ですが、ゲームの実行中はすべてコードがそのまま利用することができます。
PhotonNetwork が意図した動作と本当のエラーとを区別できるように、マニュアルでオフラインモードを有効にする必要があります。この設定はとても簡単です。

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

これで、接続することなくまたエラーも発生させずに、特定のマルチプレイヤーのメソッドを再利用することができます。さらに目立ったオーバーヘッドもありません。以下は、オフラインモードの間の PhotonNetwork 関数、変数やその結果などのリストです：

PhotonNetwork.player プレイヤーIDは常に -1　PhotonNetwork.playerName 想定通りの動作、 PhotonNetwork.playerList ローカルプレイヤーのみ含まれる、 PhotonNetwork.otherPlayers 常に空、 PhotonNetwork.time Time.timeを返す；
PhotonNetwork.isMasterClient 常に true 、PhotonNetwork.AllocateViewID() 想定通りの動作、 PhotonNetwork.Instantiate 想定通りの動作、 PhotonNetwork.Destroy 想定通りの動作、
PhotonNetwork.RemoveRPCs/RemoveRPCsInGroup/SetReceivingEnabled/SetSendingEnabled/SetLevelPrefix これらはシングルプレイヤーでは何の意味もないが、しかし特に悪さをすることもない PhotonView.RPC 想定通りの動作。
注意してほしいのは、上記以外の他のメソッドの使用は、予期せぬ結果が発生することや、また単に何もしない、という場合があり得ます。例えば、PhotonNetwork.room は明らかに null を返します。もし最初はシングルプレイヤーでゲームを開始はするが、その後のステージでマルチプレイヤーに移行したいという場合は、オフラインモードのかわりに１プレイヤーのゲームをホストしたほうがいいでしょう。そうすることで、バッファーリングされたRPCやインスタンス化のコールが保存されます。それに対し、オフラインモードではインスタンス化などは接続後に引き継がれません。
PhotonNetwork.offlineMode = false とするか、または単に Connect() をコールすることで、オフラインモードは終了します。

制限事項

Viewとプレイヤー

パフォーマンスの理由により、PhotonNetwork API は１プレイヤーあたり最高 1,000 PhotonView で最大 2,147,483 のプレイヤー数をサポートします。（ちなみに、これはハードウェアのサポート限界よりとんでもなく大きな値です。）また、プレイヤーの最大値を減らすことで、１プレイヤーあたりのPhotonViewの最大値を簡単に増やすこともできます。それはこういうことです：各PhotonView は viewID をそれぞれのすべてのネットワークメッセージ上で送ります。この viewID は整数値で、プレイヤーIDとそのプレイヤーでのプレイヤーview ID からできています。整数の最大値は 2,147,483,647 で、それを MAX_VIEW_IDS(1000) で割ると、２百万以上のプレイヤーがそれぞれ1000の viewID を持てるということになります。ですから、お分かりのように MAX_VIEW_IDSを減らすことで、最大プレイヤー数を簡単に増やすことができます。また逆に、最大プレイヤー数を減らすことで、もっと多くの VIEW_IDSをすべてのプレイヤーに与えることができます。ここでぜひ注意していただきたいのは、ほとんどのゲームでは、１プレイヤーあたり２、３個以上の viewID は決して必要としないということです。（キャラクターのために１つか２つ、ふつうはそれで終わりです。）万が一もっと多く必要だとしたら、何か間違ったことをしている可能性があります。武器が発射する弾丸のそれぞれ１つに、１つのPhotonViewとIDを割り当てるのはきわめて非効率です。そんなことをするより、プレイヤーのPhotonView もしくは武器の PhotonView を利用し、発射された各弾丸を追跡するようにしてください。
データ型を int型から short型(-32,768 ~ 32,768)に変更することで、処理パフォーマンスを向上させる余地があります。その場合、例えばMAX_VIEW_IDS を32に設定しても、それでも1023人のプレイヤーをサポートすることができます。この int viewID に関連する詳細については、「 //LIMITS NETWORKVIEWS&PLAYERS （NETWORKVIEW とプレイヤーを制限）」を検索してください。さらに、現在API は、uint/ushort を使っておらず、正数範囲だけを使用しています。これはわかりやすさからであり、特別な理由はありません。また、ほとんどの場合、viewID の使用が重大なパフォーマンス問題とはなっていません。

グループとスコープ
PhotonNetwork プラグインは、本質的なネットワークグループやスコープの概念をまだサポートしていません。Unityの「スコープ（scope）」の機能は実装されていませんが、ネットワークグループは単にクライアントサイドで実装されています。すなわち、グループ区分により無視されるべきRPCは受信されたあと廃棄されます。このように、グループは機能しますが、ネットワークの帯域幅を効率的に利用しているわけではありません。

フィードバック

このソリューションは継続的に開発しているプロジェクトですので、皆様のフィードバックが大変参考となります。もし何かが、開示されていない、欠落している、もしくは機能していない、とういうようなことがあれば、お知らせください。お知らせいただくには、我々のフォーラムにポストしてください： http://www.facebook.com/groups/photoncloudjp/[3]

よくある質問

１つの GameObject（ゲームオブジェクト）に複数の PhotonView を使用することはできますか？それなぜですか？
はい、これは何ら問題ありません。２個以上のターゲットを監視する必要があれば、PhotonViewも複数必要となります。すなわち、１つの PhotonView につき１つだけを監視することができます。RPCのためには、１つの PhotonView だけがあれば良く、それは何かを監視するのに既に使われている同じPhotonView でも大丈夫です。RPC は、監視しているターゲットと競合してしまうことはありません。

Javascriptから利用することは可能ですか？
Photon の各クラスを利用するためには、プラグインのフォルダをプロジェクトのルートに移動させる必要があります。

Unity Networking プロジェクトを Photon に変換

Unity Networking プロジェクトを Photon へ変換するのは１日で終わります。自動変換機能がスクリプトを変更しますので、念のためプロジェクトのバックアップをとってください。バックアップが終わったら、Photon エディターウインドウからコンバーター（変換機能）を実行してください。(Window -> Photon Unity Networking -> Converter -> Start)　プロジェクトのサイズやコンピューターのスペックにもよりますが、自動変換には30秒から10分ぐらいかかります。
この自動変換は次の処理をします：

- まったく同じ設定で、すべての NetworkView をPhotonView に置き換えます。これはすべてのシーンとすべてのプレハブに適用されます。
- すべてのスクリプト(JS/BOO/C#)の Network API コールが精査され、PhotonNetwork コールに置き換えられます。

小さな違いがいくつかありますので、スクリプト変換のバグを２、３手直しする必要があるでしょう。変換後、コンパイルエラーが起こることがあります。それらをまず始めに修正する必要があります。コンパイルエラーで良くあるのは：
PhotonNetwork.RemoveRPCs(player); PhotonNetwork.DestroyPlayerObjects(player)； これらは存在しなく、安全に削除できます。Photon はプレイヤーがゲームを離れると自動的にクリーンアップします。（しかし、もし必要ならこの機能を無効にしてマニュアルでクリーンアップを処理することもできます。）　..CloseConnection takes ‘2’ arguments. . .　（..CloseConnection に ‘2’ つ引数がある. . .）; このコールから2番目の引数、ブール型（boolean）を削除。　PhotonNetwork.GetPing(player)； GetPing は引数をとりません。pingリクエストは、Photon Serverに対してだけでき、他のプレイヤーにはpingできません。
myPlayerClass.transform.photonView.XXX エラー；　次のコードに変換する必要があります：
myPlayerClass.transform.GetComponent<PhotonView>().XXX　スクリプトの中で、PhotonViewを使って、添付されている PhotonView コンポーネントをゲットすることができます。しかしながら、外部のトランスフォームに直接これをコールすることはできません。　RegisterServer； マスターサーバーに対してゲームを登録する必要がありません。Photon が自動的に行います。

コンパイルエラーは、すべて 5分から30分程度で修正できるはずです。ほとんどのエラーは、IPs/Ports/Lobby GUI に関連した メインメニュー／GUIのコードから起因していることでしょう。

この部分が Photon が Unityのソリューションと大きく違っているところです：

[3] 開発元公式サイト(英文) http://forum.exitgames.com

専用のPhoton Server が一つだけあり、Roomの名前を使い接続します。ですから、IPs/ports を参照しているものはすべてコード（通常GUIのコード）から削除できます。 PhotonNetwork.JoinRoom(string room) は １つのRoom引数だけをとり、

以前の IP/port/NAT などの引数は削除する必要があります。M2H社による 「Ultimate Unity Networking project」を使っていた場合は、MultiplayerFunctions クラスを削除しなければなりません。

最後に、以前の MasterServer 関連のコールはすべて削除することができます。サーバーを登録する必要はもうありません。そしてRoomリストは PhotonNetwork.GetRoomList() をコールするだけで、簡単に取ってくることができます。そして、このリストは常に更新されます。（フェッチやポーリングなどをする必要がありません。）Roomリスティング部分を書き換えるのぐらいが大きな作業で、もしGUIをやり直す必要があれば、ゼロからGUIを書き直したほうがより簡単かもしれません。

Chapter 3

3.ネットワークシミュレーションのためのGUI

開発者用ツールとして、Photonのクライアントライブラリーは、ラグ（遅延）や損失といったネットワーク状態のシミュレーションをすることができます。

PUN のパッケージには、ゲームのランタイム時に関連する値を設定するための、簡単なGUIコンポーネントが含まれています。

これを使うには、シーンの中の有効なゲームオブジェクトに PhotonNetSimSettingsGui コンポーネントを追加してください。

ランタイム時には、画面左上に現在のラウンドトリップタイム（RTT）とネットワークシミュレーションのコントロールボタンなどが表示されます：

- RTT： ラウンドトリップタイムは、サーバーによりメッセージが確認されるまでの平均値をミリセカンド単位で表すものです。（+/-の後にある）差異の数値はRTTの安定度（低い数値ほど良い）を表します。
- "Sim" トグルスイッチ：シミュレーションを有効／無効にします。ネットワークの状況に突然の大きな変化があると、ネットワークが切断されることがあります。
- "Lag" スライダー：送受信のメッセージに固定の遅延を付加します。ミリセカンド単位。
- "Jit" スライダー：メッセージ毎に「Xミリセカンド以下」のランダムな遅延を付加します。
- "Loss" スライダー：設定したパーセントのメッセージを落として、ロス（損失）とします。ちなみに、現在のインターネットでは、想定される損失は2%未満とされています。

Chapter 4

4.ネットワークの統計値のためのGUI

PhotonStatsGui は、ランタイム時にの対象ネットワークの数的指標を簡単に表示するシンプルなGUIのコンポーネントです。

使い方

PhotonStatsGui コンポーネントを、ヒエラルキーの中のアクティブなゲームオブジェクトに追加するだけです。（ランタイム時に）ウインドウがあらわれ、メッセージのカウンターが表示されます。

いくつかのトグルスイッチがウインドウを構成します：

- buttons（ボタン）："stats on"（計測オン）、 "reset stats"（統計リセット）、と "to log"（ログへ）のボタンを表示します。
- traffic（トラフィック）：下層レベルのネットワークトラフィック（送受信毎のバイト数）を表示します。
- health（健全性）：送信、送出のタイミングとその最長の間隔を表示します。

メッセージの統計値

最上部の数値は「メッセージ」のカウンターです。オペレーション、応答そしてイベントのすべてがカウントされます。表示されるのは、送信、着信そしてその両方の合計値と測定時間あたりの平均値です。

トラフィックの統計値

これらは、バイト数とパケットのカウンターです。ネットワークを介して送信したもの、着信したものがすべて計測されます。たとえメッセージがほとんど無い状況でも、偶発的にトラフィックの測定値が膨大になり、あまり接続が安定していないクライアントの接続を切断させてしまうこともあります。特にメッセージが送信されていない時でもパケットが送信されているがわかると思います。これらは接続の維持などをしています。

健全性の統計値

"longest delta between"（間隔の最大値）で始まるブロックは、クライアントのパフォーマンスについての指標です。送信とディスパッチの連続したコール間に経過した時間を測定します。通常、これらは１秒間に10回コールされます。もし数値が１秒を超えるようであれば、なぜ Update() コールが遅延しているかを確認するべきでしょう。

”Reset”（リセット）ボタン

統計値をリセットしますが、測定は継続されます。 状況や条件が変わった場合のメッセージのカウントを追跡するような時にこれは便利です。

”To Log”（ログへ）ボタン

これを押すと現在の統計値をログに記録します。どれだけ進歩しているかという概要をつかんだり、単に参考値として参照したりするのに役立ちます。

16 Gui for Network Statistics

”Stats On”（計測オン）ボタン（トラフィックの測定を有効にする）

Photon のライブラリーは様々なネットワークの統計値を測定できますが、通常はオフとなっています。PhotonStatsGui は、測定とそれらの数値の表示をできるようにします。

GUI上の"stats on" （計測オン）のトグルスイッチは、トラフィックの統計値を計測するかどうかを決定します。インスペクターパネルの チェックボックス"Traffic Stats On" も全く同じものです。

Chapter 5

5.パブリックPUN APIとは

PUN のパブリックAPI は、開発者に役立つと考えられるあらゆるコードから構成されています。

PUNの重要な要素であるこの API を効率的に学んでいただけるように、これらの各クラスは、このレファレンスの中の「モジュール」として分類されています。

Chapter 6

6.モジュールの索引

6.1 モジュール

これはすべてのモジュールのリストです：

Chapter 7
7.ネームスペースの索引

7.1 パッケージ（Packages）

これがネームスペース（名前空間）パッケージとその短い説明（ある場合は）です。

Chapter 8
8.階層の索引

8.1クラスの階層

この継承リストは、完全ではありませんが、大まかにアルファベット順に並べられています：

Chapter 9

9.クラスの索引

9.1 クラスリスト

以下は、クラスおよび構造体、共用体、インタフェースの索引とそれぞれの簡単な説明です：

Chapter 10

10. ファイルの索引

10.1 ファイルリスト

これはすべてのファイルのリストとその簡単な説明です：

Chapter 11

11. モジュールドキュメンテーション

11.1 オプショナルGUIエレメント

クラス

- class PhotonLagSimulationGui

  この MonoBehaviour のサブクラスは、Photon クライアントでのネットワークシミュレーション機能のための基本的な GUI です。ラグ（lag、固定値の遅延)、ジッター（jitter、ランダム値の遅延) そしてパケットの損失などを変更できます。

- class PhotonStatsGui

  Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。

- class PhotonStatsTracker

  Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。

11.1.1 解説

PUN のパッケージでは、ゲーム内での GUI コンポーネントは提供されていませんが、開発者の効率的な作業に役立てられるものがいくつか用意されています。

11.2 パブリックAPI

クラス

- class PhotonMessageInfo
  特定のメッセージ、RPC／更新情報に関する情報のコンテナのクラス。
- class PhotonStream
  OnPhotonSerializeView により書かれたデータを保持するために使うコンテナクラス。
- class PhotonNetwork
  PhotonNetwork プラグインを使用するためのメインのクラス。これは静的クラスです。
- class PhotonPlayer
  Room内で actorID により識別される任意の１プレイヤーについてまとめています。
- class PhotonView
  NetworkView に代わる、PUN でネットワークに利用されるクラス。 NetworkView と同じように使用してください。
- class Room
  このクラスは、PUN が参加する（している）Roomをそのまま反映します。ただし、RoomInfo のプロパティとは違い、このプロパティは設定可能であり、「自分の」Room を閉じたり、隠したりできます。
- class RoomInfo
  リスト表示や参加するために必要な情報だけに簡素化されたRoom。ロビーでのRoomのリスティングに使用します。プロパティ（open や maxPlayersなど）は設定できません

列挙型（enum型）

- enum PeerState {
  Uninitialized, PeerCreated, Connecting, Connected,
  Queued, Authenticated, JoinedLobby, DisconnectingFromMasterserver,
  ConnectingToGameserver, ConnectedToGameserver, Joining, Joined,
  Leaving, DisconnectingFromGameserver, ConnectingToMasterserver, ConnectedComingFromGameserver,
  QueuedComingFromGameserver, Disconnecting, Disconnected, ConnectedToMaster }
  接続／ネットワークPeerの詳細な状態。PUNでは、ロードバランサーと認証のワークフローは「内部的に」実装されているため、いくつかの状態は自動的に次の状態に進められます。そのような状態には、"(will-change)"(変更になります)というコメントがついています。.
- enum PhotonNetworkingMessage {
  OnConnectedToPhoton, OnLeftRoom, OnMasterClientSwitched, OnPhotonCreateRoomFailed,
  OnPhotonJoinRoomFailed, OnCreatedRoom, OnJoinedLobby, OnLeftLobby,
  OnDisconnectedFromPhoton, OnConnectionFail, OnFailedToConnectToPhoton, OnReceivedRoomList-
  Update,
  OnJoinedRoom, OnPhotonPlayerConnected, OnPhotonPlayerDisconnected, OnPhotonRandomJoinFailed,
  OnConnectedToMaster, OnPhotonSerializeView, OnPhotonInstantiate, OnPhotonMaxCccuReached,
  OnPhotonCustomRoomPropertiesChanged, OnPhotonPlayerPropertiesChanged, OnUpdatedFriendList,
  OnCustomAuthenticationFailed }
  この列挙型は、PUNにより送られる MonoMessages のセットを作り上げます。これらの定数の名前をメソッドとして実装すると、それに応じた状況でコールされるようになります。
- enum DisconnectCause {
  ExceptionOnConnect = StatusCode.ExceptionOnConnect, SecurityExceptionOnConnect = StatusCode.-
  SecurityExceptionOnConnect, TimeoutDisconnect = StatusCode.TimeoutDisconnect, DisconnectByClient-
  Timeout = StatusCode.TimeoutDisconnect,
  InternalReceiveException = StatusCode.InternalReceiveException, DisconnectByServer = StatusCode.-
  DisconnectByServer, DisconnectByServerTimeout = StatusCode.DisconnectByServer, DisconnectByServer-
  Logic = StatusCode.DisconnectByServerLogic,
  DisconnectByServerUserLimit = StatusCode.DisconnectByServerUserLimit, Exception = StatusCode.-
  Exception, InvalidRegion = ErrorCode.InvalidRegion, MaxCcuReached = ErrorCode.MaxCcuReached }
  切断の原因を要約しています。 OnConnectionFail と OnFailedToConnectToPhotonで使用されます。

- enum PhotonTargets {
  All, Others, MasterClient, AllBuffered,
  OthersBuffered }
  RPCの「ターゲット（target）」オプションの列挙型。どのリモートクライアントがその RPC コールを受け取るかを定義します。
- enum PhotonLogLevel { ErrorsOnly, Informational, Full }
  PUN のクラスにより吐き出すログ出力レベルの定義に使います。記録するのは、エラーだけ、エラーとより詳しい情報、すべて、のいずれかです。

11.2.1 解説

PUN のパブリックAPI は、開発者に役立つと考えられるあらゆるコードから構成されています。

このドキュメンテーションでは、パブリックAPI を中心に説明しています。

これらとは別に、PUNのフレームワーク自身により内部的に使用されるクラスも複数あります。しかも、いくつかの内部使用のクラスまでも公開されています。これは使い勝手の良さのためと、ある意味 Unityが機能している効果です。

11.2.2 列挙型

11.2.2.1 DisconnectCause (enum DisconnectCause)

切断の原因を要約します。 OnConnectionFail と OnFailedToConnectToPhotonで使用されます。

ExitGames.Client.Photon.StatusCode からステータスコードを抽出しています。 **以下も、ご参照ください。**

PhotonNetworkingMessage

列挙定数リスト：

ExceptionOnConnect 接続が確立されなかった。考えられる原因：ローカルサーバーが稼働していない。

SecurityExceptionOnConnect クライアントまたはサーバーのセキュリティの設定が接続を許可しない（コメントをみてください）。良くある原因としては、ブラウザークライアントが、「クロスドメイン」ファイルをサーバーから読み込もうとしている場合があります。そのファイルが提供されていない場合やクライアントに接続させないように構成されている場合に、この例外が発生します。Photon では、通常 Unityのためにこのクロスドメインファイルを提供しています。もしうまくいかない場合は、以下を読んでください：
http://doc.exitgames.com/photon-server/PolicyApp

TimeoutDisconnect 接続のタイムアウト。考えられる原因：リモートサーバーが稼働していないか、（ルーターかファイヤーウォールが原因で）要求したポートがブロックされている。

DisconnectByClientTimeout （ACK応答が長い間受け取れていないため判断した）クライアント側が、タイムアウトにより切断。

InternalReceiveException 受信ループでの例外処理。考えられる原因：ソケットのエラー。

DisconnectByServer サーバーが能動的にクライアントを切断。

DisconnectByServerTimeout （ACK応答が長い時間ないために判断した）サーバーが、タイムアウトにより切断。

DisconnectByServerLogic サーバーが能動的にクライアントを切断。考えられる原因：サーバー側の送信バッファーがフルになった（クライアントには大きすぎるデータ）。

DisconnectByServerUserLimit サーバーが能動的にクライアントを切断。考えられる原因：サーバー側のユーザー数が上限に達したため、（接続時に）クライアントが強制的に切られた。

Exception 接続を停止する何らかの例外が発生。

InvalidRegion (32756) アプリケーションのサブスクリプションプランが特定の地域のサーバーでは許可されていないため、Photon Cloudの認証に失敗。

MaxCcuReached (32757) サブスクリプションプランでの同時接続ユーザー数（CCU）が上限になったため、Photon Cloudの認証に失敗。

11.2.2.2 PeerState (enum PeerState)

接続／ネットワークPeerの詳細な状態。PUNでは、ロードバランサーと認証のワークフローは「内部的に」実装されているため、いくつかの状態は自動的に次の状態に進められます。そのような状態には、"(will-change)"（変更になります）というコメントがついています。

列挙定数リスト：

Uninitialized 実行していない状態。初期化する前、初めての利用の時にセットされる。

PeerCreated Peerが準備され、接続ができる状態。

Connecting マスターサーバーへの最初の接続を確立するために接続処理中（この処理が終了するまで、どのオペレーションも送信されない）。(will-change) 変更になります

Connected 接続が確立し、PUN が暗号化や認証のためにキーがやりとりをする状態。(will-change) 変更になります

Queued 現在、使用されていない。

Authenticated アプリケーションが認証された状態。AutoJoinLobby が false となっていない限りは、PUN は通常すぐロビーに参加します。(will-change) 変更になります。

JoinedLobby クライアントはマスターサーバーのロビーにいて、Roomリスティングを取得する状態。プレイするためRoomに入るには、Join、Create または、JoinRandom を使用してください。

DisconnectingFromMasterserver 切断処理中。(will-change) 変更になります。

ConnectingToGameserver（プレイするためにRoomを作るかRoomに参加するために、）ゲームサーバーに接続処理中。(will-change) 変更になります。

ConnectedToGameserver Connected と同様の状態、ただしゲームサーバーに接続。まだ、Roomを作るかRoomに参加するかの処理途中。(will-change) 変更になります。

Joining ゲームサーバー上で、Roomを作成かRoomに参加の処理中の状態。(will-change) 変更になります。

Joined Roomへの参加／作成の一連の処理の最終状態。この状態になり、このクライアントは、イベントやRPCのコールを他のクライアントとやりとりができます。

Leaving Roomから退出処理中。(will-change) 変更になります。

DisconnectingFromGameserver ワークフローでは、ゲームサーバーから出て、マスターサーバーに再接続します。(will-change) 変更になります。

ConnectingToMasterserver ワークフローでは、マスターサーバーに接続し、暗号化とアプリケーションの認証を確立します。 (will-change) 変更になります。

ConnectedComingFromGameserver Connected と同じ状態、ただしゲームサーバーからきている。(will-change) 変更になります。

