ObjectNet

Table of contents

ObjectNet

EASIER MULTIPLAYER SYSTEM

Thank you for your purchase of the most powerful and flexible multiplayer solution on the Asset Store.

ObjectNet was designed to be easy and simple, nonetheless, is at the same time highly powerful and can be used on any type of project.

We are very proud of our product, and if you have any comments or suggestions feel free to contact us by email admin@onlineobject.net

Regards

***** Note *****
If you are using AD-Blocks the content index may not work properly.

OnlineObject Team

About

ObjectNet is an event solution based, created to make it easy to create and maintain multiplayer solutions.

The core engine used by **ObjectNet** was designed to be agnostic, this means that **ObjectNet** can work independently from game or application logic. Your game or application could be designed for any purpose and **ObjectNet** shall work seamlessly.

Created by a team with decades of experience in network applications and software communication, **ObjectNet** is pretty well-designed and reliable.

System requirements

ObjectNet requirements will depend on the project where he will be used, in short, **ObjectNet** uses a few resources from the system. Nonetheless, depending on how your game was designed, this could mean more or less computation and memory.

A game with a few elements to be synchronized can run on ordinary hardware, on the other hand, how many elements you need to synchronize, more CPU systems will be required.

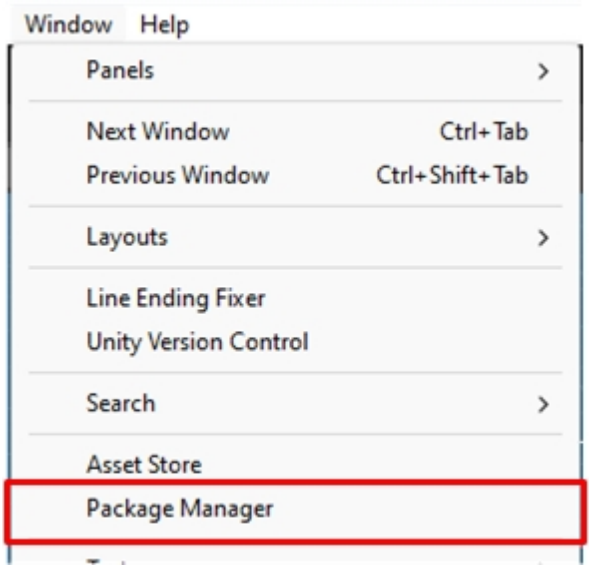
ObjectNet has a pretty smart system to detect and send events only when *changes occur, this reduces the amount of data and computation required to keep things up to date.

**Smart detection system only applies to events and data controlled by ObjectNet, for events raised or sent manually by the developer, ObjectNet will not take and measure.*

Installing

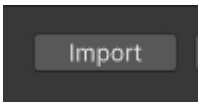
This topic will cover how to install **ObjectNet** on your project.

1. After acquiring the **ObjectNet** go to the Unity package manager.

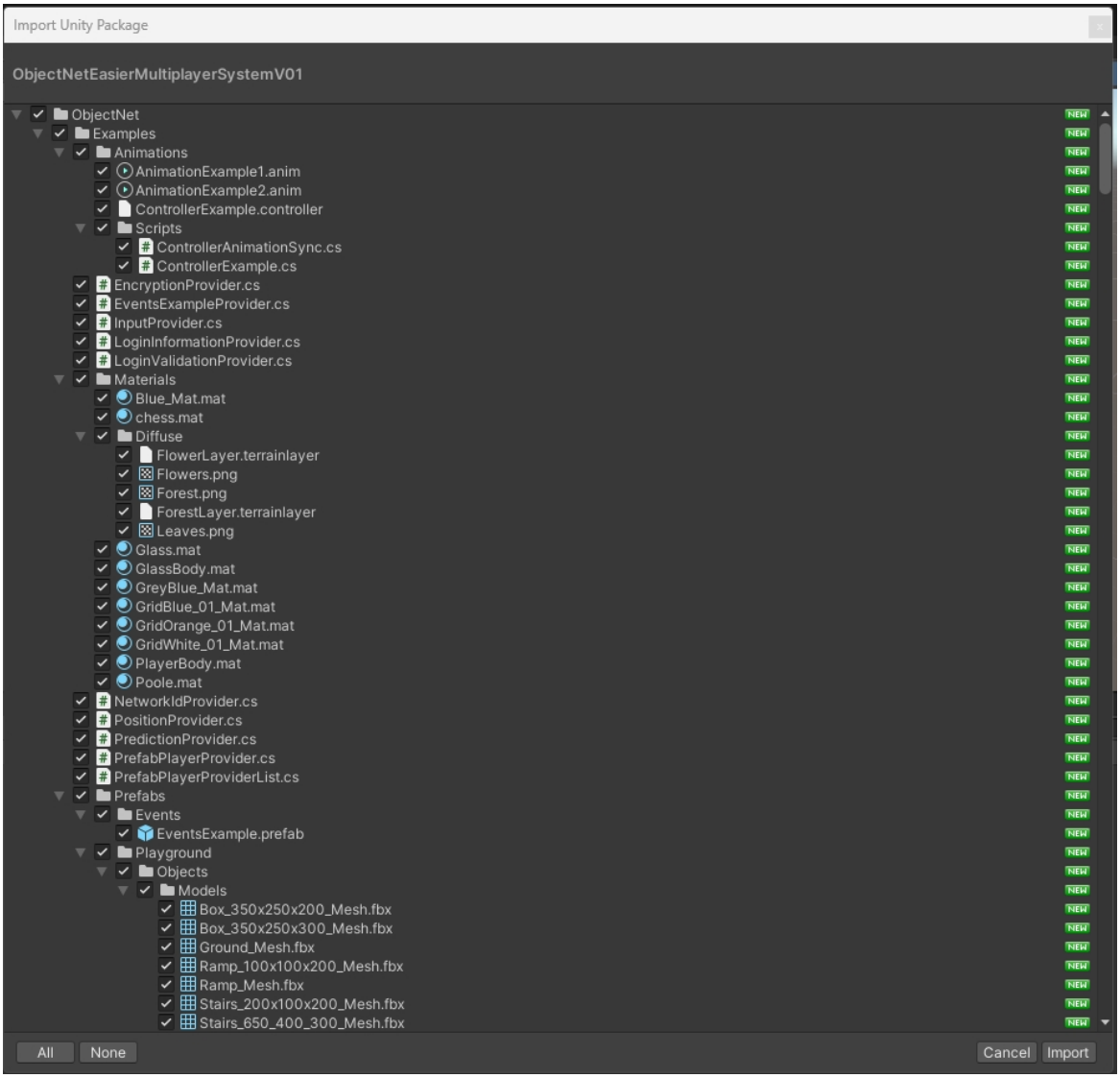


2. Locate **ObjectNet** package and click to download

3. After download was finished click on import



4. On the opened window got to the right bottom corner and press Import



Done, you have already **ObjectNet** on your project.

First Steps

Step by Step

This session is intended to be the step by step guide explaining how to start with **ObjectNet**, nonetheless, it's highly recommended to check the entirely manual to getting a more detailed overview about **ObjectNet**.

We recommend to start a new project only to having a fresh start and understand the principles of **ObjectNet**, after having a new project started yo can got to the Step 1 - Installing.

Step 1 - Installing

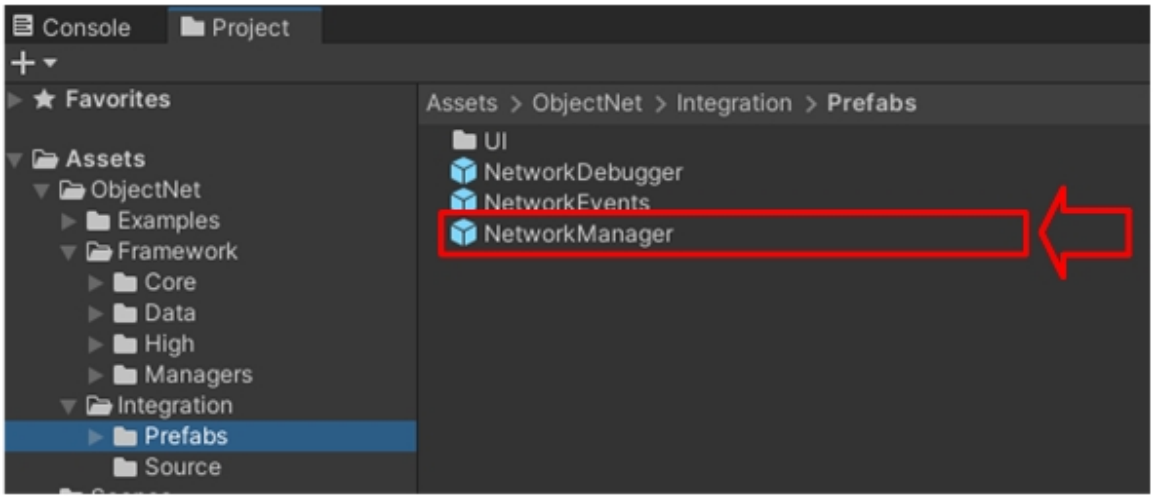
Check hot to install ObjectNet on the section Installing, and after installation, you may proceed to the next Step.

Step 2 - Configuring Scene

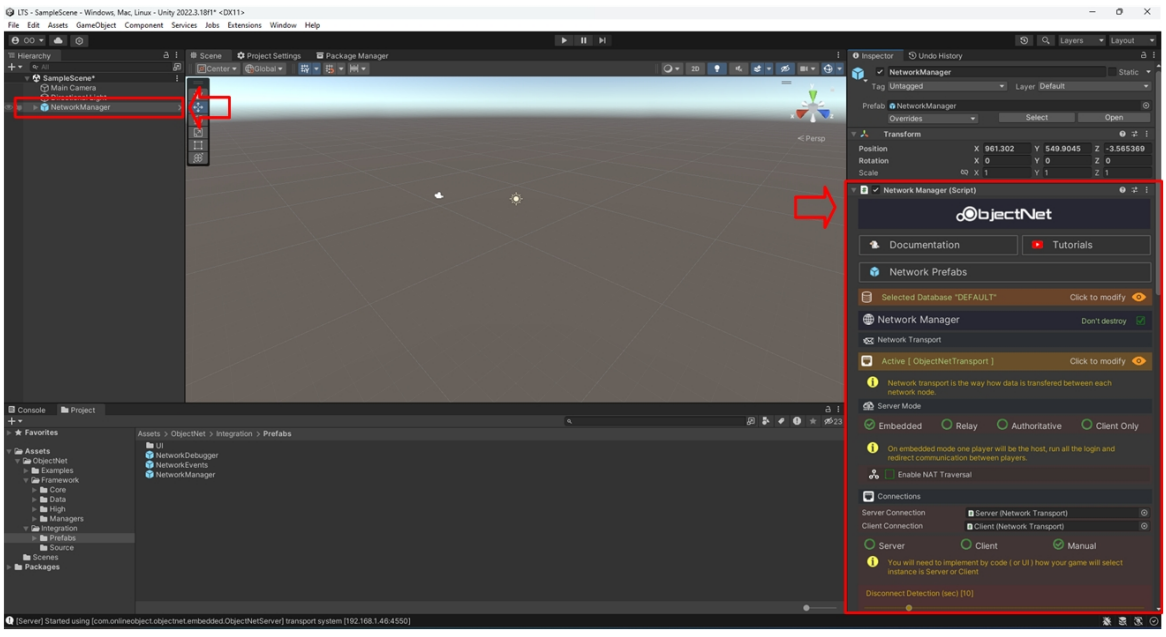
You will need to place some objects on your scene to work with **ObjectNet**, each object has its own purpose, but in this example, we will keep on basic and put only **NetworkManager**.

To know more about NetworkManager read the Network Manager section, for now, let's keep on our step by-step.

Drag NetworkPrefab to you current Scene.



Once you placed **NetworkManager** on your scene you will need to configure it.

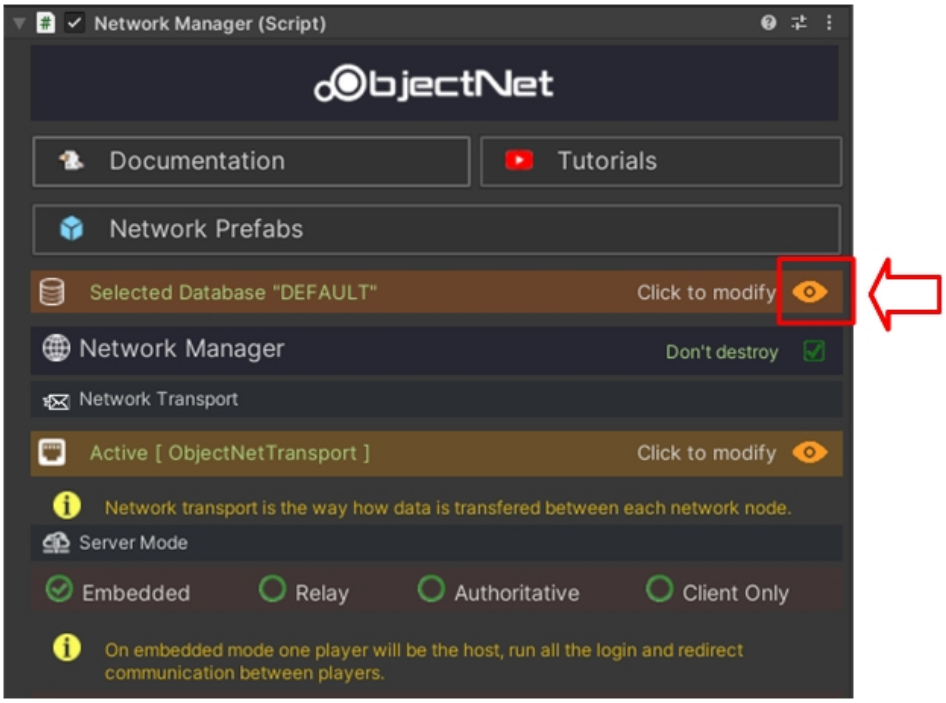


Step 3 - Defining a Database

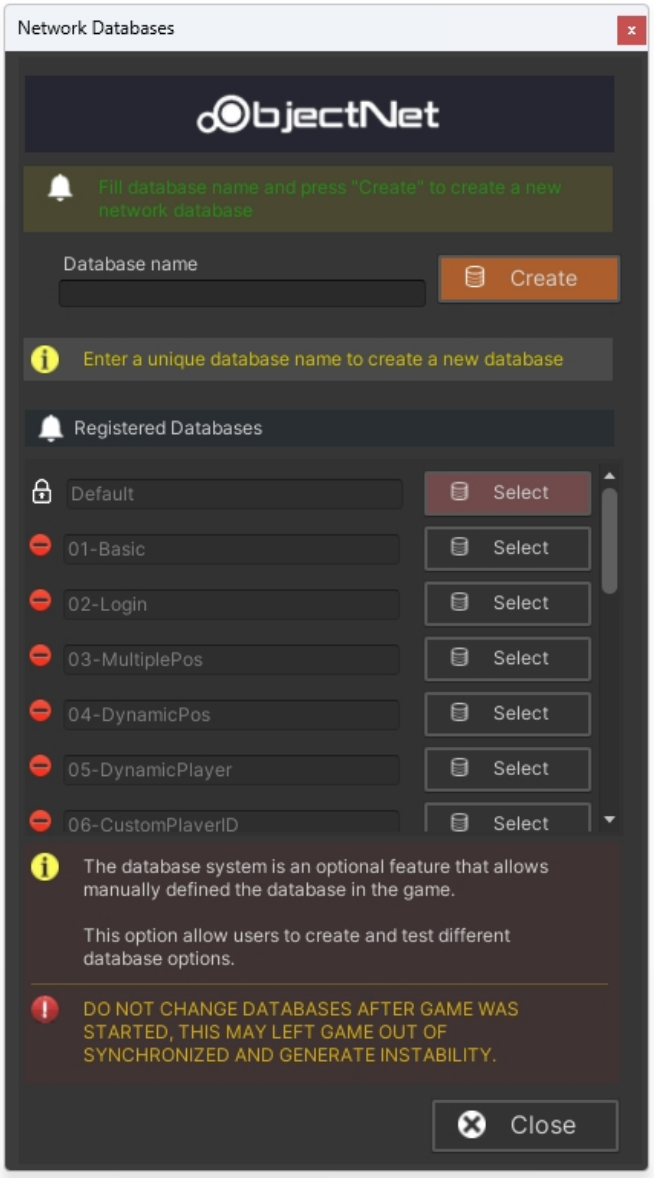
Network Manager has a lot of options, but we will keep on simple and configure only the parameter to start our multiplayer game.

ObjectNet uses a database system to provide more than one game mode on the same game, to know more about the database read the Network Database section.

Click on eye icon to setup a database.

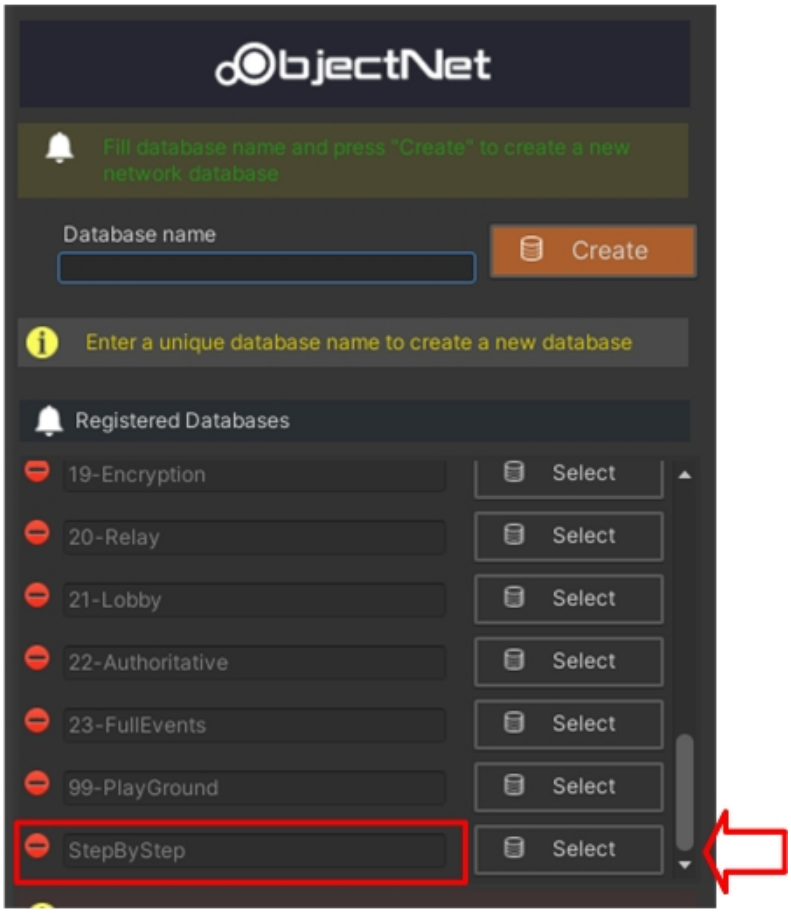


A list of existent databases will appear.



Now you need to create a new database to your example, fill the database name with you database and press Create.

A new database shall appear at bottom of list, click "Select" to set this is the current database.

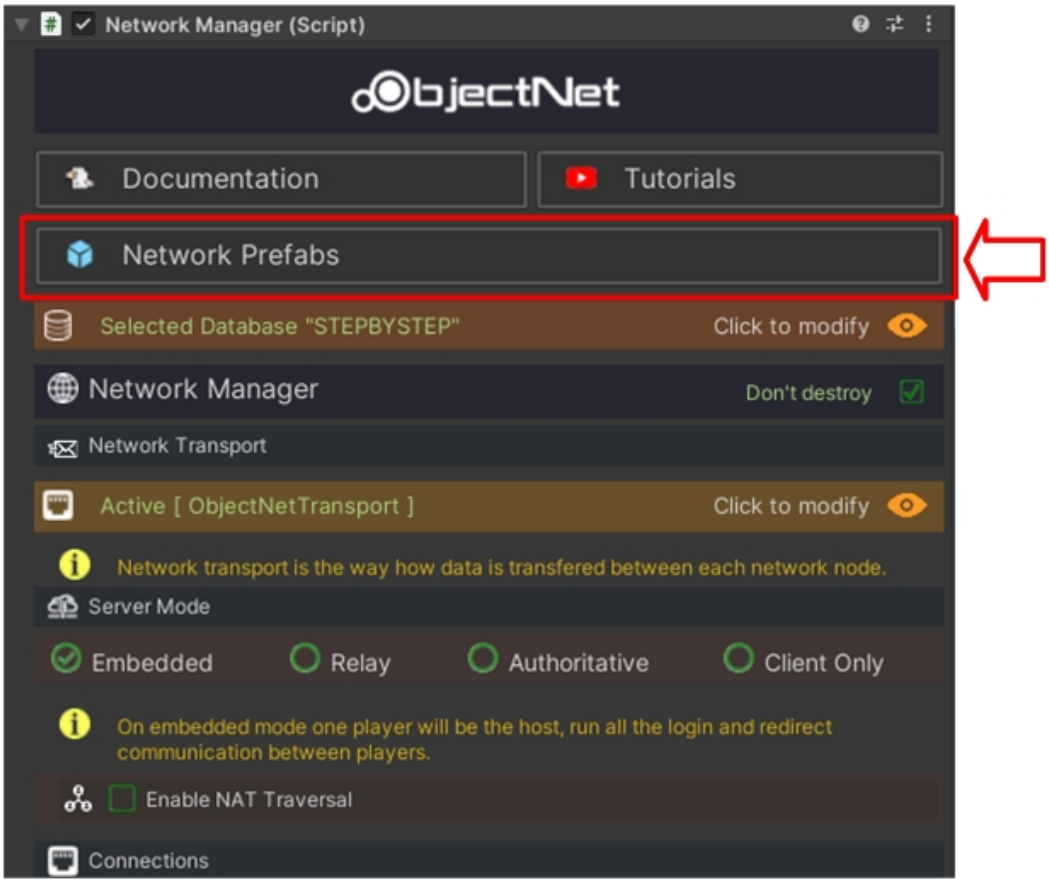


Step 4 - Network Prefabs

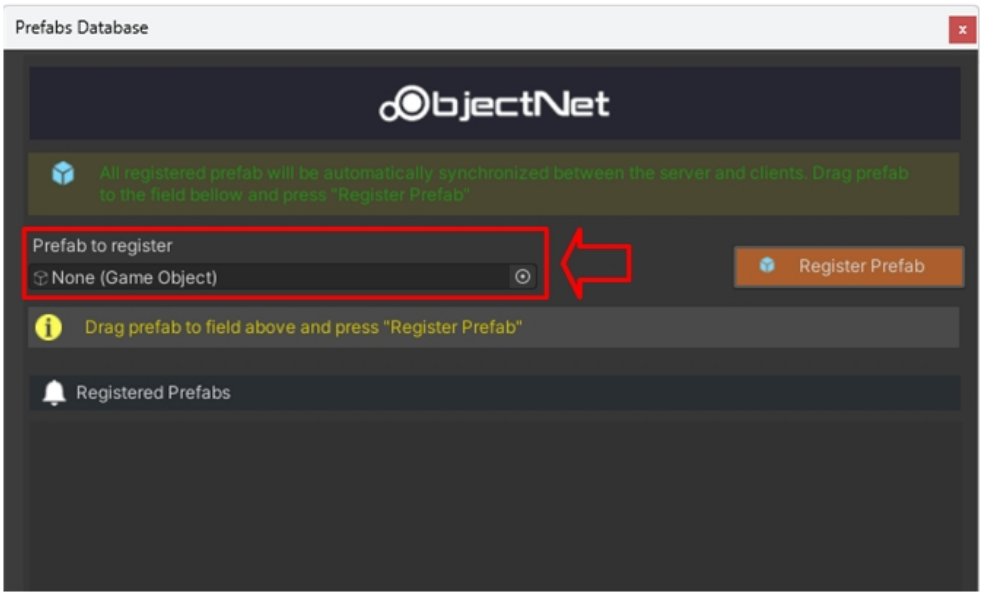
ObjectNet need to know what are the prefabs that he will handle over the network.

To know more about network prefabs read Network Prefabs section.

Click on Network Prefabs button.



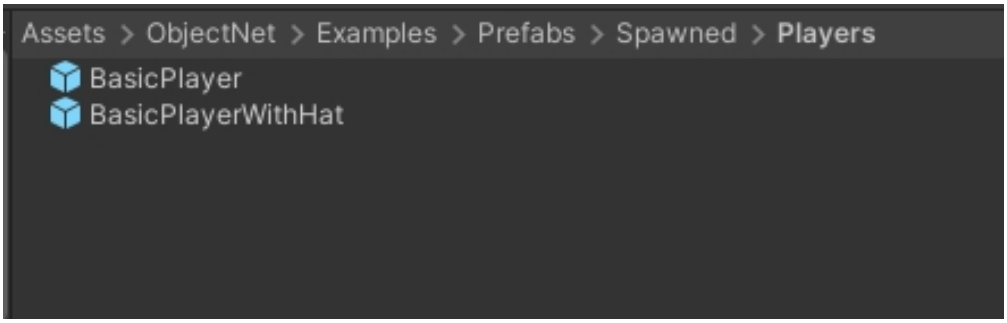
A new window shall appear with you current Network Prefabs.



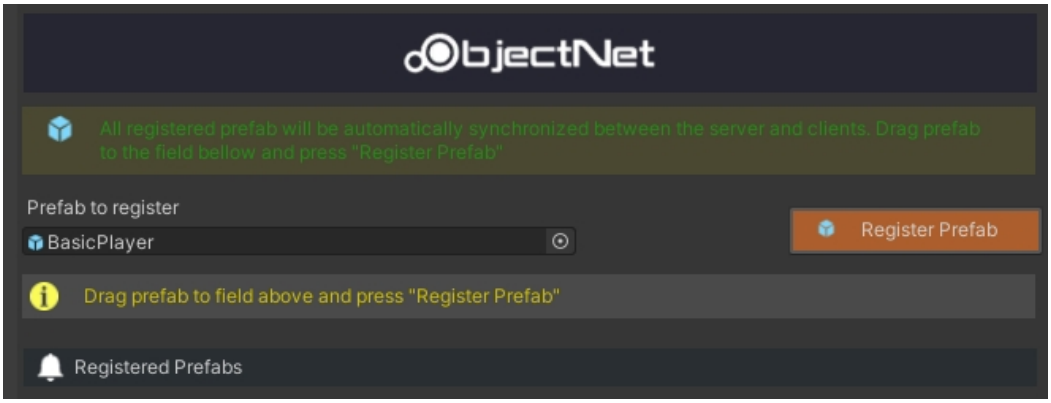
Now you need to select your player prefab, if you don't have one you can use one of the prefabs provided on the ObjectNet example.

To check how to import examples read the *Example Scenes* section.

In our step-by-step, we going to use the "Basic Player" prefab.



Drag "Basic Player" prefab to "Prefab to register" field and press "Register Prefab".



Prefab shall appear as registered on Network Prefabs list.

Step 5 - Player Prefab

Network Prefabs shall list all your network prefabs, including :

- NPC
- Weapons
- Artifacts
- Other Players

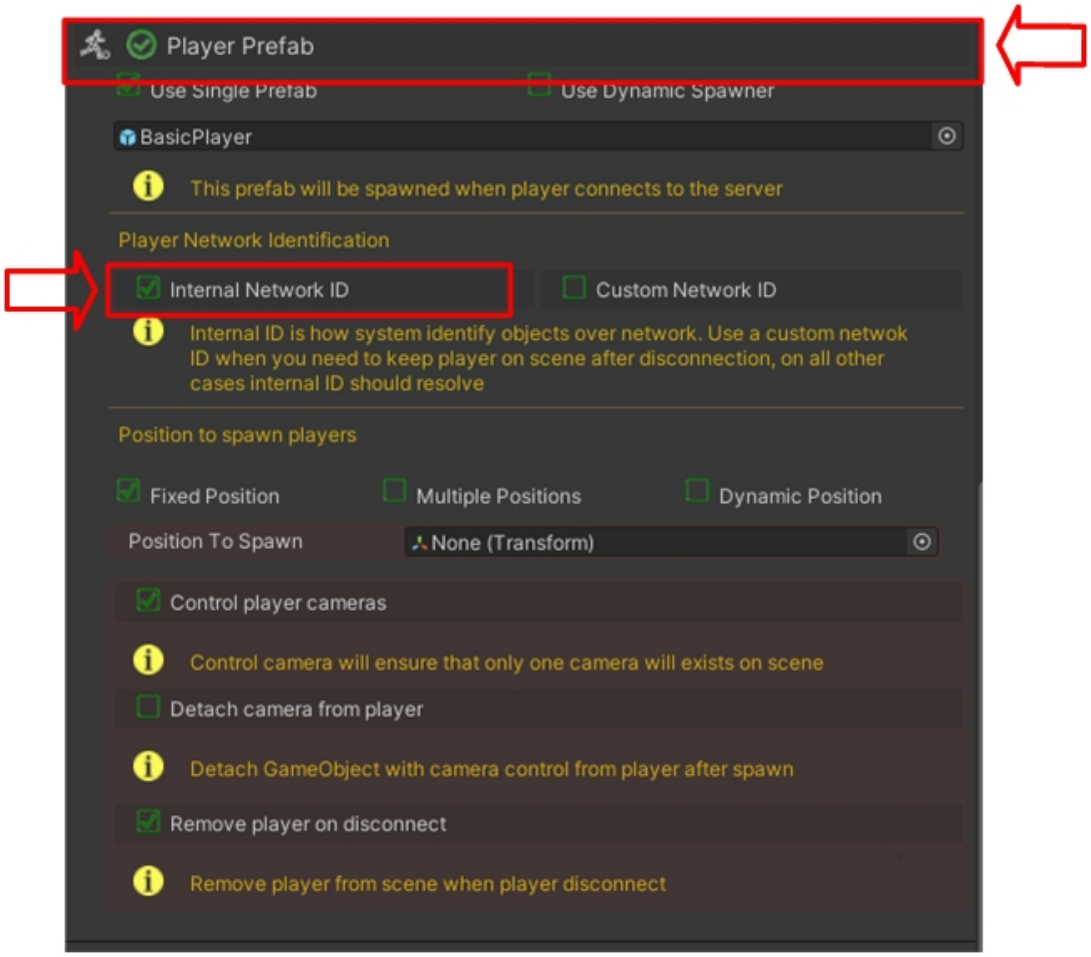
- Vehicles
- And everything that shall be spawned on all clients

Nonetheless, you still need to tell to **ObjectNet** what of those prefabs represents your player, now you going to define which prefab is the player of your game.

