photon

Search

This document is about: **QUANTUM 2**

SWITCH TO

Online Session

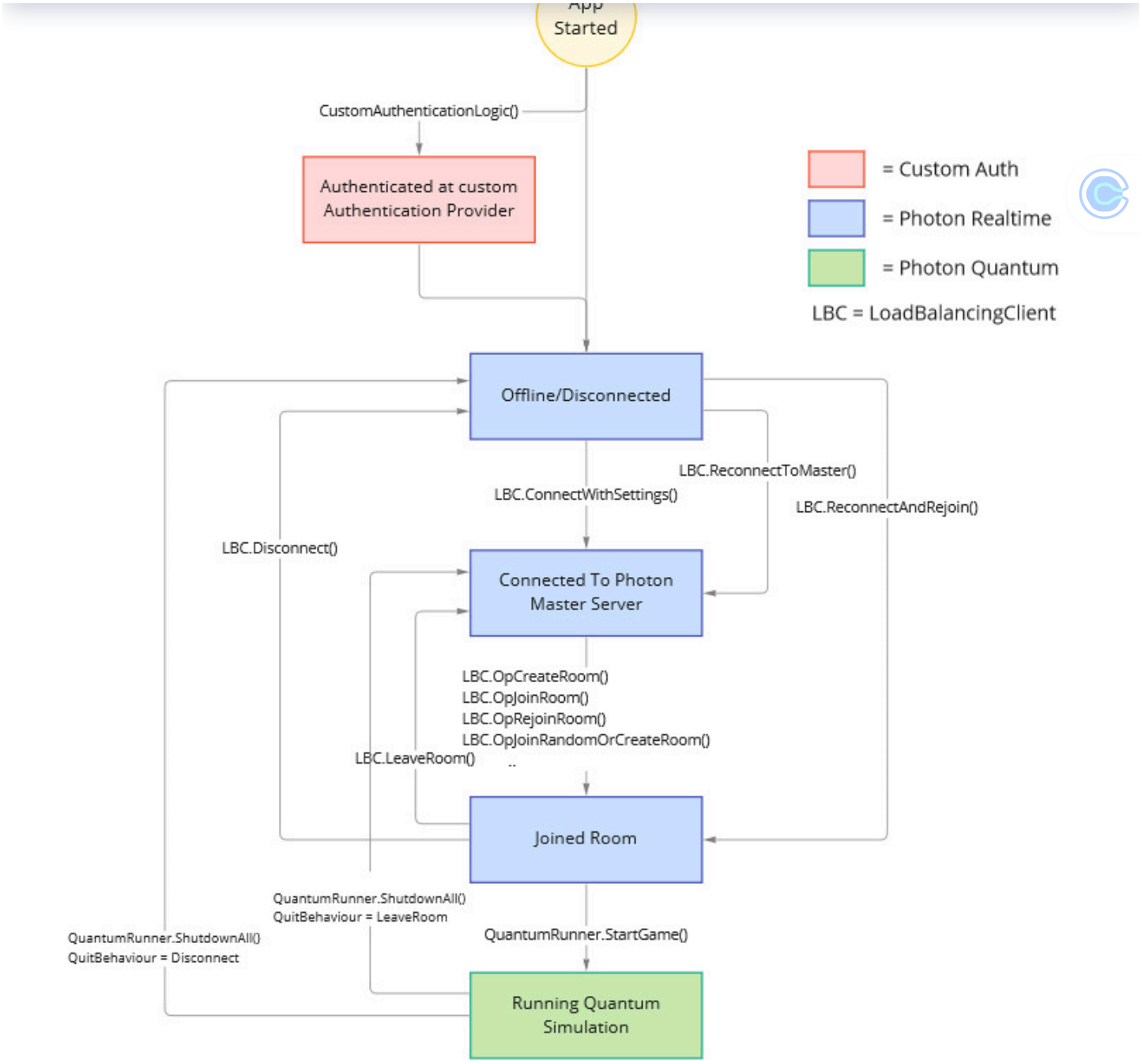
Overview

The Quantum online services are build on top of the common Photon online infrastructure (Photon Realtime). Connecting to an online session usually goes through three connection phases:

1. **Custom Authentication**: Photon does not offer player accounts and recommends securing the logins using a proprietary or third-party authentication provider and set up Photon Custom Authentication.
2. **Game Server Connection**: Before starting the online simulation the clients have to connect to the Photon Cloud and enter a Room using the Photon Realtime library. The process, including rudimentary random matchmaking, is described here, inside the `DemoMenu` from the SDK and the official Photon Realtime doc Photon Matchmaking And Lobby.
3. **Quantum Simulation Start Sequence**: In this phase the Quantum simulation is started, client configuration data is send and the Quantum session is joined and synchronized between clients.