QueuedComingFromGameserver Queued と同じ状態、ただしゲームサーバーからきている。 (will-change) 変更になります。

Disconnecting PUNは切断処理中です。この後は Disconnected になります。(will-change) 変更になります。

Disconnected 接続がされていない、接続準備の状態。PeerCreated と似たような状態。

ConnectedToMaster ロビーに参加することなく、マスターサーバーに接続する処理の最終状態。（AutoJoinLobby が false。）

11.2.2.3 PhotonLogLevel (enum PhotonLogLevel)

PUN のクラスにより作られるログ出力レベルの定義に使われます。記録するのは、エラーだけ(ErrorsOnly）、エラーとより詳しい情報（Informational）、すべて(Full)のいずれかです。

列挙定数リスト：

ErrorsOnly

Informational

Full

11.2.2.4 PhotonNetworkingMessage (**enum PhotonNetworkingMessage**)

この列挙型は、PUNにより送られる MonoMessages のセットを作り上げます。これらの定数の名前をメソッドとして実装すると、それに応じた状況でコールされるようになります。
コードの例： `public void OnLeftRoom() { //処理を記述 }`

列挙定数リスト：

OnConnectedToPhoton サーバーに接続し、クライアントが認証される前にコールされる。クライアントは他の処理をする前に、 OnJoinedLobby （または、OnConnectedToMaster）がコールされるのを待つ。
例：void OnConnectedToPhoton(){ ... } マスターサーバーからゲームサーバーへの移行（ユーザーには隠されています）の場合は、これはコールされない。

OnLeftRoom ローカルユーザーがRoomを出た時点でコールされる。 例：void OnLeftRoom(){ ... }

OnMasterClientSwitched 前のマスタークライアントがRoomを出たため（Roomに入った時ではなく）、新しいマスタークライアントにスイッチされた時にコールされる。前のマスタークライアントは、この時点で既に除されている。例：void OnMasterClientSwitched(PhotonPlayer newMasterClient){ ... }

OnPhotonCreateRoomFailed CreateRoom()コールが失敗した時にコールされる。多くの場合、既にじ名前が別のRoomで使用されていることによる。例：void OnPhotonCreateRoomFailed(){ ... }

OnPhotonJoinRoomFailed JoinRoom()コールが失敗した場合にコールされる。多くの場合、指定したRoomが存在しないか、またはRoomが定員に達していることによる。 例：void OnPhotonJoinRoomFailed(){ ... }

OnCreatedRoom CreateRoomが、Roomを作り終えた時にコールされる。これに続き、OnJoinedRoomがコールされる。（これはRoomを作ったか、単に参加するかに関わらずコールされる。） 例：void OnCreatedRoom(){ ... } ちなみに、この場合、このローカルクライアントがマスタークライアントであるということを意味します。

OnJoinedLobby マスターサーバーのロビーに入った時にコールされる。クライアントはRoomを作るか参加することができるが、OnReceivedRoomListUpdateがコールされるまで、Roomリストは提供されない。例：void OnJoinedLobby(){ ... } 注：PhotonNetwork.autoJoinLobby が false の場合、OnConnectedToMaster がかわりにコールされて、Roomリストは提供されない。また、ロビーに滞在している間、Roomリストは、決められた（変更できない）インターバルで自動的に更新される。

OnLeftLobby ロビーを出た後にコールされる。例：void OnLeftLobby(){ ... }

OnDisconnectedFromPhoton Photon Serverとの接続を切断した後にコールされる。このOnDisconnectedFromPhoton がコールされる前に、他のイベント、例としては OnConnectionFail や OnFailedToConnectToPhotonなどのイベントが起きている場合もある。例：void OnDisconnectedFromPhoton(){ ... }

OnConnectionFail （接続が確立された後に）何らかの原因で接続が失敗した場合にコールされ、この後にOnDisconnectedFromPhotonがコールされる。最初にサーバーに接続できなかった場合には、この代わりにOnFailedToConnectToPhoton がコールされる。エラーの理由は、ステータスコードで提供される。例： void OnConnectionFail(DisconnectCause cause){ ... }

OnFailedToConnectToPhoton 接続が確立される前に Photon Server への接続処理が失敗した場合にコールされ、次にOnDisconnectedFromPhoton のコールが続く。もし、接続が確立された後に失敗の場合は、 OnConnectionFail がコールされる。例：void OnFailedToConnectToPhoton(DisconnectCause cause){... }

OnReceivedRoomListUpdate Roomリストが、新規に提供されるか、または既にるリストが更新された場合にコールされる。（マスターサーバー上の）ロビーにいる状態の場合のみコールされる。例：void OnReceivedRoomListUpdate(){ ... }

OnJoinedRoom （Roomの作成かRoomへの参加により）Roomに入る時にコールされる。すべてのクライアント（マスタークライアントを含む）でコールされる。例：void OnJoinedRoom(){ ... }

OnPhotonPlayerConnected リモートプレイヤーがRoomに接続した後にコールされる。この時点で、この Photonプレイヤー はプレイヤーリストに既に加されている。例：void OnPhotonPlayerConnected(PhotonPlayer newPlayer){ ... }

OnPhotonPlayerDisconnected リモートプレイヤーがRoomから切断された後にコールされる。この時点で、この Photonプレイヤー はプレイヤーリストから既に除されている。例：void OnPhotonPlayerDisconnected(PhotonPlayer otherPlayer){ ... }

OnPhotonRandomJoinFailed JoinRandom()コールが失敗した後にコールされる。多くの場合、すべてのRoomが定員いっぱいかRoomが無いことが原因でおきる。例：void OnPhotonRandomJoinFailed(){ ... }

OnConnectedToMaster PhotonNetwork.AutoJoinLobby が false である場合に限り、マスターへの接続が確立され、認証された後にコールされる。もしAutoJoinLobby が false であると利用できるRoomのリストは提供されないが、(名前を指定してかランダムに）Roomに参加する、またはRoomを作成すること、いずれも可能である。例：void OnConnectedToMaster(){ ... }

OnPhotonSerializeView PhotonView により監視される 各MonoBehaviour へのネットワーク上の各更新情報毎にコールされる。例：void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info){ ...}

OnPhotonInstantiate PhotonNetwork.Instantiate により生成された GameObject（とその子オブジェクト）上のすべてのスクリプトでコールされる。例：void OnPhotonInstantiate(PhotonMessageInfo info){ ... }

OnPhotonMaxCccuReached 同時接続数の上限に（一時的に）達しているため、このクライアントはサーバーに許可されずに切断される。この場合、ユーザーは後で再び接続を試みることもできる。OnPhotonMaxCcuReached() の場合、クライアントは切断されるため、Roomを作成したりRoomに参加したりはできない。(サーバーを自前でホストしている場合は）ライセンス内容を変更して、また(Photon Cloud 使用の場合は）サブスクリプションプランをグレードアップさせて、同時接続数の上限（CCU limit）を増やすことができる。Photon Cloud は、CCU limit が上限に達すると、メールで通知をする。これは ダッシュボード （ウェブページ）でも状況が確認できる。 例：void OnPhotonMaxCccuReached(){ ... }

OnPhotonCustomRoomPropertiesChanged Roomにいる時、Roomのカスタムプロパティが変更されるとコールされる。また、これはローカルプレイヤーがRoomプロパティを変更した時にもコールされる。例：void OnPhotonCustomRoomPropertiesChanged(){ ... }

OnPhotonPlayerPropertiesChanged Roomにいる時、プレイヤーのだれかのカスタムプロパティが変更されるとコールされる。例：void OnPhotonPlayerPropertiesChanged(PhotonPlayer player){ ... }

OnUpdatedFriendList FindFriendsリクエストに対しサーバーが回答を返し、PhotonNetwork.Friends が更新された時にコールされる。例：void OnUpdatedFriendList(){ ... }

OnCustomAuthenticationFailed （間違ったユーザー入力、不正なトークンや暗号、または間違った設定などにより）カスタム認証が失敗した場合にコールされる。切断がこれに続く。例：void OnCustomAuthenticationFailed(string debugMessage){ ... } ダッシュボード 内で、対象アプリケーションにカスタム認証を設定しない限り、これはコールされません。もし認証が成功した場合もこのメソッドはコールされずに、（通常どおり） OnJoinedLobby や OnConnectedToMaster がコールされます。

11.2.2.5 PhotonTargets **(enum PhotonTargets)**

RPCの「ターゲット（target）」オプションの列挙型。どのリモートクライアントがその RPC コールを受け取るかを定義します。

列挙定数のリスト：

All

Others

MasterClient

AllBuffered

OthersBuffered

Chapter 12

12. ネームスペースドキュメンテーション

12.1 Photon パッケージ

クラス

- class MonoBehaviour

  このクラスは、PhotonViewプロパティを追加します。また、ゲーム内で networkViewがまだ使用されている場合は、警告をログに出します。

Chapter 13

13. クラスドキュメンテーション

13.1 ActorProperties クラス レファレンス

定数のクラス。この値（byte型）は、Actor/Playerの標準のプロパティを定義します。PUNが内部的にこれらの定数を使用します。

パブリック属性

- const byte PlayerName = 255
  (255) player/actor の名前

13.1.1 解説

定数のクラス。この（byte型）の値は、Actor/Playerの標準のプロパティを定義します。PUNが内部的にこれらの定数を使用します。
「カスタムプロパティ（Custom properties）」は、キーとして文字列型を使わなければいけません。それらは任意に割り当てることができます。

13.1.2 メンバーデータ

13.1.2.1 PlayerName const byte ActorProperties.PlayerName = 255
(255) 1 player/actor の名前

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.2 AuthenticationValues クラス レファレンス

Photon でのカスタム認証の値（デフォルト：ユーザーとトークン）のコンテナクラスです。接続する前に AuthParameters を設定、その他、認証の値に関するすべてが扱われます。

パブリックメンバー関数

- virtual void SetAuthParameters (string user, string token)
  ユーザーとトークンの値から、エスケープ処理して、デフォルトのパラメーター文字列を作成します。替わりに、AuthParameters を独自に決めることもできます。

override string ToString ()

パブリック属性

- CustomAuthenticationType AuthType = CustomAuthenticationType.Custom
  使用すべきカスタム認証サービスのタイプです。現在は、"Custom"(カスタム） か "None"(これをオフにする)だけです。.
- string AuthParameters
  この文字列は、使用する認証サービスで必要とされるすべてのHTTPのゲット（http get）パラメーターを含む必要があります。デフォルトでは、ユーザー名とトークンです。
- string Secret
  初期認証の後、Photon はクライアント／ユーザーに対し、それ以降に（キャッシュされ）バリデーションに使われる秘密のキーを提供します。

13.2.1 解説

Photon での「カスタム認証（Custom Authentication）」の値（デフォルト：ユーザーとトークン）のコンテナクラスです。
接続する前に AuthParameters を設定、またその他すべてが処理されます。
カスタム認証により、エンドユーザーをログインやトークンなどの方法で確認することができます。これらの値は Photon に送られ、そしてクライアントをアクセスさせるか切断する前に Photon により検証されます。
Photon Cloud のダッシュボード上で、この機能を有効にし、重要なサーバー値などの設定することができます。
https://cloud-jp.exitgames.com/dashboard

13.2.2 メンバー関数

13.2.2.1 SetAuthParameters **virtual void AuthenticationValues.SetAuthParameters (string user, string token)** `[virtual]`
ユーザーとトークンの値から、エスケープ処理して、デフォルトのパラメーター文字列を作成します。替わりに、AuthParameters を独自に決めることもできます。
デフォルトのパラメーターは次の通りです： `"username={user}&token={token}"`
パラメーター

user	カスタム認証で使用される名前またはエンドユーザーIDなど。
token	Photon への初期ログインで使用される認証サービスにより提供されるトークン。

13.2.2.2 ToString **override string AuthenticationValues.ToString ( )**

13.2.3 メンバーデータ

13.2.3.1 AuthParameters **string AuthenticationValues.AuthParameters**
この文字列は、使用する認証サービスで必要とされるすべての HTTP のゲット（http get）パラメーターを含む必要があります。デフォルトでは、ユーザー名とトークンです。
標準の HTTP のゲット（http get）パラメーターが使用され、サーバー（Photon Cloud のダッシュボード）で定義されているサービスに引き渡されます。

13.2.3.2 AuthType **CustomAuthenticationType AuthenticationValues.AuthType = CustomAuthenticationType.Custom**
使用すべきカスタム認証プロバイダーのタイプです。現在は、"Custom"（カスタム） か "None"（これをオフにする）だけです。

13.2.3.3 Secret **string AuthenticationValues.Secret**

初期認証の後、Photon はクライアント／ユーザーに対し、それ以降に（キャッシュされ）バリデーションに使われる秘密のキーを提供します。

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.3 ErrorCode クラス レファレンス

定数のクラス。これらの値(int型) は、Photonのロードバランサーロジックで定義され送信されるエラーコードを表します。PUNが内部的にこれらの定数を使用します。

パブリック属性

- const int Ok = 0
  は常に 「OK」を意味し、それ以外はエラーもしくは特定の状況。
- const int OperationNotAllowedInCurrentState = -3
  (-3) オペレーションがまだ実行することができない。（例 : OpJoin は認証される前にはコールできない。 RaiseEvent はRoom に入る前には使用できない。）.
- const int InvalidOperationCode = -2
  (-2) コールしたオペレーションが、接続しているサーバー（アプリケーション）で実装されていない。適合するアプリケーションを実行していることを確認する。
- const int InternalServerError = -1
  (-1) サーバーで何らかの不具合が起こった。再現を試み、Exit Gamesに連絡してください。
- const int InvalidAuthentication = 0x7FFF
  (32767) 認証の失敗。考えられる原因：アプリケーションIDが（クラウドサービスの）Photon で確認できない。
- const int GameIdAlreadyExists = 0x7FFF - 1
  (32766) ゲームID（名前）が既にわれている(同じものは作れない）。名前を変更する。
- const int GameFull = 0x7FFF - 2
  (32765) ゲーム／Roomが満員。ゲーム参加処理中に他のプレイヤーがかわりに参加した場合に起こり得る。
- const int GameClosed = 0x7FFF - 3
  (32764) ゲーム／Roomが閉じられ、参加ができない。他のゲームに参加する。
- const int AlreadyMatched = 0x7FFF - 4
- const int ServerFull = 0x7FFF - 5
  (32762) 現在、使用されていない。
- const int UserBlocked = 0x7FFF - 6
  (32761) 現在、使用されていない。
- const int NoRandomMatchFound = 0x7FFF - 7
  (32760) ランダムマッチメイキングは、閉じられていないかつ満員でないRoomが存在する場合のみに成功する。何秒後かに再トライするか、Roomを新しく作成する。
- const int GameDoesNotExist = 0x7FFF - 9
  (32758) Join は、Room（名前）が(既に存在しない場合失敗する。参加処理中に他のプレイヤーがいなくなった時に起こり得る。
- const int MaxCcuReached = 0x7FFF - 10
  (32757) アプリケーションでのサブスクリプションプランの同時接続数（CCU）が上限に達したのため、Photon Cloud での認証が失敗。
- const int InvalidRegion = 0x7FFF - 11
  (32756) アプリケーションでのサブスクリプションプランが指定の地域サーバーの利用を許可していないため、Photon Cloud での認証が失敗。
- const int CustomAuthenticationFailed = 0x7FFF - 12
  (32755) 設定上の問題（Cloud ダッシュボード 参照）か、入力されたユーザーデータ（ユーザーネームやトークン）が原因で、ユーザーのカスタム認証が失敗。詳細はエラーメッセージを確認。

13.3.1 解説

定数のクラス。これらの値(int型) は、Photonのロードバランサーロジックで定義され送信されるエラーコードを表ます。PUNが内部的にこれらの定数を使用します。
注意：Photon Core からのコードは負数です。デフォルトでアプリケーションのエラーコードは、short.max（short型整数の最大値）から下に下がっていきます。

13.3.2 メンバーデータ

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

13.3.2.2 CustomAuthenticationFailed **const int ErrorCode.CustomAuthenticationFailed = 0x7FFF - 12**
(32755) 設定上の問題（Cloud ダッシュボード 参照）か、入力されたユーザーデータ（ユーザーネームやトークン）が原因で、ユーザーのカスタム認証が失敗。詳細はエラーメッセージを確認。

13.3.2.3 GameClosed **const int ErrorCode.GameClosed = 0x7FFF - 3**
(32764) ゲームが閉じられ、参加ができない。他のゲームに参加する。

13.3.2.4 GameFull **const int ErrorCode.GameDoesNotExist = 0x7FFF - 9**
(32758) Join は、Room（名前）が(既に）存在しない場合失敗する。参加処理中に他のプレイヤーがいなくなった時に起こり得る。

13.3.2.5 GameFull **const int ErrorCode.GameFull = 0x7FFF - 2**
(32765) ゲームが満員。ゲーム参加処理中に他のプレイヤーがかわりに参加した場合に起こり得る。

13.3.2.6 GameIdAlreadyExists **const int ErrorCode.GameIdAlreadyExists = 0x7FFF - 1**
(32766) ゲーム ID（名前）が既に使われている(同じものは作れない）。名前を変更する。

13.3.2.7 InternalServerError **const int ErrorCode.InternalServerError = -1**
(-1) サーバーで何らかの不具合が起こった。再現を試み、Exit Games に連絡してください。

13.3.2.8 InvalidAuthentication **const int ErrorCode.InvalidAuthentication = 0x7FFF**
(32767) 認証の失敗。考えられる原因：アプリケーション ID が（クラウドサービスの）Photon で確認できない。

13.3.2.9 InvalidOperationCode **const int ErrorCode.InvalidOperationCode = -2**
(-2) コールしたオペレーションが、接続しているサーバー（アプリケーション）で実装されていない。適合するアプリケーションを実行していることを確認する。

13.3.2.10 InvalidRegion **const int ErrorCode.InvalidRegion = 0x7FFF - 11**
(32756) アプリケーションでのサブスクリプションプランが指定の地域サーバーの利用を許可していないため、Photon Cloud での認証が失敗。

Photon Cloud のサブスクリプションプランのいくつかは、地域が限定されています。その場合、他の地域のサーバーは使用できません。
Photon Cloud のダッシュボード（https://cloud-jp.exitgames.com/dashboard）上のサーバー情報と、設定されているマスターサーバーのアドレスを比べて、確認してください。
OpAuthorize は、Photon Cloud 上でのみ使用される接続ワークフローの一部で、このエラーが起こることがあります。自前でホストしている同時接続数限定ライセンスの Photon Server の場合は、Photon Cloud 用クライアントを接続させることはありません。

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

(32757) アプリケーションでのサブスクリプションプランの同時接続数（CCU）が上限に達したのため、Photon Cloud での認証が失敗。
「CCU Burst（CCU 制限解除）」のプランを持っていない限り、接続処理中に認証の段階でクライアントが接続失敗となります。影響を受けたユーザーはオペレーションをコールすることはできません。次のことに注意してください。あるゲームを終了しマスターサーバーに戻ったプレイヤーは、一旦切断され再接続されます。つまり、プレイを楽しんだばかりで、そのすぐ後に再接続を拒否されることがある、ということです。これは一時的なことです。同時接続数が上限より小さくなれば、プレイヤーはゲームに接続することができるようになります。
OpAuthorize は、Photon Cloud 上でのみ使用される接続ワークフローの一部で、このエラーが起こることがあります。自前でホストしている同時接続数限定ライセンスの Photon Server の場合は、Photon Cloud 用クライアントを接続させることはありません。

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

(32760) ランダムマッチメイキングは、閉じられていないかつ満員でない Room が存在する場合のみに成功します。何秒後かに再トライするか、Room を新しく作成するようにします。

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

(0) は常に 「OK」を意味し、それ以外はエラーもしくは特定の状況を意味します。

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

(-3) オペレーションがまだ実行することができません。（例 : OpJoin は認証される前にはコールできない。 RaiseEvent は Room に入る前には使用できない。）.
Photon Cloud サービスでオペレーションをコールする前に、自動化されたクライアントでのワークフローが認証処理を完了していなければいけません。
PUN では、状態が、JoinedLobby (AutoJoinLobby = true の時) もしくは ConnectedToMaster (AutoJoinLobby = false の時) になるまで待つようにしてください。

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

(32762) 現在、使用されていない。

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

(32761) 現在、使用されていない。

このクラスのドキュメンテーションは、次のファイルから生成されました：
Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.4 EventCode クラス レファレンス

定数のクラス。これらの値は、Photonのロードバランサーで定義されたイベントを表します。PUNが内部的にこれらの定数を使用します。.