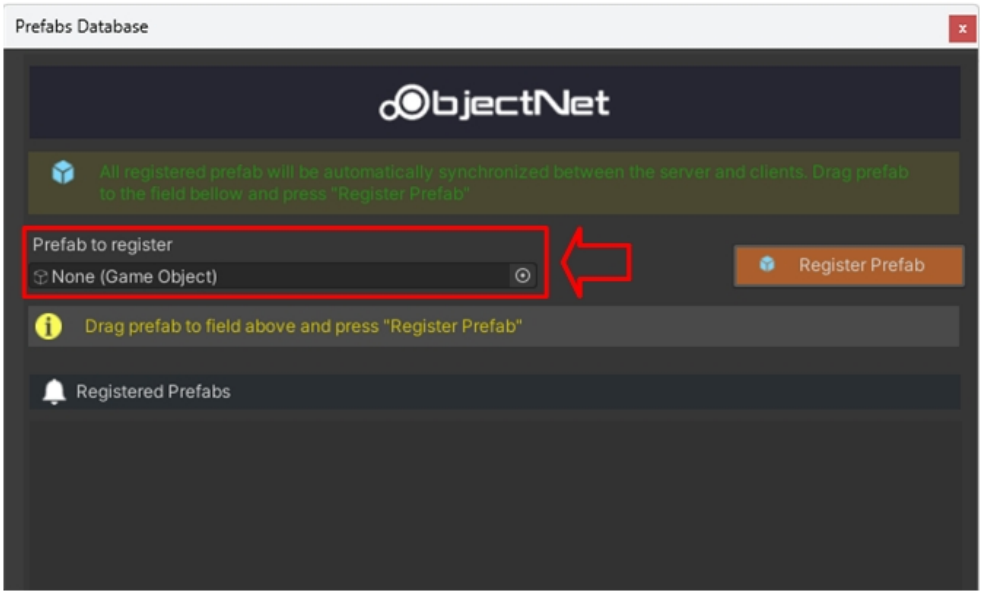
There are a lot of options regard how to spawn your player, to know more about read the Player Prefab section.

Find the "Player Prefab" option on NetworkManager and check it (a bunch of options shall appear as below).

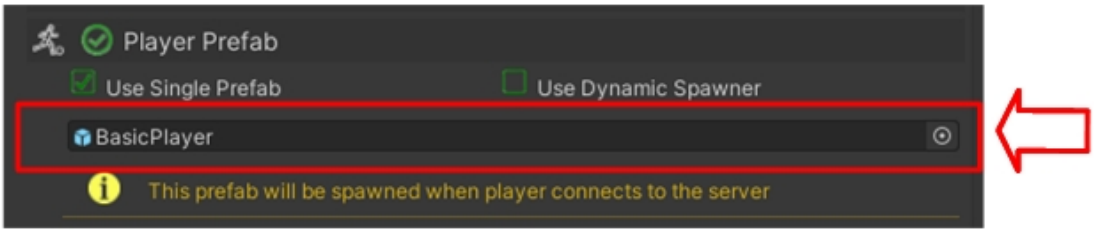
Note: Be careful, don't change "Internal Network ID" option, this may make the player not spawned on instances.



Even having many options we will keep it simple and only draw the "BasicPlayer" prefab to the field.



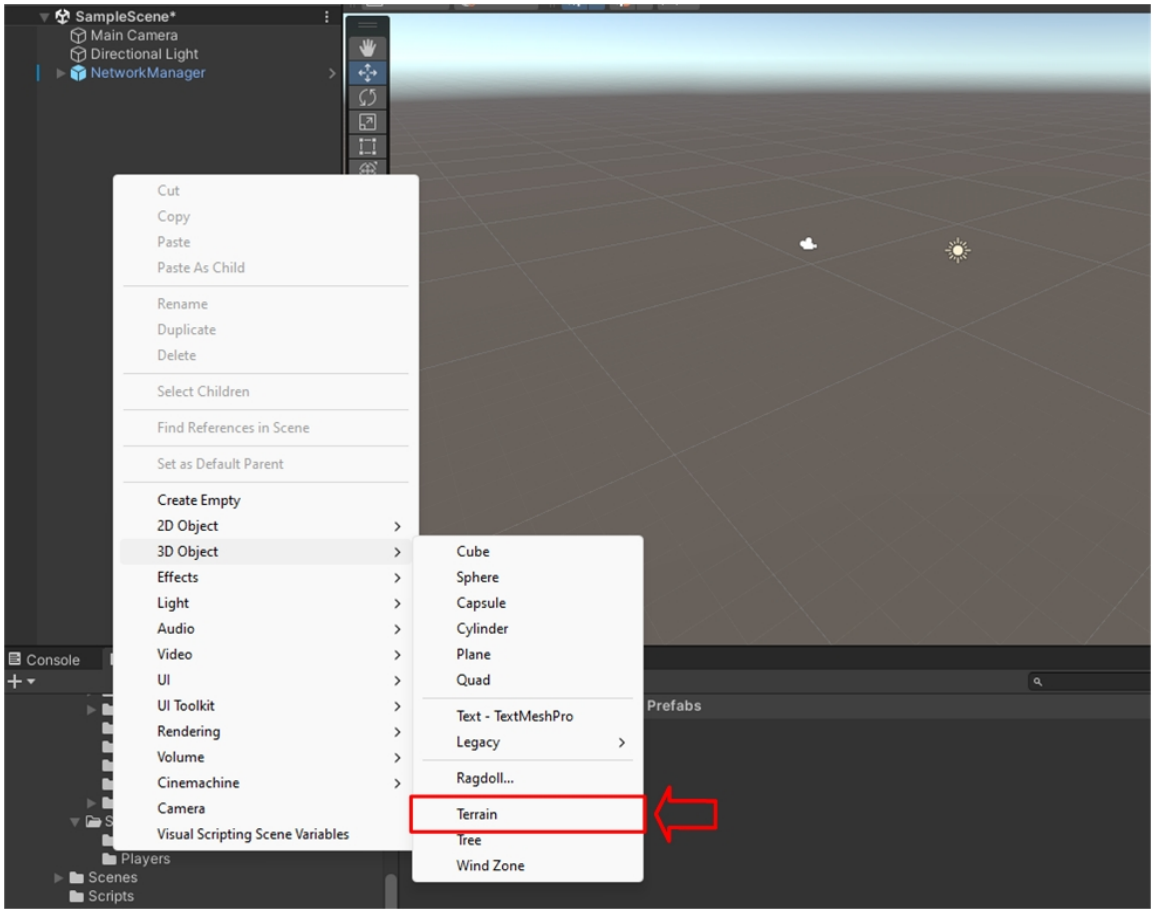
You need to use the same prefab defined on [Step 4 - Network Prefabs](#).



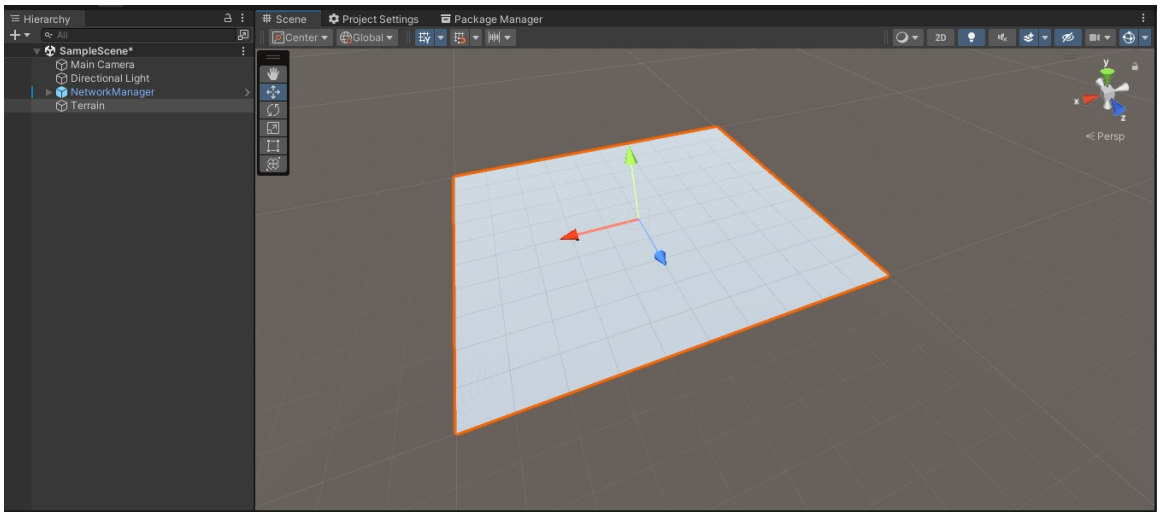
Step 6 - Spawn Position

Now we need to define where your player will be spawned, we can do this by setting "**Spawn Position**" on NetworkManager.

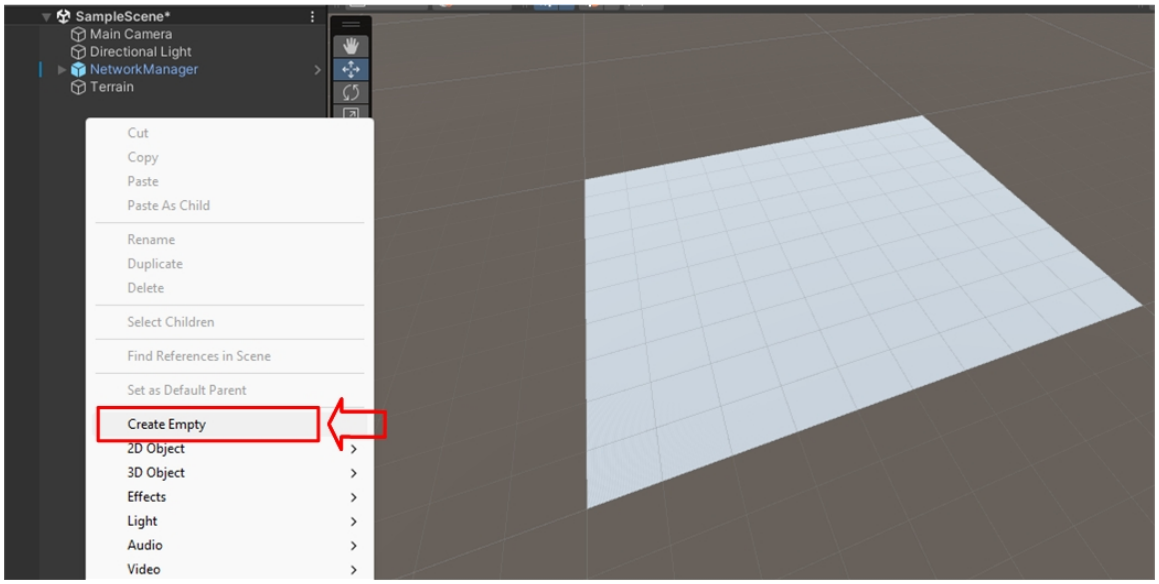
But first we going to put a terrain to step over.



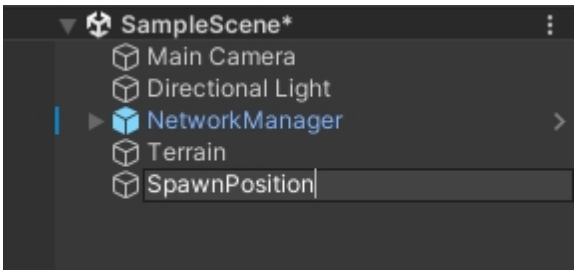
Now we have a terrain on our scene.



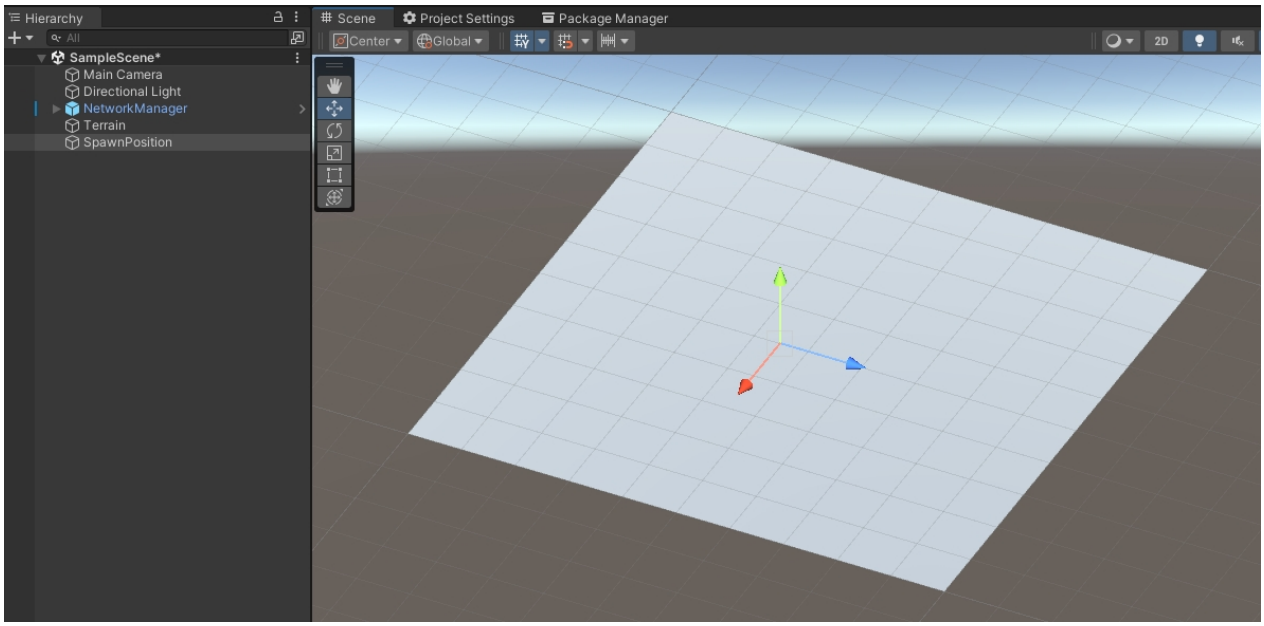
Let's to add another GameObject to work as spawn point reference.



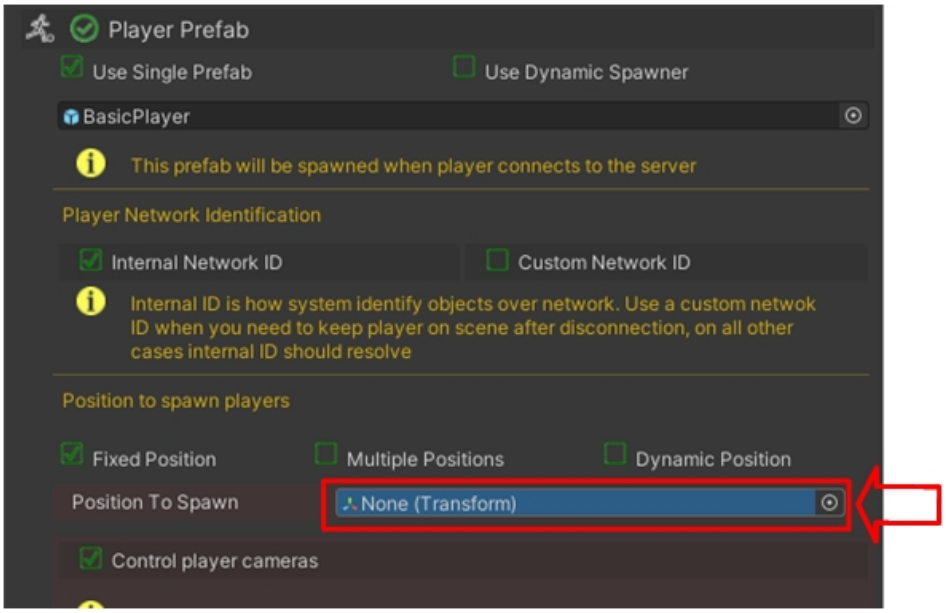
Name it as "SpawnPosition" but you can choose any name.



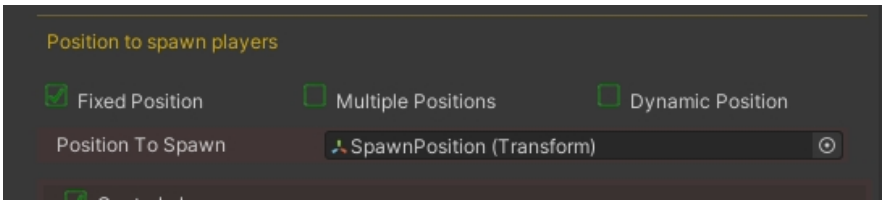
Place it at center of terrain over the terrain (not under).



Now drag "SpawnPosition" object to "Position to Spawn" field.

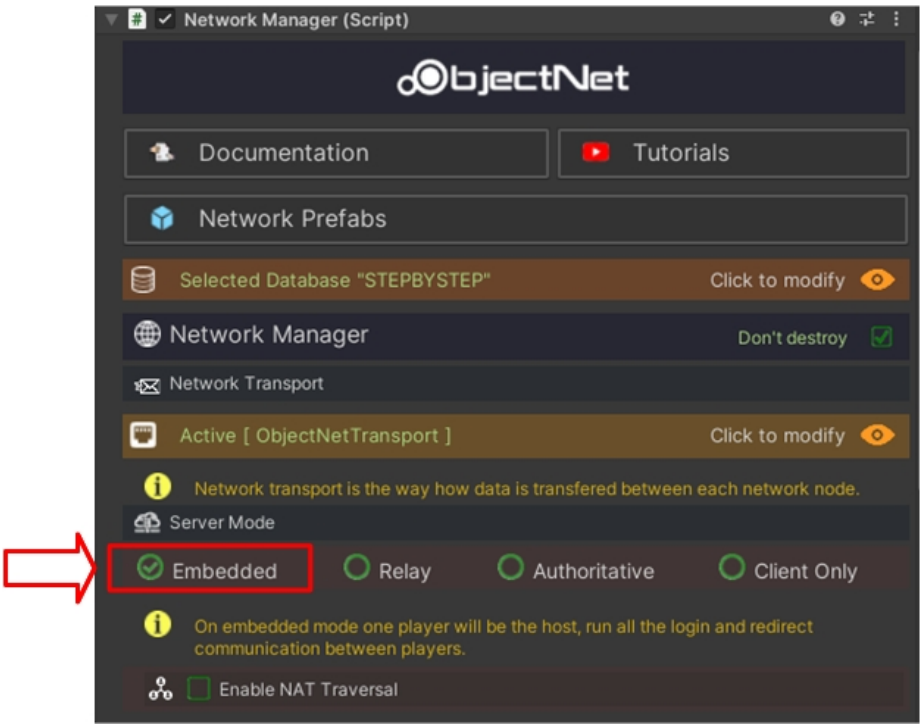


Position to Spawn shall use the transform of "SpawnPosition" object.



Step 7 - Setup Server

First, we need to define the working mode, **ObjectNet** can work with many modes, but in this example, we going to set it up as "Embedded".



Some games provide the option to the player to start as a Server or Client, ObjectNet also provides this facility, to know more read the Server Mode section.
***ObjectNet** also provides ready-to-use examples where you can interact with users to select this option in the game and check the examples scene section Example Scenes.*

For now, let's finish your configuration by following the Step 8 - Compiling Server.

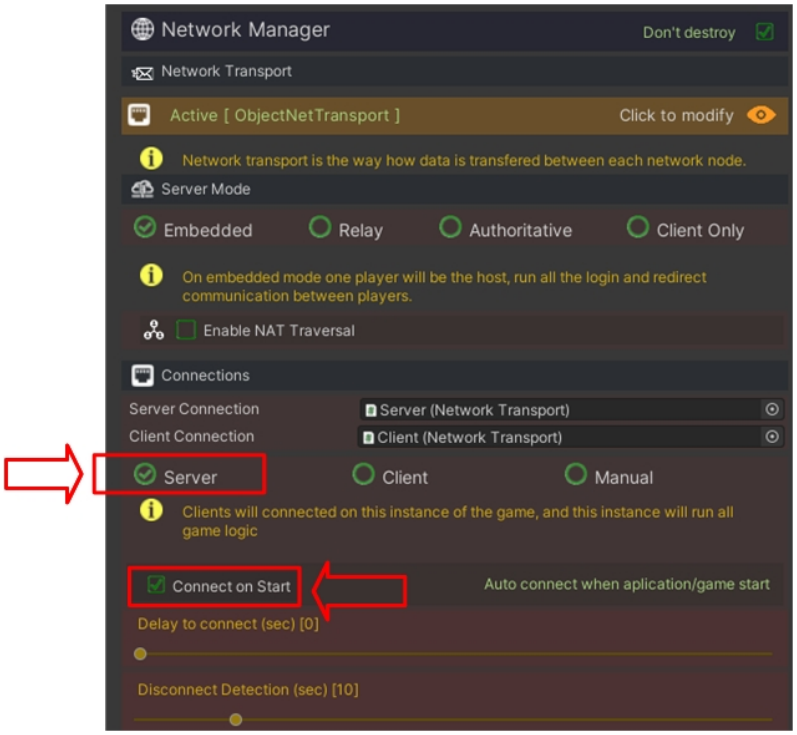
Step 8 - Compiling Server

Lets to compile out server build.

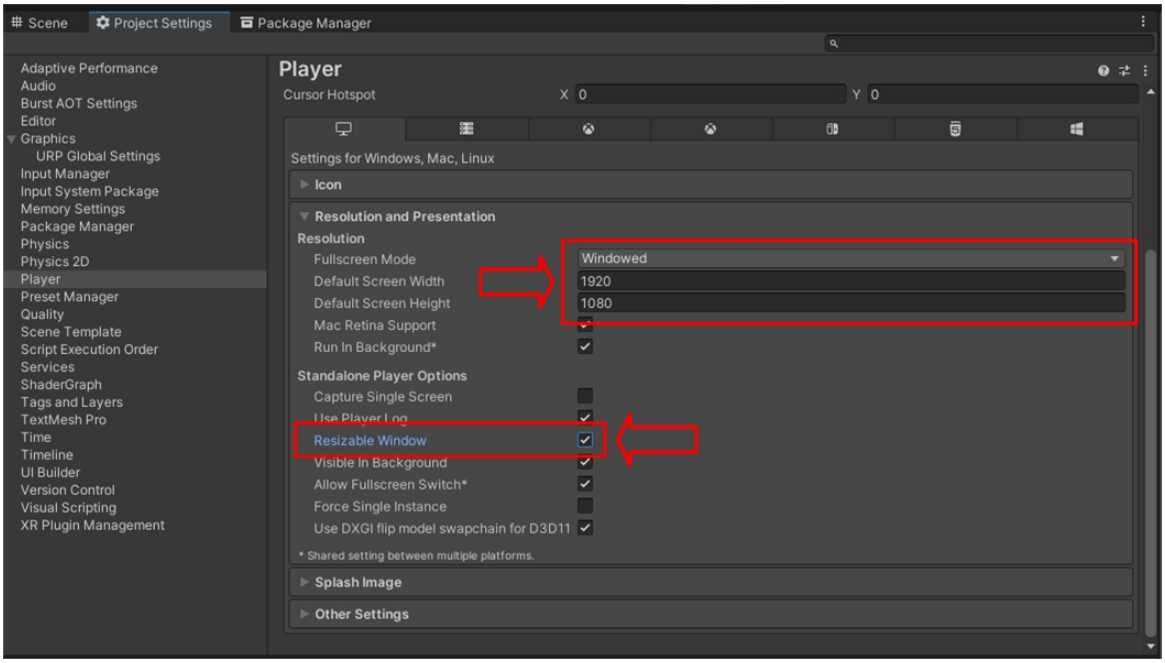
Check the following options.

"Server" shall be selected to start this instance as Server.

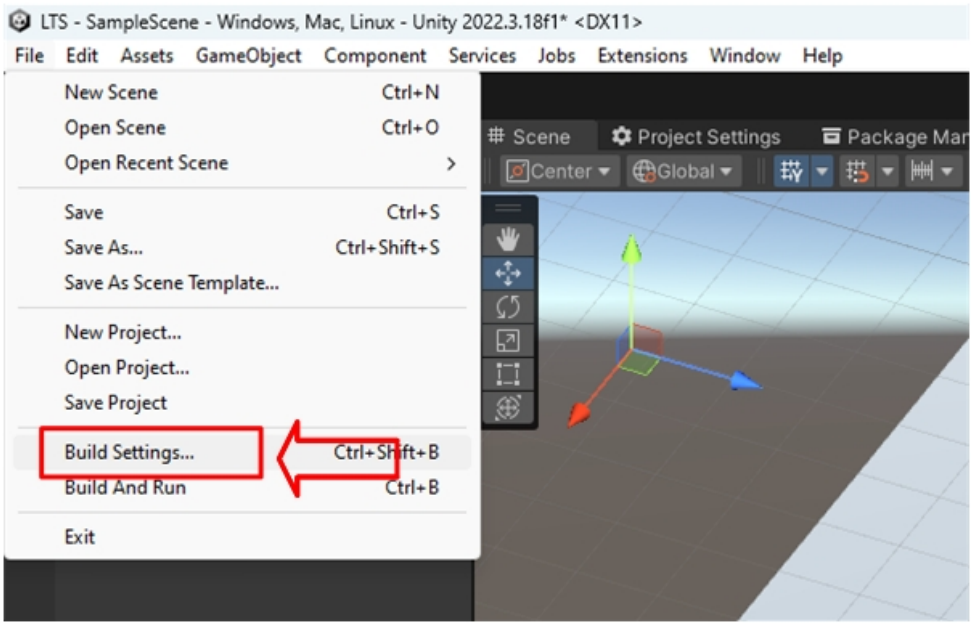
"Connect on Start" shall be selected to start server when game is launched.



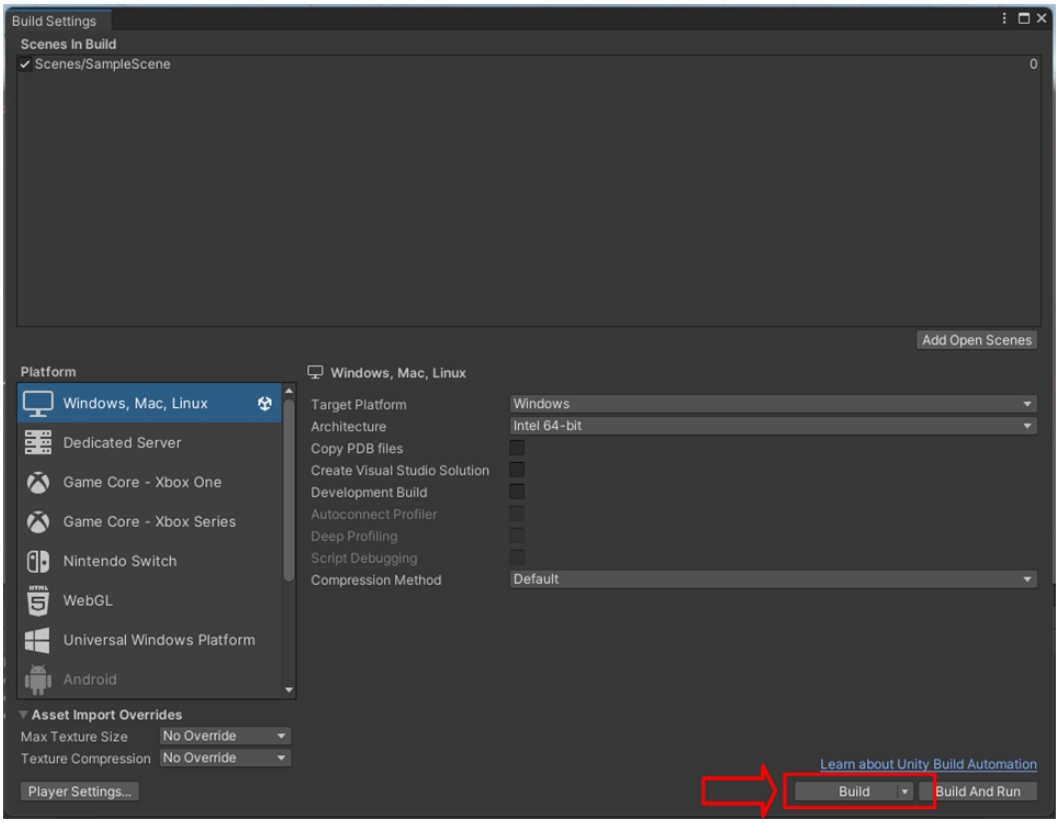
For testing purposes is recommended to build "Windowed" mode and "Re-sizable Window" option.



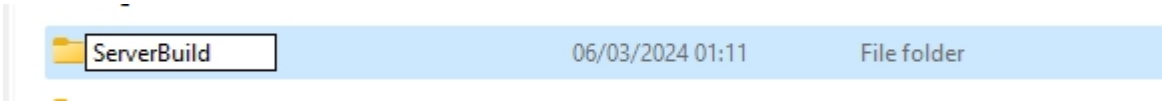
Now lets build our Server.



Click on "Build".



Create a new folder called "ServerBuild" and build it.

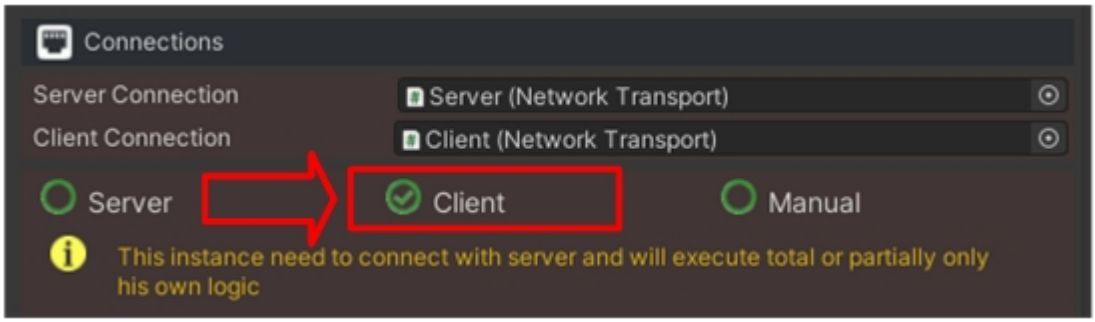


Wait to build be finished.

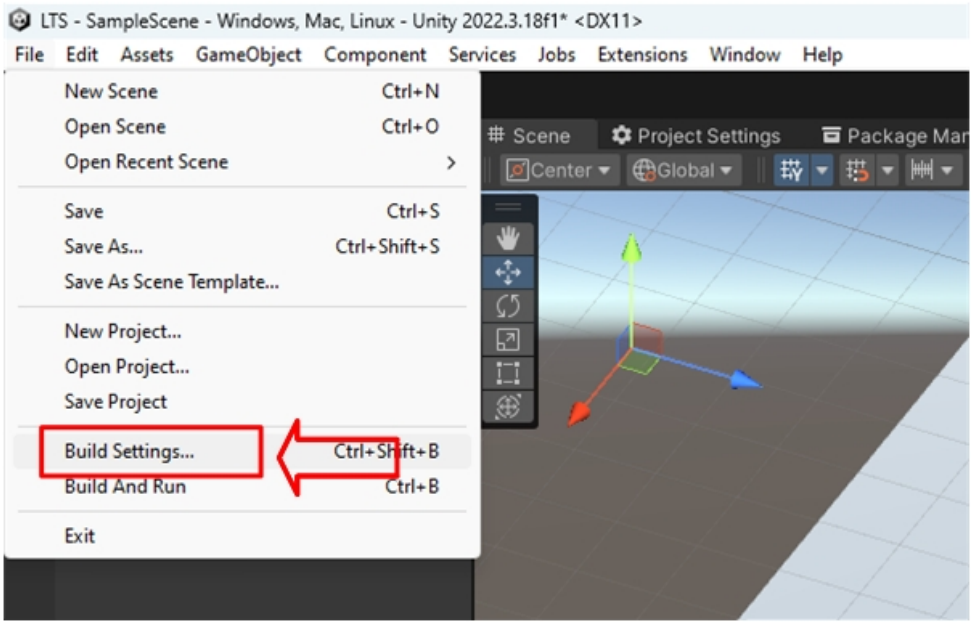
Step 9 - Compiling Clilent's

Lets to compile our client build.