How To Start A Quantum Online Session

The diagram shows the typical connection flow to give an overview of the connection handling required to start, restart and stop an online Quantum game.



Starting Online Session: Connecting With Photon

Further Readings

- Photon Realtime Quick Start
- Quantum Reconnection Manual

Reduce Allocation in Photon Realtime

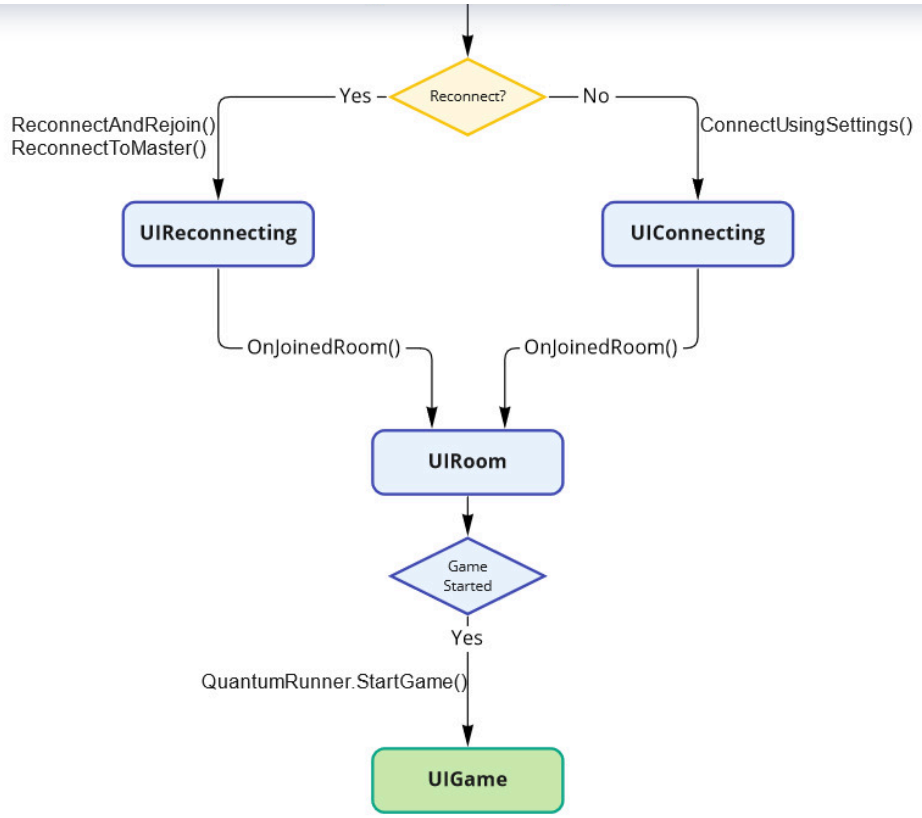
C#

```
LoadBalancingPeer.ReuseEventInstance = true;
```

The Demo Menu

The `DemoMenu` (included in the SDK) demonstrates the complete process. To better understand what's going on the individual UI classes (`UIConnect`, `UIConnecting`, `UIReconnecting`, `UIRoom`, `UIGame`) should be seen as a state machine. The similarities to the diagram above are visible.

- The player presses a button to connect to the Photon Cloud and uses random matchmaking to finally join a Photon Room.
- After waiting for other players pressing a button signals the game start and the players each will start their Quantum sessions.
- The player can press disconnect, stop the Unity editor or restart the application to then use the reconnect button to get back to the same game session.



Demo Menu Flowchart

UIConnect

There are a few extras that are supposed to make the DemoMenu a convenient development tool to start with but also make the code harder to read. For example the `RegionDropdown` and `AppVersionDropdown`.