パブリック属性

- const byte GameList = 230
  (230) （マスター上のロビー内で）RoomInfosの初期リスト
- const byte GameListUpdate = 229
  (229) （マスター上のロビー内で）初期リストと結合させる RoomInfos の更新情報
- const byte QueueState = 228
  (228) 現在、使用されていない。サーバーがフルの場合のキューされている状態
- const byte Match = 227
  (227) 現在、使用されていない。マッチメイキング用のイベント
- const byte AppStats = 226
  (226) このアプリケーションの統計（プレイヤー、Room、その他）のイベント
- const byte AzureNodeInfo = 210
  (210) Azure でホスティングされている場合に内部的に使われる。
- const byte Join = (byte)LiteEventCode.Join
  (255) Joinイベント：だれかがゲームに参加した。新しい actorNumber とアクター（actor）のプロパティが提供される(OpJoin で設定されている場合）。
- const byte Leave = (byte)LiteEventCode.Leave
  (254) Leaveイベント: ゲームを出たプレイヤーは、actorNumber で識別される。
- const byte PropertiesChanged = (byte)LiteEventCode.PropertiesChanged
  (253) ブロードキャストのオプションが「オン」で OpSetProperties をコールすると、このイベントが発生される。設定されるプロパティが含まれている。
- const byte SetProperties = (byte)LiteEventCode.PropertiesChanged
  (253) ブロードキャストのオプションが「オン」で OpSetProperties をコールすると、このイベントが発生される。設定されるプロパティが含まれている。

13.4.1 解説

定数のクラス。これらの値は、Photonのロードバランサーで定義されたイベントを表します。PUNが内部的にこれらの定数を使用します。

この番号は、255 から始まり小さくなっていきます。開発するゲーム内の独自のイベントは、0 から始めることができます。

13.4.2 メンバーデータ

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

(226) このアプリケーションの統計（プレイヤー、Room、その他）のイベント

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

(210) Azure でホスティングされている場合に内部的に使われる。

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

(230) （マスター上のロビー内で）RoomInfos の初期リスト

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

(229) （マスター上のロビー内で）初期リストと結合させる RoomInfos の更新情報

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

(255) Join イベント : だれかがゲームに参加した。新しい actorNumber とアクター（actor）のプロパティが提供される(OpJoin で設定されている場合）。

13.4.2.6 Leave **const byte EventCode.Leave = (byte)LiteEventCode.Leave**

(254) Leave イベント: ゲームを出たプレイヤーは、actorNumber で識別される。

13.4.2.7 Match **const byte EventCode.Match = 227**

(227) 現在、使用されていない。マッチメイキング用のイベント

13.4.2.8 PropertiesChanged **const byte EventCode.PropertiesChanged = (byte)LiteEventCode.PropertiesChanged**

(253) ブロードキャストのオプションが「オン」で OpSetProperties をコールすると、このイベントが発生される。設定されるプロパティが含まれている。

13.4.2.9 QueueState **const byte EventCode.QueueState = 228**

(228) 現在、使用されていない。サーバーがフルの場合のキューされている状態

13.4.2.10 SetProperties **const byte EventCode.SetProperties = (byte)LiteEventCode.PropertiesChanged**

(253) ブロードキャストのオプションが「オン」で OpSetProperties をコールすると、このイベントが発生される。設定されるプロパティが含まれている。

このクラスのドキュメンテーションは、次のファイルから生成されました：

Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.5 Extensions クラス レファレンス

この静的クラスは、他のクラスのための便利な拡張メソッドをいくつか定義しています。（例：Vector3、浮動小数型など）

静的パブリックメンバー関数

- static PhotonView[ ] GetPhotonViewsInChildren (this UnityEngine.GameObject go)
- static PhotonView GetPhotonView (this UnityEngine.GameObject go)
- static bool AlmostEquals (this Vector3 target, Vector3 second, float sqrMagnitudePrecision)
  target と second の差分（target – second）の平方マグニチュードを、任意の浮動小数値と比較します。
- static bool AlmostEquals (this Vector2 target, Vector2 second, float sqrMagnitudePrecision)
  target と second の差分（target – second）の平方マグニチュードを、任意の浮動小数値と比較します。
- static bool AlmostEquals (this Quaternion target, Quaternion second, float maxAngle)
  target と second 間の角度を、任意の浮動小数値と比較します。
- static bool AlmostEquals (this float target, float second, float floatDiff)
  ２つの浮動小数値を比較し、その差が floatDiff より小さい場合 true を返します。
- static void Merge (this IDictionary target, IDictionary addHash)
  addHash のすべてのキーを target に結合します。新しいキーを追加し、 target に存在するキーの値を更新します。
- static void MergeStringKeys (this IDictionary target, IDictionary addHash)

target のハッシュテーブルに文字列キーを結合します。

- static string ToStringFull (this IDictionary origin)
  タイプ情報も含めた IDictionary の内容の説明文字列を返します。注意：頻繁に使用すると「非常に重たい」コールとなりますが、Dictionary やハッシュテーブルの内容のデバッグにとても役立ちます。
- static Hashtable StripToStringKeys (this IDictionary original)
  このメソッドは original のすべての文字列型のキーを、新しいハッシュテーブルにコピーします。
- static void StripKeysWithNullValues (this IDictionary original)
  これは、値が null を参照しているキーとバリューのペアをすべて削除します。値が null に設定されると Photon プロパティは削除されます。元の originalの IDictionary は変更されます。
- static bool Contains (this int[ ] target, int nr)
  特定の整数値がある整数の配列の中にあるかを調べます。

13.5.1 解説

この静的クラスは、他のクラスのための便利な拡張メソッドをいくつか定義しています。（例：Vector3、浮動小数型など）

13.5.2 メンバー関数

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

target と second の差分（target - second）の平方マグニチュードを、任意の浮動小数値と比較します。

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

target と second の差分（target - second）の平方マグニチュードを、任意の浮動小数値と比較します。

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

target と second 間の角度を、任意の浮動小数値と比較します。

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

２つの浮動小数値を比較し、その差が floatDiff より小さい場合 true を返します。

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

特定の整数値がある整数の配列の中にあるかを調べます。

これは Room のプレイヤーリストの中に、特定の actorNumber があるか検索するのに役立つでしょう。

パラメーター

target	調べる対象の整数の配列。
nr	target の中で検索する番号。

戻り値

nr が target で見つかった場合、true。

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

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

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

addHash のすべてのキーを target に結合します。新しいキーを追加し、 target に存在するキーの値を更新します。

パラメーター

target	更新する IDictionary
addHash	target に結合するデータを含む IDictionary

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

target のハッシュテーブルに文字列キーを結合します。

target からキーを削除しません（ですから、文字列以外のキーが target に既にった場合は、それはそのままです）。

パラメーター

target	渡された target IDicitionary および addHash のすべての文字列キー
addHash	target の更新のため、部分的に target に結合されるべき IDictionary

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

これは、値が null を参照しているキーとバリューのペアをすべて削除します。値が null に設定されると Photon プロパティは削除されます。元の original の IDictionary は変更されます。

パラメーター

original	値が null のキーが削除される対象の IDictionary

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

このメソッドは original のすべての文字列型のキーを、新しいハッシュテーブルにコピーします。

元のルートハッシュで値となっているかもしれないハッシュに対して再帰的に処理しません。また、これは original を変更しません。

パラメーター

original	文字列キーを取得する元の IDictonary

戻り値

original を部分的に含む新しいハッシュテーブル（Hashtable）

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

タイプ情報も含めた IDictionary の内容の説明文字列を返します。注意：頻繁に使用すると「非常に重たい」コールとなりますが、Dictionary やハッシュテーブルの内容のデバッグにとても役立ちます。

パラメーター

origin	Dictionary または ハッシュテーブル

戻り値

IDictionary の内容の文字列

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs

13.6 FriendInfo クラス レファレンス

オンラインの友達の状態とどのRoomにいるかの情報を格納するために使用されます。

パブリックメンバー関数

override string ToString ()

プロパティ

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

13.6.1 解説

オンラインの友達の状態とどのRoomにいるかの情報を格納するために使用されます。

13.6.2 メンバー関数

13.6.2.1 ToString **override string FriendInfo.ToString ( )**

13.6.3 プロパティ

13.6.3.1 IsInRoom **bool FriendInfo.IsInRoom** `[get]`

13.6.3.2 IsOnline **bool FriendInfo.IsOnline** `[get]`, `[set]`

13.6.3.3 Name **string FriendInfo.Name** `[get]`, `[set]`

13.6.3.4 Room **string FriendInfo.Room** `[get]`, `[set]`

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs

13.7 GameProperties クラス レファレンス

定数のクラス。これらの値(byte型)は、Photonのロードバランサーで使用されるRoomおよびゲームの標準のプロパティのためのものです。PUNが内部的にこれらの定数を使用します。

パブリック属性

- const byte MaxPlayers = 255
  (255) Roomに収容する最大プレイヤー数。0 は 「無制限」を意味します。
- const byte IsVisible = 254
  (254) マスター上のロビー内でこのRoomをリストにのせるかどうか
- const byte IsOpen = 253
  (253) Roomにさらに多くのプレイヤーの参加を認めるかどうか
- const byte PlayerCount = 252
  (252) 現在のRoomのプレイヤー数カウント。マスター上のロビーの中でだけ使用される。
- const byte Removed = 251
  (251) Roomリスティングから、そのRoomが削除されるべき場合は true （マスターのロビー内でのRoomリストの更新で使われる）。
- const byte PropsListedInLobby = 250
  (250) ロビー内の RoomInfo リストに渡す Roomプロパティのリスト。これは、CreateRoom で利用される。CreateRoom では 1 Room毎に 1 回このリストを定義する。
- const byte CleanupCacheOnLeave = 249
  Join オペレーションのパラメーター CleanupCacheOnLeave と同じもの

13.7.1 解説

定数のクラス。これらの値(byte型)は、Photonのロードバランサーで使用されるRoomおよびゲームの標準のプロパティのためのものです。PUNが内部的にこれらの定数を使用します。
「カスタムプロパティ（Custom properties）」は、キーとして文字列型を使わなければいけません。それらは任意に割り当てることができます。

13.7.2 メンバーデータ

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

Join オペレーションのパラメーター CleanupCacheOnLeave と同じものです。

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

(253) Room にさらに多くのプレイヤーの参加を認めるかどうか、のために利用します。

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

(254) マスター上のロビー内でこの Room をリストにのせるかどうか、を定義します。

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

(255) Room に収容する最大プレイヤー数。0 は 「無制限」を意味します。

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

(252) 現在の Room のプレイヤー数カウント。マスター上のロビーの中でだけ使用されます。

13.7.2.6 PropsListedInLobby **const byte GameProperties.PropsListedInLobby = 250**

(250) ロビー内の RoomInfo リストに渡す Room プロパティのリスト。これは、CreateRoom で利用されます。CreateRoom では 1 Room 毎に 1 回このリストを定義します。

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

(251) Room リスティングから、その Room が削除されるべき場合は true （マスターのロビー内での Room リストの更新で使われます）。

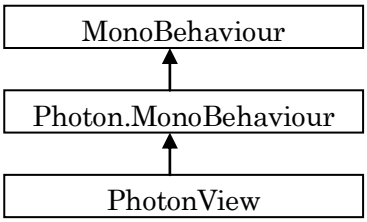
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.8 Photon.MonoBehaviour クラス レファレンス

このクラスは、photonViewプロパティを追加します。また、ゲーム内で networkViewがまだ使用されている場合は、警告をログに出します。

継承概念図： Photon.MonoBehaviour



プロパティ

PhotonView photonView `[get]`

new PhotonView networkView `[get]`

13.8.1 解説

このクラスは、photonViewプロパティを追加します。また、ゲーム内で networkViewがまだ使用されている場合は、警告をログに出します。

13.8.2 プロパティ

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

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

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs

13.9 OperationCode クラス レファレンス

定数のクラス。オペレーション関連のコードが含まれます。PUNが内部的にこれらの定数を使用します。

パブリック属性

- const byte Authenticate = 230
  (230) Peerを認証し、仮想アプリケーションに接続する
- const byte JoinLobby = 229

(229) マスター上のロビーに参加する

- const byte LeaveLobby = 228
  (228) マスター上のロビーから出る
- const byte CreateGame = 227
  (227) ゲーム（game）を作る(名前が既に存在する場合は失敗)
- const byte JoinGame = 226
  (226) （名前を指定し）ゲームに参加する
- const byte JoinRandomGame = 225
  (225) （マスター上で）ゲームにランダムに参加する
- const byte Leave = (byte)LiteOpCode.Leave
  (254) Roomから出るための OpLeave のためのコード
- const byte RaiseEvent = (byte)LiteOpCode.RaiseEvent
  (253) （他のアクター／プレイヤーのために、Room内で）イベントを発生させる
- const byte SetProperties = (byte)LiteOpCode.SetProperties
  (252) （Roomやアクター／プレイヤーの）プロパティを設定する
- const byte GetProperties = (byte)LiteOpCode.GetProperties
  (251) プロパティを取得する
- const byte ChangeGroups = (byte)LiteOpCode.ChangeGroups
  (248) 各Roomでのそれぞれのインタレストグループを変更するオペレーションコード（Liteおよびそれを拡張したアプリケーションで）
- const byte FindFriends = 222
  (222) （固有である名前による）友達リストのRoom とオンラインの状態をリクエストする

13.9.1 解説

定数のクラス。オペレーション関連のコードが含まれます。PUNが内部的にこれらの定数を使用します。

13.9.2 メンバーデータ

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

(230) Peer を認証し、仮想アプリケーションに接続します。

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

(248) 各 Room でのそれぞれのインタレストグループを変更するオペレーションコード（Lite およびそれを拡張したアプリケーションでの）となります。

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

(227) ゲーム（game）を作ります(名前が既に存在する場合は失敗となります）。

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

(222) （固有である名前による）友達リストの Room とオンラインの状態をリクエストします。

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

(251) プロパティを取得します。

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

(226) （名前を指定し）ゲームに参加する

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

(229) マスター上のロビーに参加する

13.9.2.8 JoinRandomGame **const byte OperationCode.JoinRandomGame = 225**

(225) （マスター上で）ゲームにランダムに参加する

13.9.2.9 Leave **const byte OperationCode.Leave = (byte)LiteOpCode.Leave**

(254) Room から出るための OpLeave のためのコードです。

13.9.2.10 LeaveLobby **const byte OperationCode.LeaveLobby = 228**

(228) （マスター上の）ロビーから出る

13.9.2.11 RaiseEvent **const byte OperationCode.RaiseEvent = (byte)LiteOpCode.RaiseEvent**

(253) （他のアクター／プレイヤーのために、Room 内で）イベントを発生させる

13.9.2.12 SetProperties **const byte OperationCode.SetProperties = (byte)LiteOpCode.SetProperties**

(252) （Room やアクター／プレイヤーの）プロパティを設定する

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.10 ParameterCode クラス レファレンス

定数のクラス。オペレーションとイベントのパラメーターのコードです。PUNが内部的にこれらの定数を使用します。

パブリック属性

- const byte Address = 230
  (230) 利用する（ゲーム）サーバーのアドレス
- const byte PeerCount = 229
  (229) あるRoom内のプレイヤー数（そのゲームアプリケーションで、ゲームサーバーに接続しているプレイヤー。統計イベントで使われる。）
- const byte GameCount = 228
  (228) そのアプリケーションの中のゲーム数（統計イベントで使われる。）
- const byte MasterPeerCount = 227
  (227) マスターサーバー上のプレイヤー数（そのゲームアプリケーションで、ゲームRoomを探すためマスターサーバーに接続しているプレイヤー。統計イベントで使われる。）
- const byte UserId = 225
  (225) ユーザーのID
- const byte ApplicationId = 224

(224) 開発者のアプリケーションID：自前のPhotonサーバー上の識別用の名前、もしくは Photon Cloud 上でのGUID

- const byte Position = 223
  (223) （現在、Position として）使用されていない。接続する前にキューにいれられた場合、これが「位置（Position）」となる。
- const byte MatchMakingType = 223
  (223) OpJoinRandom で利用されるマッチメイキング・アルゴリズムを変更する。選択できるパラメーター値は、 enum MatchmakingMode で定義される。
- const byte GameList = 222
  (222) オープンしている／リスティングされている全RoomのRoomInfoのリスト
- const byte Secret = 221
  (221) 暗号化を規定するために内部的に使用される
- const byte AppVersion = 220
  (220) ゲームアプリケーションのバージョン
- const byte RoomName = (byte)LiteOpKey.GameId
  (255) gameId/roomName (各Room毎に固有な名前)のためのコード。OpJoin とその類するメソッドで利用される。
- const byte Broadcast = (byte)LiteOpKey.Broadcast
  (250) OpSetProperties メソッドのブロードキャストパラメーターのためのコード
- const byte ActorList = (byte)LiteOpKey.ActorList
  (252) Room内のプレイヤーリストのためのコード。現在、使われていない。
- const byte ActorNr = (byte)LiteOpKey.ActorNr
  (254)オペレーションの Actor のコード。プロパティの get と set で利用される。
- const byte PlayerProperties = (byte)LiteOpKey.ActorProperties
  (249) プロパティ（ハッシュテーブル）set のためのコード
- const byte CustomEventContent = (byte)LiteOpKey.Data
  (245) イベントのデータ／個別内容のコード。OpRaiseEvent で利用される。
- const byte Data = (byte)LiteOpKey.Data
  (245) イベントのデータのコード。OpRaiseEvent で利用される。
- const byte Code = (byte)LiteOpKey.Code
  (244) OpRaiseEvent のイベントコードのようなコード関連のパラメーター送信の際に使用される。
- const byte GameProperties = (byte)LiteOpKey.GameProperties
  (248) プロパティ（ハッシュテーブル）set のためのコード
- const byte Properties = (byte)LiteOpKey.Properties
  (251) プロパティ（ハッシュテーブル）set のためのコード。このキーは、 1 つのプロパティセットだけを送信するために利用されます。ActorProperties か GameProperties （もしくは両方）が使われた場合、このキーを確認してください。
- const byte TargetActorNr = (byte)LiteOpKey.TargetActorNr
  (253) オペレーションの target Actor のコード。プロパティの set で利用される。”Is 0” はゲームを示す。
- const byte ReceiverGroup = (byte)LiteOpKey.ReceiverGroup
  (246) イベントの受信者を選択するコード（Liteでの Operation RaiseEvent で利用される）。
- const byte Cache = (byte)LiteOpKey.Cache
  (247) イベントを発生させながら、そのイベントをキャッシュするためのコード。
- const byte CleanupCacheOnLeave = (byte)241
  (241) CreateGame Operation のブール型パラメーター。 true の場合、退出したプレイヤーの roomcache をサーバーがクリーンアップします。（キャッシュされていたそのユーザーのイベントは削除される。）
- const byte Group = LiteOpKey.Group
  (240) グループ（Group）オペレーションのパラメーター（Op RaiseEvent で使われる）
- const byte Remove = LiteOpKey.Remove
  (239) 削除（Remove）オペレーションのパラメーター。リストから何かを削除するのに利用できる。例：プレイヤーのインタレストグループからグループを削除する。
- const byte Add = LiteOpKey.Add
  (238) 追加（Add）オペレーションのパラメーター。リストやセットに何かを追加するのに利用できる。例：プレイヤーのインタレストグループにグループを追加する。
- const byte ClientAuthenticationType = 217

(217) この(byte型)キーの値は、クライアントが接続する対象のカスタム認証のタイプやサービスを定義する。OpAuthenticate で使用される。

- const byte ClientAuthenticationParams = 216
  (216) この（文字列型）キーの値は、クライアントが接続する対象のカスタム認証のタイプやサービスに送るべきパラメーターを提供する。 OpAuthenticate で使用される。
- const byte FindFriendsRequestList = (byte)1
  (1) Op FindFriends リクエストで使われる。値は、検索する friends の文字列配列（string[]）でなければならない。
- const byte FindFriendsResponseOnlineList = (byte)1
  (1) Op FindFriends レスポンスで使われる。オンライン状態の ブール配列（bool[]） リストを含む(オンラインで無い場合は、false となる）
- const byte FindFriendsResponseRoomIdList = (byte)2
  (2) Op FindFriends レスポンスで使われる。Room名の 文字列配列（string[]） を含む(名前が不明のRoomやRoomに参加していない場合は、“”(空)となる。）

13.10.1 解説

定数のクラス。オペレーションとイベントのパラメーターのコードです。PUNが内部的にこれらの定数を使用します。

13.10.2 メンバーデータ

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

(252) Room 内のプレイヤーリストのためのコード。現在、使われていません。

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

(254)オペレーションの Actor のコード。プロパティの get と set で利用されます。

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

(238) 追加（Add）オペレーションのパラメーター。リストやセットに何かを追加するのに利用できます。例：プレイヤーのインタレストグループにグループを追加する。

13.10.2.4 Address **const byte ParameterCode.Address = 230**

(230) 利用する（ゲーム）サーバーのアドレス

13.10.2.5 ApplicationId **const byte ParameterCode.ApplicationId = 224**

(224) 開発者のアプリケーション ID：自前の Photon サーバー上の識別用の名前、もしくは Photon Cloud 上での GUID

13.10.2.6 AppVersion **const byte ParameterCode.AppVersion = 220**

(220) ゲームアプリケーションのバージョン

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

(250) OpSetProperties メソッドのブロードキャストパラメーターのためのコード

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

(247) イベントを発生させながら、そのイベントをキャッシュするためのコード。

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

(241) CreateGame Operation のブール型パラメーター。true の場合、退出したプレイヤーの roomcache をサーバーがクリーンアップします。（キャッシュされていたそのユーザーのイベントは削除される。）

13.10.2.10 ClientAuthenticationParams **const byte ParameterCode.ClientAuthenticationParams = 216**

(216) この（文字列型）キーの値は、クライアントが接続する対象のカスタム認証のタイプやサービスに送るべきパラメーターを提供する。 OpAuthenticate で使用されます。

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

(217) この（byte 型）キーの値は、クライアントが接続する対象のカスタム認証のタイプやサービスを定義する。 OpAuthenticate で使用されます。

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

(244) OpRaiseEvent のイベントコードのようなコード関連のパラメーター送信の際に使用されます。
これはオペレーションのコードとは同じではありません。Photon 3 では、オペレーションのコードは Dictionary パラメーターの一部として送信されることはもうありません。

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

(245) イベントのデータ／個別内容のコード。OpRaiseEvent で利用される。

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

(245) イベントのデータのコード。OpRaiseEvent で利用される。

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

(1) Op FindFriends リクエストで使われる。値は、検索する friends の 文字列配列（string[]） でなければならない。

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

(1) Op FindFriends レスポンスで使われる。オンライン状態の ブール配列（bool[]） リストを含む(オンラインで無い場合は、false となる）

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

(2) Op FindFriends レスポンスで使われる。Room 名の 文字列配列（string[]） を含む(名前が不明の Room や Room に参加していない場合は、”” (空) となる。）

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

(228) そのアプリケーションの中のゲーム数（統計イベントで使われる。）

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

(222) オープンしている／リスティングされている全 Room の RoomInfo のリスト

13.10.2.20 GameProperties **const byte ParameterCode.GameProperties = (byte)LiteOpKey.GameProperties**
(248) プロパティ（ハッシュテーブル）set のためのコード

13.10.2.21 Group **const byte ParameterCode.Group = LiteOpKey.Group**
(240) グループ（Group）オペレーションのパラメーター（Op RaiseEvent で使われる）

13.10.2.22 MasterPeerCount **const byte ParameterCode.MasterPeerCount = 227**
(227) マスターサーバー上のプレイヤー数（そのゲームアプリケーションで、ゲーム Room を探すためマスターサーバーに接続しているプレイヤー。統計イベントで使われる。）

13.10.2.23 MatchMakingType **const byte ParameterCode.MatchMakingType = 223**
(223) OpJoinRandom で利用されるマッチメイキング・アルゴリズムを変更する。選択できるパラメーター値は、 enum MatchmakingMode で定義される。

13.10.2.24 PeerCount **const byte ParameterCode.PeerCount = 229**
(229) ある Room 内のプレイヤー数（そのゲームアプリケーションで、ゲームサーバーに接続しているプレイヤー。統計イベントで使われる。）

13.10.2.25 PlayerProperties **const byte ParameterCode.PlayerProperties = (byte)LiteOpKey.ActorProperties**
(249) プロパティ（ハッシュテーブル）set のためのコード

13.10.2.26 Position **const byte ParameterCode.Position = 223**
(223) (現在、Position として)使用されていない。接続する前にキューにいれられた場合、これが「位置（Position）」となる。

13.10.2.27 Properties **const byte ParameterCode.Properties = (byte)LiteOpKey.Properties**
(251) プロパティ（ハッシュテーブル）set のためのコード。このキーは、１つのプロパティセットだけを送信するために利用されます。ActorProperties か GameProperties （もしくは両方）が使われた場合、このキーを確認してください。

13.10.2.28 ReceiverGroup **const byte ParameterCode.ReceiverGroup = (byte)LiteOpKey.ReceiverGroup**
(246) イベントの受信者を選択するコード（Lite での Operation RaiseEvent で利用される）。

13.10.2.29 Remove **const byte ParameterCode.Remove = LiteOpKey.Remove**
(239) 削除（Remove）オペレーションのパラメーター。リストから何かを削除するのに利用できる。例：プレイヤーのインタレストグループからグループを削除する。

13.10.2.30 RoomName **const byte ParameterCode.RoomName = (byte)LiteOpKey.GameId**
(255) gameId/roomName (各 Room 毎に固有な名前)のためのコード。OpJoin とその類するメソッドで利用される。

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

(221) 暗号化を規定するために内部的に使用される

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

(253) オペレーションの target Actor のコード。プロパティの set で利用される。”Is 0” はゲームを示す。

13.10.2.33 UserId **const byte ParameterCode.UserId = 225**

(225) ユーザーの ID

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs

13.11 PBitStream クラス レファレンス

パブリックメンバー関数

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

静的パブリックメンバー関数

- static int BytesForBits (int bitCount)

プロパティ

- int ByteCount `[get]`
- int BitCount `[get, set]`
- int Position `[get, set]`

13.11.1 コンストラクターとデストラクター

13.11.1.1 PBitStream **PBitStream.PBitStream ( )**

13.11.1.2 PBitStream **PBitStream.PBitStream (int bitCount)**

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

13.11.2 メンバー関数

13.11.2.1 Add **void PBitStream.Add (bool val)**

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

13.11.2.3 Get **bool PBitStream.Get (int bitIndex)**

13.11.2.4 GetNext **bool PBitStream.GetNext ( )**

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

13.11.2.6 ToBytes **byte [ ] PBitStream.ToBytes ( )**

13.11.3 プロパティ

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

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

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

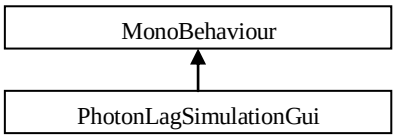
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs

13.12 PhotonLagSimulationGui クラス レファレンス

この MonoBehaviour のサブクラスは、Photon クライアントでのネットワークシミュレーション機能のための基本的な GUI です。ラグ（lag、固定値の遅延)、ジッター（jitter、ランダム値の遅延) そしてパケットの損失などを変更できます。

継承概念図： PhotonLagSimulationGui:



パブリックメンバー関数

- void Start ()
- void OnGUI ()