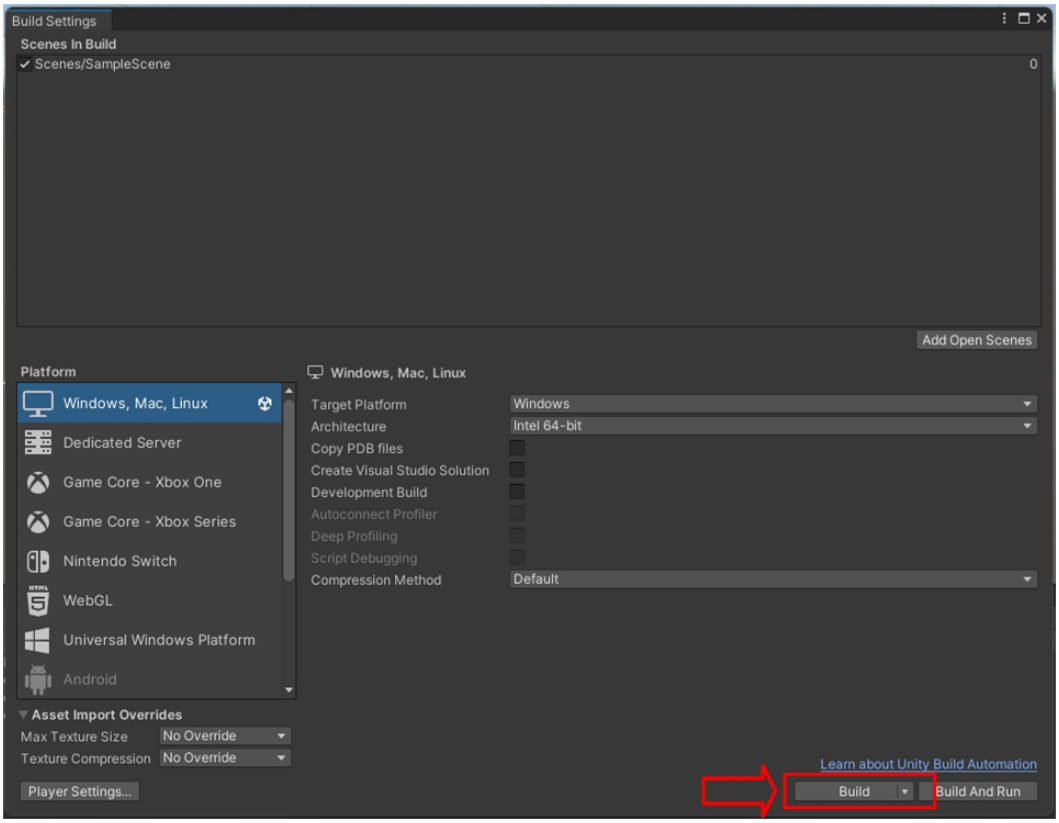
Go back to the NetworkManager and check the "**Client**" option.



Let's keep everything as it and build our Client.



Click on "Build".



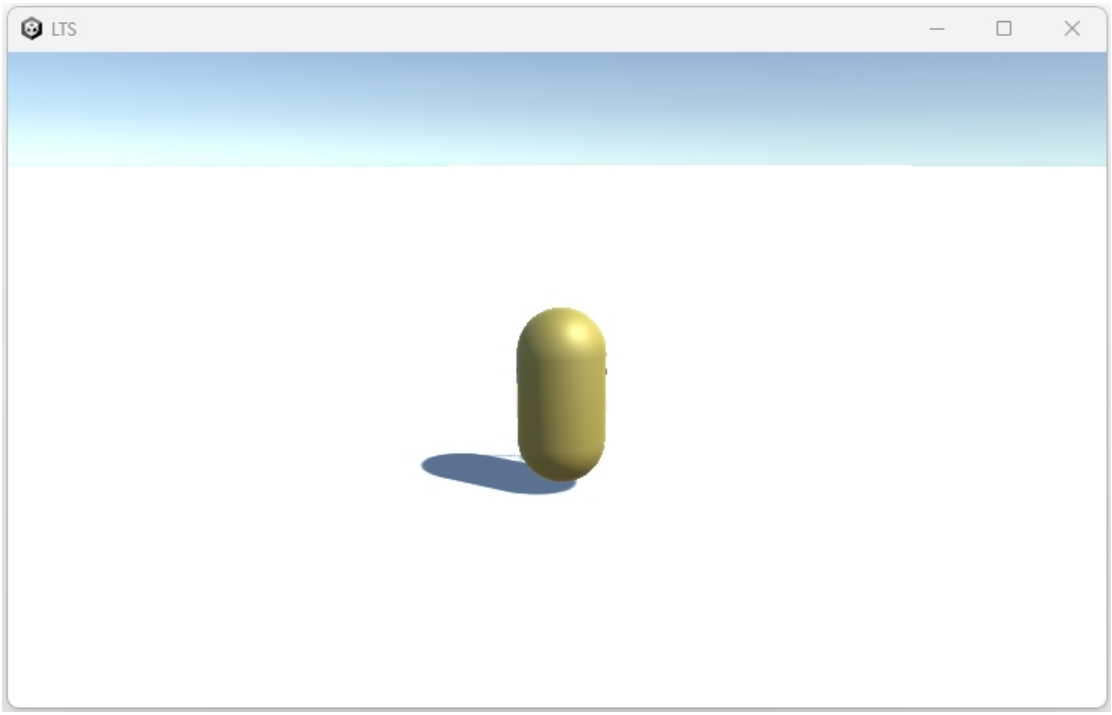
Create a new folder called "ClientBuild" and build it.

Wait build to be finished.

Step 10 - Running

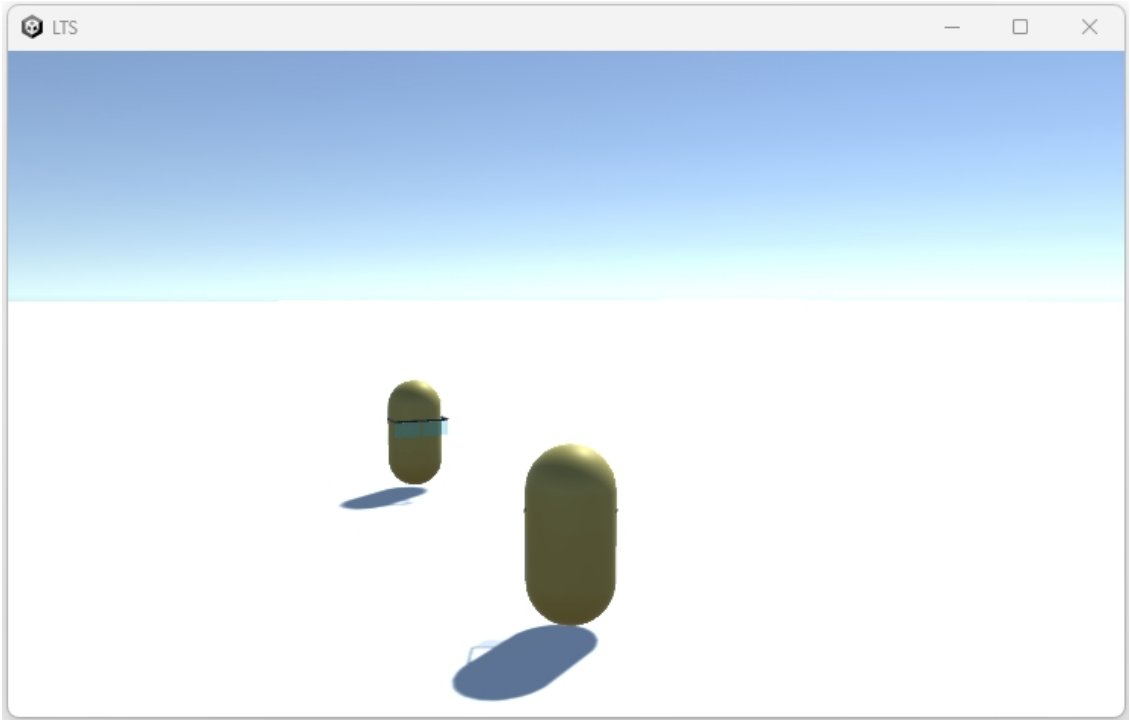
Now you should have two folders.

Open "ServerBuild" and run the game.



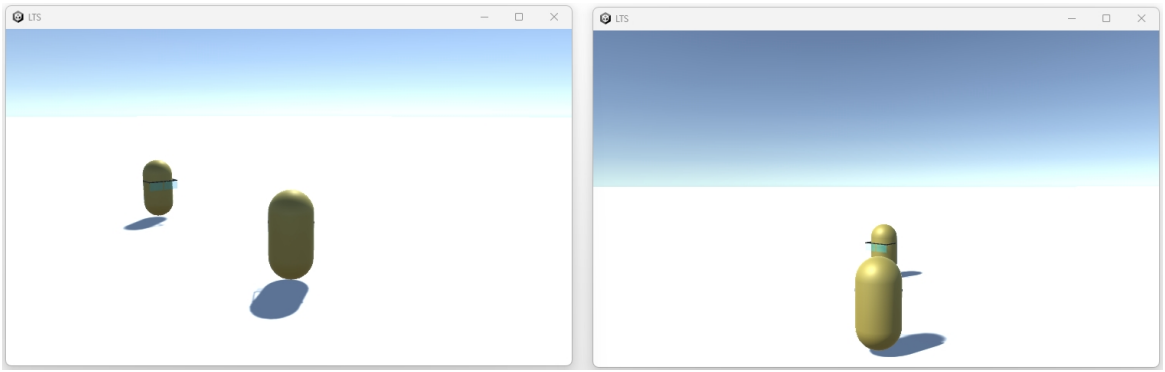
The game shall start and the player spawned (this player is the player obejct where game is running).

Open "Client Build" and run the game.

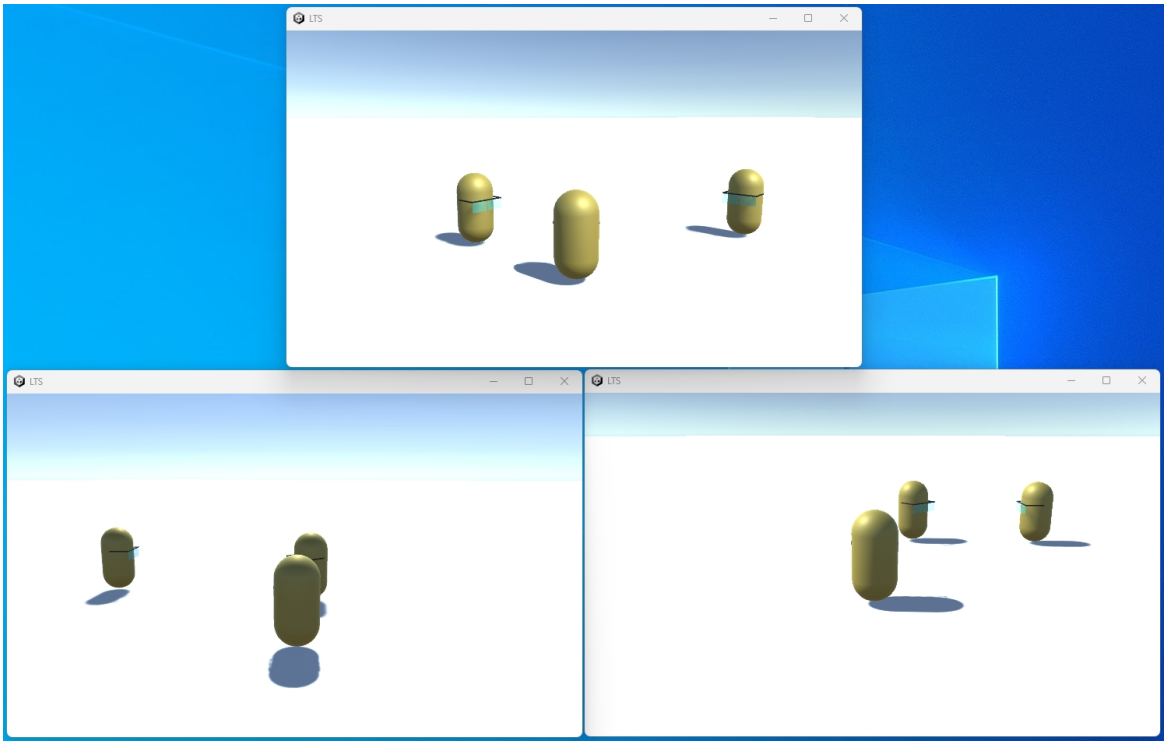


Your player shall be spawned and the previous player (on the server) will appear on your instance.

All players shall appear in both instances and all movement will be reflected in both instances (you can move by using W-A-S-D keys).



Now you also can run many instances as you wish.



Step 11 - Next Steps

Now you know the basic of basics to start, we strongly recommend checking this document and all topics to know how to extract all the power that **ObjectNet** can provide to you.

We also recommend checking each example on examples scenes, they can bring a clear view of how to work with **ObjectNet**.

Concepts

ObjectNet is based on some main pillars.

- **Network Events**
- **Network Prefabs**

- **Network Transport**

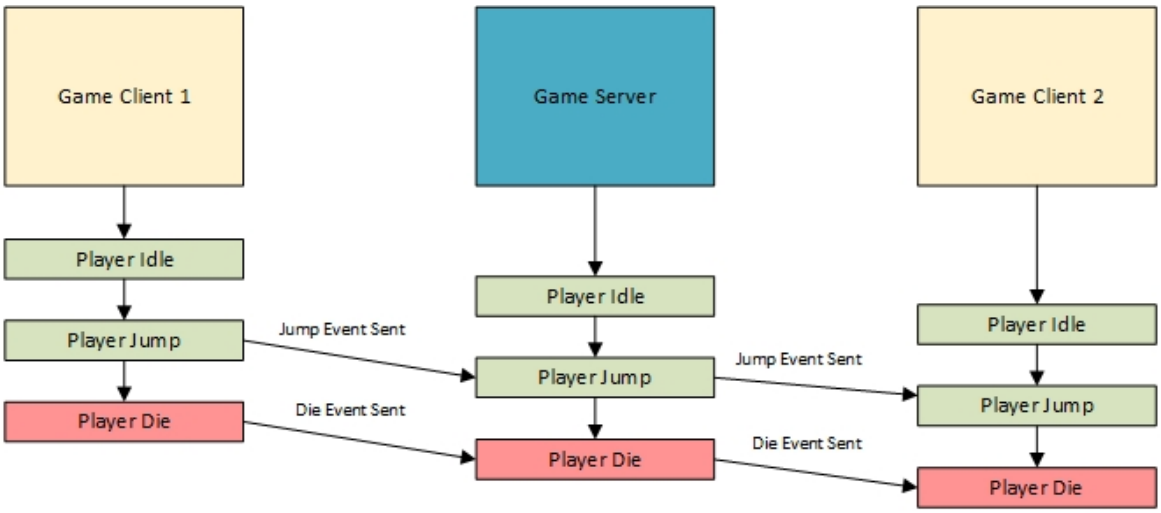
Network Events

Network Event is the base of all ObjectNet communication systems, this means that anything will happen based on events.

Question : But, what is "event" by definition?

Answer : An event happens on an instance of a game (Server or Client) and should be notified to other members in the same context.

The following drawing illustrates how events work during a game session.



In a basic game architecture, some client or server sends an event destined for one or more players, and when this event arrives the server or player will execute some action.

This simple approach provides a powerful mechanism to isolate each action in its context and remove any horizontal dependency from any object or context status.

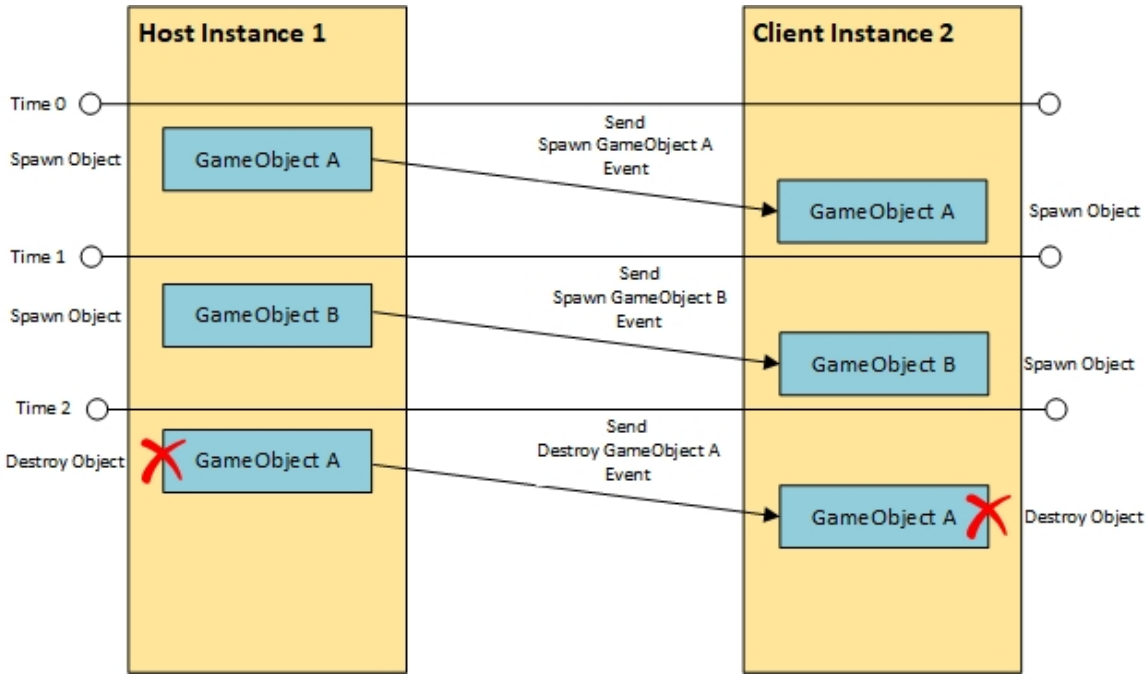
Network Prefabs

Network Prefabs is a structure used to keep objects synchronized over a network.

Question: But, what is "Network Prefab" by definition?

Answer: Network Prefab is any prefab object that the system will detect its creation or destroy and will keep synchronized on all instances of the same game.

The following drawing illustrates how ObjectNet works to keep Network Prefabs synchronized over the network.



Network Prefabs allow you to keep game objects synchronized without the need to change any script or code from your game. This simple but powerful mechanism allows you to port any game with no or fewer changes into the current source code.

Network Transport

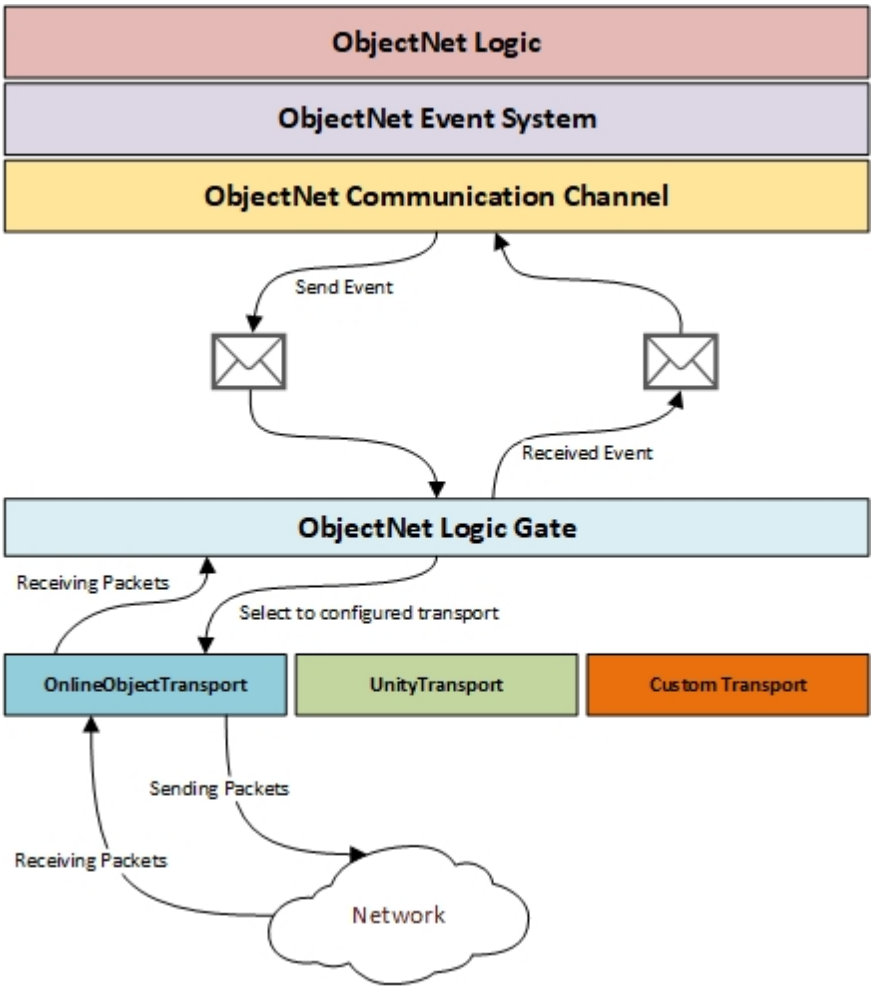
Network Transport is the way how ObjectNet use to communicate with other network peers.

ObjectNet has its embedded transport system called OnlineObjectTransport which is a light-way implementation of TCP and UDP sockets.

ObjectNet also accepts "Unity Transport System" in replace of OnlineObjectTransport (see how to install Unity Transport Unity Transport).

Additionally, you can also implement any other transport system such as Epic, Steam, or your custom system by using the interfaces provided by ObjectNet API.

The following drawing illustrates how ObjectNet abstracts its internal behavior from the configured transport system.



The logic gate layer isolates ObjectNet internal logic from the transport system. This allows the system to interchange messages using any transport system that implements needed interfaces.

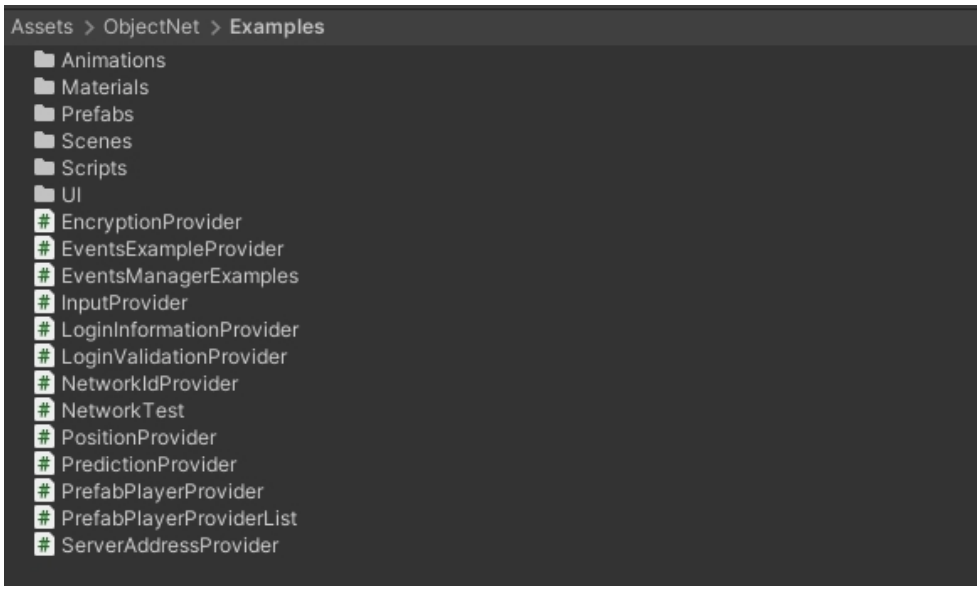
Providers Scripts

ObjectNet introduces the concept of **Provider Scripts**, provider script is scripts that allow to provide any input information to **ObjectNet**. This means that every time that you can input or override any **ObjectNet** behavior, this will be made by some **Provider Script**.

There are two types of providers :

- `IInformationProvider`
  - This provider covers almost all cases and is used to provide any information to the system.
- `IInputProvider`
  - This provider is used only to provide network Inputs (see Remote Controls).

ObjectNet provides an example of each information provider in the following example folder.



Object Update

Object Update is the way ObjectNet keeps objects updated, this is a special core event used to send and receive all updates of any NetworkPrefab.

By default, object updates can include the following information:

- Object Position
- Object Rotation
- Object Scale
- Animation Status
- Particles Status
- Variables (only for **Network Objects**)

Reliable Message

ObjectNet can deliver messages in a reliable way, this means that the Transport Layer will ensure that the message arrives on the network peer.

Normally, this message is sent over a TCP network nonetheless, this may depend on transport layer implementation and can use another protocol such as reliable UDP to ensure delivery.

This method is commonly used for punctual events that will not be sent twice, for example :

- **Player is death**
- **Player was spawned**
- **Object was picked**
- **Etc...**

For messages and events that will be repeated all the time, we suggest using the Unreliable method.

Unreliable Message

ObjectNet can deliver messages in an unreliable way, which means that the Transport Layer will not ensure that the message arrives on the network peer.

Normally, this message is always sent over the UDP network.

This method is commonly used for punctual events that will be repeated all the time, for example :

- **Player position**
- **Player rotation**
- **Player input controls**
- **Etc...**

The reason why these messages are sent all the time is because Unreliable messages are considerably faster than Reliable messages.

An unreliable message does not guarantee that each message will arrive on each peer, this is the reason why messages are sent at a frequency rate because not matter if one message is lost because another message will be sent.

Active x Passive

On **ObjectNet** the network objects can assume two statuses, **Passive** and **Active**.

- **Active**
  Active objects are the object instances where logic and scripts are executed.
  For players, each player will have his player prefab as active on his instance, on the other hand, all other objects will be active only on the server or player host.

- **Passive**
  Passive objects is the object instance spawned by the server on each client instance.
  Os game clients the unique object on Active mode is the local player prefab object.

Network Variables

ObjectNet provides the facility to keep variables synchronized over the network.

Those variables are automatically synchronized over the network from Active to Passive instances.

To implement Network Variables, the script must be inherited from NetworkBehaviour, and can be used in two ways.

- By **NetworkPrefabs** (see Script Variables)
- By **Code** (see Coding Network Variables)

Network variables can synchronize the following types:

```
int
uint
long
ulong
short
```

```
ushort
float
double
byte
byte[]
string
char
char[]
Vector3
Color
bool
```

Coding Network Variables

This section explains how to declare and use **Network Variables** by code.

1. Using Network Variables directly

Network Variable must bound a type **`NetworkVariable<T>`** where T must be included into allowed types (see Network Variables).

```
private NetworkVariable<string> variable = "";
```

2. Once defined you can use it as if any other script variable

```
this.variable = "This is a new string value";

string variableValue = this.variable;
```

3. Detecting Changes

You can detect when the object was updated on the passive instance to apply changes only when occur..

```
this.variable.OnValueChange((string oldValue, string newValue) => {
  Debug.Log(string.Format("Value was updated from [ {0} ] to [ {1} ] ",
oldValue, newValue));
 });
```

4. Synchronizing local variables

ObjectNet allows to synchronization of network variables with local variables automatically. The following script allows you to keep a local variable synchronized with a network variable, no matter where you change the value, both variables will have the same value.

```
string localVariable = "Start text";

this.variable.OnSynchonize(() => { return this.localVariable; },
                           (string value) => { this.localVariable =
value; });
```

```
 this.localVariable = "New value from local"; // In this case variable Network
will also receive the new value "New value from local"
 this.variable = "New value from Network variable"; // In this case local
variable will also receive the new value "New value from Network Variable"
```

Network Behaviour

ObjectNet provides the facility to create custom behavior that will apply to custom NetworkObjects.

This structure is a class that needs to implement NetworkEntity base class. ObjectNet has already the following types :

- PositionNetwork
- RotationNetwork
- ScaleNetwork
- AnimationNetwork
- ParticlesNetwork
- VariablesNetwork
- InputNetwork

The main idea of custom behavior is to keep objects synchronized without the need to manually send and receive each value at different points of the code.

To implement your own DataStream you can check the API document on `NetworkEntity` class (see Custom Behaviour).

DataStream

ObjectNet provides the facility to send and receive simple and complex structures over the network.

This structure is a class that needs to implement `IDataStream` interface. ObjectNet has already the following types :

- `Int16 : Int16Stream`
- `UInt16 : UInt16Stream`
- `Int32 : Int32Stream`
- `UInt32 : UInt32Stream`
- `float : FloatStream`
- `double : DoubleStream`
- `bool : BooleanStream`
- `string : StringStream`
- `Color : ColorStream`
- `byte : ByteStream`
- `byte[] : ByteArrayStream`
- `Vector2 : Vector2Stream`
- `Vector3 : Vector3Stream`

The main idea of DataStream is to write and read data without the need to write on each place, you can use

some DataStream and encapsulate how much data you need.

To implement your own **DataStream** you can check the API document on `NetworkEntity` class (see Custom DataStream).

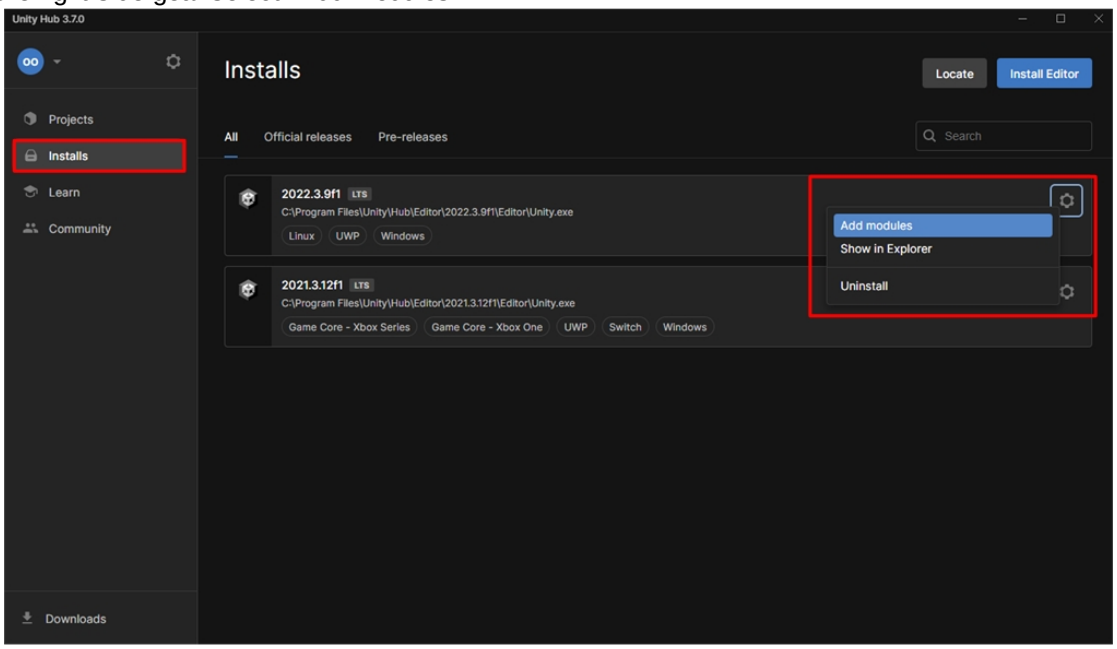
Unity Dedicated Server

ObjectNet allow to create standalone build to run dedicated servers (Relay and Authoritative).

Installing required modules

To be able to build an standalone build of you game server you need to install **Dedicated Server Build Support**.