RegionDropdown

There is a scriptable asset (`Photon/QuantumDemo/Menu/Resources/PhotonRegions.asset`) that contains a set of regions selectable by dropdown to quickly change the region when running the menu. The `LastSelectedRegion` is kept inside the `PlayerPrefs`.

AppVersionDropdown

The `AppVersion` (which is set on `AppSettings` when connecting) is a way to separate the users when performing matchmaking. This can be used for live games that use the same AppId but also during testing/development. The implementation covers two cases:

- Developers can run the app multiple times on their own machine and not want any others joining their games. For this purpose, it is recommended to use the `Private AppVersion` which uses a

- For a focus test or QA tests players should join certain groups. Names listed in the `Photon/QuantumDemo/Menu/Resources/PhotonAppVersions.asset` will be selectable in the drop down list of the menu.

AppSettings

To connect to the Photon Cloud the configuration called `AppSettings` is required. Inside the Quantum integration this is usually accessed via a Singleton pattern: `PhotonServerSettings.Instance.AppSettings` which of course can be exchanged with another way of injecting the correct data.

Notice that the settings object is copied. Otherwise there would a risk of saving changes to the asset.

C#

```
var appSettings = PhotonServerSettings.CloneAppSettings(PhotonSer
```

QuantumLoadBalancingClient

The one `instance` of the connection class `QuantumLoadBalancingClient` is stored as a static reference in `UIMain.Client`. UI development has lot's of flavors and the Singleton there and state machine here are the simplest way and should challenge developers to change the menu flow to their way of creating UI.

The `QuantumLoadBalancingClient` exists only to assign a `Nickname` more conveniently and to cache the `BestRegionSummaryFromStorage` (region ping results) to `PlayerPrefs`. To start Quantum only the inherited class is `LoadBalancingClient` is required.

ConnectUsingSettings

The Demo Menu does not use any authentication process instead is will just connect to the Photon Cloud which will generate a user id for the client.

`ConnectUsingSettings()` will use the mentioned `AppSettings` enhanced by Region and AppVersion selection to connect to the Photon Master Server. `HideScreen()` and `UIConnecting.ShowScreen()` are used to issue the state progression.

```
if (UIMain.Client.ConnectUsingSettings(appSettings, Username.text
    HideScreen();
    UIConnecting.ShowScreen();
}
```

OnReconnectClicked

The sample demonstrates three reconnection scenarios:

- The connection object (`UIMain.Client`) is valid and `PlayerTtlInSeconds` is set

The client may still be considered online in which case it can try to "fast reconnect" using `UIMain.Client.ReconnectAndRejoin()` . It is not necessary to check for `PlayerTtlInSeconds` here the reconnection has to be inside a 10 second timeout before the server removes the clients residue from the room and fast reconnect is not possible anymore.

`ReconnectAndRejoin()` would transport the client back into their original room.

- The connection object (`UIMain.Client`) is valid but the client hat been offline for more than 10 sec (or `PlayerTtlInSeconds`)

The client will reconnect to the master server `UIMain.Client.ReconnectToMaster()` and form there join back into the room (room info saved on `ReconnectInformation`).

- The connection object is lost probably because of an app restart

Data to reconnect is saved in Unity's `PlayerPrefs` using `ReconnectInformation` . Relevant data to cache is `RoomName` , `Region` , `AppVersion` , `UserId` (not if the custom authentication runs again any way) and finally the `Timeout` : when the saved data is not worth to reconnect to.

`UIMain.Client` and `AppSettings` is reconfigured then the connection to the master server is opened and a rejoin or join is executed. Rejoin only works when the UserId is the same.

C#

```
        UIMain.Client = new QuantumLoadBalancingClient(PhotonServerSe
        UIMain.Client.UserId = ReconnectInformation.Instance.UserId;

        var appSettings = PhotonServerSettings.CloneAppSettings(Photo
        appSettings.FixedRegion = ReconnectInformation.Instance.Regio
        appSettings.AppVersion = ReconnectInformation.Instance.AppVer

        if (UIMain.Client.ConnectUsingSettings(appSettings, LastUsern
            HideScreen();
            UIReconnecting.ShowScreen();
        }
    }
```

Further Readings

- Quantum Reconnection Manual
- Photon Realtime Analyzing Disconnects

UIConnecting

The state will listen to Photon connection callbacks `IConnectionCallbacks` and `IMatchmakingCallbacks` to progress the connection and to perform error handling.