パブリック属性

- Rect WindowRect = new Rect(0, 100, 120, 100)
  ウインドウのために長方形を位置づけます。
- int WindowId = 101
  Unity GUI Window ID （必ず固有、そうでないと問題を起こします。）
- bool Visible = true
  GUI を表示するかしないかの値です。（設定値などには影響しません。）

プロパティ

- PhotonPeer Peer [get, set]
  (ネットワークシミュレーションを行う対象として) 現在、使われているPeerを取得、設定します。

13.12.1 解説

この MonoBehaviour のサブクラスは、Photon クライアントでのネットワークシミュレーション機能のための基本的な GUI です。ラグ（lag、固定値の遅延)、ジッター（jitter、ランダム値の遅延) そしてパケットの損失などを変更できます。

13.12.2 メンバー関数

13.12.2.1 OnGUI　**void PhotonLagSimulationGui.OnGUI ( )**

13.12.2.2 Start　**void PhotonLagSimulationGui.Start ( )**

13.12.3 メンバーデータ

13.12.3.1 Visible　**bool PhotonLagSimulationGui.Visible = true**

GUI を表示するかしないかの値です。（設定値などには影響しません。）

13.12.3.2 WindowId　**int PhotonLagSimulationGui.WindowId = 101**

Unity GUI Window ID （必ず固有、そうでないと問題を起こします。）

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

ウインドウのために長方形を位置づけます。

13.12.4 プロパティ

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

(ネットワークシミュレーションを行う対象として) 現在、使われている Peer を取得、設定します。

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs

13.13　PhotonMessageInfo クラス レファレンス

特定のメッセージ、RPC／更新情報に関する情報のコンテナのクラス。

パブリックメンバー関数

- PhotonMessageInfo ()
  PhotonMessageInfo クラスの新しいインスタンスを初期化します。空の messageinfo インスタンスを作るだけです。
- PhotonMessageInfo (PhotonPlayer player, int timestamp, PhotonView view)
- override string ToString ()

パブリック属性

- PhotonPlayer sender
- PhotonView photonView

プロパティ

- double timestamp [get]

13.13.1 解説

特定のメッセージ、RPC／更新情報に関する情報のコンテナのクラス。

13.13.2 コンストラクターとデストラクター

13.13.2.1 PhotonMessageInfo **PhotonMessageInfo.PhotonMessageInfo ( )**

PhotonMessageInfo クラスの新しいインスタンスを初期化します。空の messageinfo インスタンスを作るだけです。

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

13.13.3 メンバー関数

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

13.13.4 メンバーデータ

13.13.4.1 photonView**PhotonView PhotonMessageInfo.photonView**

13.13.4.2 photonView **PhotonPlayer PhotonMessageInfo.photonView**

13.13.5 プロパティ

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

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs

13.14 PhotonNetwork クラス レファレンス

PhotonNetwork プラグインを使用するためのメインのクラス。これは静的クラスです。

静的パブリックメンバー関数

- static bool SetMasterClient (PhotonPlayer player)
  現在のマスタークライアントに対し、誰か別なユーザーをマスタークライアントとして指名することを許可します。カスタム設定した選択では、どのクライアント上であっても同じユーザーをマスタークライアントとして選択すべきです。
- static void NetworkStatisticsReset ()
  ネットワークトラフィックの統計値をリセットし、計測を再び有効とします。
- static string NetworkStatisticsToString ()
  NetworkStatisticsEnabled が統計値を集計するの使用されている場合のみ、利用可能です。
- static void InternalCleanPhotonMonoFromSceneIfStuck ()
  内部的に Editorスクリプトで利用されます。（シーンの保存も含めて）ヒエラルキーに変更があった時にコールされ、隠れて残っている PhotonHandlers を取り除きます。
- static void ConnectUsingSettings ()

- static void Connect (string serverAddress, int port, string uniqueGameID)
- static void ConnectUsingSettings (string gameVersion)

  設定された Photon Server に接続します：PhotonNetwork.serverSettingsAssetPath を読み取り、クラウドサービスか自前のサーバーに接続します。
- static void ConnectToBestCloudServer (string gameVersion)

  ping 計測値が最小のPhoton Cloud Server に接続します。PlayerPrefs の中にあるすべての PhotonCloud Server へのpingの結果を保存します。初めてこれをコールすると２秒余計にかかります。pingの結果は、PhotonNetwork.OverrideBestCloudServer(..) で、書き換えることができます。
- static void OverrideBestCloudServer (CloudServerRegion region)

  ConnectToBestCloudServer(string gameVersion) で利用される地域を書き換えます。そして、すべてのPhotonCloud Serverへpingして得た結果を書き換えます。
- static void RefreshCloudServerRating ()

  すべてのPhotonCloud Serverに再びpingし、（その時点での）最良のping結果のものを見つけます。
- static void Connect (string serverAddress, int port, string appID, string gameVersion)

  指定したアドレス、ポート、アプリケーションID、ゲーム（クライアント）バージョンで、Photon Server に接続します。
- static void Disconnect ()

  このクライアントをPhoton Server から切断します。Roomから退出する処理、完了時に OnDisconnectedFromPhoton をコールします。
- static void InitializeSecurity ()

  Unity Networking との互換性のためだけに使用されます。接続処理中に暗号化は自動的に初期化されます。
- static bool FindFriends (string[ ] friendsToFind)

  友達リストのRoom とオンラインの状態をリクエストします。すべてのクライアントは、PlayerNameプロパティにより固有のユーザーネーム（username）を設定している必要があります。結果は、this.Friendsで利用できます。
- static void CreateRoom (string roomName)

  指定の名前でRoomを作成します。しかし既にその名前のRoomが存在する場合は失敗します。
- static void CreateRoom (string roomName, bool isVisible, bool isOpen, int maxPlayers)

  指定の名前でRoomを作成します。しかし既にその名前のRoomが存在する場合は失敗します。
- static void CreateRoom (string roomName, bool isVisible, bool isOpen, int maxPlayers, Hashtable custom-
- RoomProperties, string[ ] propsToListInLobby)

  指定の名前でRoomを作成します。しかし既にその名前のRoomが存在する場合は失敗します。
- static void JoinRoom (RoomInfo listedRoom)

  room.Name によりRoomに参加します。Roomが定員一杯か（その瞬間に閉じられたかして）利用可能でない場合は失敗します。
- static void JoinRoom (string roomName)

  任意のタイトルによりRoomに参加します。Roomが定員一杯か（その瞬間に閉じられたかして）利用可能でない場合は失敗します。
- static void JoinRandomRoom ()

  利用可能な、どのRoomでも参加します。しかし、その時点で利用可能なものがまったく無い場合、失敗します。
- static void JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)

  カスタムプロパティを適合させて、オープンなRoomに参加を試みます。しかし、その時点で利用可能なものがまったく無い場合、失敗します。
- static void JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers,
- MatchmakingMode matchingType)

  カスタムプロパティを適合させて、オープンなRoomに参加を試みます。しかし、その時点で利用可能なものがまったく無い場合、失敗します。
- static void LeaveRoom ()

  現在のRoomを退出する。
- static RoomInfo[ ] GetRoomList ()

  RoomInfo として（その時点で）既知のRoomの配列を取得します。クライアントが、（マスターサーバー上の）ロビーにいる間、このリストは数秒毎に自動更新されます。いずれかのRoomの中にいる間は利用できません。
- static void SetPlayerCustomProperties (Hashtable customProperties)

  この（ローカル）プレイヤーのプロパティを設定します。PhotonNetwork.player.customProperties にプロパティがキャッシュされます。 CreateRoom、JoinRoom、JoinRandomRoom のすべてが、Roomに入る時、プレイヤーのカスタムプロパティを利用

します。Roomにいる間は、プレイヤーのプロパティは他のプレイヤーと共有／同期されます。ハッシュテーブルが null である場合、カスタムプロパティはクリアされます。カスタムプロパティは決して自動的にはクリアされませんので、変更しない限り、次のRoomにも引き継がれていきます。

- static int AllocateViewID ()

その時点でローカルプレイヤーに有効な viewID の１つを割り当てます。

- static void UnAllocateViewID (int viewID)
  （マニュアルにインスタンス化され、その後抹消されたネットワーク上のオブジェクトの）viewID を抹消します。
- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group)
  ネットワーク上でプレハブをインスタンス化します。このプレハブは、Resourcesフォルダのルートになければなりません。
- static GameObject Instantiate (string prefabName, Vector3 position, Quaternion rotation, int group, object[ ]
- data)
  ネットワーク上でプレハブをインスタンス化します。このプレハブは、Resourcesフォルダのルートになければなりません。
- static GameObject InstantiateSceneObject (string prefabName, Vector3 position, Quaternion rotation, int
- group, object[ ] data)
  ネットワーク上でシーンが所有するプレハブをインスタンス化します。Photon View は MasterClient によりコントロールできるようになります。このプレハブは、Resourcesフォルダのルートになければなりません。
- static int GetPing ()
  その時点での Photon Serverへのラウンドトリップタイム（RTT）
- static void FetchServerTimestamp ()
  サーバーのタイムスタンプを更新する。（非同期オペレーション、１ラウンドトリップかかる）
- static void SendOutgoingCommands ()
  作成されたばかりのRPCやインスタンス化コールを即時に送信するために利用でき、それらは他のプレイヤーに送信中となります。
- static void CloseConnection (PhotonPlayer kickPlayer)
  クライアントに切断するようにリクエストします（これを「キック」といいます）。マスタークライアントだけがこれを行えます。
- static void Destroy (PhotonView targetView)
  対象Photon Viewが、静的な場合以外、またこのクライアントのコントロール下でない場合以外は、PhotonView と関連づけられている GameObject をネットワーク上で削除します。
- static void Destroy (GameObject targetGo)
  GameObject が静的か、または、このクライアントのコントロール下でない場合以外は、そのGameObject をネットワーク上で削除します。
- static void DestroyPlayerObjects (PhotonPlayer targetPlayer)
  targetPlayer のすべての GameObjectsとPhotonViewとそれらのRPCをネットワーク上で削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に（だれに対しても）、コールすることができます。
- static void DestroyPlayerObjects (int targetPlayerId)
  （IDで指定し）対象プレイヤーのすべての GameObjectsとPhotonViewとそれらのRPCをネットワーク上で削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に（だれに対しても）、コールすることができます。
- static void DestroyAll ()
  Room内のすべての GameObjectsとPhotonViewとそれらのRPCをネットワーク上で削除します。バッファーリングされているものをすべてサーバーから削除します。マスタークライアントだけが、（だれに対しても）コールすることができます。
- static void RemoveRPCs (PhotonPlayer targetPlayer)
  バッファーリングしているRPCで targetPlayer から送られたものを、サーバーからすべて削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に（だれに対しても）、コールすることができます。
- static void RemoveRPCs (PhotonView targetPhotonView)
  バッファーリングしているRPCで targetPhotonView によって送られたものを、サーバーからすべて削除します。マスタークライアントまたは targetPhotonView のオーナーがコールすることができます。
- static void RemoveRPCsInGroup (int targetGroup)
  このクライアントが、マスタークライアントか、または個別の PhotonView をコントロールしている場合、バッファーリングしているRPCで、targetGroup内で送られたものを、すべて削除します。
- static void SetReceivingEnabled (int group, bool enabled)
  指定したグループでの受信を有効／無効にします。（PhotonView に適用される）
- static void SetSendingEnabled (int group, bool enabled)
  指定したグループでの送信を有効／無効にします。（PhotonView に適用される）
- static void SetLevelPrefix (short prefix)
  後からインスタンス化される PhotonView の前に付けるレベルを設定します。必要なレベルが１つだけの場合は、設定しな

いでください。

- static void LoadLevel (int levelNumber)
  レベルを読み込み、自動的にネットワークキューをポーズします。OnJoinedRoom でこれをコールして、キャッシュされた RPC が間違ったシーンで処理されることがないように確認してください。
- static void LoadLevel (string levelTitle)
  レベルを読み込み、自動的にネットワークキューをポーズします。OnJoinedRoom でこれをコールして、キャッシュされた RPC が間違ったシーンで処理されることがないように確認してください。

パブリック属性

- const string versionPUN = "1.21"
  PUN のバージョン番号。クライアントのバージョンを区別するために、ゲームバージョンでも利用されます。
- const string serverSettingsAssetFile = "PhotonServerSettings"
  PhotonServerSettings ファイルの名前（ロードするため、また新しいファイルを保存するため PhotonEditor により使用される）
- const string serverSettingsAssetPath = "Assets/Photon Unity Networking/Resources/" + PhotonNetwork.-serverSettingsAssetFile + ".asset"
  PhotonServerSettings ファイルへのパス（PhotonEditorにより使用される）

静的パブリック属性

- static readonly int MAX_VIEW_IDS = 1000
  １プレイヤー（または１シーン）当たりに割り当てられた PhotonView の最大上限数。この上限数を増やす方法については説明している部分を参照してください。
- static ServerSettings PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.server-
- SettingsAssetFile, typeof(ServerSettings))
  シリアライズされたサーバー設定。 ConnectUsingSettingsで利用されるために、セットアップウィザードで書き込まれる。
- static float precisionForVectorSynchronization = 0.000099f
  PhotonViewの OnSerialize/ObservingComponent で、更新の送信前に必要となるVector2 や Vector3 で必要となる最小差異（例として、トランスフォームのローテーション値）。注意：これはsqrMagnitude（平方マグニチュード）です。例：Y軸上の 0.01 の変更の後に送る場合は、0.01f*0.01f=0.0001f を使います。ただし、浮動小数の不正確性の改善措置として、0.0001f の代わりに 0.000099f を使用します。
- static float precisionForQuaternionSynchronization = 1.0f
  PhotonViewの OnSerialize/ObservingComponent を介して送られる前に、rotation値が変化していなければならない最小の角度を表します。
- static float precisionForFloatSynchronization = 0.01f
  PhotonViewの OnSerialize/ObservingComponent で、更新の送信前に必要となる最小差異、浮動小数の差分。
- static PhotonLogLevel logLevel = PhotonLogLevel.ErrorsOnly
  ネットワークのログ出力レベル。どの程度PUNが詳細な出力するかをコントロールします。
- static bool UsePrefabCache = true
  有効（true）の間、インスタンス化は PhotonNetwork.PrefabCache を使用し、ゲームオブジェクトをメモリーに保持します。（同じプレハブのインスタンス化の効率を向上させます。）
- static Dictionary< string,
- GameObject > PrefabCache = new Dictionary<string, GameObject>()
  頻繁なインスタンス化のためにGameObjects への参照を保持します。（Resources をロードする代わりに、メモリーの中からインスタンス化する。）

プロパティ

- static string ServerAddress [get]
  使用中のサーバーアドレス（マスターサーバーとゲームサーバーのどちらでも）
- static bool connected [get]
  Photon Serverに接続しているかどうか。（Roomの中か外の場合があります。）
- static ConnectionState connectionState [get]
  簡易な接続状態
- static PeerState connectionStateDetailed [get]
  詳細な接続状態（PUNを意識していないので、サーバーを切り替えている時、「切断 "disconnected"」となることがある。）
- static AuthenticationValues AuthValues [get, set]
  Photon （そしてカスタムサービス／コミュニティ）へのカスタム認証を利用した接続のために使われるユーザーの認証値。カスタム認証を利用したい場合は、Connect をコールする前にこれらを設定してください。
- static Room room [get]
  現在いるRoomを取得。もしRoomにいない場合、null。

- static PhotonPlayer player [get]
  ローカルの PhotonPlayer 。常に利用可能で、this player を表す。カスタムプロパティはRoomに入る前に設定でき、また共有／同期されます。
- static PhotonPlayer masterClient [get]
  マスタークライアントである PhotonPlayer。マスタークライアントは、Roomの「ヴァーチャルなオーナー」。オーソラティブな（ゲームロジックを支配するような）決定がプレイヤーの１人によって必要な場合は、これを利用できます。
- static string playerName [get, set]
  ローカルプレイヤーの名前
- static PhotonPlayer[ ] playerList [get]
  ローカルプレイヤーも含めた、すべての PhotonPlayer のリスト
- static PhotonPlayer[ ] otherPlayers [get]
  自身のローカルプレイヤーを含まない、他のすべての PhotonPlayers
- static List< FriendInfo > Friends [get, set]
  オンラインの状態と参加しているRoomの情報を含む、リードオンリーの友達（friends）リスト。FindFriends コールにより初期化されるまでは、null です。
- static int FriendsListAge [get]
  友達リストの経過時間（ミリセカンド単位）。友達リストが取得されるまでは、0 です。
- static bool offlineMode [get, set]
  マルチプレイヤーのコードをシングルプレイヤーゲームでも再利用できるように、オフラインモードを設定することができます。これが有効にされると、PhotonNetwork は接続を一切することなく、ほとんどオーバーヘッドが無くなります。特に RPC と PhotonNetwork.Instantiate を再利用するのにとても役立ちます。
- static int maxConnections [get, set]
  Roomの最大プレイヤーの数。より良いやり方として、CreateRoom でこれを設定するようにします。オープンなRoomが無い時は、これは0を返します。
- static bool automaticallySyncScene [get, set]
  true の場合、PUN は常に全ユーザーが同じシーンにいるようにします。もしマスタークライアントが変更されたら、すべてのクライアントは新しいシーンをロードします。また、これによりメインメニューからゲームに参加した後でも、ゲームシーンのロードがスムーズに処理されます。
- static bool autoCleanUpPlayerObjects [get, set]
  この設定により、退出したプレイヤーの生成した GameObjects と PhotonView をRoomにいるプレイヤーが取り除かなければならないかが定義されます。
- static bool autoJoinLobby [get, set]
  マスターサーバーに接続した時、PhotonNetwork がロビーに参加するかどうかを定義します。もしこれが false の場合、マスターサーバーへの接続が利用可能であれば、OnConnectedToMaster() がコールされます。これが false の場合、OnJoinedLobby() はコールされません。
- static bool insideLobby [get]
  Photon に接続し、ロビーに入った状態の時、true を返します。
- static int sendRate [get, set]
  PhotonNetwork が毎秒あたり何回更新などの情報パッケージを送信するかを定義します。これを変更した場合は、sendRateOnSerialize も変更するのを忘れないようにしてください。
- static int sendRateOnSerialize [get, set]
  PhotonView で、毎秒あたり何回 OnPhotonSerialize がコールされるかを定義します。.
- static bool isMessageQueueRunning [get, set]
  （RPC, インスタンス化など、 その他すべての）着信するイベントの処理をポーズするのに使用されます。まず始めにレベルだけをロードし、その後、続いて PhotonView と RPC のデータを受け取るような時に、これは役に立ちます。クライアントは受信を続け、着信してくる情報パッケージやRPC／イベントへの受信確認（ACK）を送り返します。これはラグを発生させ、すべての着信メッセージがキューされるため、ポーズがより長くなると問題を起こすこともあり得ます。
- static int unreliableCommandsLimit [get, set]
  チャネル毎にアンリライアブルコマンド（unreliable commands）を制限するため、コマンド発行毎に１度使用されます。（ですから、ポーズの後は、多くのチャネルがたくさんのアンリライアブルコマンドを引き起こすことがあり得ます。）
- static double time [get]
  Photon Network タイム、 サーバーと同期されている

- static bool isMasterClient [get]
  自分たちがマスタークライアントかどうか？
- static bool isNonMasterClientInRoom [get]
  Roomの中にいて、かつ Roomのマスタークライアントで無い場合、true。

- static int countOfPlayersOnMaster [get]
  現在Roomを探しているプレイヤーの数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。
- static int countOfPlayersInRooms [get]
  現在、Roomの中にいるプレイヤーの数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。
- static int countOfPlayers [get]
  現在、このゲームアプリケーションを利用しているプレイヤー数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。
- static int countOfRooms [get]
  現在使用中のRoomの数。ロビー内にいる時は、これは PhotonNetwork.GetRoomList().Length に基づいています。ロビーにいない時は、これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。
- static bool NetworkStatisticsEnabled [get, set]
  このクライアントのトラフィックの統計値の集計を有効または無効にする。クライアントで何か問題に直面した場合、解決策のために、まずはトラフィックの統計から調べるといいでしょう。
- static int ResentReliableCommands [get]
  （ローカルでACK受信前のリピートのタイミングにより）リピートされたコマンドの数。

13.14.1 解説

PhotonNetwork プラグインを使用するためのメインのクラス。これは静的クラスです。

13.14.2 メンバー関数

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

その時点でローカルプレイヤーに有効な viewID の１つを割り当てます。

戻り値

新しい PhotonView に使用できる１つの viewID。

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

クライアントに切断するようにリクエストします（これを「キック」といいます）。マスタークライアントだけがこれを行えます。

パラメーター

kickPlayer	キックする PhotonPlayer

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

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

指定したアドレス、ポート、アプリケーション ID、ゲーム（クライアント）バージョンで、Photon Server に接続します。このメソッドは、ConnectUsingSettings と ConnectToBestCloudServer により使用されます。

Photon Cloud に接続するためには、設定ファイルの中に有効なアプリケーション ID がなければいけません。（アプリケーション ID は、Photon Cloud ダッシュボード で表示されています）https://cloud-jp.exitgames.com/dashboard

Photon Cloud への接続は次の原因により失敗することがあります：

- ネットワークの問題（OnFailedToConnetToPhoton() をコールする）
- サブスクリプションの同時接続数の制限のため。（DisconnectCause.MaxCcuReachedと一緒にOnConnectionFail() がコールされます。また、OnPhotonMaxCccuReached() もコールされます）

詳細な接続の制限については以下リンクを参照してください：http://doc.exitgames.com/photon-cloud/

パラメーター

serverAddress	サーバーのアドレス（自前のサーバー、もしくは Photon Cloud のアドレス）
port	接続するサーバーのポート
appID	アプリケーション ID （Photon Cloud はゲームアプリケーション用の GUID とともに提供します。）
gameVersion	クライアントのバージョン番号。ユーザーはゲームバージョンにより別々に管理されます。（これによりゲームアプリケーションに急な変更を盛り込むことができます。）

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

ping 計測値が最小の Photon Cloud Server に接続します。PlayerPrefs の中にあるすべての PhotonCloud Server への ping の結果を保存します。初めてこれをコールすると２秒余計にかかります。ping の結果は、PhotonNetwork.OverrideBestCloudServer(..) で、書き換えることができます。

初めてこれを使用する場合は、このコールは２秒ほどかかることがあります。最適な地域サーバー確認のため、すべての PhotonCloud Server が ping されます。

PUN の設定ウィザードが、アプリケーション ID を設定ファイルに保存し、任意のサーバーアドレスとポートを適用します。これらは Connect(string serverAddress, int port, string appID, string gameVersion) で利用されます。

Photon Cloud に接続するためには、設定ファイルの中に有効なアプリケーション ID がなければいけません。（アプリケーション ID は、Photon Cloud ダッシュボード で表示されます）

https://cloud-jp.exitgames.com/dashboard

Photon Cloud への接続は次の原因により失敗することがあります：

- ネットワークの問題（OnFailedToConnetToPhoton() をコールします）
- 無効な地域（OnConnectionFail() が、DisconnectCause.InvalidRegionと一緒にコールされます）
- サブスクリプションの同時接続数の制限のため。（DisconnectCause.MaxCcuReachedと一緒にOnConnectionFail() がコールされます。また、OnPhotonMaxCccuReached() もコールされます）

詳細な接続の制限については以下リンクを参照のこと：http://doc-jp.exitgames.com/photon-cloud/

パラメーター

gameVersion	クライアントのバージョン番号。ユーザーはお互いにゲームバージョンにより別々に管理されます。（これにより、ゲームアプリケーションに急な変更を盛り込むことができます。）

13.14.2.6 ConnectUsingSettings **static void PhotonNetwork.ConnectUsingSettings ( ) [static]**

13.14.2.7 ConnectUsingSettings **static void PhotonNetwork.ConnectUsingSettings (string gameVersion) [static]**

設定された Photon Server に接続します : PhotonNetwork.serverSettingsAssetPath を読み取り、クラウドサービスか自前のサーバーに接続します。