1. On Unity Hub select you unity install
2. On the right side gear select "Add modules"

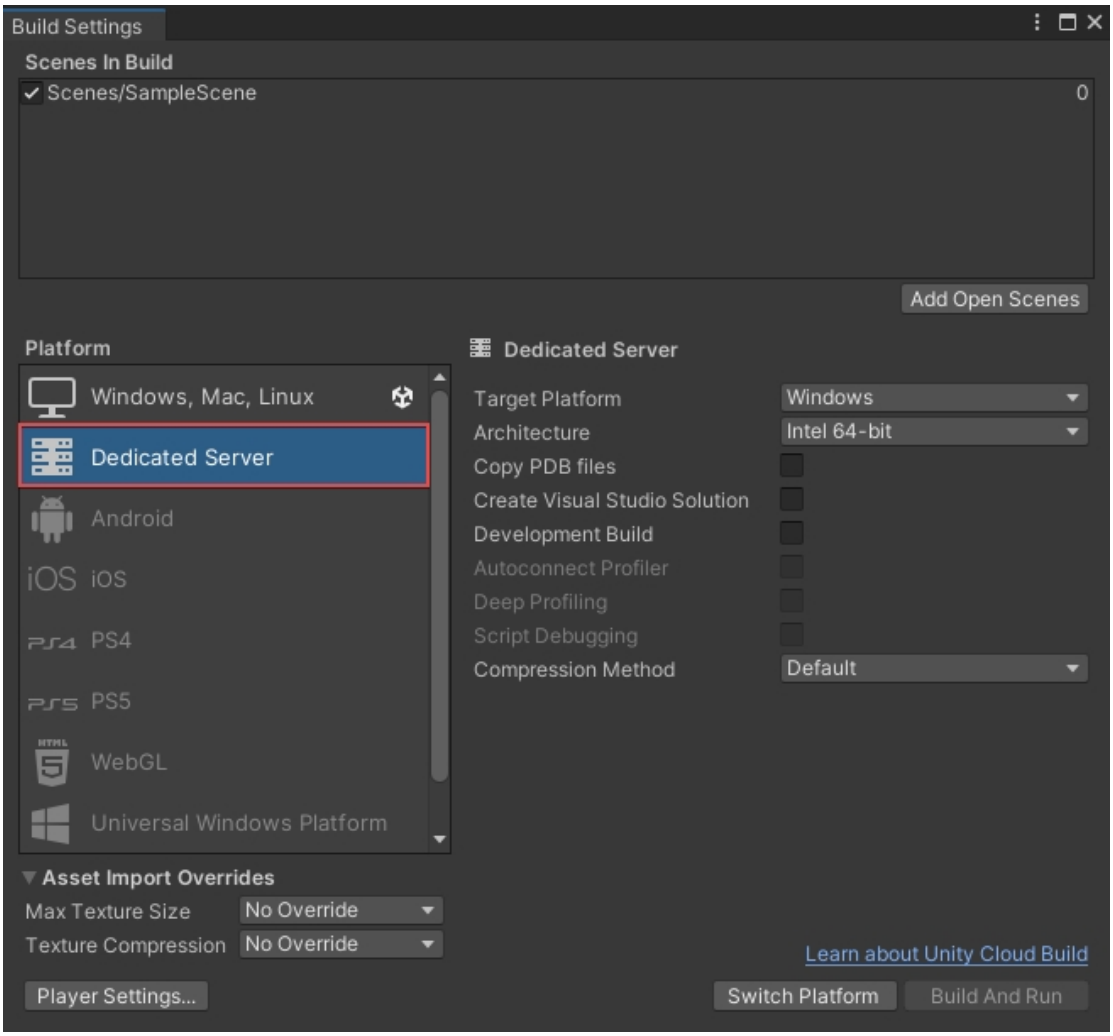


3. Add Dedicated Server Build Support module

Build your application for Dedicated Server

You can create a Dedicated Server build through either of the following ways:

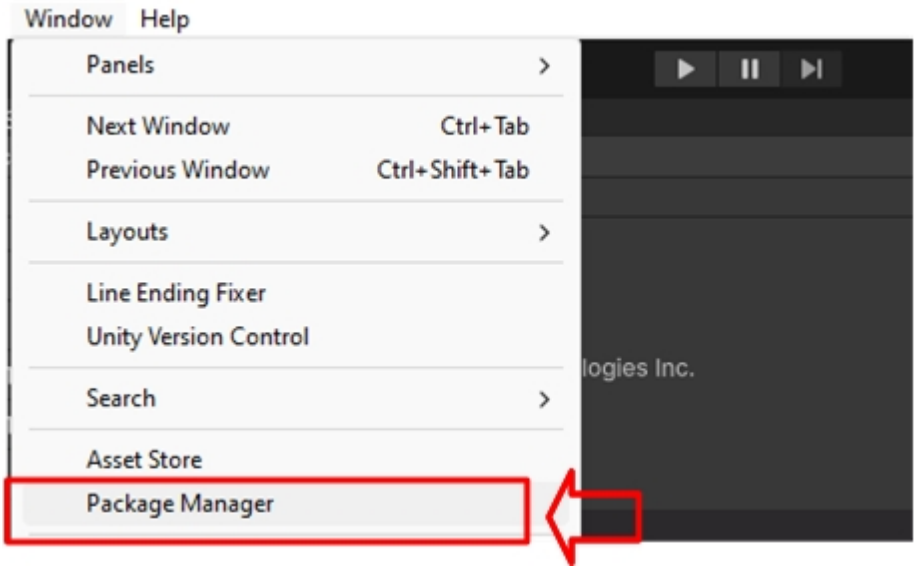
1. From the Unity Editor, select File > Build Settings.
2. Select Dedicated Server.



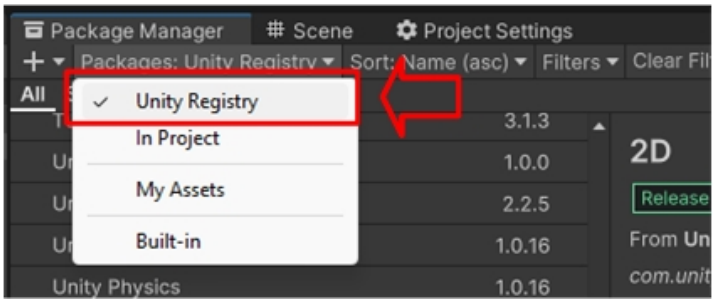
Unity Transport

ObjectNet allow to use Unity Transport System as in use transport system.

1. Open the Unity Package Manager by navigating to Window > Package Manager along the top bar.

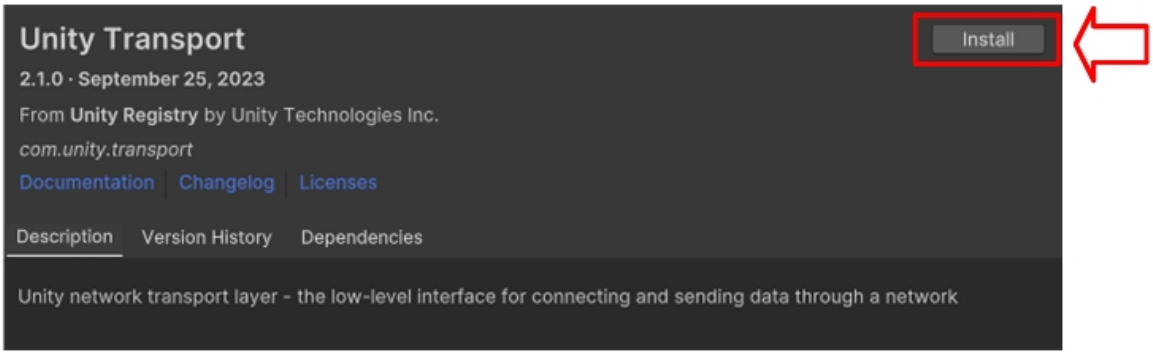


2. Click Add in the status bar.



3. Find "Unity Transport Package"

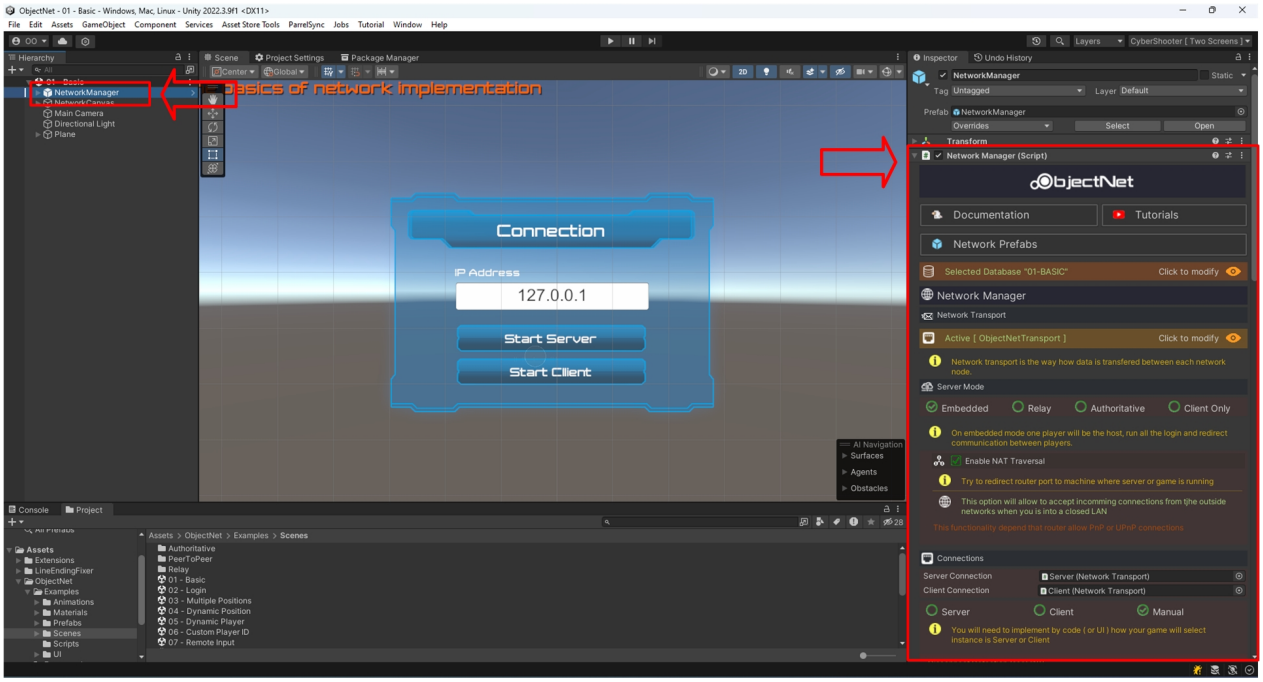
4. Click to install package.



Is this options not appear see official Unity documentation (see Installing Unity Transport System).

Network Manager

NetworkManager is the main controller of ObjectNet and he is responsible to control and manage the main aspects of the multiplayer system.



NetworkManager is the final layer of ObjectNet core system, and he is responsible for handling all elements of your multiplayer game.

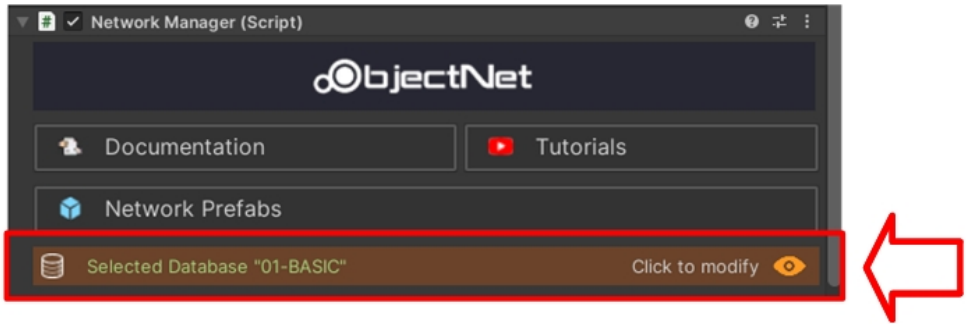
On NetworkManager you can define or configure all aspects of your multiplayer. On the NetworkManager editor, you can configure the following properties.

- Network Database
- Network Transport
- Server Mode
- Connections
- Connection Parameters
- Send Configurations
- Delivery Mode
- Network Prefabs
- Enable Login
- Enable Encryption
- Network Clock
- Latency Tolerance
- Movement Prediction
- Remote Controls
- Measure Latency
- Player Prefab

Network Database

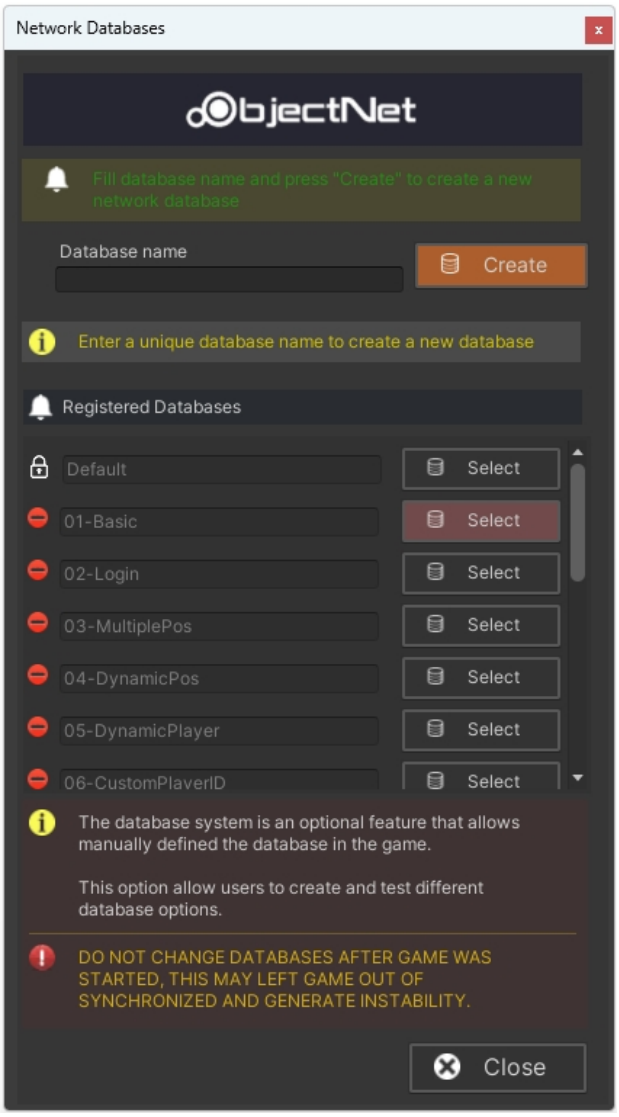
NetworkDatabase is the structure responsible for storing all NetworkPrefabs and NetworkEvents databases into a unique internal database. Each NetworkManager and NetworkEventsManager can target just one database per game.

The selected database appears on the top of NetworkManager object.



To modify the current Database you shall click on the eye icon close to the "Click to modify" message.

A new "Network Database" window shall appear.

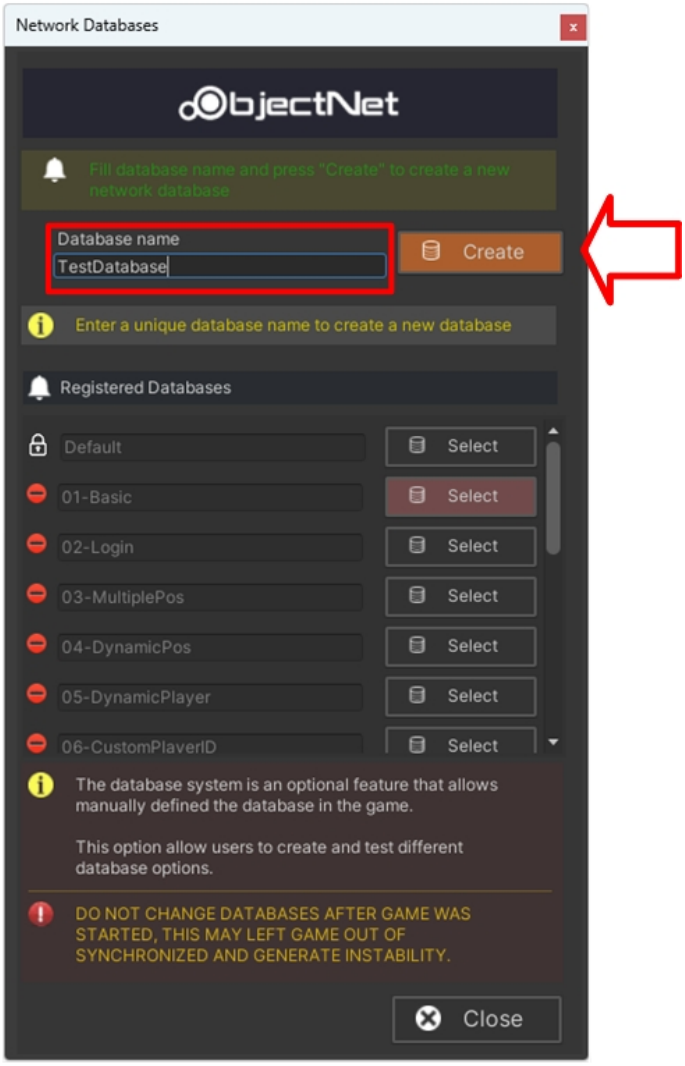


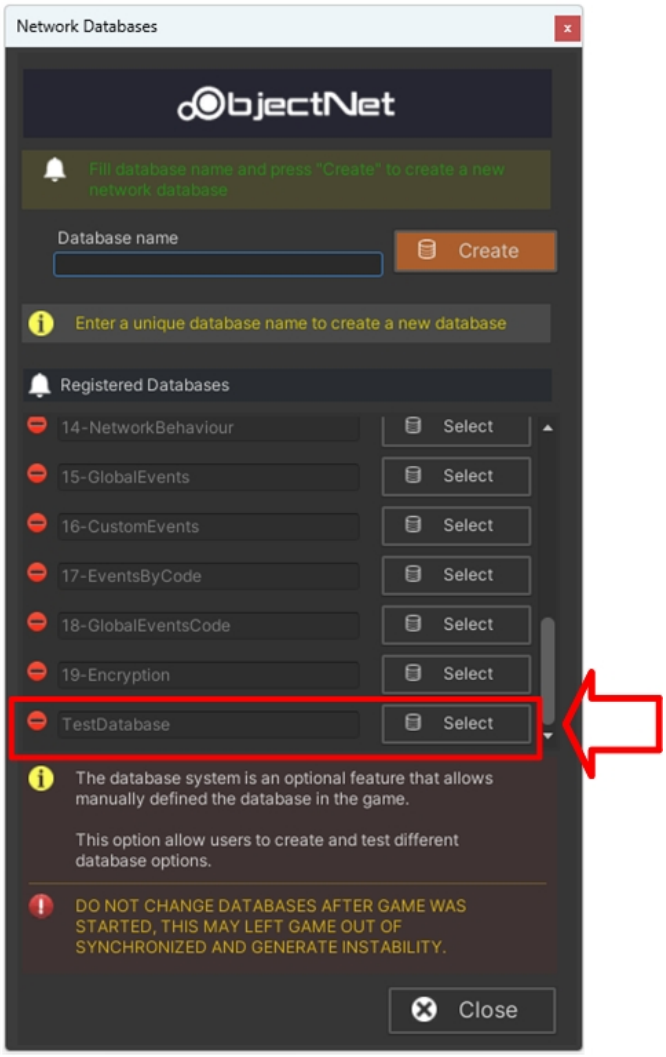
On this window, you can :

- Create a new Database
- Delete an existing Database
- Change selected database

Creating Database

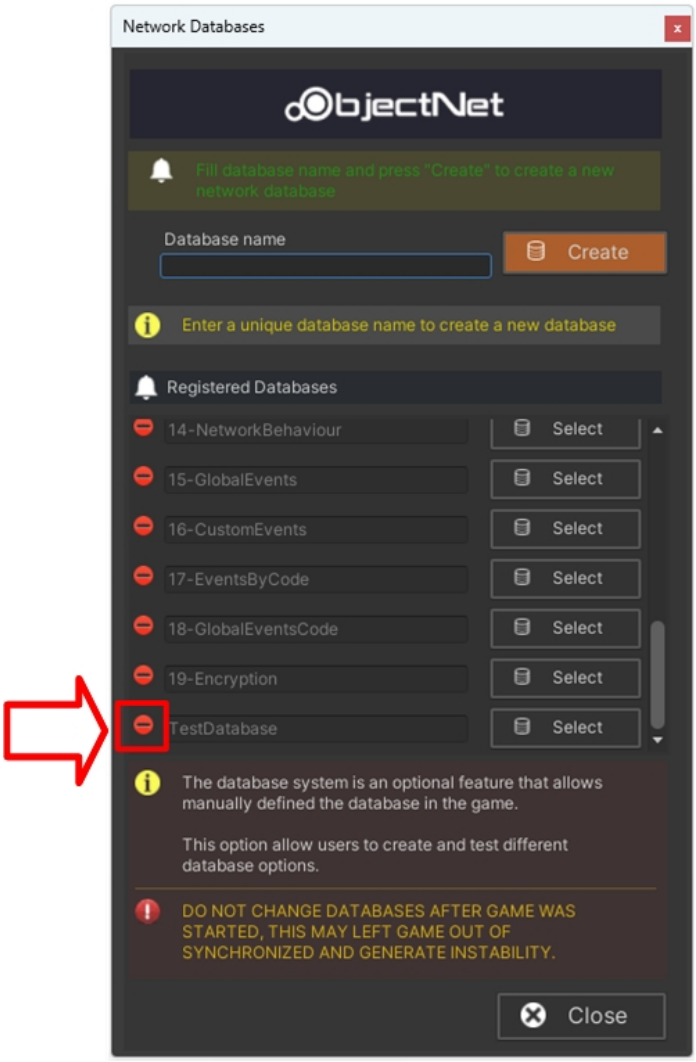
1. To create a new Database you shall fill in the Database Name field and press "Create"
2. A new database shall appear on the last record on the listed database.





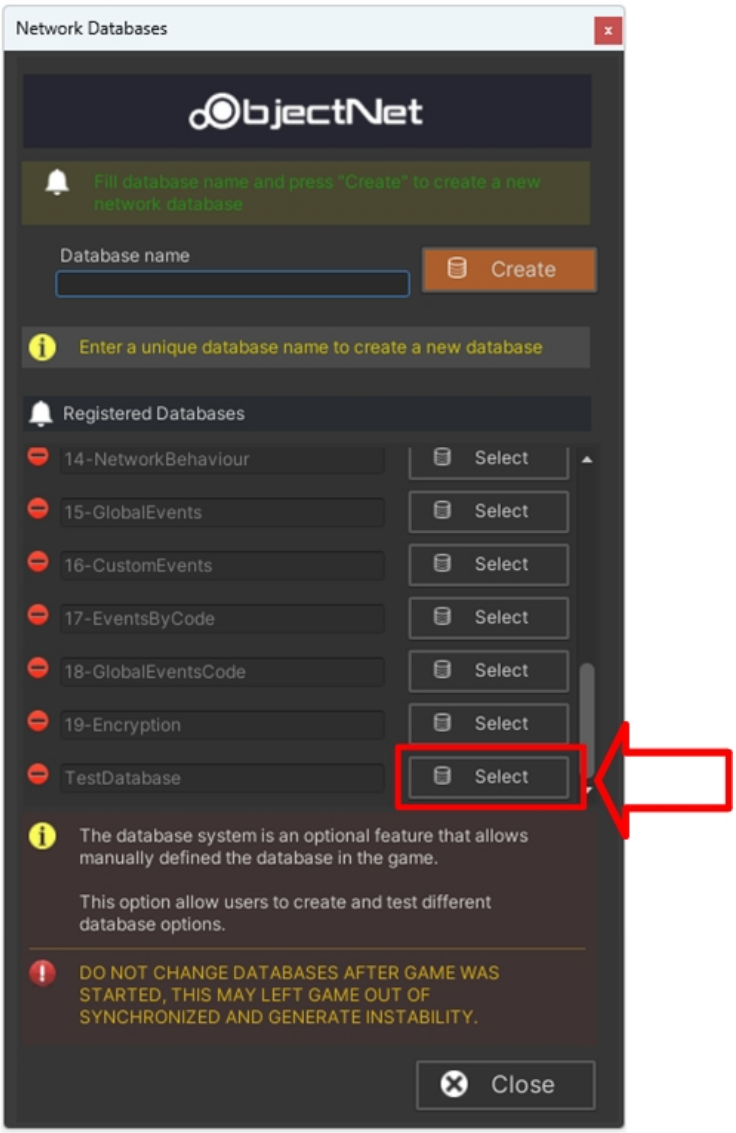
Deleting Database

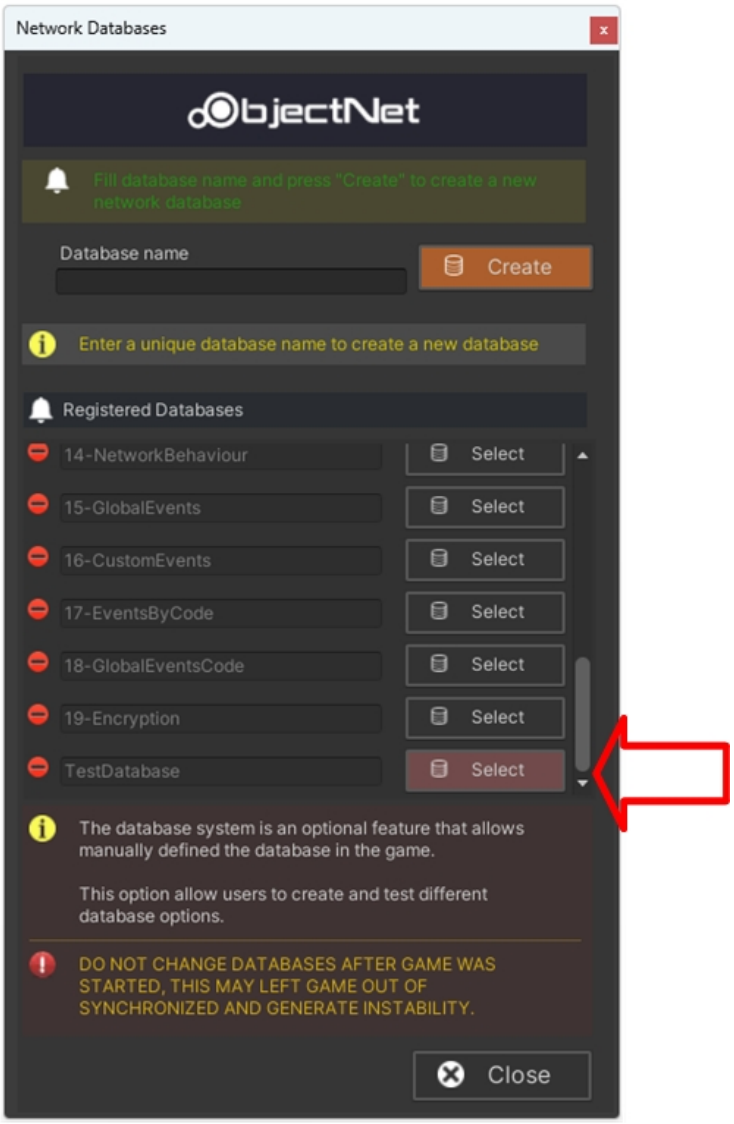
1. Locate the database that you wish to delete
2. Click on the remove button on the left side of the database name



Changing Database

1. Locate the database that you wish to select
2. Click on the select button on the right side of the database name
3. The selected database shall appear in red color


Network Databases
ObjectNet
Fill database name and press "Create" to create a new network database
Database name
Create
Enter a unique database name to create a new database
Registered Databases
14-NetworkBehaviour
Select
15-GlobalEvents
Select
16-CustomEvents
Select
17-EventsByCode
Select
18-GlobalEventsCode
Select
19-Encryption
Select
TestDatabase
Select
The database system is an optional feature that allows manually defined the database in the game.
This option allow users to create and test different database options.
DO NOT CHANGE DATABASES AFTER GAME WAS STARTED, THIS MAY LEFT GAME OUT OF SYNCHRONIZED AND GENERATE INSTABILITY.
Close



Important Note : Database system is a mechanism to version and control the different modes of you game.

You can use different databases on different scenes to provide more than one game mode depending on how players start the game, nonetheless, once the game is started, and NetworkManager is initialized, is not recommended to change the selected database while the game is running for the following reasons.

- You can't ensure that all connected players are using the same database.
- You can't ensure that currently created elements match with the new database.
- And many other reasons.

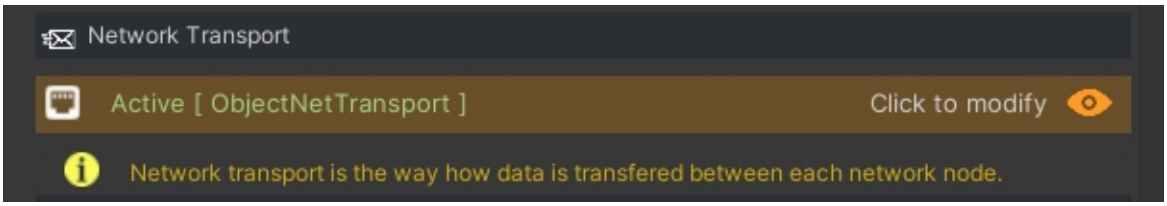
Network Transport

NetworkTransport is the logic used to send and receive messages between each network node in-game.

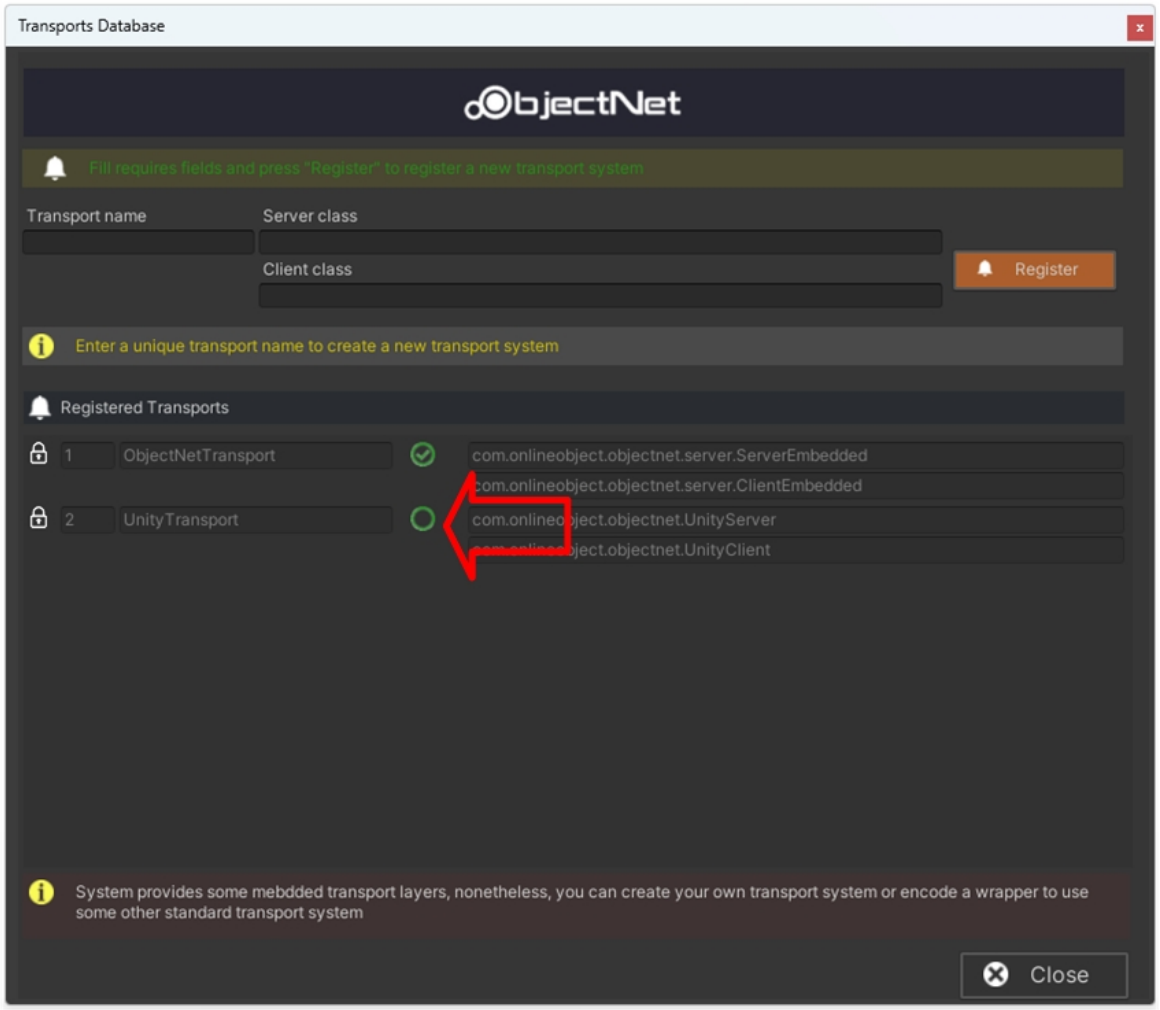
ObjectNet brings with it its own embedded transport system called OnlineObjectTransport, and also native support Unity Transport system.