OnConnectedToMaster

Once the connection to the master server is successful the random matchmaking is initiated right away and a `OpJoinRandomRoomParams` object is created. In this example the `CustomRoomProperties` are used to negotiate the map selection which can be changed after entering the game room.

- `RoomOptions.IsVisible` indicates that the room is open for matchmaking
- `RoomOptions.MaxPlayers` normally reflects the same number as Quantum players
- `RoomOptions.Plugins` has to be `new string[] { "QuantumPlugin" }`
- `RoomOptions.PlayerTtl` and
- `RoomOptions.EmptyRoomTtl` reflect the settings on the PhotonServerSettings

C#

```
_enterRoomParams = new EnterRoomParams();
_enterRoomParams.RoomOptions = new RoomOptions();
_enterRoomParams.RoomOptions.IsVisible  = true;
_enterRoomParams.RoomOptions.MaxPlayers = Input.MAX_COUNT;
_enterRoomParams.RoomOptions.Plugins    = new string[] { "Quantum
_enterRoomParams.RoomOptions.CustomRoomProperties = new Hashtable
    { "HIDE-ROOM", false },
    { "MAP-GUID", defaultMapGuid },
};
_enterRoomParams.RoomOptions.PlayerTtl = PhotonServerSettings.Ins
_enterRoomParams.RoomOptions.EmptyRoomTtl = PhotonServerSettings.

if (!UIMain.Client.OpJoinRandomOrCreateRoom(joinRandomParams, _en
    UIMain.Client.Disconnect();
}
```

`OpJoinRandomOrCreateRoom()` initiates the random matchmaking and the connection to the game server.

OnJoinRandomFailed

There is a way to mitigate `ErrorCode.NoRandomMatchFound` by just creating a new room.

OnJoinedRoom

When successfully joined a room the state progresses to `UIRoom`.

Other error callbacks report and show a dialog which disconnects the client and returns the players to the main menu.

UIReconnecting

OnConnectedToMaster

When running `ReconnectAndRejoin()` this is skipped and `OnJoinedRoom()` is called directly.

In the other cases of reconnecting two paths are possible:

The different is that `Rejoin` requires the client to be still considered to be active on the server (based on the 10 second timeout and the addition PlayerTTL). Both will have error handling in `OnJoinRoomFailed()`.

OnJoinedRoom

Success, continue with the `UIRoom` state.

OnJoinRoomFailed()

Two errors are very common to handle here:

- `ErrorCode.JoinFailedFoundActiveJoiner`: Tried to `join` but the client is still marked active in the room (the server does not know that it disconnected). Mitigation: retry until 10 seconds are over.
- `ErrorCode.JoinFailedWithRejoinerNotFound`: Tried to `rejoin` but there is not client marked active in the room. Mitigation: join normally.

UIRoom

In addition to matchmaking and connection callbacks UIRoom listens to `IInRoomCallbacks` and `IOnEventCallback`.

This is the state before the Quantum simulation where clients are already in the Photon Room. The sample uses room properties to communicate some game configuration like the map selection. Only the master client can change the settings.

To start a Quantum session usually a `RuntimeConfig` is required which includes custom settings for each game. By default it can be changed inside the Menu.scene (Menu > RuntimeConfig).

Start The Game

The sample uses the Photon communication tool `OpRaiseEvent(UIMain.PhotonEventCode.StartGame)` to communicate the starting of the game which is then dispatched by all client in `OnEvent()`.

C#

```
  switch (photonEvent.Code) {
    case (byte)UIMain.PhotonEventCode.StartGame:
```

For client coming after the start or are rejoining the information that the game has started is saved in the room properties:

C#

```
var ht = new ExitGames.Client.Photon.Hashtable {{"STARTED", true}
UIMain.Client.CurrentRoom.SetCustomProperties(ht);
```

All clients returning to the room initially check if the game has already started.