PUN の設定ウィザードが、アプリケーション ID を設定ファイルに保存し、任意のサーバーアドレスとポートを適用します。これらは Connect(string serverAddress, int port, string appID, string gameVersion) で利用されます。

Photon Cloud に接続するためには、設定ファイルの中に有効なアプリケーション ID がなければいけません。（アプリケー

ション ID は、Photon Cloud ダッシュボード で表示されます）
https://cloud-jp.exitgames.com/dashboard

Photon Cloud への接続は次の原因により失敗することがあります：

- ネットワークの問題（OnFailedToConnetToPhoton() をコールします）
- 無効な地域（OnConnectionFail() が、DisconnectCause.InvalidRegionと一緒にコールされます）
- サブスクリプションの同時接続数の制限のため。（DisconnectCause.MaxCcuReachedと一緒にOnConnectionFail() がコールされます。また、OnPhotonMaxCccuReached() もコールされます）

詳細な接続の制限については以下リンクを参照してください：http://doc.exitgames.com/photon-cloud/

パラメーター

gameVersion	クライアントのバージョン番号。ユーザーはお互いにゲームバージョンにより別々に管理されます。（これによりゲームアプリケーションに急な変更を盛り込むことができます。）

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

指定の名前で Room を作成します。しかし既にその名前の Room が存在する場合は失敗します。

自分で Room 名をつけたくない場合は、名前として null か “”(空) を渡すとサーバーが 1 つの roomName（1 つの文字列 GUID）を割り当てます。マスターサーバー上でのみこれをコールしてください。内部的には、マスターサーバーが 1 つのサーバーアドレス（と、もし必要であれば roomName も）を返します。それは内部的に利用され、割り当てられたゲームサーバー、そして roomName の Room に切り替えられます。

PhotonNetwork.autoCleanUpPlayerObjects がこの Room の AutoCleanUp プロパティとなり、この Room に参加するすべてのクライアントにより使用されます。

パラメーター

roomName	作成する Room の固有な名前

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

指定の名前で Room を作成します。しかし既にその名前の Room が存在する場合は失敗します。

自分で Room 名をつけたくない場合は、名前として null か “”(空) を渡すとサーバーが 1 つの roomName（1 つの文字列 GUID）を割り当てます。マスターサーバー上でのみこれをコールしてください。内部的には、マスターサーバーが 1 つのサーバーアドレス（と、もし必要であれば roomName も）を返します。どちらも、割り当てられたゲームサーバー、そして roomName の Room に、切り替えるために内部的に利用されます。

パラメーター

roomName	作成する Room の固有な名前。サーバーに名前を生成させるには、null か “” （空） を渡す
isVisible	ロビーでの Room リスト上で、この Room を表示する（または、表示しない）
isOpen	この Room への他のプレイヤーの参加を許可する（または、許可しない）
maxPlayers	Room に参加できる最大プレイヤー数

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

指定の名前で Room を作成します。しかし既にその名前の Room が存在する場合は失敗します。

自分で Room 名をつけたくない場合は、名前として null か “”(空) を渡すとサーバーが 1 つの roomName（1 つの文字列 GUID）を割り当てます。マスターサーバー上でのみこれをコールしてください。内部的には、マスターサーバーが 1 つのサーバーアドレス（と、もし必要であれば roomName も）を返します。どちらも、割り当てられたゲームサーバー、そして

roomName の Room に、切り替えるために内部的に利用されます。

PhotonNetwork.autoCleanUpPlayerObjects がこの Room の AutoCleanUp プロパティとなり、この Room に参加するすべてのクライアントにより使用されます。

パラメーター

roomName	作成する Room の固有な名前。サーバーに名前を生成させるには、null か “”（空） を渡す
isVisible	ロビーでの Room リスト上で、この Room を表示する（または、表示しない）
isOpen	この Room への他のプレイヤーの参加を許可する（または、許可しない）
maxPlayers	Room に参加できる最大プレイヤー数
customRoom-Properties	新規 Room のカスタムプロパティ（作成時に設定するので、すぐに利用できる）
propsToListIn-Lobby	ロビーに転送されるカスタムプロパティ名の配列（有用なものだけを含めるようにします）

13.14.2.11 Destroy **static void PhotonNetwork.Destroy (PhotonView targetView) [static]**

対象 Photon View が、静的な場合以外、またこのクライアントのコントロール下でない場合以外は、PhotonView と関連づけられている GameObject をネットワーク上で削除します。

ネットワーク上の GameObject を削除するということは、次を含みます：

- サーバーのRoom用バッファーからインスタンス化コールを削除
- PhotonNetwork.Instantiate により間接的に生成された PhotonView のための、バッファーリングされたRPCの削除
- 他のクライアントに対象GameObjectを同じく削除するようにというメッセージの送信（ネットワークのラグに影響される）

ネットワークオブジェクトの削除は、そのオブジェクトが PhotonNetwork.Instantiate() により生成された場合のみ機能します。

シーンと一緒にロードされたオブジェクトは、たとえ PhotonView コンポーネントを持っていても、無視されます。

また GameObject は、このクライアントのコントロール下になければなりません：

- コントロール下とは、このクライアントによりインスタンス化され所有されていること。
- Roomを退出したプレイヤーによりインスタンス化されたオブジェクトはマスタークライアントによりコントロールされる。
- シーンが所有しているゲームオブジェクトは、マスタークライアントによりコントロールされる。

戻り値

無し。問題がある場合は、デバッグ用エラーログを確認してください。

13.14.2.12 Destroy **static void PhotonNetwork.Destroy (GameObject targetGo) [static]**

GameObject が、静的な場合以外、またこのクライアントのコントロール下でない場合以外は、その GameObject をネットワーク上で削除します。

ネットワーク上の GameObject を削除するということは、次を含みます：

- サーバーのRoom用バッファーからインスタンス化コールを削除
- PhotonNetwork.Instantiate により間接的に生成された PhotonView のための、バッファーリングされたRPCの削除
- 他のクライアントに対象GameObjectを同じく削除するようにというメッセージの送信（ネットワークのラグに影響される）

ネットワークオブジェクトの削除は、そのオブジェクトが PhotonNetwork.Instantiate() により生成された場合のみ機能します。

シーンと一緒にロードされたオブジェクトは、たとえ PhotonView コンポーネントを持っていても、無視されます。

また GameObject は、このクライアントのコントロール下になければなりません：

- コントロール下とは、このクライアントによりインスタンス化され所有されていること。
- Roomを退出したプレイヤーによりインスタンス化されたオブジェクトはマスタークライアントによりコントロールされる。
- シーンが所有しているゲームオブジェクトは、マスタークライアントによりコントロールされる。

戻り値

無し。問題がある場合は、デバッグ用エラーログを確認してください。

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

Room 内のすべての GameObjects と PhotonView とそれらの RPC をネットワーク上で削除します。バッファーリングされているものをすべてサーバーから削除します。マスタークライアントだけが、(だれに対しても) コールすることができます。マスタークライアントだけが、 (すべてのクライアントに対し) コールすることができます。Destroy メソッドとは違い、これはサーバーの Room バッファーからすべてのものを削除します。もしゲーム内でインスタンス化コールと RPC 以外のものをバッファーリングしている場合も、それらも同様にサーバーから削除されます。

すべて削除する (DestroyAll) とは次を含みます :

- サーバーのRoomバッファーからすべてのもの(Instantiate、 RPC、その他のバッファーリングされているもの)を削除する
- 他のクライアントに、各クライアントでもローカルですべてを同じように削除するようにというメッセージの送信 (ネットワークのラグに影響される)

ネットワークオブジェクトの削除は、そのオブジェクトが PhotonNetwork.Instantiate() により生成された場合のみ機能します。シーンと一緒にロードされたオブジェクトは、たとえ PhotonView コンポーネントを持っていても、無視されます。

戻り値

無し。問題がある場合は、デバッグ用エラーログを確認してください。

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

targetPlayer のすべての GameObjects と PhotonView とそれらの RPC をネットワーク上で削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に (だれに対しても) 、コールすることができます。

ネットワーク上の GameObject を削除するということは、次を含みます :

- サーバーのRoom用バッファーからインスタンス化コールを削除
- PhotonNetwork.Instantiate により間接的に生成された PhotonView のための、バッファーリングされたRPCの削除
- 他のクライアントに対象GameObjectを同じく削除するようにというメッセージの送信 (ネットワークのラグに影響される)

ネットワークオブジェクトの削除は、そのオブジェクトが PhotonNetwork.Instantiate() により生成された場合のみ機能します。シーンと一緒にロードされたオブジェクトは、たとえ PhotonView コンポーネントを持っていても、無視されます。

戻り値

無し。問題がある場合は、デバッグ用エラーログを確認してください。

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

(ID で指定し) 対象プレイヤーのすべての GameObjects と PhotonView とそれらの RPC をネットワーク上で削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に (だれに対しても) 、コールすることができます。

ネットワーク上の GameObject を削除するということは、次を含みます :

- サーバーのRoom用バッファーからインスタンス化コールを削除
- PhotonNetwork.Instantiate により間接的に生成された PhotonView のための、バッファーリングされたRPCの削除
- 他のクライアントに対象GameObjectを同じく削除するようにというメッセージの送信 (ネットワークのラグに影響される)

ネットワークオブジェクトの削除は、そのオブジェクトが PhotonNetwork.Instantiate() により生成された場合のみ機能します。

シーンと一緒にロードされたオブジェクトは、たとえ PhotonView コンポーネントを持っていても、無視されます。

戻り値

無し。問題がある場合は、デバッグ用エラーログを確認してください。

13.14.2.16 Disconnect **static void PhotonNetwork.Disconnect ( ) [static]**

このクライアントを Photon Server から切断します。Room から退出する、そして完了時に OnDisconnectedFromPhoton をコールする処理をします。

クライアントを切断させると、サーバーに「切断中（"disconnecting"）」のメッセージが送られます。これは同じ Room にいるプレイヤーへの退出／切断のメッセージを迅速化します。（そうしないと、サーバーはこのクライアントの接続でタイムアウトを起こすかもしれません。）

オフラインモードで使われた場合、瞬時に、状態が遷移し、OnDisconnectedFromPhoton がイベントコールされます。オフラインモードも、false に設定されます。一度切断されたクライアントは、再び接続することができます。ConnectUsingSettings を利用してください。

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

サーバーのタイムスタンプを更新する。（非同期オペレーション、1 ラウンドトリップかかる）

接続の不具合などにより、タイムスタンプが利用できない場合や不正確な場合に役立ちます。

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

友達リストの Room とオンラインの状態をリクエストします。すべてのクライアントは、PlayerName プロパティにより固有のユーザーネーム（username）を設定している必要があります。結果は、this.Friends で利用できます。

選択したリストの各ユーザーがプレイしている Room を見つけるために、マスターサーバー上で使用されます。リクエストした結果が提供された場合、それは LoadBalancingClient.Friends にマッピングされます。リストは、初めて使用する際に OpFindFriends により初期化されます。（その前は、null です。）

LoadBalancingClient インスタンスの中の PlayerName を設定することにより、各ユーザーはそれぞれ識別されます。これにより、（マスターサーバーとゲームサーバーへの）すべての接続毎に OpAuthenticate の中の名前が、次々と送られるようになります。注意：友達リストを使う時にプレイヤー名の変更をすることは意味がありません。

ユーザーネームのリスト（友達リスト）は、（Photon から提供されるものではなく）別に何らかの方法で用意される必要があります。

内部処理：サーバーのレスポンス情報は次の２つの配列を含みます。（それぞれの配列の各インデックスは、リクエストの各友達とマッチしています。）：ParameterCode.FindFriendsResponseOnlineList = bool[] オンライン状態のブール配列、ParameterCode.FindFriendsResponseRoomIdList = string[] Room 名の文字列の配列（どの Room にもいない場合は空文字列となる）

パラメーター

friendsToFind	友達の名前の配列（それぞれ固有なことを確認のこと）

戻り値

このオペレーションが送信されたかどうか。（接続している必要があります。どんな時でもこのリクエストは１つだけが認められています。）オフラインモードでは、常に false となります。

13.14.2.19 GetPing **static int PhotonNetwork.GetPing ( ) [static]**

Photon Server への現在のラウンドトリップタイム

戻り値

ラウンドトリップタイム（サーバーへの往復）

13.14.2.20 GetRoomList **static RoomInfo [ ] PhotonNetwork.GetRoomList ( ) [static]**

RoomInfo として（現在）既知の Room の配列を取得します。クライアントが、（マスターサーバー上の）ロビーにいる間、このリストは数秒毎に自動更新されます。いずれかの Room の中にいる間は利用できません。

コールされる度にリストの新しいインスタンスを作ります。 networkingPeer.mGameList からコピーされます。

戻り値

現在のロビー内の Room の RoomInfo[] 配列

13.14.2.21 InitializeSecurity **static void PhotonNetwork.InitializeSecurity ( ) [static]**

Unity Networking との互換性のためだけに使用されます。接続処理中に暗号化は自動的に初期化されます。

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

ネットワーク上でプレハブをインスタンス化します。このプレハブは、Resources フォルダのルートになければなりません。Resources フォルダのプレハブを利用する代わりに、マニュアルで Instantiate をコールし、PhotonView を割り当てることもできます。関連の説明を参照してください。

パラメーター

prefabName	インスタンス化するプレハブの名前
position	インスタンス化の時に、Vector3 に適用される位置
rotation	インスタンス化の時に、Quaternion に適用される回転
group	この PhotonView のグループ

戻り値

初期化された PhotonView 付きの GameObject の新しいインスタンス

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

ネットワーク上でプレハブをインスタンス化します。このプレハブは、Resources フォルダのルートになければなりません。Resources フォルダのプレハブを利用する代わりに、マニュアルで Instantiate をコールし、PhotonView を割り当てることもできます。関連の説明を参照してください。

パラメーター

prefabName	インスタンス化するプレハブの名前
position	インスタンス化の時に、Vector3 に適用される位置
rotation	インスタンス化の時に、Quaternion に適用される回転
group	この PhotonView のグループ
data	インスタンス化のオプションデータ。PhotonView.instantiationData に保存される。

戻り値

初期化された PhotonView 付きの GameObject の新しいインスタンス

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

ネットワーク上でシーンが所有するプレハブをインスタンス化します。Photon View はマスタークライアントによりコントロールできるようになります。このプレハブは、Resources フォルダのルートになければなりません。

マスタークライアントのみが、シーンオブジェクトをインスタンス化できます。Resources フォルダのプレハブを利用する代わりに、マニュアルで Instantiate をコールし、PhotonView を割り当てることもできます。関連の説明を参照してください。

パラメーター

prefabName	インスタンス化するプレハブの名前
position	インスタンス化の時に、Vector3 に適用される位置
rotation	インスタンス化の時に、Quaternion に適用される回転
group	この PhotonView のグループ
data	インスタンス化のオプションデータ。PhotonView.instantiationData に保存される。

戻り値

初期化された PhotonView 付きの GameObject の新しいインスタンス

13.14.2.25 InternalCleanPhotonMonoFromSceneIfStuck **static void PhotonNetwork.InternalCleanPhotonMonoFromSceneIfStuck ( )** [static]

内部的に Editor スクリプトで利用されます。（シーンの保存も含めて）ヒエラルキーに変更があった時にコールされ、隠れて残っている PhotonHandlers を取り除きます。

13.14.2.26 JoinRandomRoom **static void PhotonNetwork.JoinRandomRoom ( )** [static]

利用可能な、どの Room でも参加します。しかし、その時点で利用可能なものがまったく無い場合、失敗します。

失敗した場合でも、Room を作ることができます。（そして、次に JoinRandomRoom を使う人にこの Room を利用可能とします。）あるいは、少したってから再びトライします。

13.14.2.27 JoinRandomRoom **static void PhotonNetwork.JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers)** [static]

カスタムプロパティを適合させて、オープンな Room に参加を試みます。しかし、その時点で利用可能なものがまったく無い場合、失敗します。

失敗した場合でも、Room を作ることができます。（そして、次に JoinRandomRoom を使う人にこの Room を利用可能とします。）あるいは、少したってから再びトライします。

パラメーター

expected-CustomRoom-Properties	Room をマッチさせるためのフィルター条件となるカスタムプロパティ（文字列キーと値）。無効にするには、null を渡す。
expectedMax-Players	フィルター条件とする最大プレイヤー数の設定。最大プレイヤー数がいくつでも良い場合は、0 を使う。

13.14.2.28 JoinRandomRoom **static void PhotonNetwork.JoinRandomRoom (Hashtable expectedCustomRoomProperties, byte expectedMaxPlayers, MatchmakingMode matchingType)** [static]
カスタムプロパティを適合させて、オープンな Room に参加を試みます。しかし、その時点で利用可能なものがまったく無い場合、失敗します。
失敗した場合でも、Room を作ることができます。（そして、次に JoinRandomRoom を使う人にこの Room を利用可能とします。）あるいは、少したってから再びトライします。

パラメーター

expected-CustomRoom-Properties	Room をマッチさせるためのフィルター条件となるカスタムプロパティ（文字列キーと値）。無効にするには、null を渡す。
expectedMax-Players	フィルター条件とする最大プレイヤー数の設定。最大プレイヤー数がいくつでも良い場合は、0 を使う。
matchingType	利用可能なマッチメイキング・アルゴリズムから 1 つを選択。選択肢は、MatchmakingMode enum を参照のこと。

13.14.2.29 JoinRoom **static void PhotonNetwork.JoinRoom (RoomInfo listedRoom)** [static]
room.Name により Room に参加します。Room が定員一杯か（その瞬間に閉じられたかして）利用可能でない場合は失敗します。

パラメーター

roomName	参加する Room インスタンス（listedRoom.Name だけが使われる）

13.14.2.30 JoinRoom **static void PhotonNetwork.JoinRoom (string roomName)** [static]
任意のタイトルにより Room に参加します。Room が定員一杯か（その瞬間に閉じられたかして）利用可能でない場合は失敗します。

パラメーター

roomName	作成する Room の固有な名前

13.14.2.31 LeaveRoom **static void PhotonNetwork.LeaveRoom ( )** [static]
現在の Room を退出します。

13.14.2.32 LoadLevel **static void PhotonNetwork.LoadLevel (int levelNumber)** [static]
レベルを読み込み、自動的にネットワークキューをポーズします。OnJoinedRoom でこれをコールして、キャッシュされた RPC が間違ったシーンで処理されることがないように確認してください。

パラメーター

levelNumber	ロードするレベル番号（番号がビルド設定にあることを確認すること）

13.14.2.33 LoadLevel **static void PhotonNetwork.LoadLevel (string levelTitle)** [static]
レベルを読み込み、自動的にネットワークキューをポーズします。OnJoinedRoom でこれをコールして、キャッシュされた RPC が間違ったシーンで処理されることがないように確認してください。

パラメーター

levelTitle	ロードするレベルの名前

13.14.2.34 NetworkStatisticsReset **static void PhotonNetwork.NetworkStatisticsReset ( ) [static]**

ネットワークトラフィックの統計値をリセットし、計測を再び有効とします。

13.14.2.35 NetworkStatisticsToString **static string PhotonNetwork.NetworkStatisticsToString ( ) [static]**

NetworkStatisticsEnabled が統計値を集計するの使用されている場合のみ、利用可能です。

戻り値

稼働中ネットワークの統計値の文字列

13.14.2.36 OverrideBestCloudServer **static void PhotonNetwork.OverrideBestCloudServer (CloudServerRegion region) [static]**

ConnectToBestCloudServer(string gameVersion) で利用される地域を書き換えます。そして、すべての PhotonCloud Server へ ping して得た結果を書き換えます。

ユーザーに自分の地域サーバーを指定、入力させるために使用してください。

13.14.2.37 RefreshCloudServerRating **static void PhotonNetwork.RefreshCloudServerRating ( ) [static]**

すべての PhotonCloud Server に再び ping し、（その時点での）最良の ping 結果のものを見つけます。

13.14.2.38 RemoveRPCs **static void PhotonNetwork.RemoveRPCs (PhotonPlayer targetPlayer) [static]r**

バッファーリングしている RPC で targetPlayer から送られたものを、サーバーからすべて削除します。ローカルプレイヤー上で自分自身向けての場合か、マスタークライアントの場合に（だれに対しても）、コールすることができます。

このメソッドはいずれかの条件が必要です：

- このクライアントが targetPlayerのクライアントである
- このクライアントがマスタークライアントである。(すべての PhotonPlayer's RPC を削除できる。）

これがコールされると同時に、もし targetPlayer が RPC をコールした場合、その RPC がバッファーリングされるか、もしくは他のものと同じようにクリアされてしまうかは、ネットワークのラグにより決まります。

パラメーター

targetPlayer	このプレイヤーのバッファーリングされている RPC が、サーバーバッファーから削除される

13.14.2.39 RemoveRPCs **static void PhotonNetwork.RemoveRPCs (PhotonView targetPhotonView) [static]**

バッファーリングしている RPC で targetPhotonView によって送られたものを、サーバーからすべて削除します。マスタークライアントまたは targetPhotonView のオーナーがコールすることができます。

このメソッドはいずれかの条件が必要です：

- targetPhotonView がこのクライアントにより所有されている（これによりインスタンス化された）
- •このクライアントがマスタークライアントである(すべての PhotonPlayer's RPC を削除できる）

パラメーター

targetPhoton-View	この PhotonView のためのバッファーリングされた RPC が、サーバーバッファーから削除される

13.14.2.40 RemoveRPCsInGroup **static void PhotonNetwork.RemoveRPCsInGroup (int targetGroup)** [static]

このクライアントが、マスタークライアントか、または個別の PhotonView をコントロールしている場合、バッファーリングしている RPC で、targetGroup 内で送られたものを、サーバーからすべて削除します。

このメソッドはいずれかの条件が必要です：

- このクライアントがマスタークライアントである（グループのすべての RPC を削除できる）
- その他のクライアントの場合、個々の PhotonView がこのクライアントのコントロール下にあるかが調べられる。あてはまるRPC だけが削除される。

パラメーター

targetGroup	すべての RPC が削除されるインタレストグループ

13.14.2.41 SendOutgoingCommands **static void PhotonNetwork.SendOutgoingCommands ( )** [static]

作成されたばかりの RPC やインスタンス化コールを即時に送信するために利用できます。対象の RPC やコールは他のプレイヤーに送信中となります。

あるレベルをロードさせる RPC をコールし、そして自分自身でもそのレベルをロードする場合に、これは役に立つでしょう。レベルをロードしている間は、他のプレイヤーに RPC を送信できませんので、「レベルをロードさせる」RPC を遅延させることになります。このメソッドを使って RPC を「他の人」にまず送信して、（isMessageQueueRunning 利用して）メッセージキューを無効にし、その後ロードを行うということができます。

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

後からインスタンス化される PhotonView の前に付けるレベルを設定します。必要なレベルが 1 つだけの場合は、設定しないでください。

重要：複数のレベルを使用しない場合、絶対にこの値を設定しないでください。デフォルト値は、トラフィックのために最適化されています。

これは、既に存在している PhotonView には影響しません。（既存の PhotonView に対しては変更することができません。）違うレベルが前に付けられて送信されたメッセージは、受信されますが実行はされません。RPC、インスタンス化および同期処理が、これにより影響されます。

PUN は決してこの値をリセットしませんので、必ず自分自身でリセットしなければならないことに注意してください。

パラメーター

prefix	最大値は short.MaxValue = 32767

13.14.2.43 SetMasterClient **static bool PhotonNetwork.SetMasterClient (PhotonPlayer player)** [static]

現在のマスタークライアントに対し、誰か別なユーザーをマスタークライアントとして指名することを許可します。カスタム設定した選択では、どのクライアント上であっても同じユーザーをマスタークライアントとして選択すべきです。