Modifying Transport

1. To select another transport system click on the eye icon located on the right side of NetworkManager
2. A new "Transport Database" window shall appear.



3. Click on the green selector to change the selected transport system.



Creating custom Transport System

ObjectNet provides the possibility to anyone create and use their own Transport System or implement any 3-part transport system.

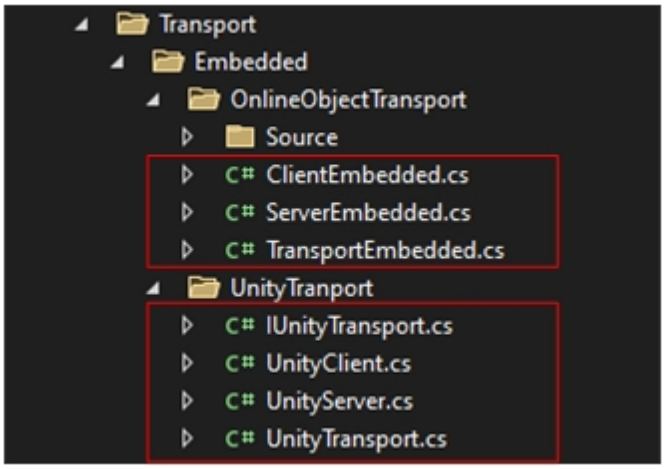
To create a new transport system, you shall implement a minimum of 2 classes inherited from the following interfaces.

- `ITransportClient`
  Provides an implementation of connection from the client perspective.
- `ITransportServer`

Provides an implementation of connection from the server perspective.

The both interfaces inherited from `ITransport` which has the signature on needed methods to be implemented.

You can use as a reference the already contained Transport Systems.



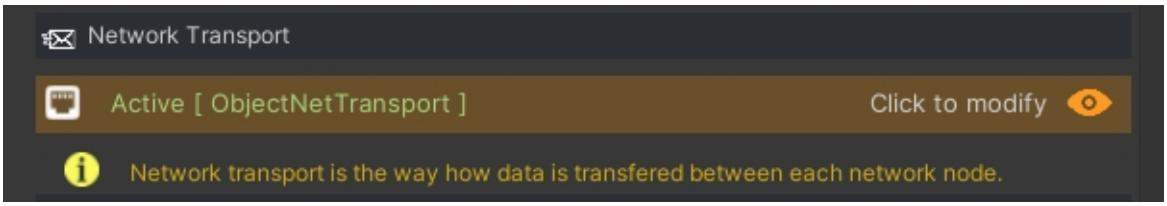
`ClientEmbedded, ServerEmbedded` and `TransportEmbedded` is the wrapper of **OnlineObjectTransport** system to work with **ObjectNet**.

`UnityClient, UnityServer` and `UnityTransport` is the wrapper of UnityTransport system to work with ObjectNet.

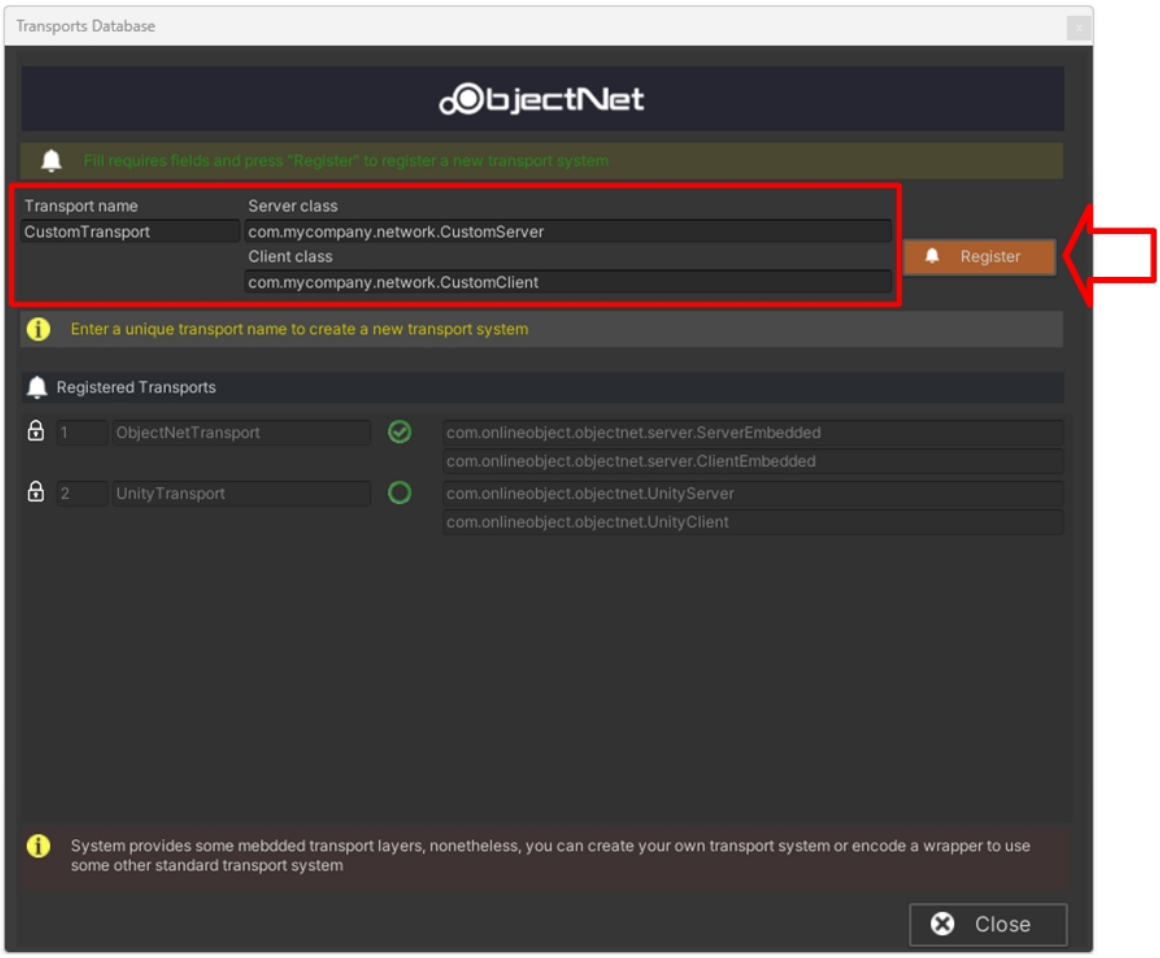
You can use any of them as a reference to implement your transport system.

Registering custom Transport System

1. To select another transport system click on the eye icon located on the right side of **NetworkManager**
2. A new "Transport Database" window shall appear.



3. The following fields must be filled
   - Transport Name :
     - The literal name of the transport system
   - Server Class :
     - Name space and class which implements `ITransportServer` interface.
   - Client Class :
     - Name space and class which implements `ITransportClient` interface.
4. Click on **Register** button

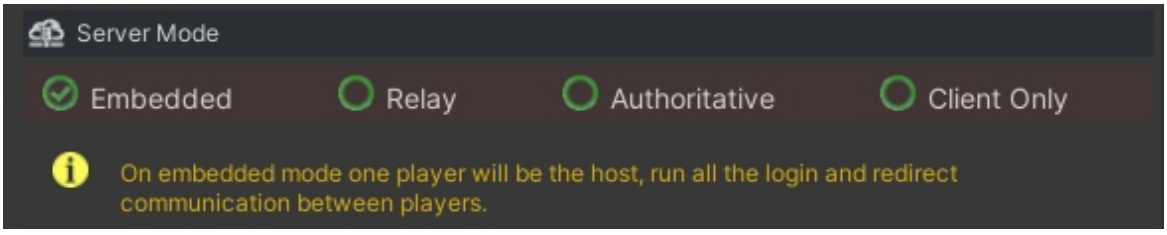


Server Mode

ObjectNet allows you to design your game in different approaches.

ObjectNet server can run in the most popular design modes, currently, ObjectNet can run on the following modes:

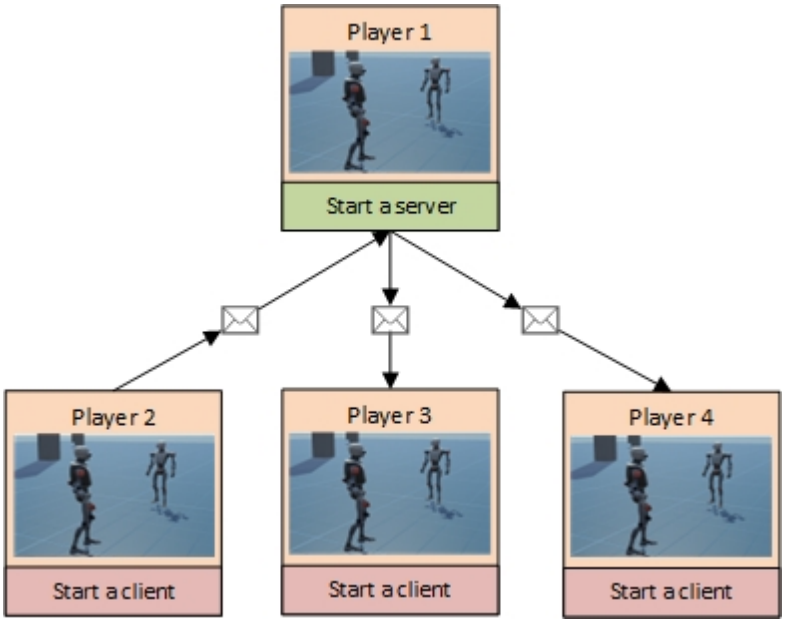
- Embedded
- Relay
- Authoritative
- Client



Embedded Mode

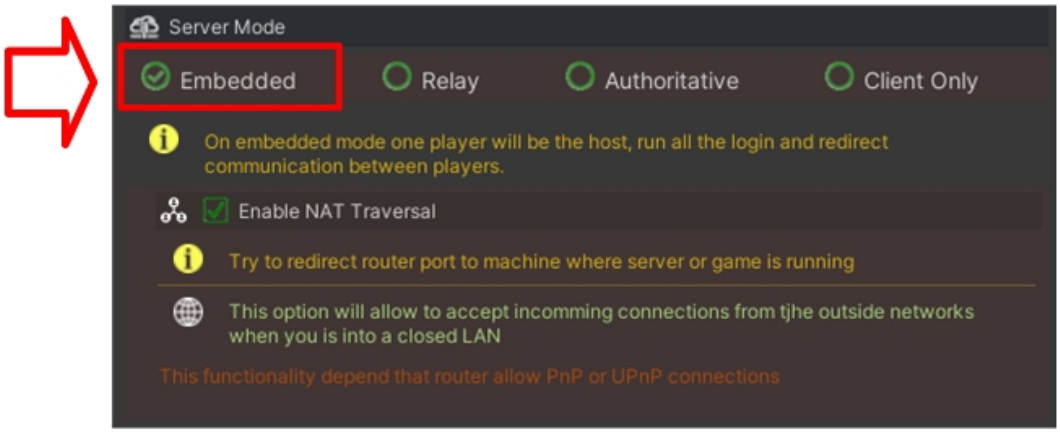
Embedded mode is a mode where one player starts a server at the same time when he will also be a player.

The following design illustrates how this works from a topology point of view.



This mode allows to any player be a server of the game, at the same time that they can be another player. This means that when a player starts a server he will always enter the game being another player.

To use the Embedded mode you must ensure that the "Embedded" option is selected on NetworkManager.

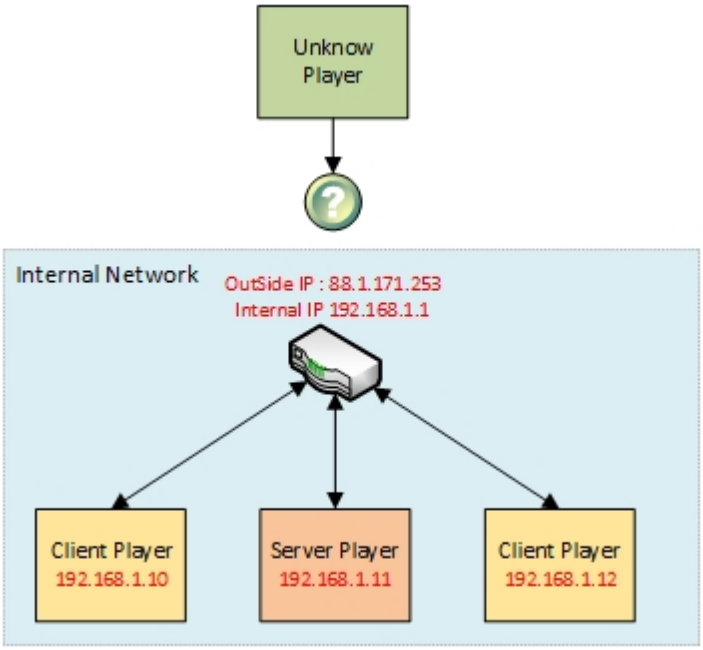


Enable NAT Traversal

NAT Traversal allows players to run the game server in their own internal network and be visible to players out of their own internal LAN.

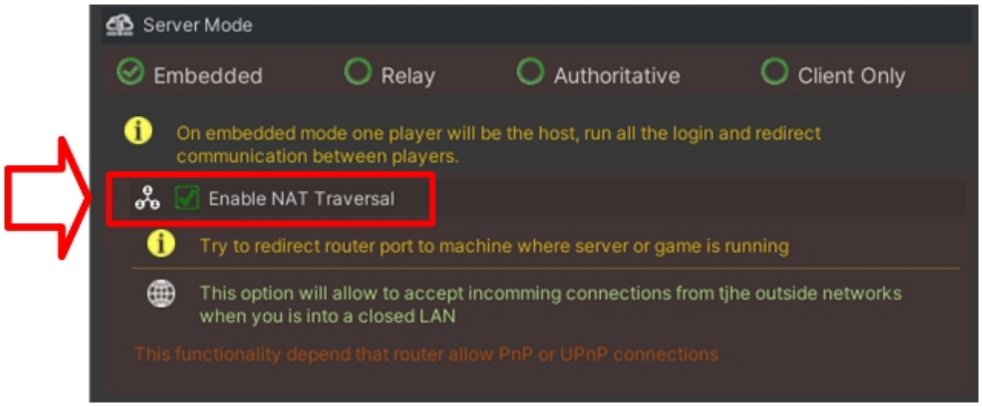
When players run the game on their internal network (by how TCP and UDP are building) players out of this network are unable to reach the machine where the game is running.

The following drawing illustrates a player out of the internal network trying to reach the server where the server is running.



Even if the **Unknown Player** knows the router IP Address, they are unable to reach machine 192.168.1.11 where the game server is running.

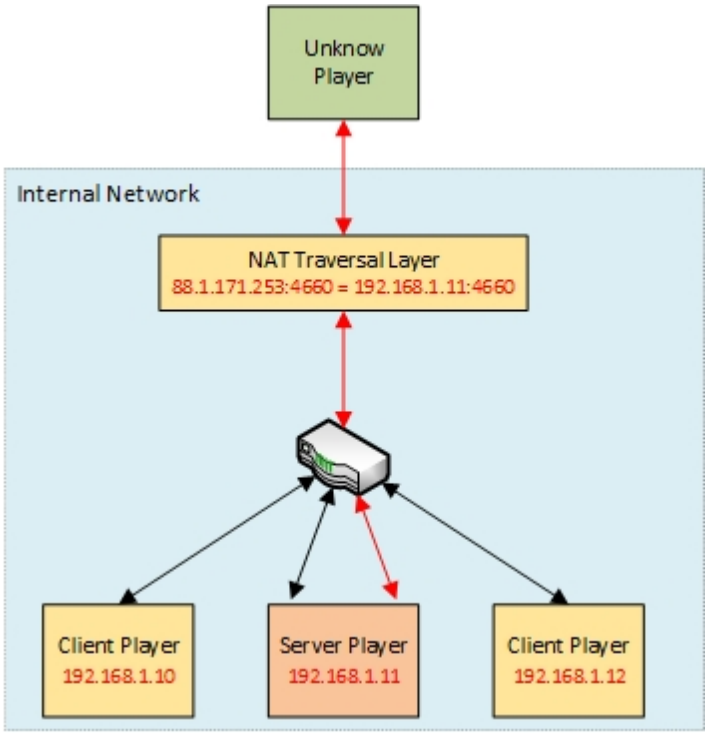
You can **Enable Nat Traversal** by checking the option on **NetworkManager**.



When NAT Traversal is enabled, the system will try to open the game port used to accept client connections on the server.

The system will map the Server Player port in the router targeting the same port on the router, this means that when some player out of the internal network tries to connect to the public IP on the port used by Server Player, the communication will be redirected to the machine where the server is running.

The following drawing illustrates how NAT Traversal works.



This tells the router that redirect communication from an unknown player to IP 192.168.1.11 and vice-versa.

In this mode, players can run the server on their internal network and accept players on the other network to play on the same server.

Relay Mode

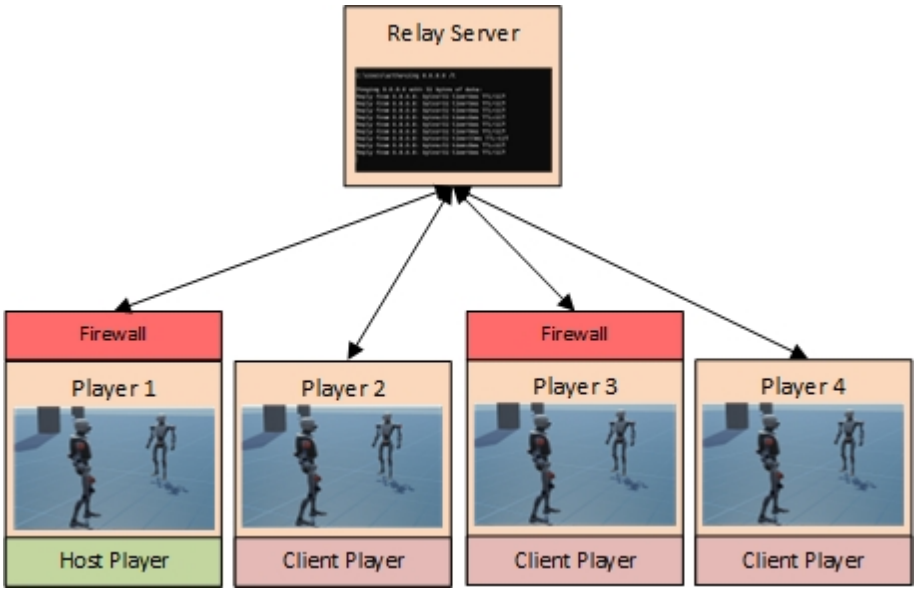
Relay mode is a mode where a server works only to redirect communication between players.

This mode is pretty useful to allow players on a private network to play online multiplayer games on a dedicated server host. Nonetheless, in this mode, one player needs to execute all game logic since the server only works to redirect communication.

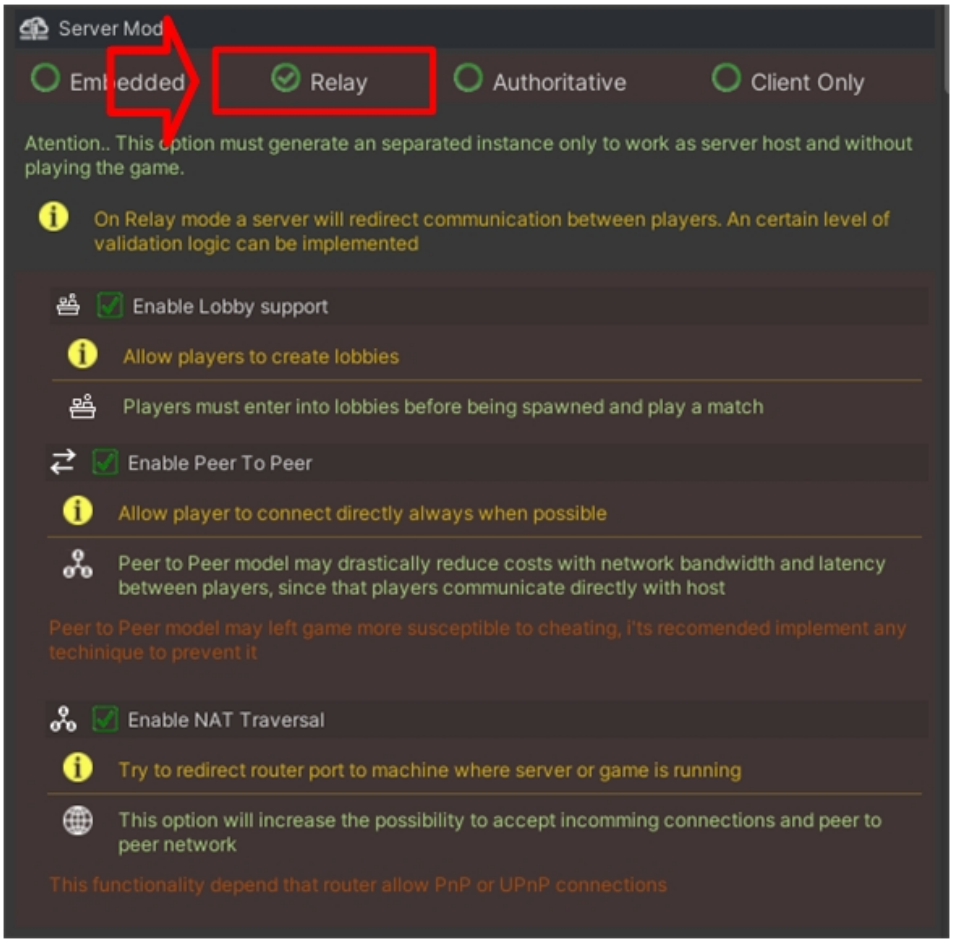
This mode is especially useful to allow players behind firewalls to play online games.

On requirement to allow Relay mode is that server must be on a public network accessible to all players.

The following design illustrates how this works from a topology point of view.



To use Relay mode you must ensure that the "Relay" option is selected on NetworkManager.



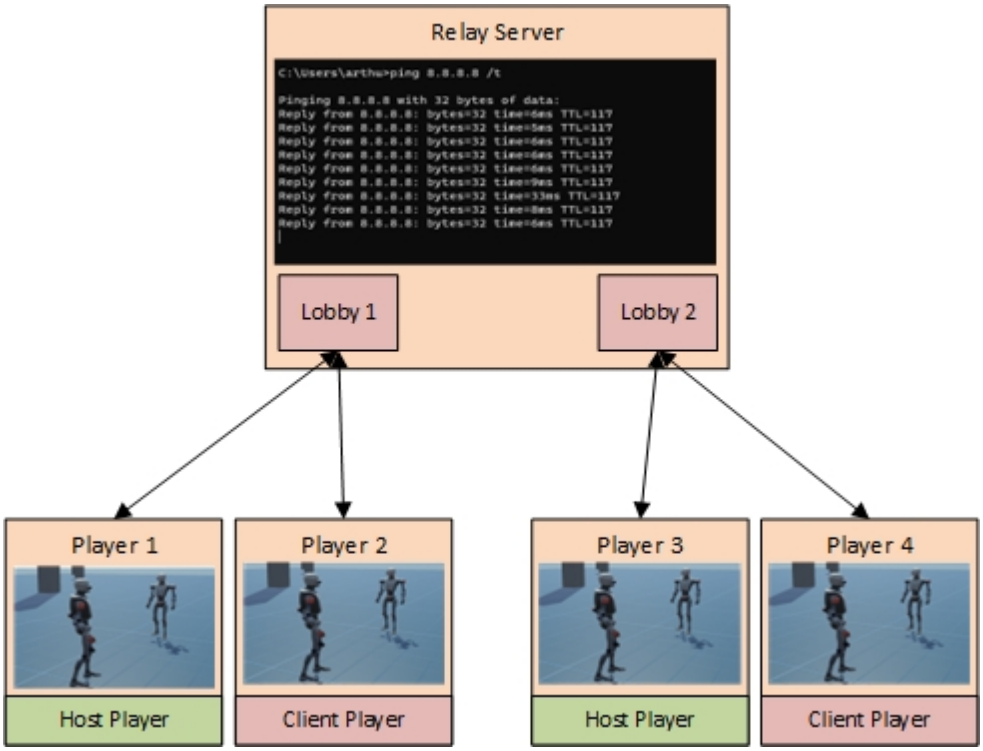
Lobby System

A Lobby System is a way to support multiple players, playing in different groups of players when connected to the same server.

The Lobby System works only on Relay mode, the reason is that in other modes (Embedded and Authoritative) the system shall always handle and send information from all players to all players.

In short, by default Relay mode is the only mode that supports the possibility to create Lobbies, this option possibility to use the same server to run multiple players one separated from each other.

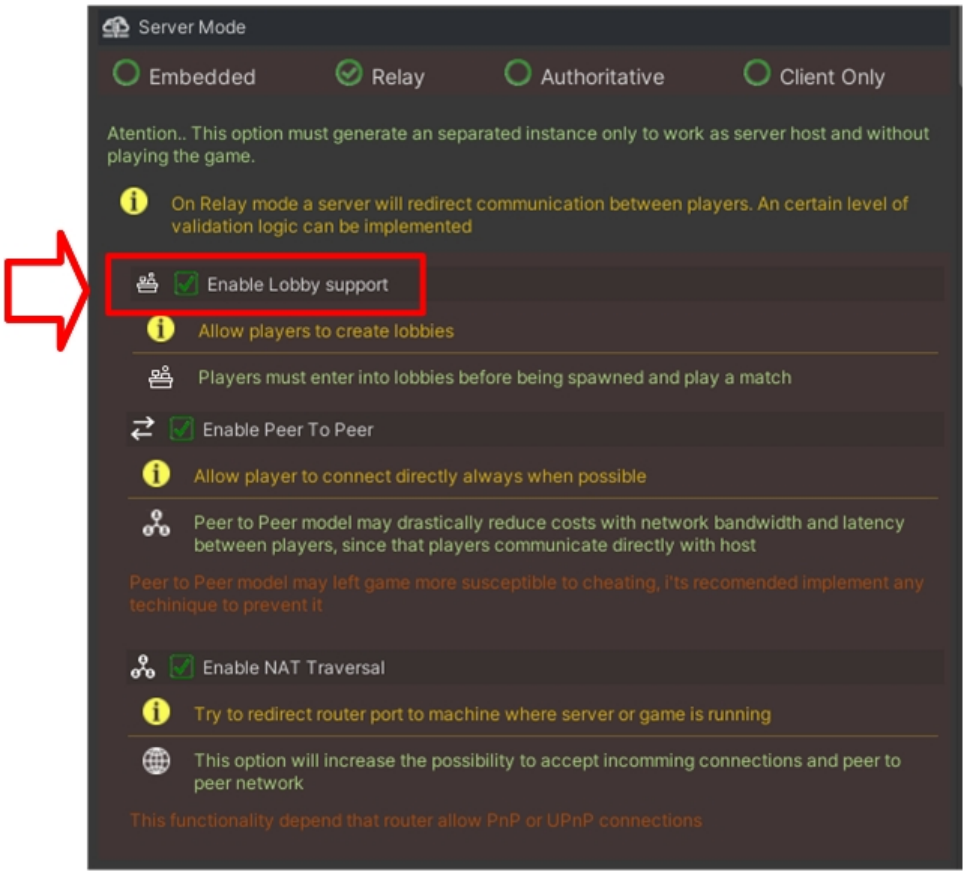
The following drawing illustrates how Lobbies isolates the group of players, on this scope players will only see events from their groups, instead of seeing all events of Relay Server.



In this scenario, Player1 and Player2 will see only events that came from Lobby1, and Player3 and Player4 will see only events that came from Lobby2.

This option is pretty useful if your game has a small number of players for each match and there's no memory or computation difference from having the same number of players with or without lobbies.

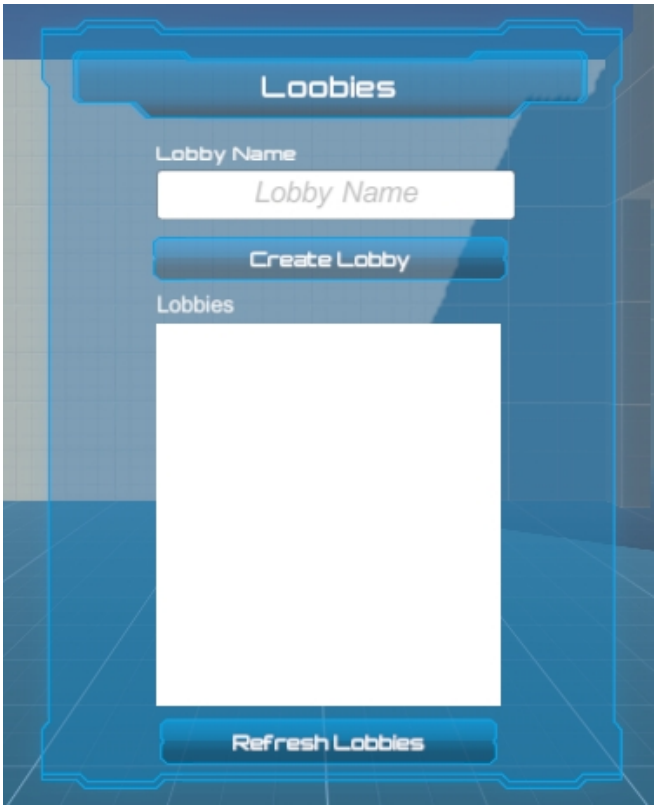
You can Enable Lobby support by checking the option on NetworkManager.



When Lobby support is enabled, a lobby must be created before the player enters on any match, this means that the player has the option to create a new lobby or enter into an existing one.