C#

```
UIMain.Client.CurrentRoom.CustomProperties.TryGetValue("MAP-GUID"
UIMain.Client.CurrentRoom.CustomProperties.TryGetValue("STARTED",
```

Starting or restarting the Quantum session is done inside `StartQuantumGame()`.

Initially the RuntimeConfig is copied using `FromByteArray(ToByteArray())` to not accidentally write on the source and a map guid is set.

C#

```
var config = RuntimeConfigContainer != null ? RuntimeConfig.FromB
config.Map.Id = mapGuid;
```

The `StartParameters` are configured. There are different settings when joining as a spectator and for re-joiners there is an optional local snapshot that can be send (the recoding happens in UIGame).

```
var param = new QuantumRunner.StartParameters {
RuntimeConfig              = config,
DeterministicConfig        = DeterministicSessionConfigAsset.Insta
ReplayProvider             = null,
GameMode                   = Spectate ? Photon.Deterministic.Deter
FrameData                  = IsRejoining ? UIGame.Instance?.FrameS
InitialFrame               = IsRejoining ? (UIGame.Instance?.Frame
PlayerCount                = UIMain.Client.CurrentRoom.MaxPlayers,
LocalPlayerCount           = Spectate ? 0 : 1,
RecordingFlags             = RecordingFlags.None,
NetworkClient              = UIMain.Client,
StartGameTimeoutInSeconds = 10.0f
};
```

The `clientId` used to start the Quantum game is usually the Photon `UserId`. It is important to be the same for reconnecting players in order to be assigned to the same Quantum player slot. For testing proposes the `ClientIdProvider` script located in the menu scene (`Menu > UICanvas > Menu > IdProvider`) can be configured to use different sources.

C#

```
var clientId = ClientIdProvider.CreateClientId(IdProvider, UIMain
QuantumRunner.StartGame(clientId, param);
```

After calling `QuantumRunner.StartGame()` the Quantum starting sequence is executed automatically. It will also trigger the map scene loading configured in the `SimulationConfig`. The demo menu state transitions to `UIGame`.

UIGame

The game UI state handles:

- Displaying and listening to a disconnect button
- Recording snapshot on the `OnDisconnected()` callback

photon

Testing Disconnects

Inside `OnLeaveClicked()` change `UIMain.Client.Disconnect()` to something else to simulate slightly more natural disconnects or exceptions in the networking thread.

C#

```
public void OnLeaveClicked() {
    UIMain.Client.Disconnect();
    // Debugging: use these instead of UIMain.Client.Disconnect()
    //UIMain.Client.SimulateConnectionLoss(true);
    //UIMain.Client.LoadBalancingPeer.StopThread();
}
```