ReceiverGroup.MasterClient（RPC で利用できる）は、これにより影響を受けません(引き続き、Room 内で一番小さい player.ID を参照します）。この値（列挙型）を使用するのは避けてください（そのかわりに特定のプレイヤーに送るようにしてください）。

もし現在のマスタークライアントが退出した場合、PUN は「一番小さい player ID」で新しいマスタークライアントを見つ

けだします。この場合のコールバックを取得するため OnMasterClientSwitched を実装してください。PUN により選ばれたマスタークライアントが別の新しいマスタークライアントを指名するかもしれません。
マスタークライアント指名の無限ループを作らないように確認してください。カスタム設定してマスタークライアントを選択する場合は、どのクライアントが実際に指名をするかに関係なく、すべてのクライアントで同じプレイヤーを指名するようにすべきです。

ローカルでマスタークライアントはただちに変更されますが、一方リモートクライアントはイベントを取得します。ということは、マスタークライアントが退出した時と同じように、ゲームに一時的にマスタークライアントがいない状態をこれは意味します。
マスタークライアントをマニュアルで変更する時は、他のマスタークライアントと同じように、この実行担当のユーザーもゲームから退出して変更処理を実行しない、ということも起こり得るという点を忘れないようにしてください。

パラメーター

playerId	次のマスタークライアントの player.ID

戻り値

この同期アクションが処理されない時は、false。オンライン状態かつマスタークライアントでなければならない。

13.14.2.44 SetPlayerCustomProperties **static void PhotonNetwork.SetPlayerCustomProperties (Hashtable customProperties)** `[static]`
この（ローカル）プレイヤーのプロパティを設定します。PhotonNetwork.player.customProperties にプロパティがキャッシュされます。 CreateRoom、JoinRoom、JoinRandomRoom のすべてが、Room に入る時、プレイヤーのカスタムプロパティを利用します。Room にいる間は、プレイヤーのプロパティは他のプレイヤーと共有／同期されます。ハッシュテーブルが null である場合、カスタムプロパティはクリアされます。カスタムプロパティは決して自動的にはクリアされませんので、変更しない限り、次の Room にも引き継がれていきます。
決して PhotonNetwork.player.customProperties を直接修正して、プロパティを設定しようとしないでください。

パラメーター

custom-Properties	このハッシュテーブルの文字列型キーだけが使用される。 null の場合、カスタムプロパティはすべて削除される。

13.14.2.45 SetReceivingEnabled **static void PhotonNetwork.SetReceivingEnabled (int group, bool enabled)** `[static]`
指定したグループでの受信を有効／無効にします。（PhotonView に適用される）

パラメーター

group	影響されるインタレストグループ
enabled	このグループからの受信を有効（または無効）に設定する。

13.14.2.46 SetSendingEnabled **static void PhotonNetwork.SetSendingEnabled (int group, bool enabled)** `[static]`
指定したグループでの送信を有効／無効にします。（PhotonView に適用される）

パラメーター

group	影響されるインタレストグループ
enabled	このグループからの送信を有効（または無効）に設定する。

13.14.2.47 UnAllocateViewID **static void PhotonNetwork.UnAllocateViewID (int viewID)** `[static]`
（マニュアルにインスタンス化され、その後抹消されたネットワーク上のオブジェクトの）viewID を抹消します。

パラメーター

viewID	このプレイヤーによりマニュアルで割り当てられた viewID

13.14.3 メンバーデータ

13.14.3.1 logLevel **PhotonLogLevel PhotonNetwork.logLevel = PhotonLogLevel.ErrorsOnly** [static]
ネットワークのログ出力レベル。どの程度 PUN が詳細な出力するかをコントロールします。

13.14.3.2 MAX_VIEW_IDS **readonly int PhotonNetwork.MAX VIEW IDS = 1000** [static]
１プレイヤー（または 1 シーン）当たりに割り当てられた PhotonView の最大上限数。この上限を増やす方法については説明している部分を参照してください。

13.14.3.3 PhotonServerSettings **ServerSettings PhotonNetwork.PhotonServerSettings = (ServerSettings)Resources.Load(PhotonNetwork.-serverSettingsAssetFile, typeof(ServerSettings))** [static]
シリアライズされたサーバー設定。 ConnectUsingSettings で利用されるために、セットアップウィザードで書き込まれる。

13.14.3.4 precisionForFloatSynchronization **float PhotonNetwork.precisionForFloatSynchronization = 0.01f** [static]
PhotonView の OnSerialize/ObservingComponent で、更新の送信前に必要となる最小差異、浮動小数の差分。

13.14.3.5 precisionForQuaternionSynchronization **float PhotonNetwork.precisionForQuaternionSynchronization = 1.0f** [static]
PhotonView の OnSerialize/ObservingComponent で、更新の送信前に必要となる rotation 値の変更の最小角度を表します。

13.14.3.6 precisionForVectorSynchronization **float PhotonNetwork.precisionForVectorSynchronization = 0.000099f** [static]
PhotonView の OnSerialize/ObservingComponent で、更新の送信前に必要となる Vector2 や Vector3 で必要となる最小差異（例として、トランスフォームのローテーション値）。
注意 : これは sqrMagnitude（平方マグニチュード）です。例 : Y 軸上の 0.01 の変更の後に送る場合は、0.01f*0.01f=0.0001f を使います。ただし、浮動小数の不正確性の改善措置として、0.0001f の代わりに 0.000099f を使用します。

13.14.3.7 PrefabCache **Dictionary<string, GameObject> PhotonNetwork.PrefabCache = new Dictionary<string, GameObject>()** [static]
頻繁なインスタンス化のためにGameObjects への参照を保持します。（Resources をロードする代わりに、メモリーの中からインスタンス化します。）
Instantiate が実行中の時以外、このキャッシュはいつでも変更できます。メインスレッドだけで行うのが最も良いでしょう。

13.14.3.8 serverSettingsAssetFile **const string PhotonNetwork.serverSettingsAssetFile = ”PhotonServerSettings”**
PhotonServerSettings ファイルの名前（ロードするため、また新しいファイルを保存するため PhotonEditor により使用される）

13.14.3.9 serverSettingsAssetPath **const string PhotonNetwork.serverSettingsAssetPath = ”Assets/Photon Unity Networking/Resources/” +PhotonNetwork.serverSettingsAssetFile + ”.asset”**
PhotonServerSettings ファイルへのパス（PhotonEditor により使用される）

13.14.3.10 UsePrefabCache **bool PhotonNetwork.UsePrefabCache = true** [static]
有効（true）の間、インスタンス化は PhotonNetwork.PrefabCache を利用し、ゲームオブジェクトをメモリーに保持します。（同じプレハブのインスタンス化の効率を向上させます。）
ランタイム中に UsePrefabCache を false に設定しても、PrefabCache はクリアされず、その設定はただちに無視されます。このキャッシュは開発者がクリアし変更することができます。コメントを読んでください。

13.14.3.11 versionPUN **const string PhotonNetwork.versionPUN = "1.21"**
PUN のバージョン番号。クライアントのバージョンを区別するために、ゲームバージョンでも利用されます。

13.14.4 プロパティ

13.14.4.1 AuthValues **AuthenticationValues PhotonNetwork.AuthValues** [static], [get], [set]
Photon （およびカスタムサービス／コミュニティ）へのカスタム認証を利用した接続のために使われるユーザーの認証値。カスタム認証を利用したい場合は、Connect をコールする前にこれらを設定してください。
どのような値を入力しても認証が失敗してしまう場合、PUN は、開発者が実装した OnCustomAuthenticationFailed(string debugMsg) をコールします。PhotonNetworkingMessage.OnCustomAuthenticationFailed を参照してください。

13.14.4.2 autoCleanUpPlayerObjects **bool PhotonNetwork.autoCleanUpPlayerObjects** [static], [get], [set]
退出したプレイヤーが生成した GameObject と PhotonView を、Room に残っているプレイヤーが取り除かなければならないかどうかを、この設定で定義します。
"this client" が Room／ゲームを作る時、autoCleanUpPlayerObjects は（どんな値であっても）Room プロパティにコピーされ、Room 内のすべての PUN クライアントにより使用されます。
もし room.AutoCleanUp がその Room 内で有効の場合、退出時にそのプレイヤーのオブジェクトを PUN クライアントが削除します。
有効の時、退出したプレイヤーの RPC、インスタンス化した GameObjects と PhotonView はサーバーにより削除されます。その後、参加しているプレイヤーはそれらを取得するはもう無くなります。
いったん Room が作成されたら、この設定は変更できません。
デフォルトは有効です。.

13.14.4.3 autoJoinLobby **bool PhotonNetwork.autoJoinLobby** [static], [get], [set]
マスターサーバーに接続した時、PhotonNetwork がロビーに参加するかどうかを定義します。もしこれが false の場合、マスターサーバーへの接続が可能であれば、OnConnectedToMaster() がコールされます。これが false の場合、OnJoinedLobby() はコールされません。
デフォルトは有効です。
Room リストは利用できません。ロビーに参加していなくても（つまり Room リストが利用できなくても）、Room を作成したり、Room に参加したり（ランダムに）することはできます。

13.14.4.4 automaticallySyncScene **bool PhotonNetwork.automaticallySyncScene** [static], [get], [set]
true の場合、PUN は常に全ユーザーが同じシーンにいるようにします。もしマスタークライアントが変更されたら、すべてのクライアントは新しいシーンをロードします。また、これによりメインメニューからゲームに参加した後でも、ゲームシーンのロードがスムーズに処理されます。
もし自動的にシーンを同期させる場合は true ；そうでなければ false とします。

13.14.4.5 connected　**bool PhotonNetwork.connected** [static], [get]
自分たちが Photon Server に接続しているかどうか。（Room 内の場合と Room 外の場合があります。）

13.14.4.6 connectionState　**ConnectionState PhotonNetwork.connectionState** [static], [get]
簡易な接続状態

13.14.4.7 connectionStateDetailed　**PeerState PhotonNetwork.connectionStateDetailed** [static], [get]
詳細な接続状態（PUN を意識していないので、サーバーを切り替えている時、「切断 "disconnected"」となることがあります。）

13.14.4.8 countOfPlayers　**int PhotonNetwork.countOfPlayers** [static], [get]
現在、このゲームアプリケーションを利用しているプレイヤー数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。

13.14.4.9 countOfPlayersInRooms　**int PhotonNetwork.countOfPlayersInRooms** [static], [get]現在、Room の中にいるプレイヤーの数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。

13.14.4.10 countOfPlayersOnMaster　**int PhotonNetwork.countOfPlayersOnMaster** [static], [get]
現在 Room を探しているプレイヤーの数。これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。
13.14.4.11 countOfRooms　**int PhotonNetwork.countOfRooms** [static], [get]
現在使用中の Room の数。ロビー内にいる時は、これは PhotonNetwork.GetRoomList().Length に基づいています。ロビーにいない時は、これはマスターサーバー上で（のみ、変化があった場合に）５秒間隔で更新されます。

13.14.4.12 Friends　**List<FriendInfo> PhotonNetwork.Friends** [static], [get], [set]
オンラインの状態と参加している Room の情報を含む、リードオンリーの友達（friends）リスト。FindFriends コールにより初期化されるまでは、null です。
決してこのリストを修正しないでください！これは内部的に FindFriends に扱われ、値を読み込むためだけに利用できます。FriendsListAge が、ミリセカンド単位でデータがどの程度経過しているかを知らせます。
必要以上に、頻繁にこのリストを取得しないでください（実用的な間隔は最低でも 10 秒を超える間隔でしょう）。取得するリストをとても短いものにするのが、一番良いでしょう。例としては、まずリスト全体を１回取得し、その後オンライン状態の友達だけの情報を、２、３回リクエストします。少したって（例えば１分後ぐらいに）、オンラインステータスを更新するため、リスト全体をもう一度取得するようにする、というやり方があります。

13.14.4.13 FriendsListAge　**int PhotonNetwork.FriendsListAge** [static], [get]
友達リストの経過時間（ミリセカンド単位）。友達リストが取得されるまでは、0 です。

13.14.4.14 insideLobby　**bool PhotonNetwork.insideLobby** [static], [get]
Photon に接続し、ロビーに入った状態の時、true を返します。

13.14.4.15 isMasterClient **bool PhotonNetwork.isMasterClient** [static], [get]
自分たちがマスタークライアントかどうか

13.14.4.16 isMessageQueueRunning **bool PhotonNetwork.isMessageQueueRunning** [static], [get], [set]
（RPC, インスタンス化など、 その他すべての）着信するイベントの処理をポーズするのに使用されます。まず始めにレベルだけをロードし、その後、続いて PhotonView と RPC のデータを受け取るような時に、これは役に立ちます。クライアントは受信を続け、着信してくる情報パッケージや RPC／イベントへの受信確認（ACK）を送り返します。これはラグを発生させ、すべての着信メッセージがキューされるため、ポーズがより長くなると問題を起こすこともあり得ます。

13.14.4.17 isNonMasterClientInRoom **bool PhotonNetwork.isNonMasterClientInRoom** [static], [get]
Room の中にいて、かつ Room のマスタークライアントで無い場合、true。

13.14.4.18 masterClient **PhotonPlayer PhotonNetwork.masterClient** [static], [get]
マスタークライアントである PhotonPlayer。マスタークライアントは、Room の「ヴァーチャルなオーナー」。オーソラティブな（ゲームロジックを支配するような）決定がプレイヤーの 1 人によって必要な場合は、これを利用できます。
masterClient は、誰かが Room 内に参加するまでは null で、また Room から誰もいなくなったら、また null になります。

13.14.4.19 maxConnections **int PhotonNetwork.maxConnections** [static], [get], [set]
Room の最大プレイヤーの数。より良いやり方として、CreateRoom でこれを設定するようにします。オープンな Room が無い時は、これは 0 を返します。

13.14.4.20 NetworkStatisticsEnabled **bool PhotonNetwork.NetworkStatisticsEnabled** [static], [get], [set]
このクライアントのトラフィックの統計値の集計を有効または無効にする。クライアントで何か問題に直面した場合、解決策のために、まずはトラフィックの統計から調べるといいでしょう。
統計が有効な時だけ、GetVitalStats を利用できます。

13.14.4.21 offlineMode **bool PhotonNetwork.offlineMode** [static], [get], [set]
マルチプレイヤーのコードをシングルプレイヤーゲームでも再利用できるように、オフラインモードを設定することができます。これが有効にされると、PhotonNetwork は接続を一切することなく、ほとんどオーバーヘッドが無くなります。特に RPC と PhotonNetwork.Instantiate を再利用するのにとても役立ちます。

13.14.4.22 otherPlayers **PhotonPlayer [ ] PhotonNetwork.otherPlayers** [static], [get]
自身のローカルプレイヤーを含まない、他のすべての PhotonPlayers

13.14.4.23 player **PhotonPlayer PhotonNetwork.player [static], [get]**
ローカルの PhotonPlayer 。常に利用可能で、this player を表す。カスタムプロパティは Room に入る前に設定でき、また共有／同期されます。

13.14.4.24 playerList **PhotonPlayer [ ] PhotonNetwork.playerList** [static], [get]
ローカルプレイヤーも含めた、すべての PhotonPlayer のリスト

13.14.4.25 playerName **string PhotonNetwork.playerName** [static], [get], [set]
ローカルプレイヤーの名前
接続している時は、名前を設定すると自動的に送信されます。 null を設定しても、名前は変更されません。

13.14.4.26 ResentReliableCommands **int PhotonNetwork.ResentReliableCommands** [static], [get]
（ローカルで ACK 受信前のリピートのタイミングにより）リピートされたコマンドの数。

13.14.4.27 room **Room PhotonNetwork.room** [static], [get]
現在いる Room を取得。もし Room にいない場合、null。

13.14.4.28 sendRate **int PhotonNetwork.sendRate** [static], [get], [set]
PhotonNetwork が毎秒あたり何回更新などの情報パッケージを送信するかを定義します。これを変更した場合は、sendRateOnSerialize も変更するのを忘れないようにしてください。
回数が少ないほど、オーバーヘッドも小さくなりますが、遅延がより大きくなります。 sendRate を 50 にすると、毎秒 50 の情報パッケージ（とても多い！）を作成します。ターゲットするプラットフォームを考慮してくださいさい：モバイルのネットワークは、より遅く信頼性も低いです。

13.14.4.29 sendRateOnSerialize **int PhotonNetwork.sendRateOnSerialize** [static], [get], [set]
PhotonView で、毎秒あたり何回 OnPhotonSerialize がコールされるかを定義します。
sendRate の関係から、この値を選択してください。 OnPhotonSerialize は、送信パッケージに含まれる各コマンドを作りだします。より低いレートですと、あまり高いパフォーマンスを必要としませんが、ラグをより大きくすることとなります。

13.14.4.30 ServerAddress **string PhotonNetwork.ServerAddress** [static], [get]
使用中のサーバーアドレス（マスターサーバーとゲームサーバーのどちらでも）

13.14.4.31 time **double PhotonNetwork.time** [static], [get]
Photon Network タイム、 サーバーと同期される
v1.3：サーバーが稼働してからの経過時間が、ミリセカンド単位、4 バイトで表されています。これは 49 日毎にオーバーフローし、上限の大きな数字から 0 に戻されます。このオーバーフローに関しては（まだ）対応していません。マスターサーバーとゲームサーバーは、別々の時間となります。 v1.10：大きなサーバータイム値の精度の問題は修正されました。デフォルトで、15ms の精度で更新されます。

13.14.4.32 unreliableCommandsLimit **int PhotonNetwork.unreliableCommandsLimit** [static], [get], [set]
チャネル毎にアンリライアブルコマンドを制限するため、コマンド発行毎に 1 度使用されます。（ですから、ポーズの後は、多くのチャネルがたくさんのアンリライアブルコマンドを引き起こすことがあり得ます。）

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs

13.15 PhotonPlayer クラス レファレンス

Room内で actorID により識別される任意の「1 プレイヤー（player）」についてまとめて扱っています。

パブリックメンバー関数

- PhotonPlayer (bool isLocal, int actorID, string name)
  PhotonPlayer インスタンスを生成する
- override string ToString ()
  名前文字列を渡す
- override bool Equals (object p)
  PhotonPlayer の比較をする
- override int GetHashCode ()
- void SetCustomProperties (Hashtable propertiesToSet)
  この Player の名前付きのプロパティを propertiesToSet の値で、更新または同期する。

静的パブリックメンバー関数

- static PhotonPlayer Find (int ID)
  IDにより特定のプレイヤーを取得しようとする。

パブリック属性

- readonly bool isLocal = false

それぞれ 1 つのクライアントは、1 プレイヤーだけをコントロールする。その他はローカルクライアントではない。

プロテクトメンバー関数

- PhotonPlayer (bool isLocal, int actorID, Hashtable properties)
  Join イベントからプレイヤー作成するため内部的に使用される

プロパティ

- int ID [get]
  このプレイヤーの actorID
- string name [get, set]
  このプレイヤーのニックネーム
- bool isMasterClient [get]
  最小の actorID をもつプレイヤーがマスターとなり、特別なタスクのために利用される。
- Hashtable customProperties [get, set]
  プレイヤーのカスタムプロパティ用のリードオンリーのキャッシュ。Player.SetCustomProperties から設定される。
- Hashtable allProperties [get]
  すべてのプロパティを含むハッシュテーブルを作成する（カスタムと標準のプロパティ）

13.15.1 解説

Room内で actorID により識別される任意の 1 プレイヤーについてまとめています。
それぞれのプレイヤーはそのRoomの中で有効な 1 つの actorId （または、ID）を持っています。 サーバーに割り当てられるまでは、それは -1です。Room の中に入る前であっても、各クライアントは SetCustomProperties でプレイヤーのカスタムプロパティを設定することができます。Roomに参加する時、それらは共有／同期されます。

13.15.2 コンストラクターとデストラクター

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

PhotonPlayer インスタンスを生成する

パラメーター

isLocal	これがローカル Peer のプレイヤーか（または、リモートのプレイヤーか）
actorID	現在の Room での、このプレイヤーの ID か ActorNumber（Room 内の個々のプレイヤーを識別するショートカット）
name	プレイヤーの名前（標準のプロパティの１つ）

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

Join イベントからプレイヤー作成するため内部的に使用される

13.15.3 メンバー関数

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

PhotonPlayer の比較をする

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

ID により特定のプレイヤーを取得しようとする。

パラメーター

ID	ActorID

戻り値

actorID がマッチするプレイヤー、または、actorID が使用されていない場合は、null。.

13.15.3.3 GetHashCode **override int PhotonPlayer.GetHashCode ( )**

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

この Player の名前付きのプロパティを propertiesToSet の値で、更新または同期する。.