ObjectNet container is an example scene called "XX - Lobbies" which has a clear implementation illustrating how to include lobby support in your own game.

To create a Lobby, fill in the lobby name and press "Create Lobby", this will create a lobby and spawn a player into this new lobby.



When a player enters a server with existing lobbies, he needs to press "Refresh Lobbies" and a list of existent lobbies shall appear with the option to "Join".

Peer to Peer

Peer-to-peer allows sending direct messages to players instead of passing through the Relay Server.

Peer To Peer works only in Relay mode since other modes (Embedded and Authoritative) there is no make sense in having direct communication between peers.

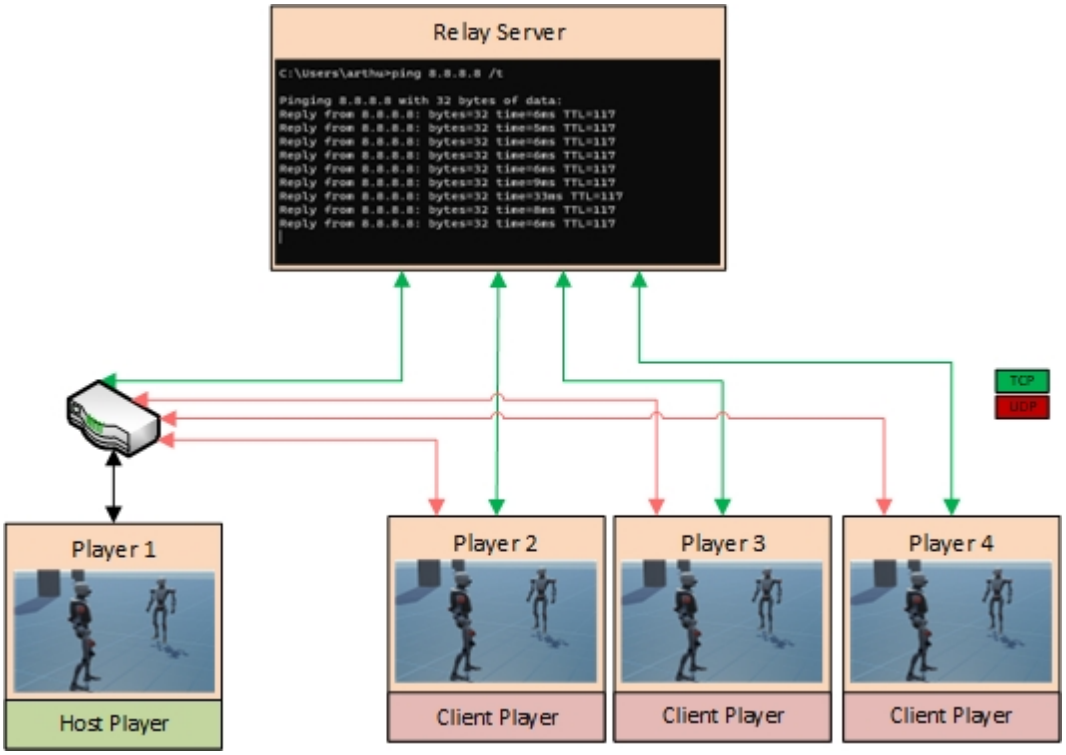
But how ObjectNet work with Peer to Peer?

In relay mode, one player is selected to be the host of the match, this means that all logic shall run on this player. What ObjectNet will do is check if this player is visible outside (public IP or NAT Traversal supported by the router) and communicate directly with this player instead of sending it to the Relay server.

Not only unreliable messages will take this route, but reliable messages will still be sent using the Relay server, this has a significant reduction of traffic messages and as a consequence, shall reduce the needed traffic on your game.

Another good collateral effect of peer-to-peer use is the reduction of latency since messages will be sent directly to the host player instead of passing through the Relay server.

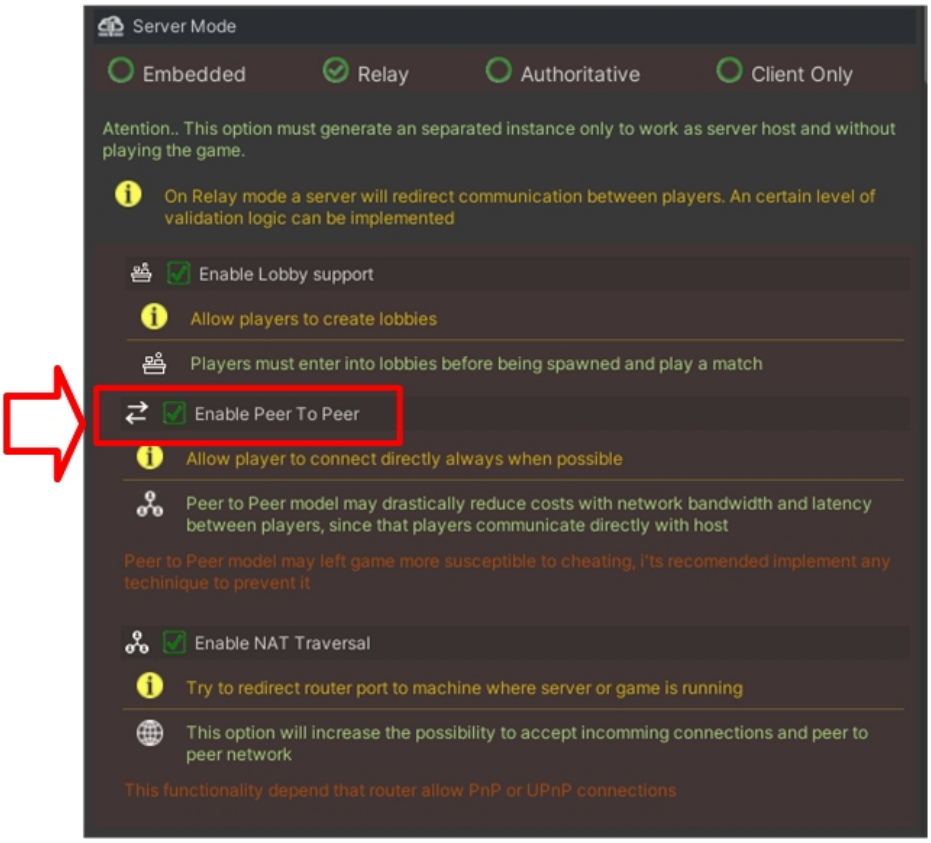
The following drawing illustrates how Lobbies on Peer to Peer works.



In this scenario, Player1 is the host of the match, and Player2, Player3, and Player4 will be notified that Player1 is network visible and try to send unreliable messages directly to Player1.

This may provide an amazing reduction of your costs with bandwidth if you are hosting your game into a hosting service such as Azure, Amazon, or any other.

You can Enable Peer to Peer by checking the option on NetworkManager.



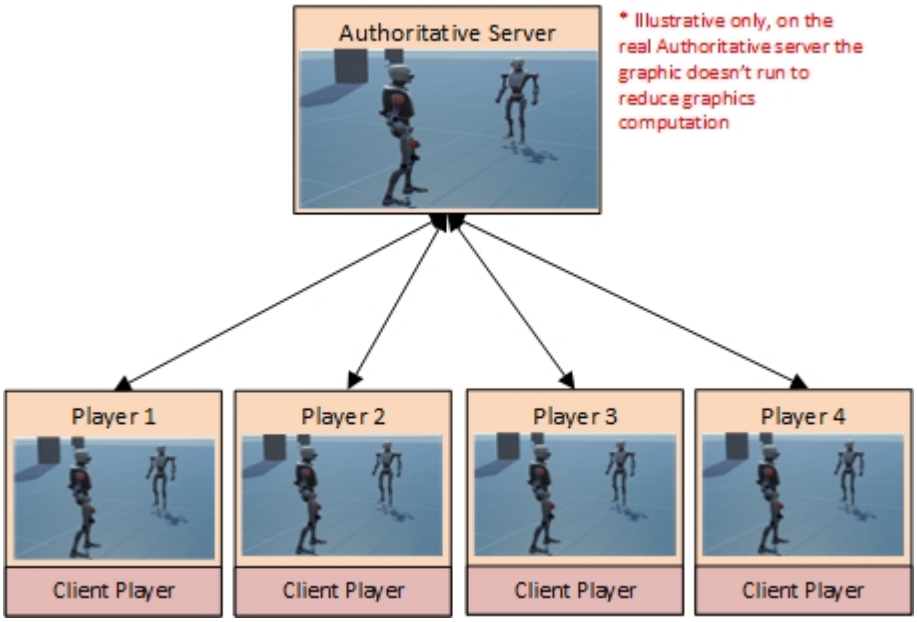
Authoritative

Authoritative mode is a mode where all game logic and physics run on the server and the client doesn't make any decision regarding the game.

Authoritative mode is the more trusted way to run the game without suffering from cheating, nonetheless, is high CPU and memory expensive since a copy of the game runs on the server.

In Authoritative mode, each player will send his information to the server, server will execute all logic and decisions and send the real status of the game back to the players.

The following drawing illustrates how the Authoritative server works.



In this scenario, all players send their information to the server, and the server processes all physics and game logic and sends the current status of the game back to all players.

To use the Authoritative mode you must ensure that the "Authoritative" option is selected on NetworkManager.

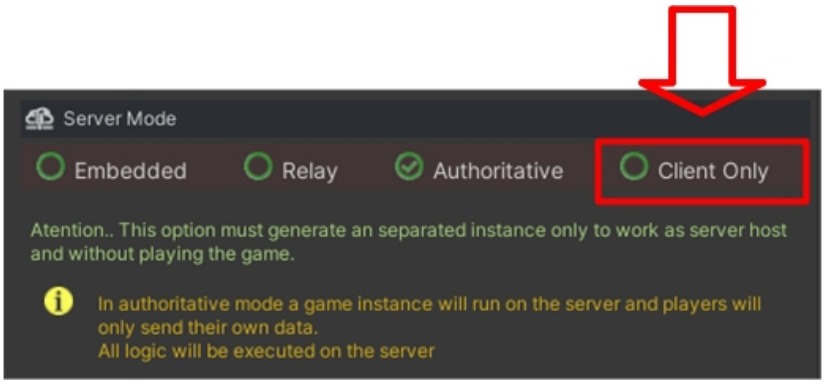
Client Only

Client Only is a mode that does not allow to any player run a server over any hypothesis.

Client Only mode must be used when you have an official and your game build not offer the possibility to run a LAN or personal server.

In Client Only mode, players must have a server to connect always, nonetheless, you can also provide a copy of your server to be hosted by clients, but needs to be a separate build of Authoritative or Relay modes.

To use Client Only mode you must ensure that the "Client Only" option is selected on NetworkManager.

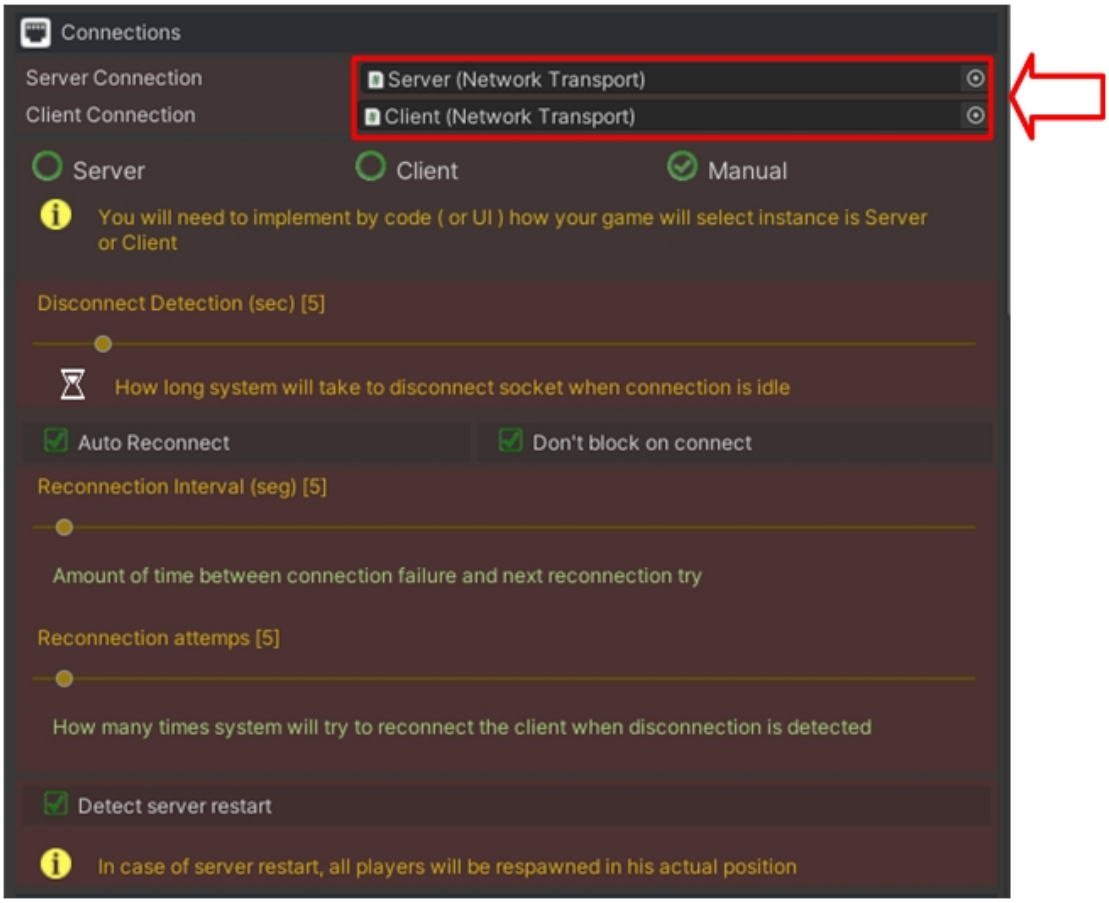


Connections

ObjectNet uses two connections to work, a Server connection and a Client connection. The server connection is the network instance that runs on the server side to handle all connected clients, on the other hand, the client connection is the entity used to enter into a game as invited.

Server Connection : TCP/UDP Socket used on the server side to accept all incoming data on the server side.

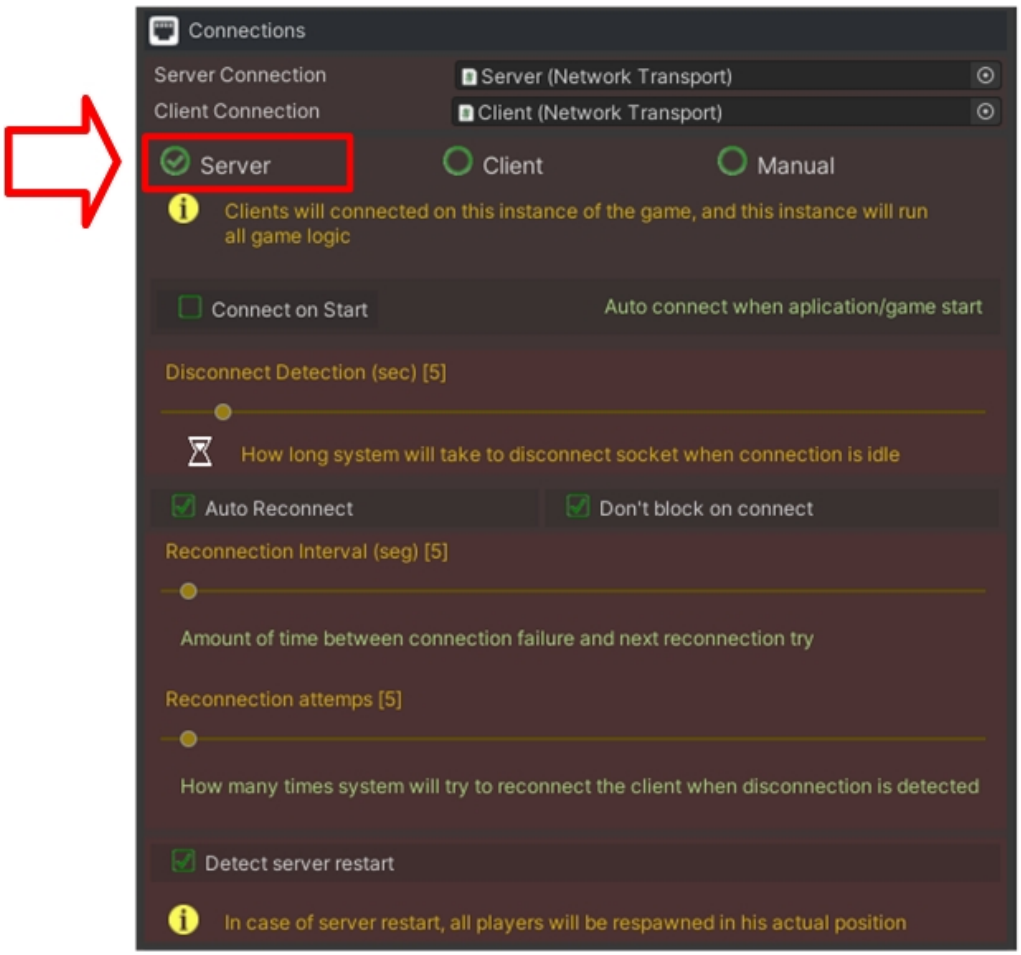
Client Connection : TCP/UDP connection used to enter into a server as client.



Server Connection is a mode running on instances directly as server, this option shall be used on build to run server only on Authoritative or Relay server modes. This option can also be used to run a server on any

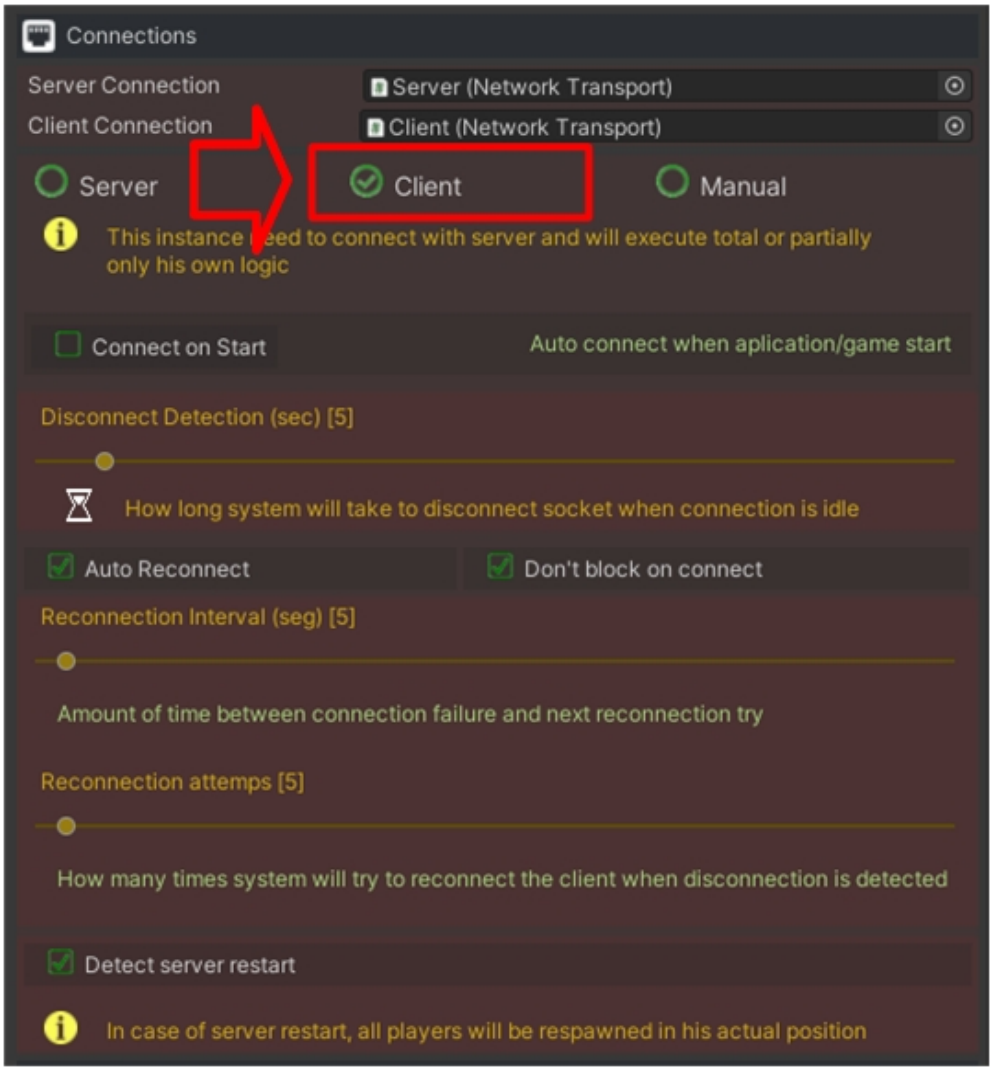
mode designed by the developer.

Server mode allow to select how to listen for incoming connections over network, depending how you host you server you may need a different type of binding (see Server Connection)

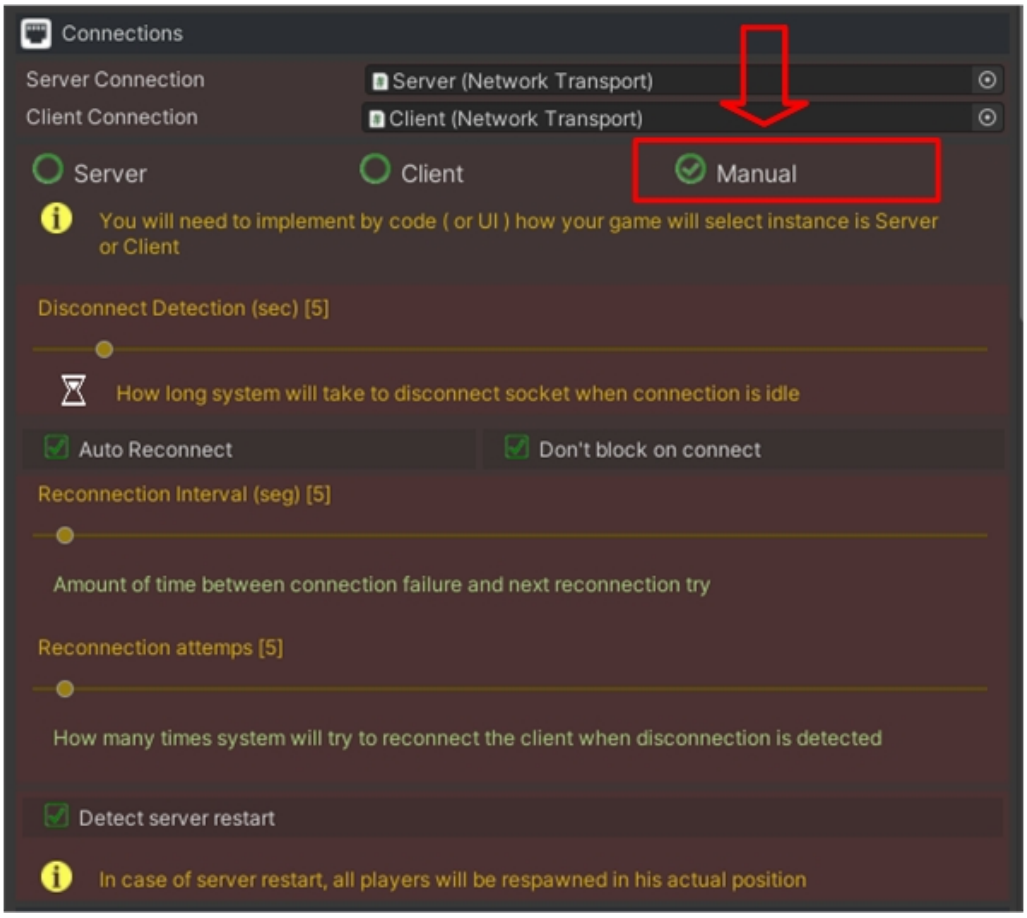


Client Connection is a mode running on instances directly as client, this option shall be used on game compiled to run client-only instances. A game compiled with Client Connection needs a server to connect to.

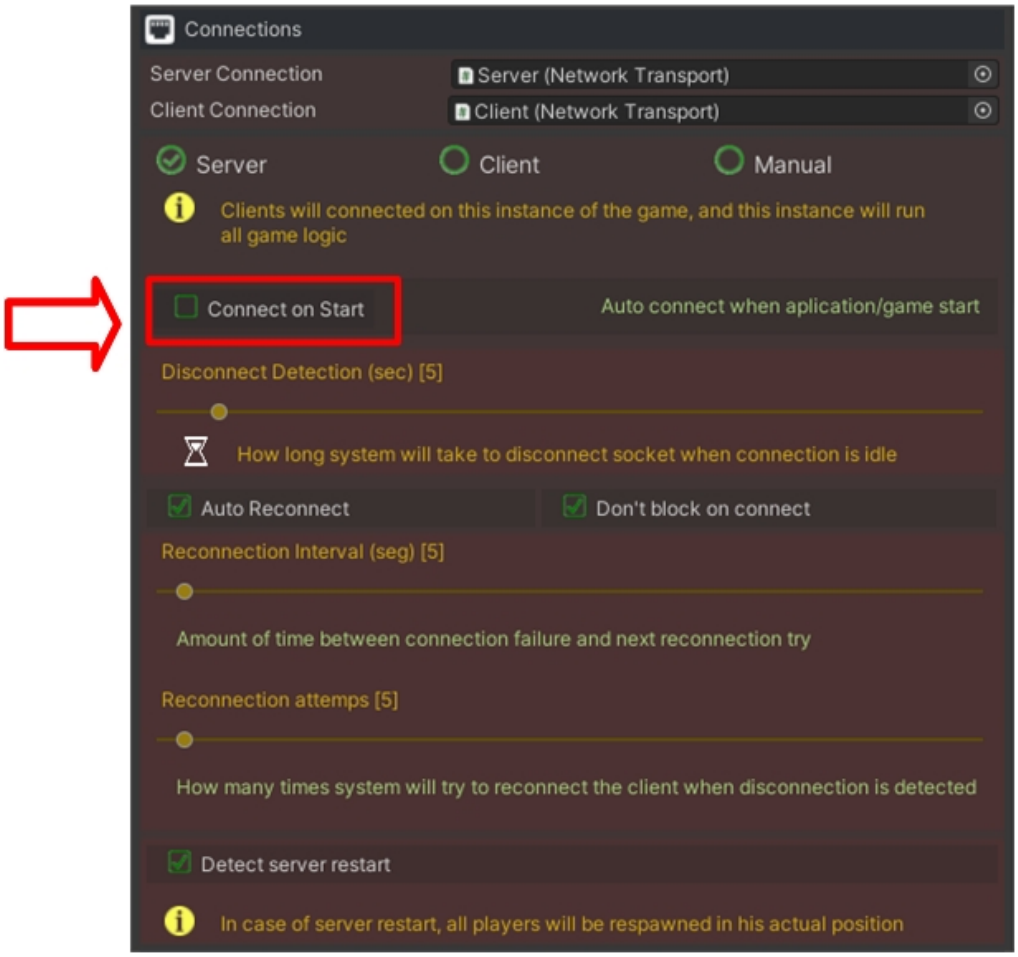
Client connections allow to statically or dynamic select the IP address where server is located (see Client Connection)



Manual Mode is a mode used to let the client decide which mode he wishes to run after the game starts. This option shall be used when the user needs to run the Server on the Client depending on how he wishes to run, is commonly used to run embedded mode where the same player is Server and Client at the same time.



The option "Connect on Start" starts the socket connection automatically when the game is started.

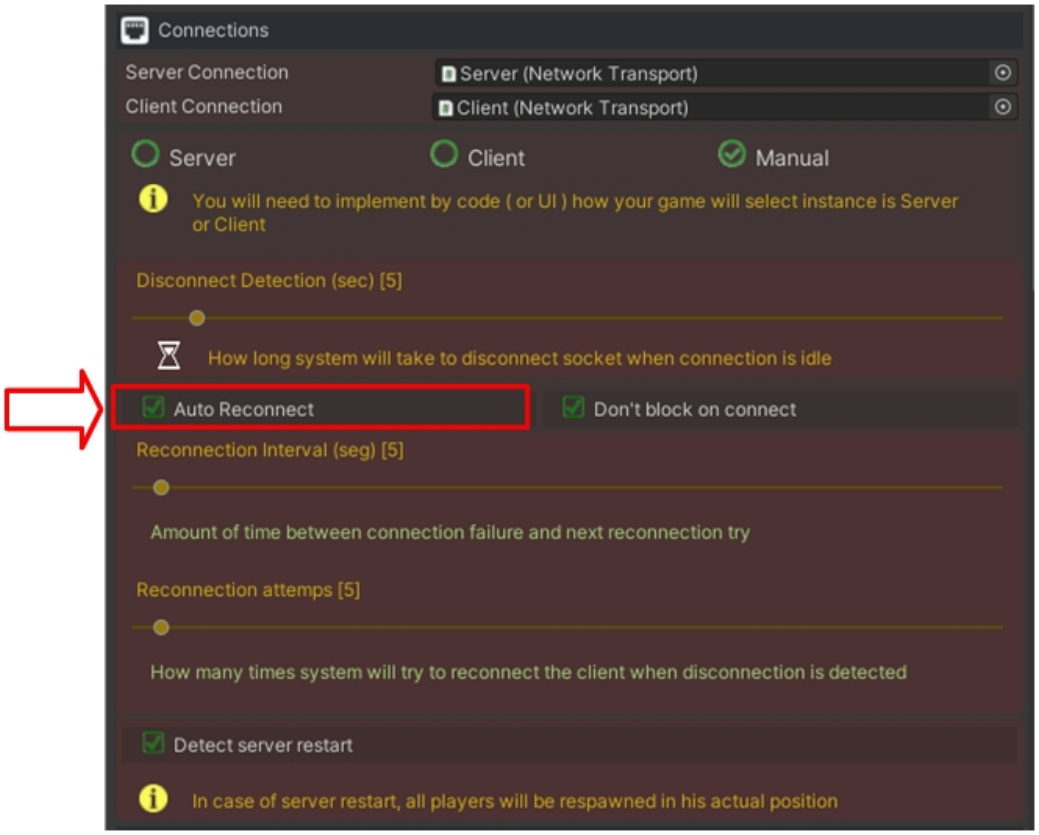


This option is useful and mandatory when a dedicated server is built for Authoritative and Relay modes, otherwise, the server will not be started.