Quantum Start Sequences

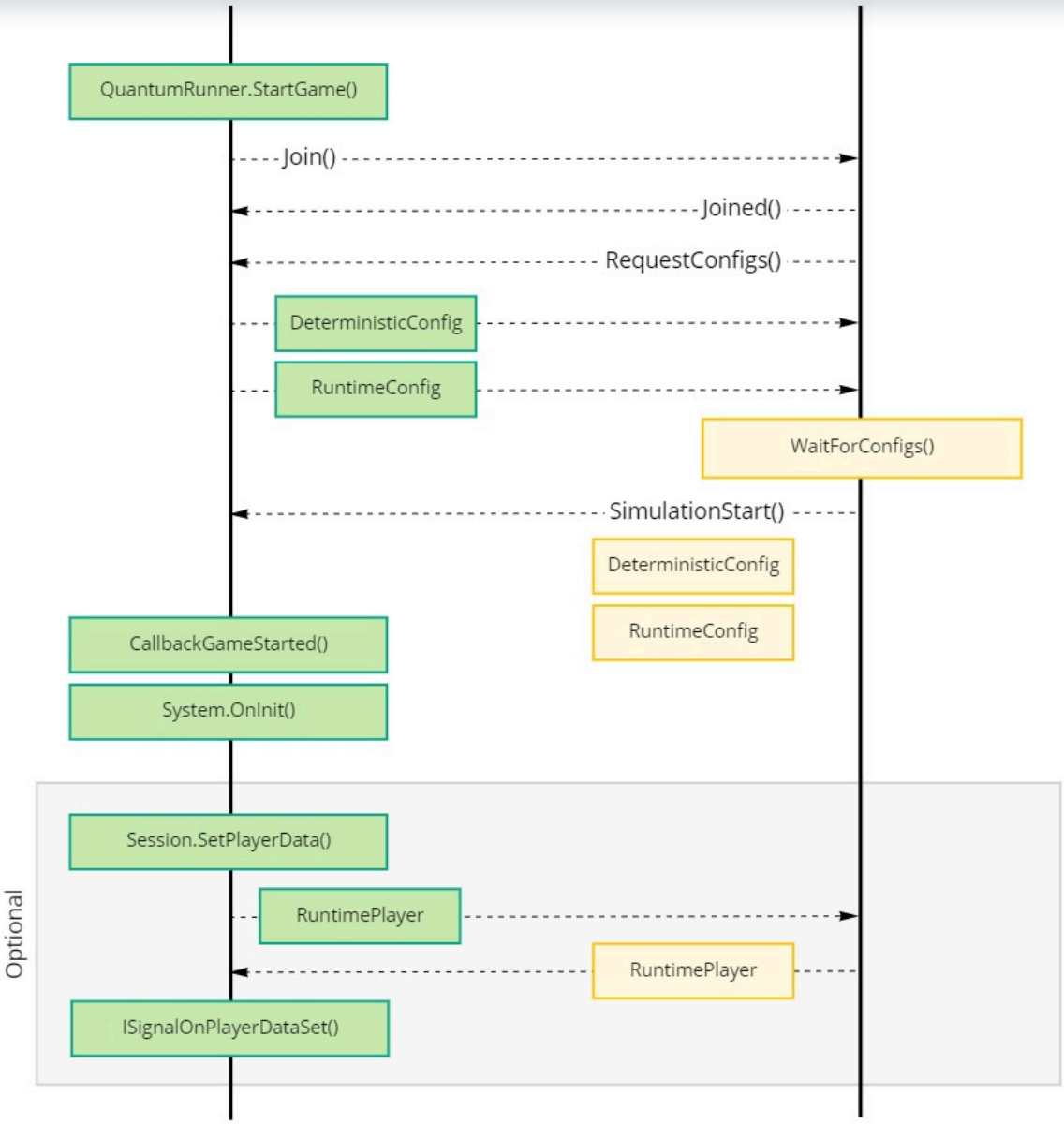
What happens after `QuantumRunner.StartGame()` is called? The following diagram visualizes the protocol and order of callbacks.

Starting A Quantum Online Session

The client joins the server session and is selected to upload the `DeterministicConfig` and `RuntimeConfig` (can be overwritten on the server authoritatively by running an enterprise Quantum server). The server uses the first configs it received and progresses to the simulation start while down-streaming the accepted configs to all clients.

On the client the `GameStarted` callback is invoked followed by `OnInit()` on all systems.

Clients optionally run `SetPlayerData()` to upload their `RuntimePlayer` data, which in turn is returned by the server resulting in `OnPlayerDataSet` signal for everyone.