プレイヤーが Room 内にいて切断または退出するまで、どのプレイヤーのプロパティも Room 内だけで利用可能です。Player.CustomProperties（リードオンリー）でどのプレイヤーのプロパティにもアクセスできますが、このリードオンリーのハッシュテーブルは変更しないでください。

新規のプロパティは追加され、既にるものは更新されます。含まれていない他の値は変更されませんので、変更するまたは新規に追加するものだけを提供するようにしてください。このプレイヤーの名前付き（カスタム）プロパティを削除するには、null を値としてください。文字列型のキーが使用されます（それ以外は無視されます）。

ローカルキャッシュはすぐに更新され、他のプレイヤーは対応したオペレーションで Photon を通じて更新されます。ネットワークのトラフィックを抑えるために、変更される値だけを使用するようにしてくださない。

パラメーター

propertiesToSet	更新、設定、同期するプロパティのハッシュテーブル。説明を参照のこと。

13.15.3.5 ToString　**override string PhotonPlayer.ToString ( )**

名前文字列を渡す

13.15.4 メンバーデータ

13.15.4.1 isLocal　**readonly bool PhotonPlayer.isLocal = false**

それぞれ 1 つのクライアントは、1 プレイヤーだけをコントロールします。その他はローカルクライアントではありません。

13.15.5 プロパティ

13.15.5.1 allProperties　**Hashtable PhotonPlayer.allProperties** `[get]`

すべてのプロパティを含むハッシュテーブルを作成します（カスタムと標準のプロパティ）

頻繁に使われる場合、キャッシュされるべきです。

13.15.5.2 customProperties　**Hashtable PhotonPlayer.customProperties** `[get]`, `[set]`

プレイヤーのカスタムプロパティ用のリードオンリーのキャッシュ。 Player.SetCustomProperties から設定されます。

このハッシュテーブルの内容を修正しないでください。SetCustomProperties とこのクラスのプロパティ情報を利用して、値を変更してください。これらを使用する時、クライアントはこれらの値をサーバーと同期させます。

13.15.5.3 ID　**int PhotonPlayer.ID** `[get]`

このプレイヤーの actorID

13.15.5.4 isMasterClient　**bool PhotonPlayer.isMasterClient** `[get]`

最小の actorID をもつプレイヤーがマスターとなり、特別なタスクのために利用されます。

13.15.5.5 name　**string PhotonPlayer.name** `[get]`, `[set]`

このプレイヤーのニックネーム

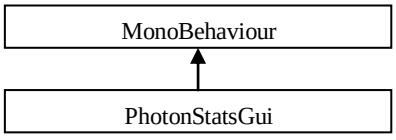
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs

13.16　PhotonStatsGui クラス レファレンス

Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。

継承概念図： PhotonStatsGui:



パブリックメンバー関数

- void Start ()
- void Update ()
  （statsOn をトグルさせる）shift+tab ショートカットキーの入力をチェックする
- void OnGUI ()
- void TrafficStatsWindow (int windowID)

パブリック属性

- bool statsWindowOn = true
  GUI の表示／非表示（統計値収集のオンオフには影響しない）
- bool statsOn = true
  統計値収集のオンオフの別のやり方（Update()で利用される）
- bool healthStatsVisible
  追加の接続の「健全性」の値を表示
- bool trafficStatsOn
  追加の「下層レベル」のトラフィック統計値を表示
- bool buttonsOn
  統計管理とリセットのボタンを表示
- Rect statsRect = new Rect(0, 100, 200, 50)
  ウインドウのために長方形を位置づけます。
- int WindowId = 100
  Unity GUI Window ID （固有のものでなければならない、そうでないと問題を起こす）

13.16.1 解説

Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。
表示される健全性の数値は、接続のロスやパフォーマンスの問題を特定するのに役立ちます。例：２つの連続した SendOutgoingCommandsコールの間の差分時間が１秒以上の場合、切断がこのコールにより引き起こされた可能性が浮上します(なぜなら、サーバーへの確認応答（ACK）は一定時間内に返さなければならないからです）。

13.16.2 メンバー関数

13.16.2.1 OnGUI **void PhotonStatsGui.OnGUI ( )**

13.16.2.2 Start **void PhotonStatsGui.Start ( )**

13.16.2.3 TrafficStatsWindow **void PhotonStatsGui.TrafficStatsWindow (int windowID)**

13.16.2.4 Update **void PhotonStatsGui.Update ( )**

（statsOn をトグルさせる）shift+tab ショートカットキーの入力をチェックします。

13.16.3 メンバーデータ

13.16.3.1 buttonsOn **bool PhotonStatsGui.buttonsOn**

統計管理とリセットのボタンを表示します。

13.16.3.2 healthStatsVisible **bool PhotonStatsGui.healthStatsVisible**
追加の接続の「健全性」の値を表示

13.16.3.3 statsOn **bool PhotonStatsGui.statsOn = true**
統計値収集のオンオフの別のやり方（Update()で利用される）

13.16.3.4 statsRect **Rect PhotonStatsGui.statsRect = new Rect(0, 100, 200, 50)**
ウインドウのために長方形を位置づけます。

13.16.3.5 statsWindowOn **bool PhotonStatsGui.statsWindowOn = true**
GUI の表示／非表示（統計値収集のオンオフには影響しない）

13.16.3.6 trafficStatsOn **bool PhotonStatsGui.trafficStatsOn**
追加の「下層レベル」のトラフィック統計値を表示

13.16.3.7 WindowId **int PhotonStatsGui.WindowId = 100**
Unity GUI Window ID （固有のものでなければならない、そうでないと問題を起こす）

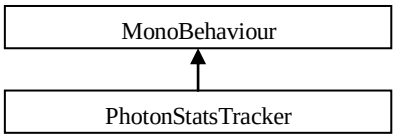
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs

13.17 PhotonStatsTracker クラス レファレンス

Photon への接続に関するトラフィックや状態の統計値を表示するための基本GUI。shift+tab により、トグルします。

継承概念図： PhotonStatsTracker:



パブリックメンバー関数

- void Start ()
- void Update ()
  （TrafficStatsEnabled をトグルさせる）left-shift+tab ショートカットキーの入力をチェックする
- void OnGUI ()
- void TrackEvent (string eventTxt)
- void OnApplicationQuit ()
- void OnDisconnectedFromPhoton ()
- void OnConnectedToPhoton ()
- void OnJoinedRoom ()

- void OnConnectionFail (DisconnectCause disconnectCause)
- long UtcUnixTimestamp ()

静的パブリックメンバー関数

- static string ToJsArray (string[ ] txts)
- static string ToJsArray (int[ ] values)

パブリック属性

- string version = "0.4"
- bool TrackingEnabled = true
  統計収集オンオフの別のやり方（Update() で使用される）
- int TrackingInterval = 2000
  統計値トラッキングする間隔、ミリセカンド単位
- int TrackingQueueLimit = 150
  トラッキングの履歴をとるエントリー数を制限する
- bool SendStatisticsOnTimeout = true
  接続のタイムアウト時に、自動的に計測値を送る
- bool SendStatisticsOnExit = true
  アプリケーション終了時に、自動的に計測値を送る
- bool StatisticsButton = false
  GUI の表示／非表示（統計値収集のオンオフには影響しない）
- Rect ButtonPosRect = new Rect(0, 100, 200, 50)
  ウインドウのために長方形を位置づけます。
- int SaveToPrefsIntervals = 5
  N番目の間隔ごとに、現在のセッションを PlayerPrefs に保存する。無効にするには、0 を使用。
- string outputFileName = "stats_output.txt"

13.17.1 解説

Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。
表示される健全性の数値は、接続のロスやパフォーマンスの問題を特定するのに役立ちます。例：２つの連続した SendOutgoingCommandsコールの間の差分時間が１秒以上の場合、切断がこれにより引き起こされた可能性が浮上します(なぜなら、サーバーへの確認応答（ACK）は一定時間内に返さなければならないからです）。

13.17.2 メンバー関数

13.17.2.1 OnApplicationQuit **void PhotonStatsTracker.OnApplicationQuit ( )**

13.17.2.2 OnConnectedToPhoton **void PhotonStatsTracker.OnConnectedToPhoton ( )**

13.17.2.3 OnConnectionFail **void PhotonStatsTracker.OnConnectionFail (DisconnectCause disconnectCause)**

13.17.2.4 OnDisconnectedFromPhoton **void PhotonStatsTracker.OnDisconnectedFromPhoton ( )**

13.17.2.5 OnGUI **void PhotonStatsTracker.OnGUI ( )**

13.17.2.6 OnJoinedRoom **void PhotonStatsTracker.OnJoinedRoom ( )**

13.17.2.7 Start **void PhotonStatsTracker.Start ( )**

13.17.2.8 ToJsArray　**static string PhotonStatsTracker.ToJsArray (string[ ] txts)** [static]

13.17.2.9 ToJsArray　**static string PhotonStatsTracker.ToJsArray (int[ ] values)** [static]

13.17.2.10 TrackEvent　**void PhotonStatsTracker.TrackEvent (string eventTxt)**

13.17.2.11 Update　**void PhotonStatsTracker.Update ( )**

（TrafficStatsEnabled をトグルさせる）left-shift+tab ショートカットキーの入力をチェックする

13.17.2.12 UtcUnixTimestamp　**long PhotonStatsTracker.UtcUnixTimestamp ( )**

13.17.3 メンバーデータ

13.17.3.1 ButtonPosRect　**Rect PhotonStatsTracker.ButtonPosRect = new Rect(0, 100, 200, 50)**

ウインドウのために長方形を位置づけます。

13.17.3.2 outputFileName　**string PhotonStatsTracker.outputFileName = ”stats output.txt”**

13.17.3.3 SaveToPrefsIntervals　**int PhotonStatsTracker.SaveToPrefsIntervals = 5**

N 番目の間隔ごとに、現在のセッションを PlayerPrefs に保存する。無効にするには、0 を使用。

13.17.3.4 SendStatisticsOnExit　**bool PhotonStatsTracker.SendStatisticsOnExit = true**

アプリケーション終了時に、自動的に計測値を送る

13.17.3.5 SendStatisticsOnTimeout　**bool PhotonStatsTracker.SendStatisticsOnTimeout = true**

接続のタイムアウト時に、自動的に計測値を送る

13.17.3.6 StatisticsButton　**bool PhotonStatsTracker.StatisticsButton = false**

GUI の表示／非表示（統計値収集のオンオフには影響しない）

13.17.3.7 TrackingEnabled　**bool PhotonStatsTracker.TrackingEnabled = true**

統計収集オンオフの別のやり方（Update() で使用される）

13.17.3.8 TrackingInterval　**int PhotonStatsTracker.TrackingInterval = 2000**

統計値トラッキングする間隔、ミリセカンド単位

13.17.3.9 TrackingQueueLimit　**int PhotonStatsTracker.TrackingQueueLimit = 150**

トラッキングの履歴をとるエントリー数を制限する

13.17.3.10 version **string PhotonStatsTracker.version = ”0.4”**

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsTracker.cs

13.18 PhotonStream クラス レファレンス

OnPhotonSerializeView により書かれたデータを保持するために使うコンテナクラス

パブリックメンバー関数

- PhotonStream (bool write, object[ ] incomingData)
- object ReceiveNext ()
- void SendNext (object obj)
- object[ ] ToArray ()
- void Serialize (ref bool myBool)
- void Serialize (ref int myInt)
- void Serialize (ref string value)
- void Serialize (ref char value)
- void Serialize (ref short value)
- void Serialize (ref float obj)
- void Serialize (ref PhotonPlayer obj)
- void Serialize (ref Vector3 obj)
- void Serialize (ref Vector2 obj)
- void Serialize (ref Quaternion obj)

プロパティ

- bool isWriting `[get]`
- bool isReading `[get]`
- int Count `[get]`

13.18.1 解説

OnPhotonSerializeView により書かれたデータを保持するために使うコンテナクラス
以下もご参照ください。
PhotonNetworkingMessage

13.18.2 コンストラクターとデストラクター

13.18.2.1 PhotonStream **PhotonStream.PhotonStream (bool write, object[ ] incomingData)**

13.18.3 メンバー関数

13.18.3.1 ReceiveNext **object PhotonStream.ReceiveNext ( )**

13.18.3.2 SendNext **void PhotonStream.SendNext (object obj)**

13.18.3.3 Serialize **void PhotonStream.Serialize (ref bool myBool)**

13.18.3.4 Serialize **void PhotonStream.Serialize (ref int myInt)**

13.18.3.5 Serialize **void PhotonStream.Serialize (ref string value)**

13.18.3.6 Serialize **void PhotonStream.Serialize (ref char value)**

13.18.3.7 Serialize **void PhotonStream.Serialize (ref short value)**

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

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

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

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

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

13.18.3.13 Serialize **object [ ] PhotonStream.ToArray ( )**

13.18.4 プロパティ

13.18.4.1 Count **int PhotonStream.Count** `[get]`

13.18.4.2 isReading **bool PhotonStream.isReading** `[get]`

13.18.4.3 isWriting **bool PhotonStream.isWriting** `[get]`

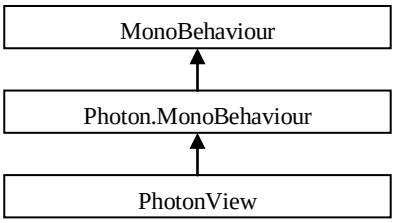
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs

13.19 PhotonView クラス レファレンス

NetworkView に代わる、PUN でネットワークに利用されるクラス。 NetworkView と同じように使用してください。

継承概念図： PhotonView:



パブリックメンバー関数

- void Awake ()
- アプリケーション開始時に Unity からコールされる。PhotonView の設定をする。
- void OnApplicationQuit ()
- void OnDestroy ()

- void RPC (string methodName, PhotonTargets target, params object[ ] parameters)
- void RPC (string methodName, PhotonPlayer targetPlayer, params object[ ] parameters)
- override string ToString ()

静的パブリックメンバー関数

- static PhotonView Get (Component component)
- static PhotonView Get (GameObject gameObj)
- static PhotonView Find (int viewID)

パブリック属性

- int subId
- int ownerId
- int group = 0
- int prefixBackup = -1
- Component observed
- ViewSynchronization synchronization
- OnSerializeTransform onSerializeTransformOption = OnSerializeTransform.PositionAndRotation
- OnSerializeRigidBody onSerializeRigidBodyOption = OnSerializeRigidBody.All
- int instantiationId

プロテクトメンバー関数

- void ExecuteOnSerialize (PhotonStream pStream, PhotonMessageInfo info)

プロパティ

- int prefix `[get, set]`
- object[ ] instantiationData `[get, set]`

  PhotonNetwork.Instantiate＊をコールした時に、渡された instantiationData （＊このプレハブ ”this prefab” を生成するのに使われた場合）
- int viewID `[get, set]`
- bool isSceneView `[get]`
- PhotonPlayer owner `[get]`
- int OwnerActorNr `[get]`
- bool isMine `[get]`

  この photonView は自分のものですか？ 所有者がローカルの PhotonPlayer と一致の場合、true 。もしマスタークライアント上のシーンの photonview の場合も、true 。

13.19.1 解説

NetworkView に代わる、PUN でネットワークに利用されるクラス。 NetworkView と同じように使用してください。

13.19.2 メンバー関数

13.19.2.1 Awake **void PhotonView.Awake ( )**

アプリケーション開始時に Unity からコールされる。PhotonView の設定をする。

13.19.2.2 ExecuteOnSerialize **void PhotonView.ExecuteOnSerialize (PhotonStream pStream, PhotonMessageInfo info)** [protected]

13.19.2.3 Find **static PhotonView PhotonView.Find (int viewID)** [static]

13.19.2.4 Get **static PhotonView PhotonView.Get (Component component)** [static]

13.19.2.5 Get **static PhotonView PhotonView.Get (GameObject gameObj)** [static]

13.19.2.6 OnApplicationQuit **void PhotonView.OnApplicationQuit ( )**

13.19.2.7 OnDestroy **void PhotonView.OnDestroy ( )**

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

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

13.19.2.10 ToString **override string PhotonView.ToString ( )**

13.19.3 メンバーデータ

13.19.3.1 group **int PhotonView.group = 0**

13.19.3.2 instantiationId **int PhotonView.instantiationId**

13.19.3.3 observed **Component PhotonView.observed**

13.19.3.4 onSerializeRigidBodyOption **OnSerializeRigidBody PhotonView.onSerializeRigidBodyOption = OnSerializeRigidBody.All**

13.19.3.5 onSerializeTransformOption **OnSerializeTransform PhotonView.onSerializeTransformOption = OnSerializeTransform.PositionAndRotation**

13.19.3.6 ownerId **int PhotonView.ownerId**

13.19.3.7 prefixBackup **int PhotonView.prefixBackup = -1**

13.19.3.8 subId **int PhotonView.subId**

13.19.3.9 synchronization **ViewSynchronization PhotonView.synchronization**

13.19.4 プロパティ

13.19.4.1 instantiationData **object [ ] PhotonView.instantiationData** [get], [set]

PhotonNetwork.Instantiate＊をコールした時に、渡された instantiationData（＊this prefab を生成するのに使われた場合）

13.19.4.2 isMine **bool PhotonView.isMine** [get]

この photonView は自分のものですか？ 所有者がローカルの PhotonPlayer と一致の場合、true 。もしマスタークライアント上のシーンの photonview の場合も、true 。

13.19.4.3 isSceneView **bool PhotonView.isSceneView** [get]

13.19.4.4 owner **PhotonPlayer PhotonView.owner** [get]

13.19.4.5 OwnerActorNr **int PhotonView.OwnerActorNr** [get]

13.19.4.6 prefix **int PhotonView.prefix** [get], [set]

13.19.4.7 viewID **int PhotonView.viewID** [get], [set]

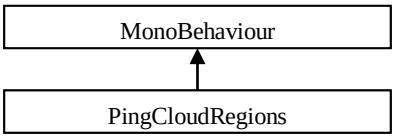
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs

13.20 PingCloudRegions クラス レファレンス

このスクリプトは、PUNにより自動的に PhotonHandler ゲームオブジェクトに追加されます。（世界中の）各地域 ExitGames Cloud に、Awakeから自動的に pingします。 これはクライアント毎に１度だけ実行され、結果は PlayerPrefs に保存されます。最良の接続状態の地域 ExitGames Cloudに接続するために、PhotonNetwork.ConnectToBestCloudServer(gameVersion) を利用してください。

継承概念図： PingCloudRegions:



パブリックメンバー関数

- IEnumerator PingAllRegions ()

静的パブリックメンバー関数

- static void OverrideRegion (CloudServerRegion region)
- static void RefreshCloudServerRating ()
- static void ConnectToBestRegion (string gameVersion)
- static string ResolveHost (string hostString)
- ホストネームをIP文字列への変換を試みる。もし失敗したら、空を返す。

静的パブリック属性

- static PingCloudRegions SP

13.20.1 解説

このスクリプトは、PUNにより自動的に PhotonHandler ゲームオブジェクトに追加されます。（世界中の）各地域 ExitGames Cloud に、Awakeから自動的に ping （接続確認を）します。 これはクライアント毎に１度だけ実行され、結果は PlayerPrefs に保存されます。最良の接続状態の地域 ExitGames Cloud に接続するために、PhotonNetwork.ConnectToBestCloudServer(gameVersion) を利用してください。

13.20.2 メンバー関数

13.20.2.1 ConnectToBestRegion **static void PingCloudRegions.ConnectToBestRegion (string gameVersion)** [static]

13.20.2.2 OverrideRegion **static void PingCloudRegions.OverrideRegion (CloudServerRegion region)** [static]

13.20.2.3 PingAllRegions　**IEnumerator PingCloudRegions.PingAllRegions ( )**

13.20.2.4 RefreshCloudServerRating　**static void PingCloudRegions.RefreshCloudServerRating ( )** `[static]`

13.20.2.5 ResolveHost　**static string PingCloudRegions.ResolveHost (string hostString)** `[static]`

ホストネームを IP 文字列への変換を試みる。もし失敗したら、空を返す。

パラメーター

hostString	変換するホストネーム

戻り値

IP 文字列か、変換に失敗の場合は空の文字列

13.20.3 メンバーデータ

13.20.3.1 SP　**PingCloudRegions PingCloudRegions.SP** `[static]`

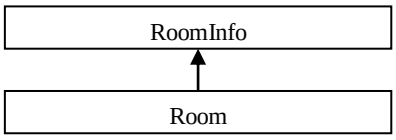
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs

13.21　Room クラスのレファレンス

このクラスは、PUN が参加する（している）Roomをそのまま反映します。RoomInfo のプロパティとは違い、このプロパティは設定可能であり、「自分の」Room を閉じたり、隠したりできます。.

継承概念図： Room:



パブリックメンバー関数

- void SetCustomProperties (Hashtable propertiesToSet)
  propertiesToSet の値でこの Room の名前付きのプロパティを、更新および同期させます。

プロパティ

- new int playerCount `[get]`
  この Roomのプレイヤーの数
- new string name `[get, set]`
  Room の名前。Room／対戦の（1 ロードバランサーグループ内で）固有な識別子。
- new int maxPlayers `[get, set]`
  この Room のプレイヤー数の上限を設定する。このプロパティもロビーで表示されます。Roomが満員（プレイヤーの数＝＝maxplayers）の場合、このRoomへの参加は失敗となります。
- new bool open `[get, set]`

Roomが参加できるかどうかを定義。これはロビーでのリスト表示には影響を与えませんが、もし open でない場合はRoomへの参加は失敗します。 open でない場合は、そのRoomはランダムマッチメイキングから除外されます。ゲームの状況により、参加する前にRoom／対戦が closed となってしまうこともあります。単にマスターに再接続して、他を見つけるようにしましょう。Roomをリストしないためには、"visible" プロパティを利用してください。

- new bool visible [get, set]
  Roomがロビーでリストされるかどうかを定義。Roomは、「見えない（リストしない）状態」として作成したり、また作成後 「見えない状態」に変更したりできます。Roomが参加できるかどうかの変更は、"open" プロパティを利用してください。
- string[ ] propertiesListedInLobby [get, set]
  ロビーに転送されリストで表示されるカスタムプロパティのリスト。
- bool autoCleanUp [get]
  プレイヤーの退出時にすべての(バッファーリングされた)RPC や GameObjects のインスタンス化コールを削除するために、autoCleanUp を使用しているかどうかを取得する。

追加の継承メンバー

13.21.1 解説

このクラスは、PUN が参加する（している）Roomをそのまま反映します。RoomInfo のプロパティとは違い、このプロパティは設定可能であり、「自分の」Room を閉じたり、隠したりできます。

13.21.2 メンバー関数

13.21.2.1 SetCustomProperties **void Room.SetCustomProperties (Hashtable propertiesToSet)**

propertiesToSet の値でこの Room の名前付きのプロパティを、更新および同期させます。

すべてのプレイヤーが Room のプロパティを設定できます。Room プロパティは、変更や削除されるまで、かつ最後のプレイヤーが Room から退出するまで、利用が可能です。Room.CustomProperties （リードオンリー！）を利用し、アクセスしてください。

新規のプロパティは追加され、既にるものは更新されます。他の値は変更されませんので、変更するまたは新規に追加するものだけを提供するようにしてください。この Room の名前付き（カスタム）プロパティを削除するには、null を値としてください。文字列型のキーだけが利用でされます（それ以外は無視されます）。

ローカルキャッシュはすぐに更新され、他のクライアントは対応したオペレーションで Photon を通じて更新されます。ネットワークのトラフィックを抑えるために、変更される値だけを設定するようにしてくださいい。

パラメーター

propertiesToSet	更新、設定、同期するプロパティのハッシュテーブル。説明を参照のこと。

13.21.3 プロパティ

13.21.3.1 autoCleanUp **bool Room.autoCleanUp** [get]

プレイヤーの退出時にすべての（バッファーリングされた）RPC や GameObjects のインスタンス化コールを削除するために、autoCleanUp を使用しているかどうかを取得します。

13.21.3.2 maxPlayers **new int Room.maxPlayers** [get], [set]

この Room のプレイヤー数の上限を設定する。このプロパティもロビーで表示されます。Room が満員（プレイヤーの数＝＝maxplayers）の場合、この Room への参加は失敗となります。

13.21.3.3 name **new string Room.name** [get], [set]

Room の名前。Room／対戦の（１ロードバランサーグループ内で）固有な識別子。

13.21.3.4 open **new bool Room.open** [get], [set]

Room が参加できるかどうかを定義。これはロビーでのリスト表示には影響を与えませんが、もし open でない場合は Room への参加は失敗します。 open でない場合は、その Room はランダムマッチメイキングから除外されます。ゲームの状況により、参加する前に Room／対戦が closed となってしまうこともあります。単にマスターに再接続して、他を見つけるようにしましょう。Room をリストしないためには、"visible" プロパティを利用してください。

13.21.3.5 playerCount **new int Room.playerCount** [get]

この Room のプレイヤーの数

13.21.3.6 propertiesListedInLobby **string [ ] Room.propertiesListedInLobby** [get], [set]

ロビーに転送されリストで表示されるカスタムプロパティのリスト。

13.21.3.7 visible **new bool Room.visible** [get], [set]

Room がロビーでリストされるかどうかを定義。Room は、「見えない（リストしない）状態」 として作成したり、また作成後「見えない状態」に変更したりできます。Room が参加できるかどうかの変更は、"open" プロパティを利用してください。

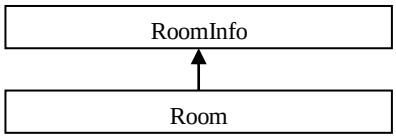
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/Room.cs

13.22 RoomInfo クラス レファレンス

リスト表示や参加するために必要な情報だけに簡素化されたRoom。ロビーでのRoomのリスティングに使用します。プロパティ（open や maxPlayersなど）は設定できません。

継承概念図： RoomInfo:



パブリックメンバー関数

- override bool Equals (object p)
  RoomInfo を（名前で）比較する
- override int GetHashCode ()
  Equals と伴に使われる。名前のハッシュコードを戻り値として使う。
- override string ToString ()
  表示出力メソッド

プロテクト属性

- byte maxPlayersField = 0
  プロパティのバックフィールド
- bool openField = true

プロパティのバックフィールド

- bool visibleField = true
  プロパティのバックフィールド
- bool autoCleanUpField = false
  プロパティのバックフィールド。GameProperty が true に設定（true でない場合は送信されない）されていない限り、false。
- string nameField
  プロパティのバックフィールド

