Photon

PhotonNetwork.ConnectUsingSettings(); //Connect to Master Photon Server.

Callback Function: OnConnectedToMaster

*public* *override* void OnConnectedToMaster()

    {

        Debug.Log("Connected To Master Server");

        battleButton.SetActive(true);

    }

Callback Fucntion: OnConnected

    public override void OnConnected()

    {

        Debug.Log("Connected To Internet");

    }

Setting Player Name in Photon Network.

PhotonNetwork.NickName = PlayerName/\*PlayerName must be string. \*/;

PhotonNetwork.IsConnected

PhotonNetwork.IsConnected 🡪 False until you connected to Photon initially. True immediately after Connect-call, in offline mode, while connected to any server and even while switching servers.

Lobbies 🡪 Place where rooms are organized.

There exist default lobby.

In a lobby, the clients only get the room list of that lobby when applicable. Nothing else. There is no way to communicate with others in a lobby.

When a client is joined to a lobby and tries to create (or JoinOrCreate) a room without explicitly setting a lobby, if the creation succeeds/happens, the room will be added to the currently joined lobby. When a client is not joined to a lobby and tries to create (or JoinOrCreate) a room without explicitly setting a lobby, if the creation succeeds/happens, the room will be added to [the default lobby](https://doc.photonengine.com/en-us/pun/v2/lobby-and-matchmaking/matchmaking-and-lobby#the_default_lobby). When a client is joined to a lobby and tries to create (or JoinOrCreate) a room by explicitly setting a lobby, if the creation succeeds/happens:

Join Random Room

PhotonNetwork.JoinRandomRoom();

Callback Function: OnJoinRandomFailed(short returnCode, string message)

*public* *override* void OnJoinRandomFailed(short returnCode, string message)

{

     Debug.Log("Failed To Join Random Room");

 }

Create Rooms

void CreateRoom()

    {

int RoomName = Random.Range(0, 10000);

*RoomOptions* roomOptions = new *RoomOptions*(){IsVisible=true, IsOpen=true, MaxPlayers=10};

      PhotonNetwork.CreateRoom("Room"+RoomName, roomOptions);

}

*/\* IsVisible: Defines if this room is listed in the lobby. If not, it also is not joined randomly.*

*A room that is not visible will be excluded from the room lists that are sent to the clients in lobbies.*

*An invisible room can be joined by name but is excluded from random matchmaking. Use this to "hide" a room and simulate "private rooms".*

*Players can exchange a roomname and create it invisble to avoid anyone else joining it.\*/*

*/\* Defines if this room can be joined at all.*

*If a room is closed, no player can join this.*

*As example this makes sense when 3 of 4 possible players start their gameplay early and don't want anyone to join during the game.*

*The room can still be listed in the lobby (set isVisible to control lobby-visibility).\*/*

Callback Function: Failed Room Creation

*public* *override* void OnCreateRoomFailed(short returnCode, string message)

    {

        Debug.Log("Room Creation failed. Room already exist.");

        CreateRoom();

    }

Callback Function: OnPlayerEnteredRoom (When remote player joins the room this callback function is executed.)

    public override void OnPlayerEnteredRoom(Player newPlayer)

    {

        Debug.Log(newPlayer.NickName + ": joined the room" + " Player Count: " +  PhotonNetwork.CurrentRoom.PlayerCount);

    }

PhotonNetwork.AutomaticallySyncScene = true;

All clients in the room will automatically load the same level as master client.

Clients will load the same scene as the Master Client.

Master Client should use PhotonNetwork.LoadLevel to synchronize the loaded level because it notifies other clients before starting to load the scene.

If Master Client loads a level directly using Unity API, PUN will notify to other players after the scene loading completed.

Callback Function: Leave Room

*public* void OnCancelButtonClicked()

    {

        cancelButton.SetActive(false);

        battleButton.SetActive(true);

        PhotonNetwork.LeaveRoom();

    }

/\* PhotonNetwork.LeaveRoom()

Leave the current room and return to the Master Server where you can join or create rooms (see remarks).

This will clean up all (network) GameObjects with a PhotonView, unless you changed autoCleanUp to false. Returns to the Master Server. In OfflineMode, the local "fake" room gets cleaned up and OnLeftRoom gets called immediately. In a room with playerTTL < 0, LeaveRoom just turns a client inactive. The player stays in the room's player list and can return later on. Setting becomeInactive to false deliberately, means to "abandon" the room, despite the playerTTL allowing you to come back. In a room with playerTTL == 0, become inactive has no effect (clients are removed from the room right away).

\*/

Callback Function: OnJoinedRoom

*public* *override* void OnJoinedRoom()

    {

        Debug.Log("Joined Room Successfully");

    }

Instantiate Gameobjects in Photon Network Game.

\*Remember Gameobject should be placed within Resources folder so Photon can Access it.

\*GameObject must have PhotonView component attached to get instantiated.

PhotonNetwork.Instantiate(“GameObject name”, new Vector3(randomPoint, 0, randomPoint), Quaternion.identity);

PhotonView

*private* *PhotonView* photonView; *//Used To do RPC calls*

*/\**A PhotonView identifies an object across the network (viewID) and configures how the controlling client updates remote instances *\*/*

[PunRPC]

Remote Procedure Calls are exactly what the name implies: Method-calls on remote clients (in the same room)

[PunRPC] is a flag. When it is applied a function then it means function is meant to be called on remote cilents.

**Showing List Of Rooms in the Lobby**

Step 1: Important thing bcoz we have to create the lobby and then only we can see list of rooms.

    public void OnShowRoomListButtonClick()

    {

        if (!PhotonNetwork.InLobby)

        {

            PhotonNetwork.JoinLobby();

        }

}

Step 2: Photon Callbacks

This will be called when a new room is created or a room is deleted.

    public override void OnRoomListUpdate(List<RoomInfo> roomList)

    {

        foreach (RoomInfo room in roomList)

        {

            Debug.Log(room.Name);

            if (!room.IsOpen || !room.IsVisible || room.RemovedFromList)

            {

                if (cachedRoomList.ContainsKey(room.Name))

                {

                    cachedRoomList.Remove(room.Name);

                }

            }

        }

    }