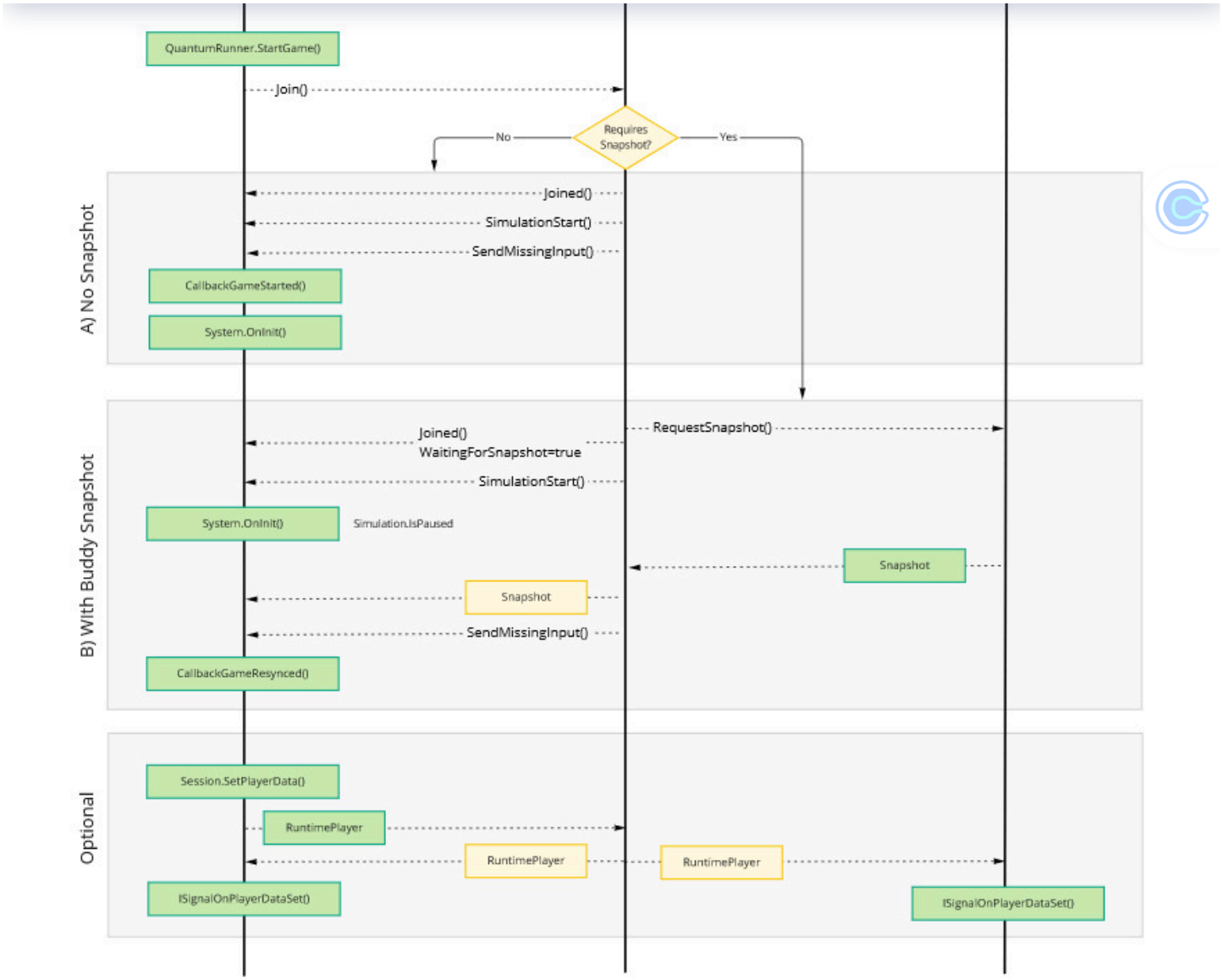
Online Session: Start Sequence

Reconnecting Into A Quantum Online Session

During the reconnection or late-joining sequence the protocol changes if a snapshot has to be send to the client. Read the Reconnection Manual for more details about snapshots timings.

For buddy-snapshots another client is tasked with uploading a recent snapshot. In the meantime the reconnecting client is signaled a successful start and `OnInit()` runs for the systems. Once the snapshot has been received the `GameResynced` callback is executed and input is coming in to catch up the last "second".

If desired `SetPlayerData()` can be invoked again.



Online Session: Restart Sequence

Further Readings

- Quantum Cheat Protection Manual
- Quantum Reconnection Manual
- Quantum Config Files

Stopping The Session And Disconnecting

To stop the Quantum simulation locally run `QuantumRunner.ShutdownAll(bool immediate)`. Only set `immediate:true` when it's **not** called from within a Quantum callback. When set to `false` the shutdown is postponed until the next Unity update.

what is set as `StartParameters.QuitBehaviour`.

If the client should exit the game gracefully, for example to clean up the player avatar for remote clients, extra logic has to be implemented into the simulation. Either a client issued command or monitoring the player connected state (see `PlayerConnectedSystem`).

Considering that players also close their app or Alt+F4 their games there might not always be an opportunity to send a graceful disconnect.

Back to top

photon

Public Discord
YouTube
Facebook
Twitter
Blog
Contact Us

English
日本語
한국어
简体中文
繁体中文