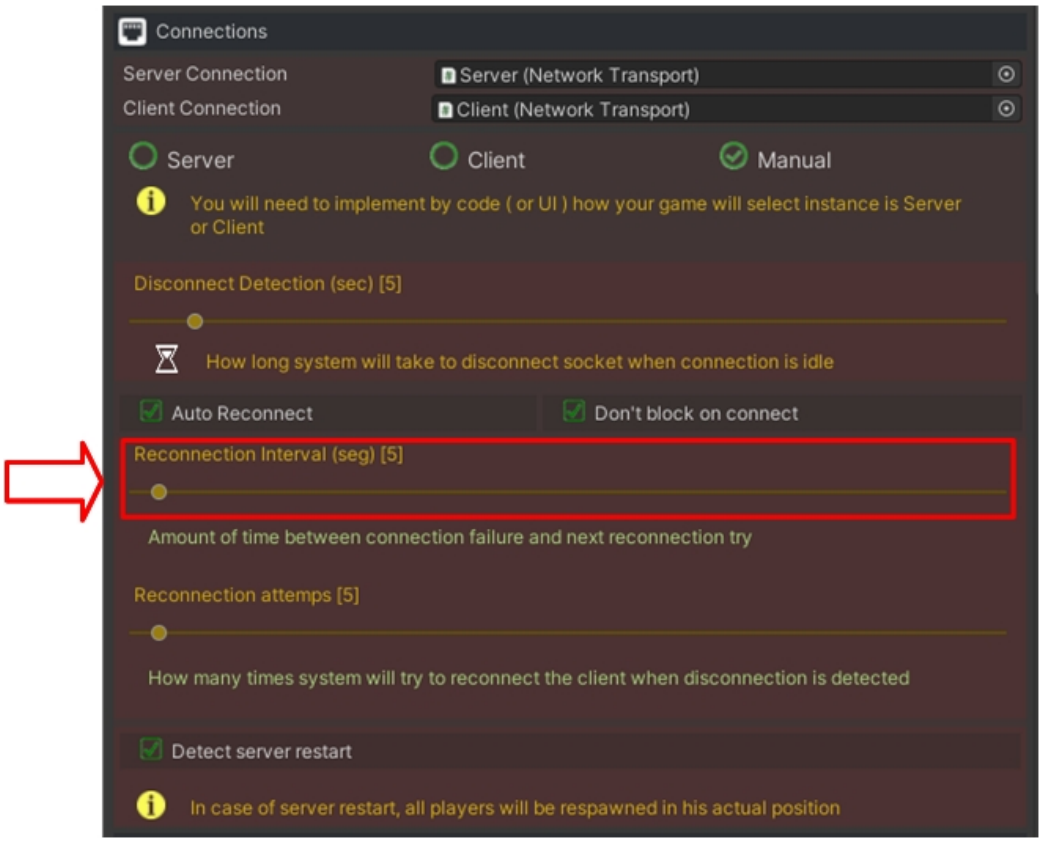
Auto Reconnect

ObjectNet allows to clients auto-reconnect on the server when the connection is lost. This option can be enabled by checking the "Auto Reconnect" option.

When activated, the client will try to reconnect at the server after the time defined on the "Reconnection Interval" configuration.

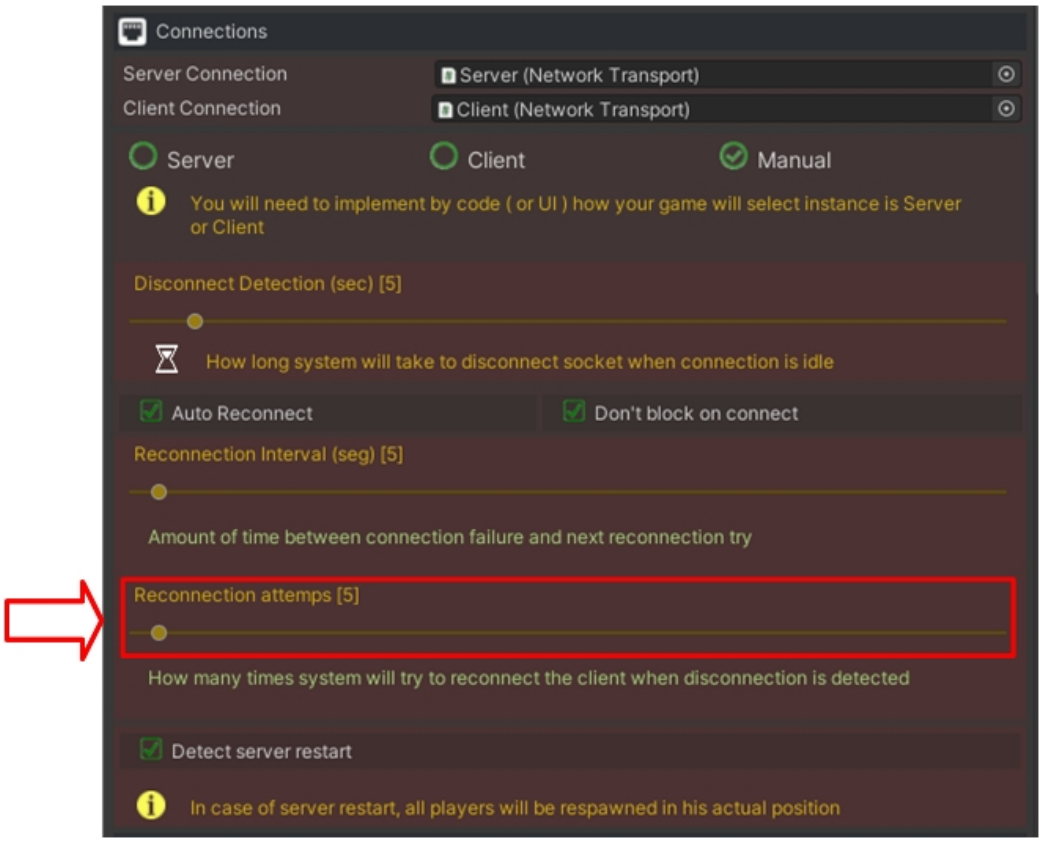


Once disconnection is detected on the client side, the system will wait the amount of time defined on "Reconnection Interval" before trying to reconnect, and in case of failure will try this again up to reconnecting is established..

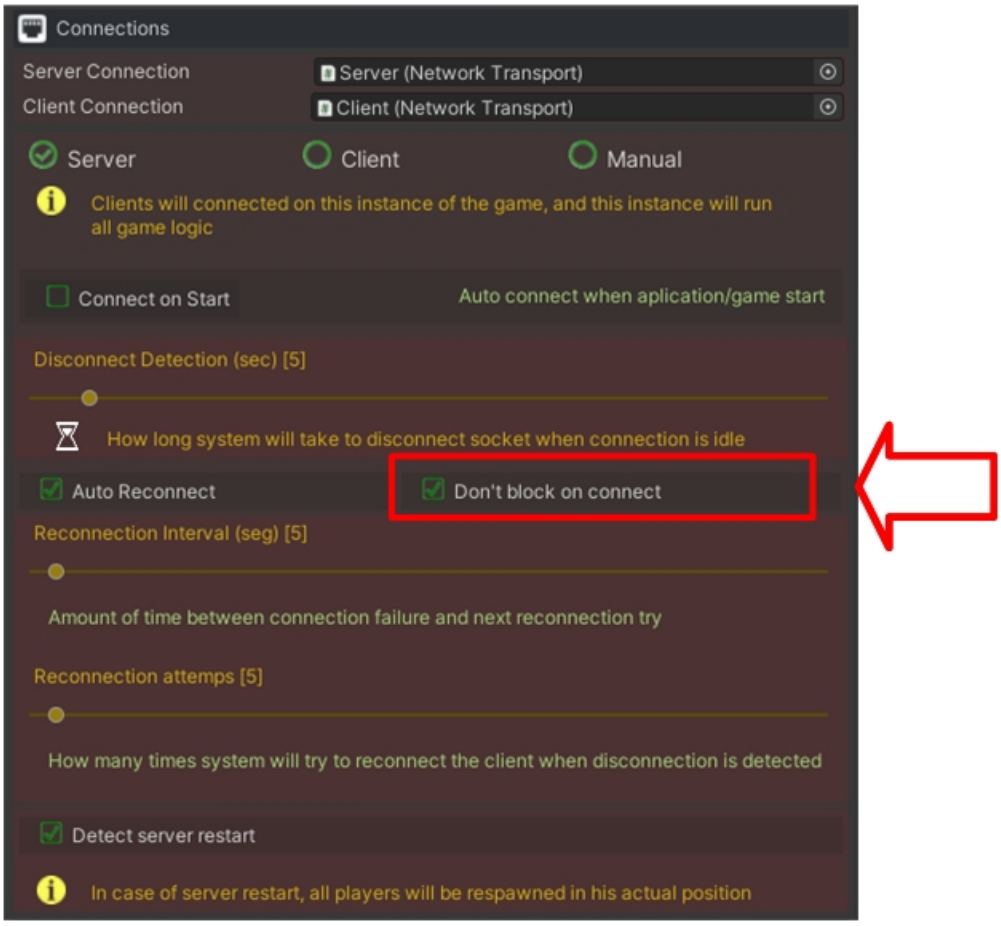


During the re-connection procedure, the system will try to reconnect a certain number of times before

stopping re-connection, this can be configured on the "Reconnection Attempts" slider.



The option "Don't Block UI" starts the socket connection into a separate coroutine to avoid blocking user interact operations.



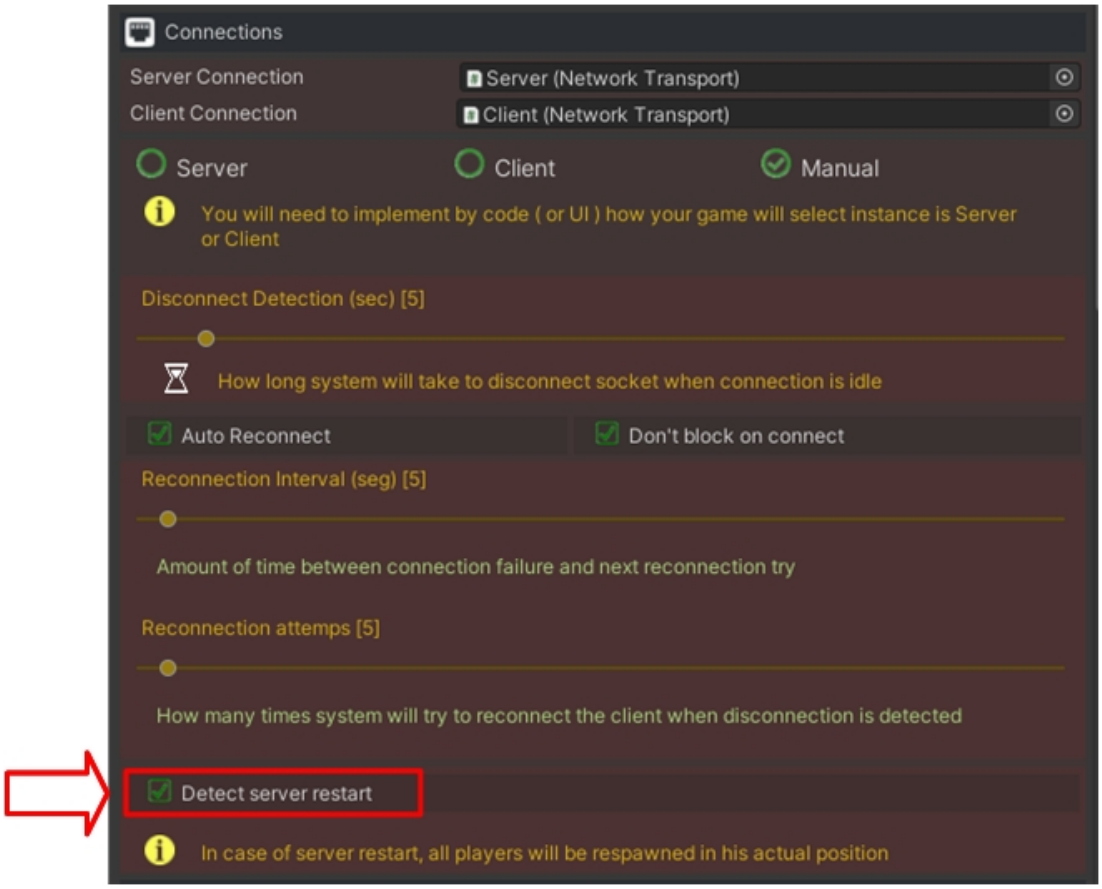
This option should be used if some connection process UI will be displayed to the user, otherwise, UI will freeze while the connection process is in progress.

Detect server restart

ObjectNet provides a facility to clients detect when the server is restarted. This can be useful to finish the game on the client side when the client reconnects after the server is restarted.

ObjectNet can bring the player back restoring all player status after the connection is the player reconnects after being released from the server side, nonetheless, this can't be done if the server restarts due to the fact that the server lost all game status. In this case, the client needs to leave the match is a server restart if detected.

You can Detect server restart by checking the option on NetworkManager.

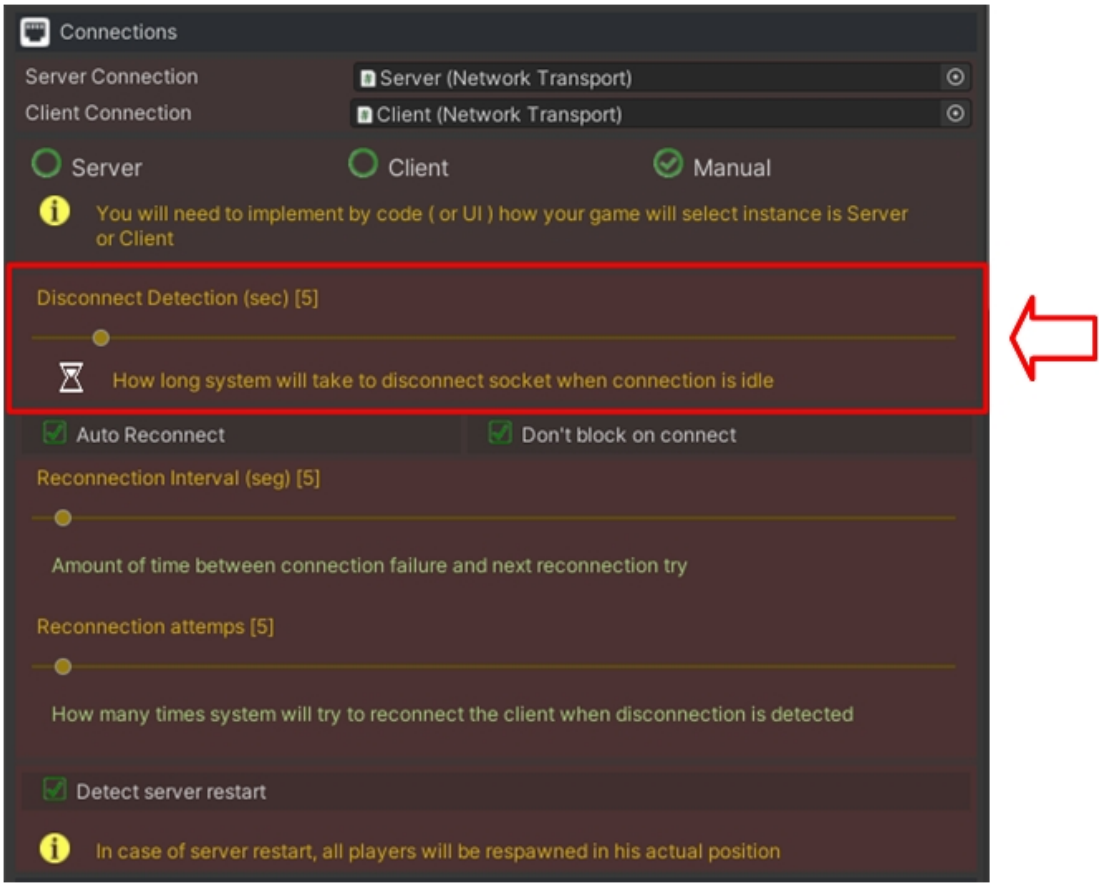


When activated, an event will be raised when a server restart is detected. This event can be caught on **NetworkEventsManager** or by the API method `OnServerRestarted` on `NetworkTransport` class (see API reference to more information's).

Disconnect Detection

ObjectNet allows users to decide how much time the server will take to detect when the client disconnects from the server. This option can be useful to not disconnect the client when the connection is lost and reconnected in a short period of time due to some network problem.

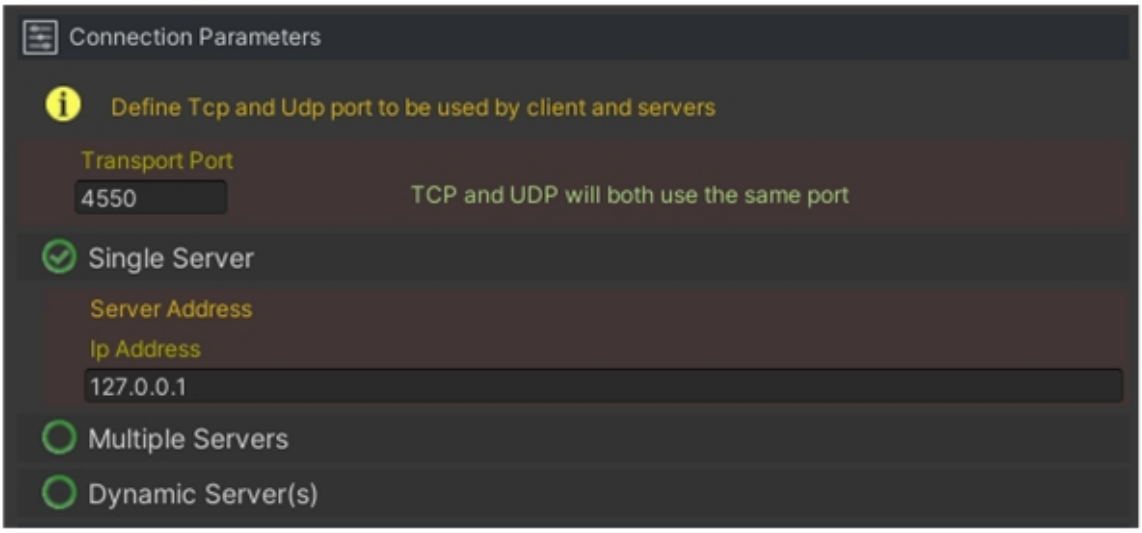
ObjectNet when using OnlineObjectTranport implements a keep-alive message to detect if the client is still connected to the server, when this message does not arrive after this period of time the system assumes that this client was disconnected.



To update this value you need to slide the horizontal according to the desired value, and each unity represents 1 second on time.

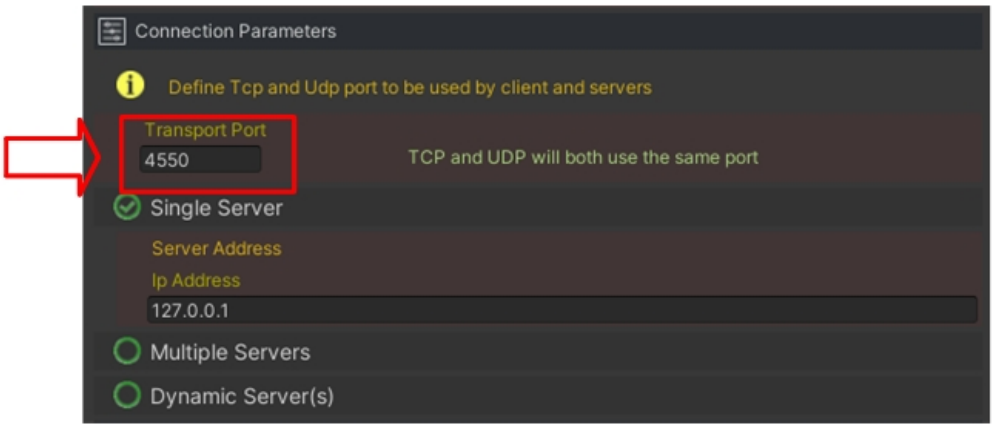
Client Connection

ObjectNet provides a flexible interface to handle servers, this can be configured on the "Connection Parameters" section.



Transport port

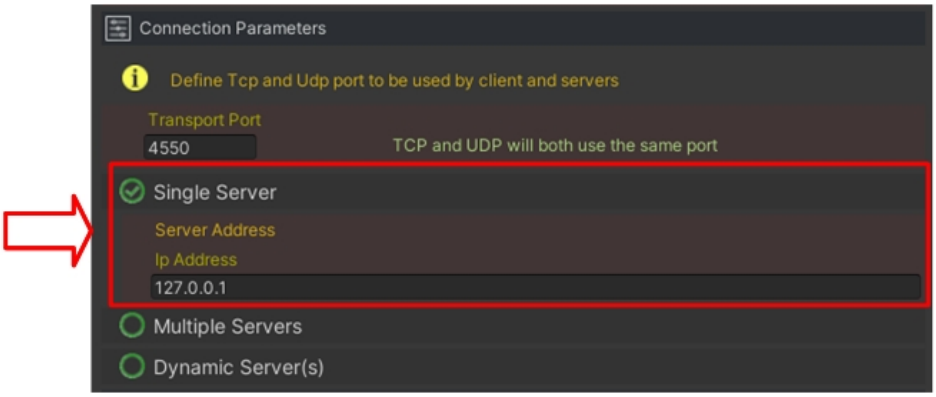
You can define the port used on your game by setting the port on the "Transport Port" field.



Once defined, both transport systems (OnlineObjectTranport and UnityTransport) will use this port as a reliable and unreliable delivery method. Nonetheless, ObjectNet is flexible and in case your custom transport system needs different ports for reliable and unreliable messages you can define this by setting `TransportPortType` provided by `ITransport` interface (see API reference for more information's).

Single Server

Single Server mode tells to client to connect to a specific IP.



This option uses a unique and unchangeable server address and uses hard code address information as the server target. This option should only be used if you have some type of internal load balance to redirect communication into some Kubernetes or other technology.

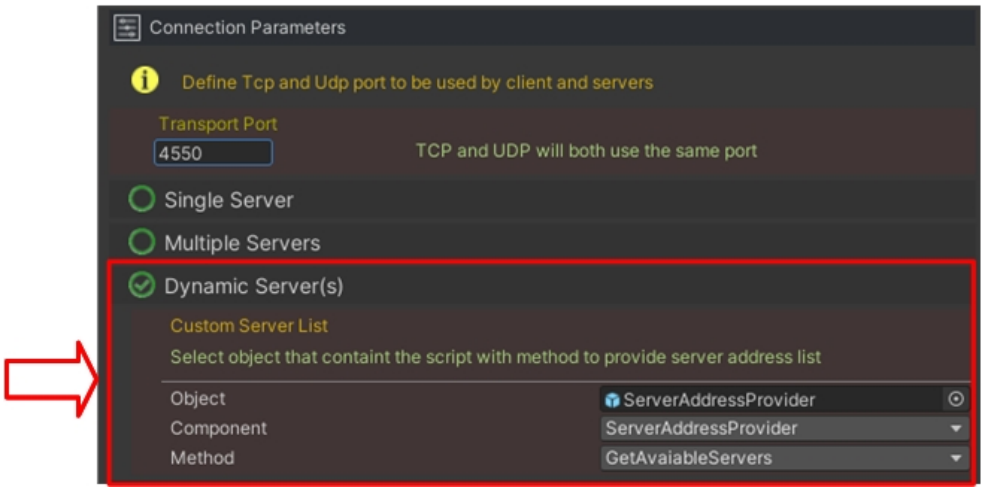
Multiple Servers

Multiple Servers mode provide a list of possible server to the client connect.

In this scenario, the system will try to connect with the server host with the lowest ping possible. This ensures that players will also be connected to the best server possible.

Dynamic Server(s)

Dynamic Server(s) mode provides a smart way to decide which server players will try to connect.



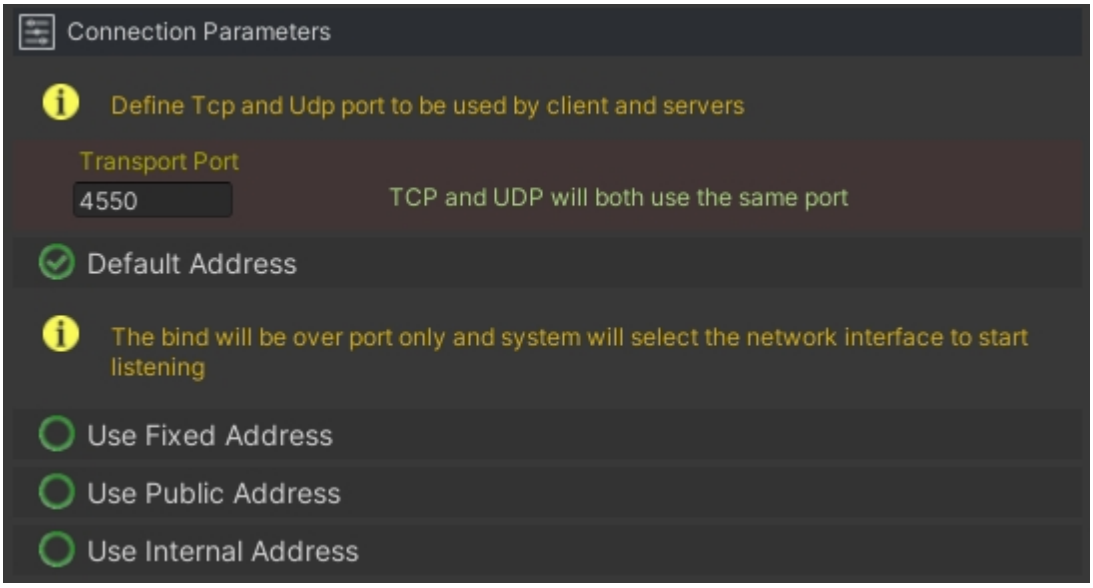
In this scenario, developers can create any logic to decide which server the client will try to connect to.

To create a Dynamic Server provider you need to create a script to provide server information, this script shall implement the interface `IInformationProvider` to return the possible list of servers to try to connect (see Providers Scripts).

You can also check the example script `ServerAddressProvider` on examples folder. This script has a full implementation of how to provide server addresses to dynamic servers.

Server Connection

ObjectNet provides a flexible interface to handle servers, this can be configured on the "Connection Parameters" section.

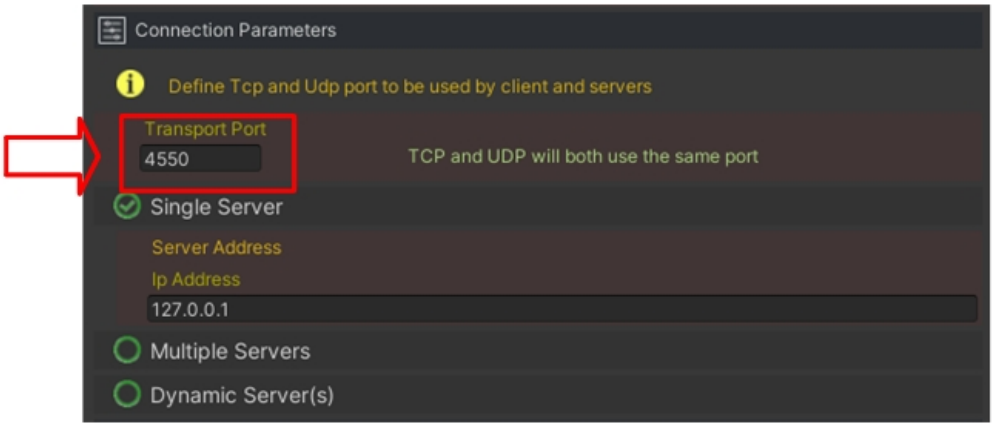


Server mode provides some options when the server starts to listen for incoming connections, in general, no option is better than the other, nonetheless, some options can fit better depending on the scenario:

- Default Address
- Fixed Address
- Public Server
- Internal Address

Transport port

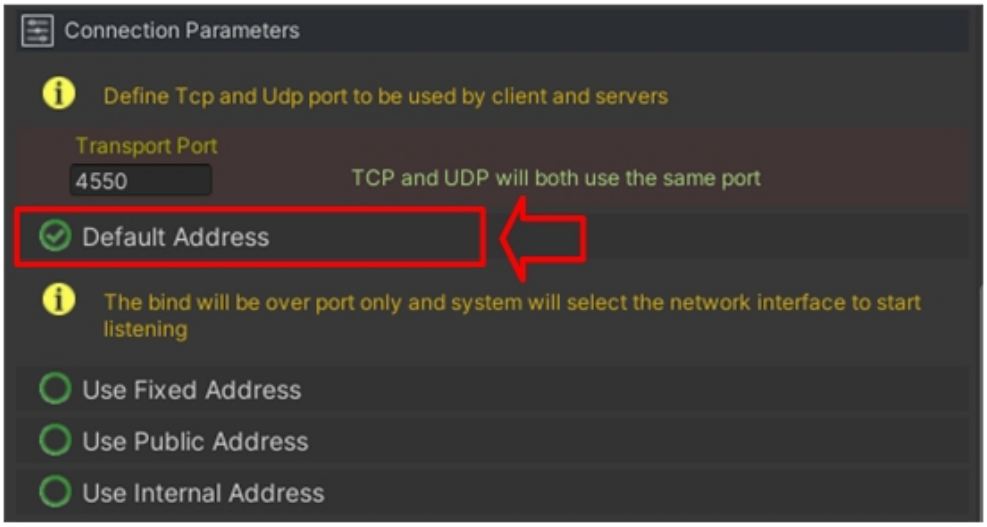
You can define the port used on your game by setting the port on the "Transport Port" field.



Once defined, both transport systems (OnlineObjectTranport and UnityTransport) will use this port as a reliable and unreliable delivery method. Nonetheless, ObjectNet is flexible and in case your custom transport system needs different ports for reliable and unreliable messages you can define this by setting `TransportPortType` provided by `ITransport` interface (see API reference for more information's).

Default Address

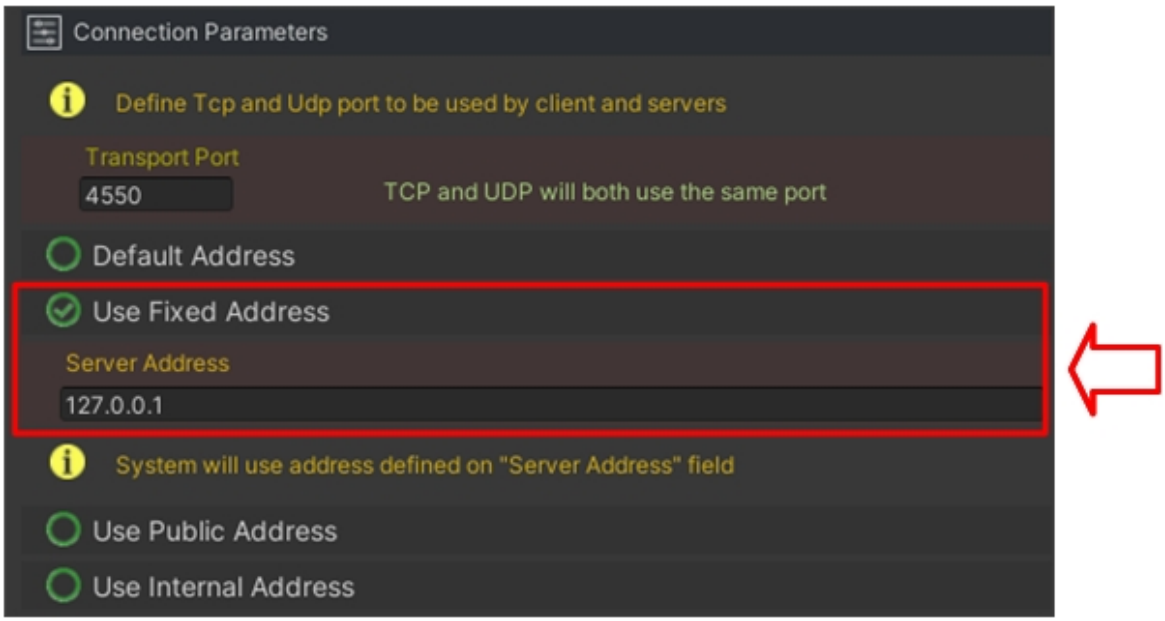
Default Address left to the operation system the responsibility to choose the correct network interface to listen.



Note: This is the best option if you don't know which option to choose.