プロパティ

- bool removedFromList [get, set]
  既にリストされなくなっているRoomをマークするために、内部的にロビーで利用される。
- Hashtable customProperties [get]
  Roomのカスタムプロパティのリードオンリーのキャッシュ。Room.SetCustomProperties （RoomInfoクラスからは利用できません）から設定します。
- string name [get]
  Room の名前。Room／対戦の（1 ロードバランサーグループ内で）固有な識別子。
- int playerCount [get, set]
  （自身がRoomにいない間の）Room内のプレイヤー数を表示するために、内部的にロビーで使用される。
- bool isLocalClientInside [get, set]
  ローカルクライアントがゲーム（Room）内に既にいる、またはゲームサーバー上でゲームに参加しようとしている状態（ロビー内の場合は、常に false。）
- byte maxPlayers [get]
  この Room のプレイヤー数の上限。このプロパティもロビーで表示されます。Roomが満員（プレイヤーの数＝＝maxplayers）の場合、このRoomへの参加は失敗となります。
- bool open [get]
  Roomが参加できるかどうかを定義。これはロビーでのリスト表示には影響を与えませんが、もし open でない場合はRoomへの参加は失敗します。 open でない場合は、そのRoomはランダムマッチメイキングから除外されます。ゲームの状況により、参加する前にRoom／対戦が closed となってしまうこともあります。単にマスターに再接続して、他を見つけるようにしましょう。Roomをリストしないためには、"IsVisible" プロパティを利用してください。
- bool visible [get]
  Roomがロビーでリストされるかどうかを定義。Roomは、「見えない（リストしない）状態」として作成したり、また作成後「見えない状態」に変更したりできます。Roomが参加できるかどうかの変更は、"open" プロパティを利用してください。

13.22.1 解説

リスト表示や参加するために必要な情報だけに簡素化されたRoom。ロビーでのRoomのリスティングに使用します。プロパティ（open や maxPlayersなど）は設定できません
マスターサーバーのロビーから送られてくる、利用可能なRoomの情報を、このクラスは反映しています。すべての値はリードオンリーであると考えてください。どの値も共有／同期されません（サーバーのイベントによってのみ更新されます。）

13.22.2 メンバー関数

13.22.2.1 Equals **override bool RoomInfo.Equals (object p)**

RoomInfo を（名前で）比較します。

13.22.2.2 GetHashCode **override int RoomInfo.GetHashCode ( )**

Equals と伴に使われる。名前のハッシュコードを戻り値として使う。

戻り値

13.22.2.3 ToString　**override string RoomInfo.ToString ( )**
表示出力メソッド

戻り値

RoomInfo を表示する文字列

13.22.3 メンバーデータ

13.22.3.1 autoCleanUpField　**bool RoomInfo.autoCleanUpField = false** [protected]
プロパティのバックフィールド。GameProperty が true に設定（true でない場合は送信されない）されていない限り、false。

13.22.3.2 maxPlayersField　**byte RoomInfo.maxPlayersField = 0** [protected]
プロパティのバックフィールド

13.22.3.3 nameField　**string RoomInfo.nameField** [protected]
プロパティのバックフィールド

13.22.3.4 openField　**bool RoomInfo.openField = true** [protected]
プロパティのバックフィールド

13.22.3.5 visibleField　**bool RoomInfo.visibleField = true** [protected]
プロパティのバックフィールド

13.22.4 プロパティ

13.22.4.1 customProperties　**Hashtable RoomInfo.customProperties** [get]
Room のカスタムプロパティのリードオンリーのキャッシュ。Room.SetCustomProperties （RoomInfo クラスからは利用できません）から設定します。
すべてのキーは文字列型で、値は　ゲーム／アプリケーションにより異なります。

13.22.4.2 isLocalClientInside　**bool RoomInfo.isLocalClientInside** [get], [set]
ローカルクライアントがゲーム（Room）内に既にいる、またはゲームサーバー上でゲームに参加しようとしている状態（ロビー内の場合は、常に false。）

13.22.4.3 maxPlayers　**byte RoomInfo.maxPlayers** [get]
この Room のプレイヤー数の上限。このプロパティもロビーで表示されます。Room が満員（プレイヤーの数＝＝maxplayers）の場合、この Room への参加は失敗となります。
RoomInfo としては、これは設定できません。（プレイヤーが参加済みの）Room としては設定でき、サーバーとすべてのクライアントを設定値で更新されます。

13.22.4.4 name **string RoomInfo.name** [get]

Room の名前。Room／対戦の（1 ロードバランサーグループ内で）固有な識別子。

13.22.4.5 open **bool RoomInfo.open** [get]

Room が参加できるかどうかを定義。これはロビーでのリスト表示には影響を与えませんが、もし open でない場合は Room への参加は失敗します。 open でない場合は、その Room はランダムマッチメイキングから除外されます。ゲームの状況により、参加する前に Room／対戦が closed となってしまうこともあります。単にマスターに再接続して、他を見つけるようにしましょう。Room をリストしないためには、"IsVisible" プロパティを利用してください。

RoomInfo としては、これは設定できません。（プレイヤーが参加済みの）Room としては設定でき、サーバーとすべてのクライアントを設定値で更新されます。

13.22.4.6 playerCount **int RoomInfo.playerCount** [get], [set]

（自身が Room にいない間の）Room 内のプレイヤー数を表示するために、内部的にロビーで使用される。

13.22.4.7 removedFromList **bool RoomInfo.removedFromList** [get], [set]

既にリストされなくなっている Room をマークするために、内部的にロビーで利用される。

13.22.4.8 visible **bool RoomInfo.visible** [get]

Room がロビーでリストされるかどうかを定義。Room は、「見えない（リストしない）状態」として作成したり、また作成後「見えない状態」に変更したりできます。Room が参加できるかどうかの変更は、"open" プロパティを利用してください。

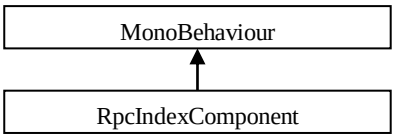
RoomInfo としては、これは設定できません。（プレイヤーが参加済みの）Room としては設定でき、サーバーとすべてのクライアントを設定値で更新されます。

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs

13.23 RpcIndexComponent クラス レファレンス

継承概念図： RpcIndexComponent:



パブリックメンバー関数

- void Awake ()
- byte GetShortcut (string method)

パブリック属性

- string[ ] RpcIndex

13.23.1 メンバー関数

13.23.1.1 Awake **void RpcIndexComponent.Awake ( )**

13.23.1.2 GetShortcut **byte RpcIndexComponent.GetShortcut (string method)**

13.23.2 メンバーデータ

13.23.2.1 RpcIndex **string [ ] RpcIndexComponent.RpcIndex**

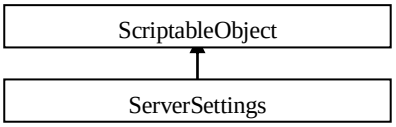
このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs

13.24 ServerSettings クラス レファレンス

PhotonNetwork.ConnectUsingSettings により内部的に利用される接続に関連する設定情報

継承概念図： ServerSettings:



パブリックタイプ

- enum HostingOption { NotSet, PhotonCloud, SelfHosted, OfflineMode }

パブリックメンバー関数

- void UseCloud (string cloudAppid, int regionIndex)
- void UseMyServer (string serverAddress, int serverPort, string application)
- override string ToString ()

静的パブリックメンバー関数

- static int FindRegionForServerAddress (string server)
- static string FindServerAddressForRegion (int regionIndex)
- static string FindServerAddressForRegion (CloudServerRegion regionIndex)

パブリック属性

- HostingOption HostType = HostingOption.NotSet
- string ServerAddress = DefaultServerAddress
- int ServerPort = 5055
- string AppID = ""
- List< string > RpcList
- bool DisableAutoOpenWizard

静的パブリック属性

- static string DefaultCloudServerUrl = "app-eu.exitgamescloud.com"
- static readonly string[ ] CloudServerRegionPrefixes = new string[ ] {"app-eu", "app-us", "app-asia", "app-jp"}
- static string DefaultServerAddress = "127.0.0.1"
- static int DefaultMasterPort = 5055
- static string DefaultAppID = "Master"

13.24.1 解説

PhotonNetwork.ConnectUsingSettings により内部的に利用される接続に関連する設定情報

13.24.2 メンバー列挙型

13.24.2.1 HostingOption **enum ServerSettings.HostingOption**

列挙定数リスト：

NotSet

PhotonCloud

SelfHosted

OfflineMode

13.24.3 メンバー関数

13.24.3.1 FindRegionForServerAddress **static int SerrverSettings.FindRegionForServerAddress (string server)** `[static]`

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

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

13.24.3.4 ToString **override string ServerSettings.ToString ( )**

13.24.3.5 UseCloud **void ServerSettings.UseCloud (string cloudAppid, int regionIndex)**

13.24.3.6 UseMyServer **void ServerSettings.UseMyServer (string serverAddress, int serverPort, string application)**

13.24.4 メンバーデータ

13.24.4.1 AppID **string ServerSettings.AppID = ””**

13.24.4.2 CloudServerRegionPrefixes **readonly string [ ] ServerSettings.CloudServerRegionPrefixes = new string[ ] f”app-eu”, ”app-us”, ”app-asia”, ”app-jp”g** `[static]`

13.24.4.3 DefaultAppID **string ServerSettings.DefaultAppID = ”Master”** `[static]`

13.24.4.4 DefaultCloudServerUrl **string ServerSettings.DefaultCloudServerUrl = ”app-eu.exitgamescloud.com”** `[static]`

13.24.4.5 DefaultMasterPort **int ServerSettings.DefaultMasterPort = 5055** `[static]`

13.24.4.6 DefaultServerAddress **string ServerSettings.DefaultServerAddress = ”127.0.0.1”** `[static]`

13.24.4.7 DisableAutoOpenWizard **bool ServerSettings.DisableAutoOpenWizard**

13.24.4.8 HostType **HostingOption ServerSettings.HostType = HostingOption.NotSet**

13.24.4.9 RpcList **List<string> ServerSettings.RpcList**

13.24.4.10 ServerAddress **string ServerSettings.ServerAddress = DefaultServerAddress**

13.24.4.11 ServerPort **int ServerSettings.ServerPort = 5055**

このクラスのドキュメンテーションは、次のファイルから生成されました：

- Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs

Chapter 14

14. ファイルドキュメンテーション

14.1 Doc/general.md ファイルレファレンス

14.2 Doc/main.md ファイルレファレンス

14.3 Doc/optionalGui.md ファイルレファレンス

14.4 Doc/photonStatsGui.md ファイルレファレンス

14.5 Doc/publicApi.md ファイルレファレンス

14.6 Photon Unity Networking/Plugins/PhotonNetwork/CustomTypes.cs ファイルレファレンス

クラス

- class CustomTypes

Unity 固有の各クラスのシリアライゼーション/デシリアライゼーションメソッドを含む、内部的に利用されるクラス。Photon シリアライゼーションプロトコルにこれらを加えることで、イベントその他で送信できるようになります。

14.7 Photon Unity Networking/Plugins/PhotonNetwork/Enums.cs ファイルレファレンス

列挙型

- enum ConnectionState {
  Disconnected, Connecting, Connected, Disconnecting,
  InitializingApplication }
  クライアントの上層レベルの接続状態。より詳細な PeerState の使用が推奨されます。
- enum PeerState {
  Uninitialized, PeerCreated, Connecting, Connected,
  Queued, Authenticated, JoinedLobby, DisconnectingFromMasterserver,
  ConnectingToGameserver, ConnectedToGameserver, Joining, Joined,
  Leaving, DisconnectingFromGameserver, ConnectingToMasterserver, ConnectedComingFromGameserver,
  QueuedComingFromGameserver, Disconnecting, Disconnected, ConnectedToMaster }
  接続/ネットワークPeerの詳細な状態。PUNでは、ロードバランサーと認証のワークフローは「内部的に」実装されているため、いくつかの状態は自動的に次の状態に進められます。そのような状態には、"(will-change)"(変更になります)というコメントがついています。

- enum PhotonNetworkingMessage {
  OnConnectedToPhoton, OnLeftRoom, OnMasterClientSwitched, OnPhotonCreateRoomFailed,
  OnPhotonJoinRoomFailed, OnCreatedRoom, OnJoinedLobby, OnLeftLobby,
  OnDisconnectedFromPhoton, OnConnectionFail, OnFailedToConnectToPhoton, OnReceivedRoomList-
  Update,
  OnJoinedRoom, OnPhotonPlayerConnected, OnPhotonPlayerDisconnected, OnPhotonRandomJoinFailed,
  OnConnectedToMaster, OnPhotonSerializeView, OnPhotonInstantiate, OnPhotonMaxCccuReached,
  OnPhotonCustomRoomPropertiesChanged, OnPhotonPlayerPropertiesChanged, OnUpdatedFriendList,
  OnCustomAuthenticationFailed }
  この列挙型は、PUNにより送られる MonoMessages のセットを作り上げます。これらの定数の名前をメソッドとして実装すると、それに応じた状況でコールされるようになります。
- enum DisconnectCause {
  ExceptionOnConnect = StatusCode.ExceptionOnConnect, SecurityExceptionOnConnect = StatusCode.-
  SecurityExceptionOnConnect, TimeoutDisconnect = StatusCode.TimeoutDisconnect, DisconnectByClient-
  Timeout = StatusCode.TimeoutDisconnect,
  InternalReceiveException = StatusCode.InternalReceiveException, DisconnectByServer = StatusCode.-
  DisconnectByServer, DisconnectByServerTimeout = StatusCode.DisconnectByServer, DisconnectByServer-
  Logic = StatusCode.DisconnectByServerLogic,
  DisconnectByServerUserLimit = StatusCode.DisconnectByServerUserLimit, Exception = StatusCode.-
  Exception, InvalidRegion = ErrorCode.InvalidRegion, MaxCcuReached = ErrorCode.MaxCcuReached }
  切断の原因を要約します。 OnConnectionFail と OnFailedToConnectToPhoton で使用されます。

14.7.1 列挙型

14.7.1.1 ConnectionState **enum ConnectionState** クライアントの上層レベルの接続状態。より詳細な PeerState の使用が推奨されます。

列挙定数リスト

Disconnected

Connecting

Connected

Disconnecting

InitializingApplication

14.8 Photon Unity Networking/Plugins/PhotonNetwork/Extensions.cs ファイルレファレンス

クラス

- class Extensions

この静的クラスは、他のクラスのための便利な拡張メソッドをいくつか定義しています。(例：Vector3、浮動小数型など)

タイプデフ

- using SupportClass = ExitGames.Client.Photon.SupportClass
-

14.8.1 タイプデフ

14.8.1.1 SupportClass **using SupportClass = ExitGames.Client.Photon.SupportClass**

14.9 Photon Unity Networking/Plugins/PhotonNetwork/FriendInfo.cs ファイルレファレンス

クラス

- class FriendInfo
  オンラインの友達の状態とどのRoomにいるかの情報を格納するために使用されます。

14.10 Photon Unity Networking/Plugins/PhotonNetwork/LoadbalancingPeer.cs ファイルレファレンス

クラス

- class LoadbalancingPeer
  PUN により内部的に使用されます。LoadbalancingPeer は、Photon ロードバランサー(もしくは、Photon Cloud)を使用するために必要となるオペレーションと列挙型 enum の定義を提供します。
- class ErrorCode
  定数のクラス。これらの値(int型) は、Photonのロードバランサーロジックで定義され送信されるエラーコードを表ます。PUNが内部的にこれらの定数を使用します。
- class ActorProperties
  定数のクラス。この値（byte型）は、Actor/Playerの標準のプロパティを定義します。PUNが内部的にこれらの定数を使用します。
- class GameProperties
  定数のクラス。これらの値(byte型)は、Photonのロードバランサーで使用されるRoomおよびゲームの標準のプロパティのためのものです。PUNが内部的にこれらの定数を使用します。
- class EventCode
  定数のクラス。これらの値は、Photonのロードバランサーで定義されたイベントを表します。PUNが内部的にこれらの定数を使用します。
- class ParameterCode
  定数のクラス。オペレーションとイベントのパラメーターのコードです。PUNが内部的にこれらの定数を使用します。
- class OperationCode
  定数のクラス。オペレーション関連のコードが含まれます。PUNが内部的にこれらの定数を使用します。
- class AuthenticationValues
  Photon でのカスタム認証の値 (デフォルト : ユーザーと トークン) のコンテナクラス。接続する前に AuthParameters を設定、その他、認証の値に関するすべてが扱われます。

列挙型

- enum MatchmakingMode : byte { FillRoom = 0, SerialMatching = 1, RandomMatching = 2 }
  OpJoinRandomで使われるマッチメイキングルールのオプション
- enum CustomAuthenticationType : byte { Custom = 0, None = byte.MaxValue }
  Photon で使用されるカスタム認証サービスのオプション。Photon に接続した後に OpAuthenticate で使用されます。

14.10.1 列挙型

14.10.1.1 CustomAuthenticationType **enum CustomAuthenticationType : byte**

Photon で使用されるカスタム認証サービスのオプション。Photon に接続した後に OpAuthenticate で使用されます。

列挙定数リスト：

Custom カスタム認証サービスを使用する。現在、ただ 1 つ実装されているオプション。

None カスタム認証を無効とする。接続(具体的には、OpAuthenticate)のために AuthenticationValues を提供しないのと同じこと。

14.10.1.2 MatchmakingMode **enum MatchmakingMode : byte**

OpJoinRandom で使われるマッチメイキングルールのオプション

列挙定数リスト：

FillRoom できるだけ早くプレイヤーを集め(先に作成されたものから）各 Room を定員一杯にします。デフォルト。MaxPlayers が 0 でない(無制限ではない）場合により多くのプレイヤーが開始に必要なゲームで、最も意味があります。

SerialMatching フィルター条件を含めて、利用できる各 Room にプレイヤーを順番に分配します。フィルター条件が無い場合、各 Room はプレイヤーが均等に分配されます。

RandomMatching (完全に）ランダムに Room に参加します。想定されるプロパティがマッチしなければなりませんが、それ以外は、利用できるどの Room でも選ばれます。

14.11 Photon Unity Networking/Plugins/PhotonNetwork/NetworkingPeer.cs ファイルレファレンス

クラス

- class NetworkingPeer
  PUNで利用される Photonロードバランサーを実装します。このクラスは内部的に利用され、パブリック API となる予定はありません。

14.12 Photon Unity Networking/Plugins/PhotonNetwork/PhotonClasses.cs ファイルレファレンス

クラス

- class PunEvent
  定数のクラス。PUNで使用する photon-event-codes を定義します。
- class Photon.MonoBehaviour
  このクラスは、photonViewプロパティを追加します。また、ゲーム内で networkViewがまだ使用されている場合は、警告をログに出します。
- class PhotonMessageInfo
  特定のメッセージ、RPC/更新情報に関する情報のコンテナのクラス。
- class PBitStream
- class PhotonStream
  OnPhotonSerializeView により書かれたデータを保持するために使うコンテナクラス

ネームスペース

- package Photon

列挙型

- enum PhotonTargets {
  All, Others, MasterClient, AllBuffered,
  OthersBuffered }
  RPCの「ターゲット(target）」オプションの列挙型。どのリモートクライアントがその RPC コールを受け取るかを定義します。
- enum PhotonLogLevel { ErrorsOnly, Informational, Full }
  PUN のクラスにより作られるログ出力レベルの定義に使われます。記録するのは、エラーだけ(ErrorsOnly）、エラーとより詳しい情報（Informational）、すべて(Full)のいずれかです。

14.13 Photon Unity Networking/Plugins/PhotonNetwork/PhotonHandler.cs ファイルレファレンス

クラス

- class PhotonHandler
  Photon に Update ループを実行させる、内部的 Monobehaviour

14.14 Photon Unity Networking/Plugins/PhotonNetwork/PhotonLagSimulationGui.cs ファイルレファレンス

クラス

- class PhotonLagSimulationGui
  この MonoBehaviour のサブクラスは、Photon クライアントでのネットワークシミュレーション機能のための基本的な GUI です。ラグ(lag、固定値の遅延)、ジッター(jitter、ランダム値の遅延) そしてパケットの損失などを変更できます。

14.15 Photon Unity Networking/Plugins/PhotonNetwork/PhotonNetwork.cs ファイルレファレンス

クラス

- class PhotonNetwork
  PhotonNetwork プラグインを使用するためのメインのクラス。これは静的クラスです。

タイプデフ

- using Debug = UnityEngine.Debug
-

14.15.1 タイプデフ

14.15.1.1 Debug **using Debug = UnityEngine.Debug**

14.16 Photon Unity Networking/Plugins/PhotonNetwork/PhotonPlayer.cs ファイルレファレンス

クラス

- class PhotonPlayer
  Room内で actorID により識別される任意の 1 プレイヤーについてまとめています。

14.17 Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsGui.cs ファイルレファレンス

クラス

- class PhotonStatsGui
  Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab によりトグルします。

14.18 Photon Unity Networking/Plugins/PhotonNetwork/PhotonStatsTracker.cs ファイルレファレンス

クラス

- class PhotonStatsTracker
  Photon への接続に関するトラフィックや健全性の統計値を表示するための基本GUI。shift+tab により、トグルします。

タイプデフ

- using Debug = UnityEngine.Debug

14.18.1 タイプデフ

14.18.1.1 Debug **using Debug = UnityEngine.Debug**

14.19 Photon Unity Networking/Plugins/PhotonNetwork/PhotonView.cs ファイルレファレンス

クラス

- class PhotonView
  NetworkView に代わる、PUN でネットワークに利用されるクラス。 NetworkView と同じように使用してください。

列挙型

- enum ViewSynchronization { Off, ReliableDeltaCompressed, Unreliable }
- enum OnSerializeTransform {
  OnlyPosition, OnlyRotation, OnlyScale, PositionAndRotation,
  All }
- enum OnSerializeRigidBody { OnlyVelocity, OnlyAngularVelocity, All }

14.19.1 列挙型

14.19.1.1 OnSerializeRigidBody **enum OnSerializeRigidBody**

列挙定数リスト：

OnlyVelocity

OnlyAngularVelocity

All

14.19.1.2 OnSerializeTransform **enum OnSerializeTransform**

列挙定数リスト：

OnlyPosition

OnlyRotation

OnlyScale

PositionAndRotation

All

14.19.1.3 ViewSynchronization **enum ViewSynchronization**

列挙定数リスト：

Off

ReliableDeltaCompressed

Unreliable

14.20 Photon Unity Networking/Plugins/PhotonNetwork/PingCloudRegions.cs ファイルレファレンス

クラス

- class PingCloudRegions
  このスクリプトは、PUNにより自動的に PhotonHandler ゲームオブジェクトに追加されます。(世界中の）各地域 ExitGames Cloud に、Awakeから自動的に ping (接続確認を)します。 これはクライアント毎に１度だけ実行され、結果は PlayerPrefs に保存されます。最良の接続状態の地域 ExitGames Cloud に接続するために、PhotonNetwork.ConnectToBestCloudServer(gameVersion) を利用してください。

14.21 Photon Unity Networking/Plugins/PhotonNetwork/Room.cs ファイルレファレンス

クラス

- class Room
  このクラスは、PUN が参加する(している）Roomをそのまま反映します。RoomInfo のプロパティとは違い、このプロパティは設定可能であり、「自分の」Room を閉じたり、隠したりできます。

14.22 Photon Unity Networking/Plugins/PhotonNetwork/RoomInfo.cs ファイルレファレンス

クラス

- class RoomInfo
  リスト表示や参加するために必要な情報だけに簡素化されたRoom。ロビーでのRoomのリスティングに使用します。プロパティ(open や maxPlayersなど)は設定できません。

14.23 Photon Unity Networking/Plugins/PhotonNetwork/RpcIndexComponent.cs ファ

イルレファレンス

クラス

- class RpcIndexComponent

14.24 Photon Unity Networking/Plugins/PhotonNetwork/ServerSettings.cs ファイルレファレンス

クラス

- class ServerSettings
  PhotonNetwork.ConnectUsingSettings により内部的に利用される接続に関連する設定情報

列挙型

- enum CloudServerRegion { EU, US, Asia, Japan }
  現時点で利用可能な Photon Cloudの地域が列挙されている。

14.24.1 列挙型

14.24.1.1 CloudServerRegion **enum CloudServerRegion**

現時点で利用可能な Photon Cloud の地域が列挙されている。

CloudServerRegionNames および CloudServerRegionPrefixes の順番と一致していなければなりません。

列挙定数リスト：

EU

US

Asia

Japan

索引

P

Q

R

S

T

U

V

W