Use Fixed Address

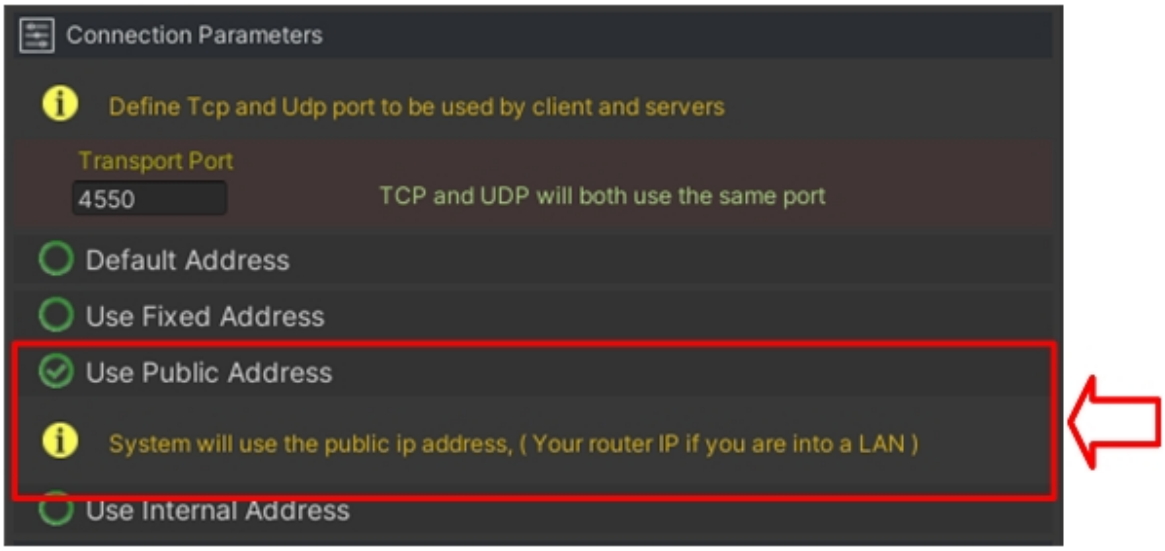
Use Fixed Address will only listen coming packet arrived on the network interface connected to this address ip.



This can be used when you have many network interfaces and you need to define the correct one to use.

Use Public Address

The Public Server will use the address visible to the internet, normally the IP Address on your router.

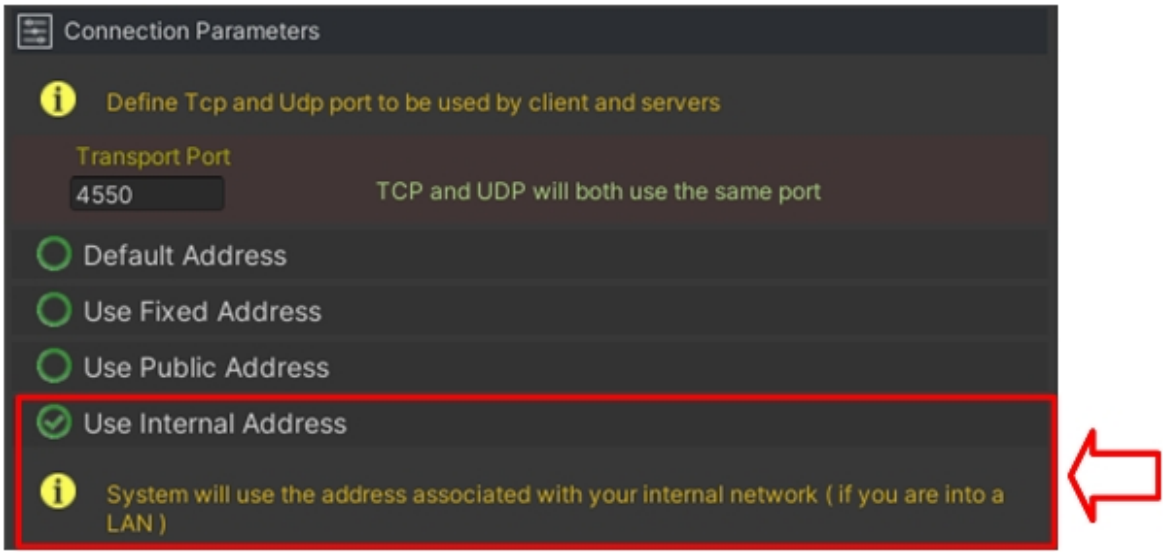


This option only makes sense if you have NAT Traversal enabled or manually created the route on your router.

Note: This does not ensure that connections outside of your LAN can connect with your server, this option only makes sense if your computer has a public IP.

Use Internal Address

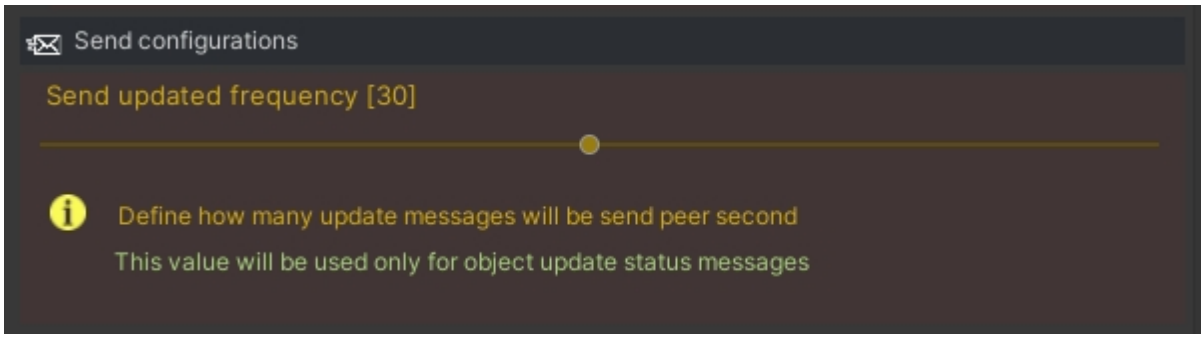
Using an Internal Address will force you to use the network address connected with the router, this will make use of the IP Address visible for your own LAN Network.



This is also the best option to use if you use NAT Traversal because the router will redirect all connections from outside to this network interface.

Send configurations

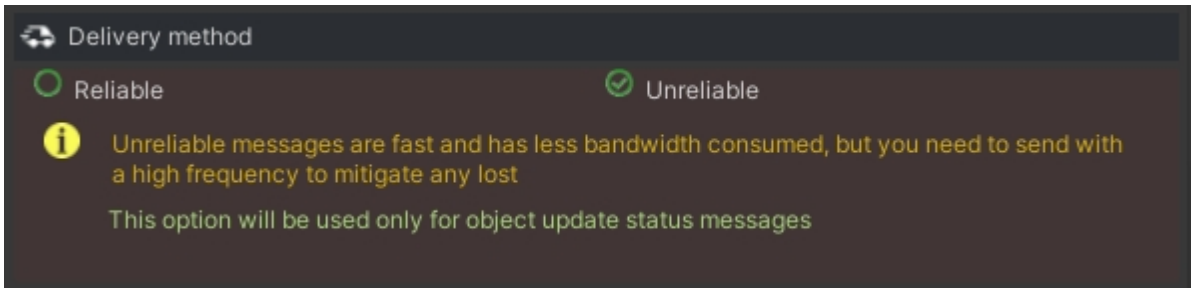
This option allows to configuration of how many updated systems will try to send peer seconds. This option applies only to internal object update messages controlled by the system



You can control the number of update object messages peer seconds **ObjectNet** will send to keep the client synchronized, depending on how your game was designed this number can be greater or smaller, games like FPS or Racing may require a large number of messages to provide accuracy, although, card games, point and click and another type may require a just a few messages, this option allows a fine twin on this aspect of your game.

Delivery Mode

This option allows one to decide how ObjectNet will deliver object update messages.



Object Update messages can be delivered in two ways Reliable and Unreliable.

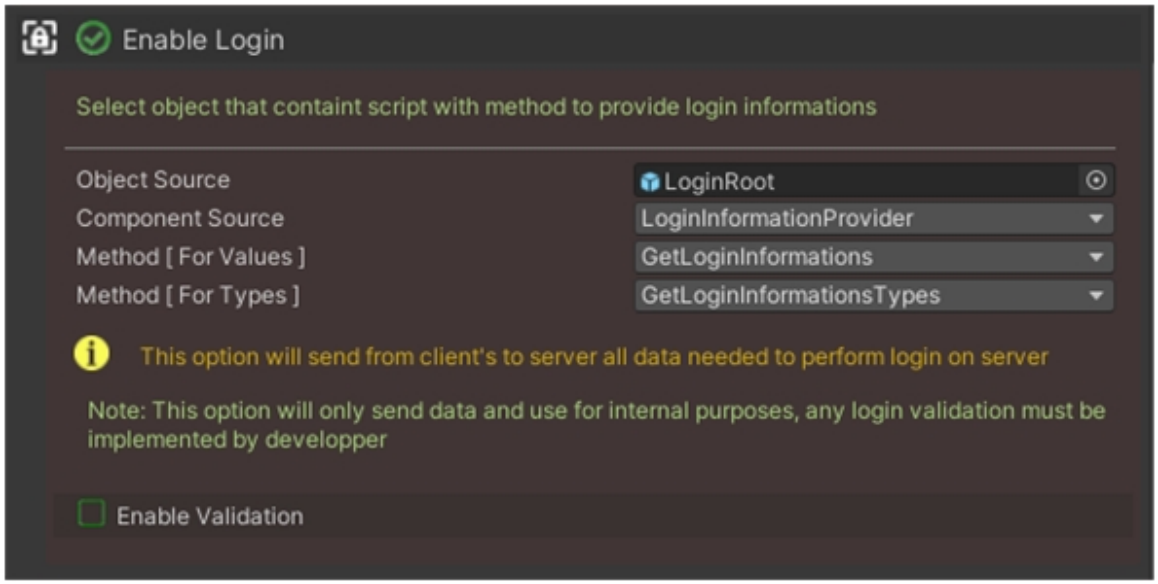
This option when combined with update frequency may provide a better and more reliable experience to the player at the same time that they reduce network costs.

Enable Login

ObjectNet provides a full login feature to protect your game ensuring that players on the game need to be logged.

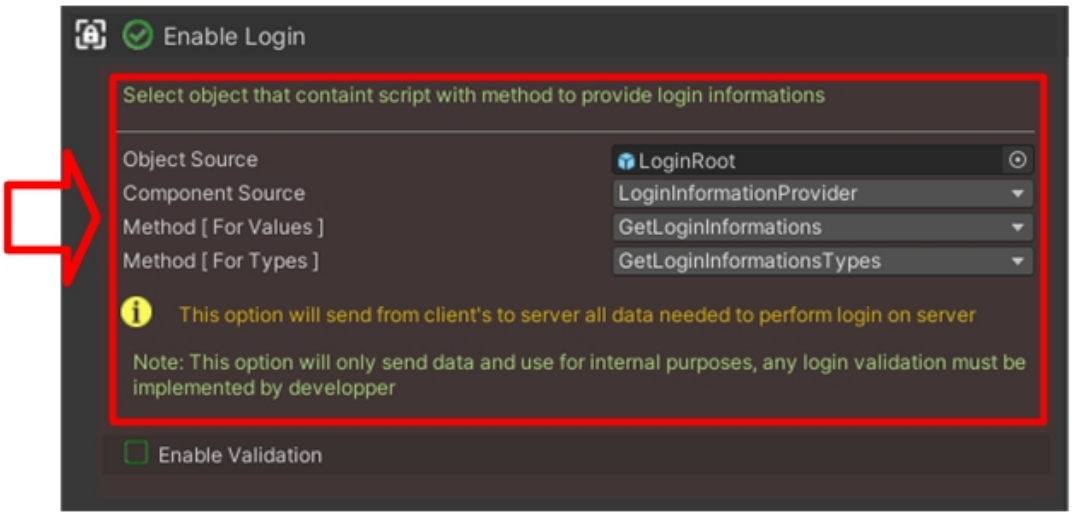
Login provided work in two steps:

- Login Provider
- Enable Validation



Login Provider

Login Provider is how ObjectNet collects a login information to perform login authentication operation.



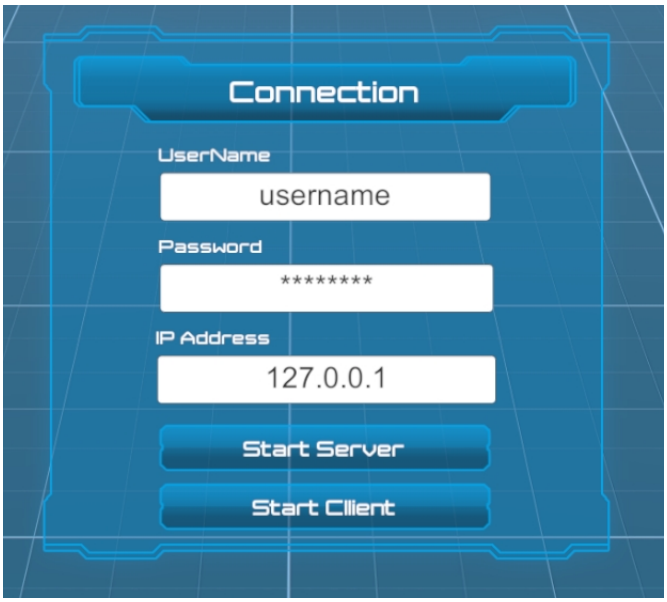
To collect login information you need to create a login provider script that implements the interface `IInformationProvider` to interact with your UI (see Providers Scripts).

This script shall implement two methods, one to collect information and another to identify each parameter type (Boolean, String, Integer, etc ...)..

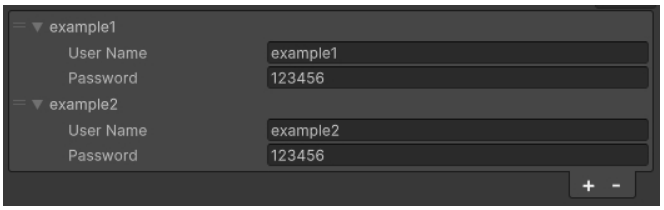
You can also check the example script `LoginInformationProvider` in the examples folder. This script has a full implementation of how to provide login information.

You can also check a full example scene "**02 - Login**".

In this example, you have a full example of how to implement your custom login including a UI.

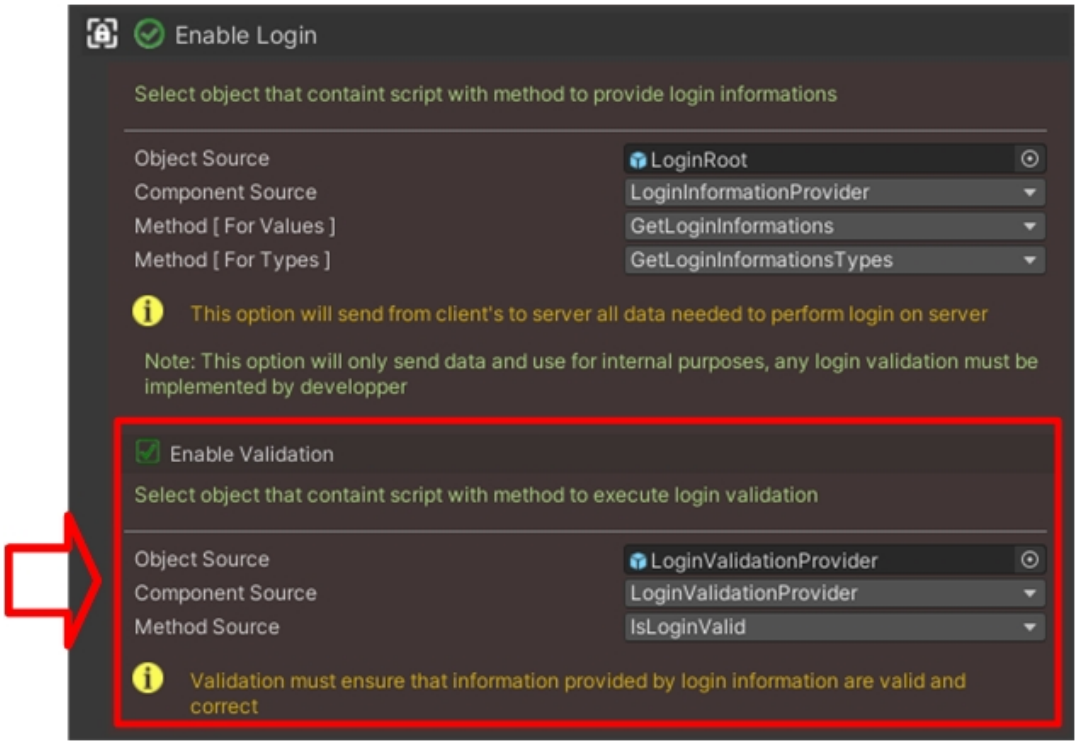


You can use the following users and passwords to test it.



Enable Validation

Login Validation is how ObjectNet checks if authentication data is valid or not.



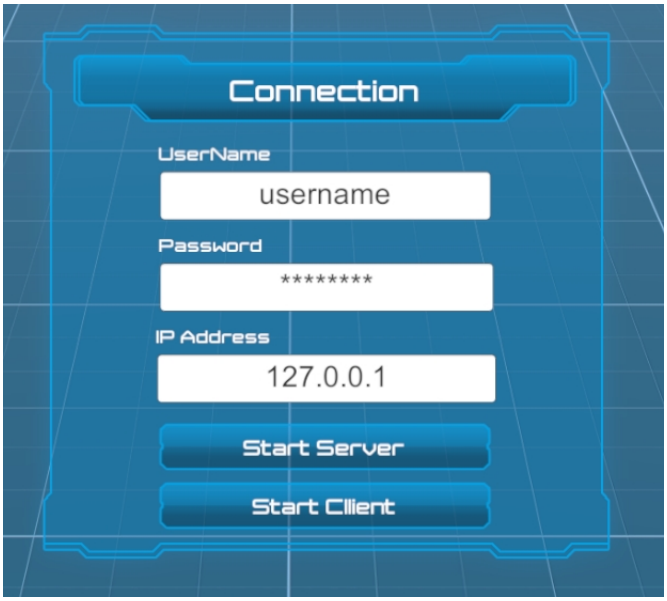
After data is collected, ObjectNet needs to check if this information is valid. To do this ObjectNet uses another script that implements the interface `IInformationProvider` to validate the information (see Providers Scripts).

This script shall implement a method to check this user against any authentication service such as Database, WebService, PlayFab, Steam, or any other.

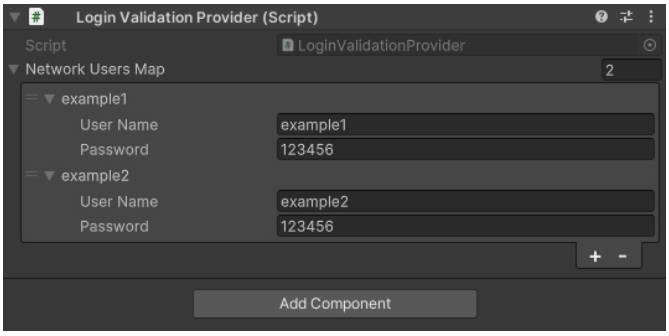
You can also check the example script `LoginValidationProvider` in the examples folder. This script has a full implementation of how to validate login information.

You can also check the full example scene "**02 - Login**".

In this example, you have a full example of how to implement your custom login including a UI.

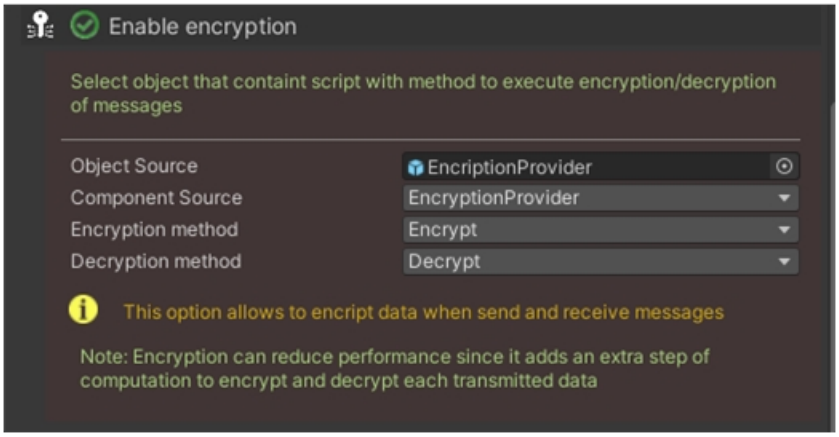


You can use the following users and passwords to test it.



Enable Encryption

ObjectNet provides an optional feature to allow the developer to encrypt data information.



Before sending data and after receiving data, ObjectNet will try to Encrypt or Decrypt data before sending by the selected transport system. Do you need to implement a script inherited from `IInformationProvider` (see Providers Scripts).

This script shall implement two methods, one to Encrypt and another to Decrypt.

You can also check the example script `EncryptionProvider` on examples folder. This script has a full implementation of how to encrypt and Decrypt data using provided C# classes, nonetheless, you can also implement your encryption method.

You can also check a full example scene "**19 - Encryption**" which has the encryption implemented.

Network Clock

ObjectNet provides one implementation of a clock designed to be more accurate than the internal unity clock where the target is multiplayer operations.

This internal clock has the same functionalities as the Unity Time class, we recommend using ObjectNet time wrapper which will decide which clock they will use.

We strongly recommend using those functions if you are starting your development :

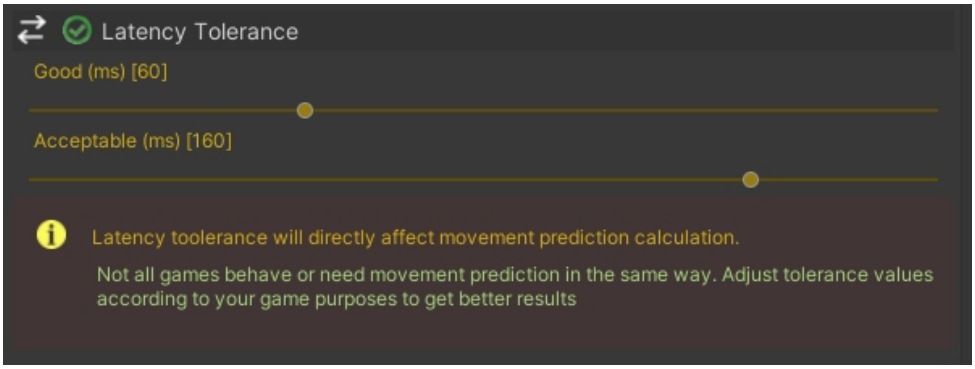
```
NetworkClock.deltaTime

NetworkClock.fixedDeltaTime

NetworkClock.time

NetworkClock.deltaFrames

NetworkClock.tick
```

If Network Clock is enabled, ObjectNet will use its internal clock, nonetheless, if Network Clock is not enabled ObjectNet will use the Unity default clock.

To see each function in detail check ObjectNet API documentation.

Latency Tolerance

ObjectNet provides an embedded Movement Prediction system to predict object movement and mitigate latency.



Movement Prediction uses those values to calculate the predicted position of an object over the network. In this configuration you have two options :

- Good Latency
  - Good latency is the latency between 0ms and the selected value, this latency is considered a good value for players playing your game.

- Acceptable Latency
  - Acceptable latency starts on the end of good latency up to the selected value, and movement prediction will predict in a different way if latency is on the acceptable bounds.

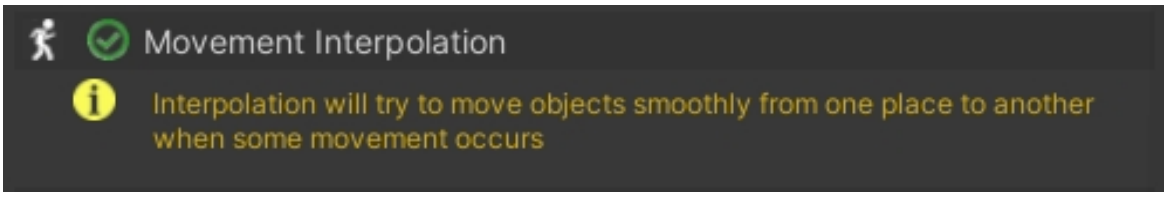
If values exceed the acceptable value the movement prediction will use his extrapolated prediction without affecting game-play.

Movement Interpolation

Always when remote object position was received, the new received position became the current object position this may cause some teleportation or jittering on objects.

ObjectNet provide the facility to smoothly move objects during network movement to avoid jittering or flicker on game objects.

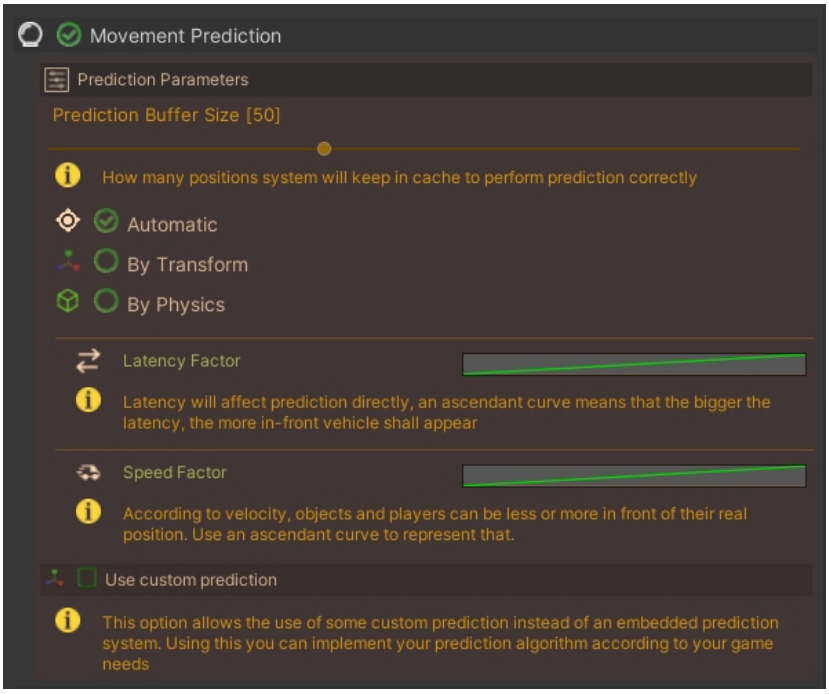
To enable internal latency measure you need to enable "**Movement Interpolation**" on **NetworkPrefab**.



Note: ObjectNet will always work to mitigate and reduce flicker and jittering, nonetheless, depending of the latency or packets lost you may experiment certain issues.

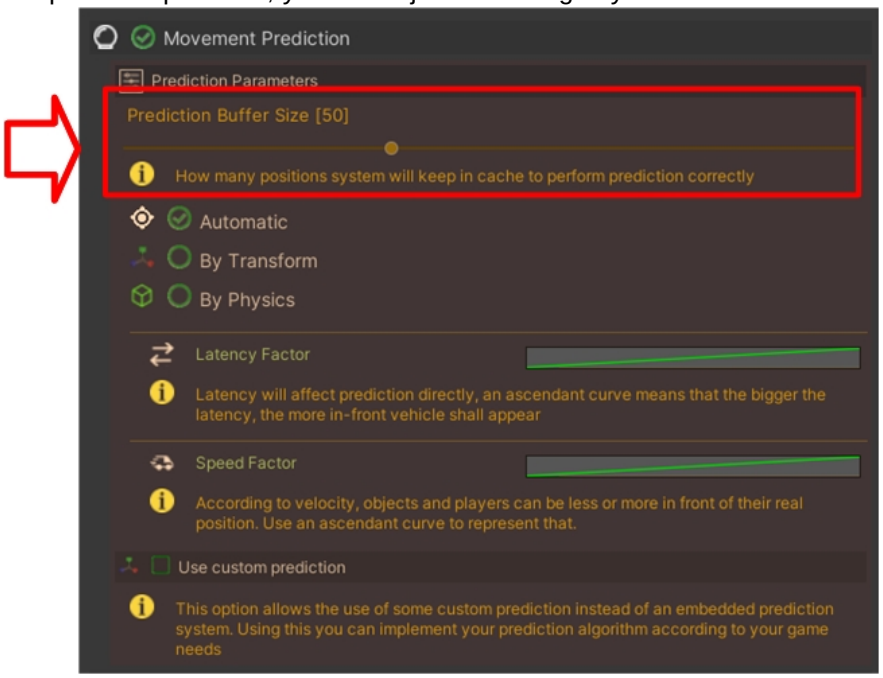
Movement Prediction

ObjectNet provides an embedded Movement Prediction system to allow users of competitive games to reduce or mitigate latency problems.



Movement Prediction has a few options to adjust a fine-tuning.

- Prediction Buffer Size
  - This option creates a buffer where each past position is stored to be used as a database of predicted positions, you can adjust according to your needs.



- Prediction Mode
  - Prediction mode tells the ObjectNet how the system will check the parameters (such as distance, angular velocity, linear velocity, etc..) needed to predict movement and the values

can be :

- Automatic
  - The system will automatically select the best approach
- By Transform
  - The system will use transform to calculate needed values, this can be useful if your objects don't have a rigid body.
- By Physics
  - The system will use physics objects (such as Rigid Body) to calculate needed values.

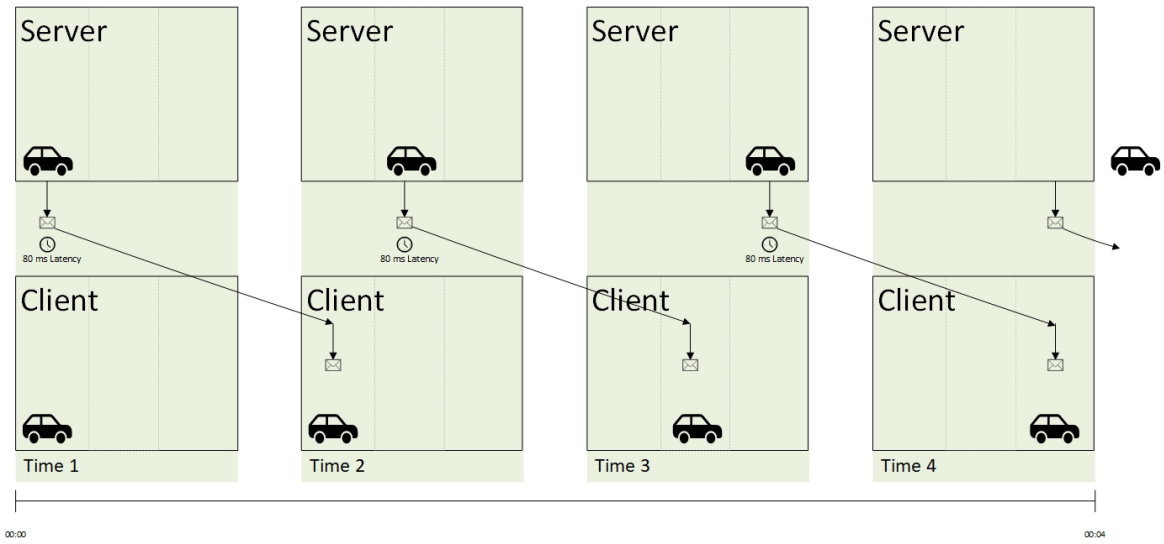
- Latency Factor
  - This parameter tells the system how they shall handle latency (see Latency Factor).

Latency Factor

This topic explains how latency affects your game and how the Latency factor parameter can be used to mitigate this problem.

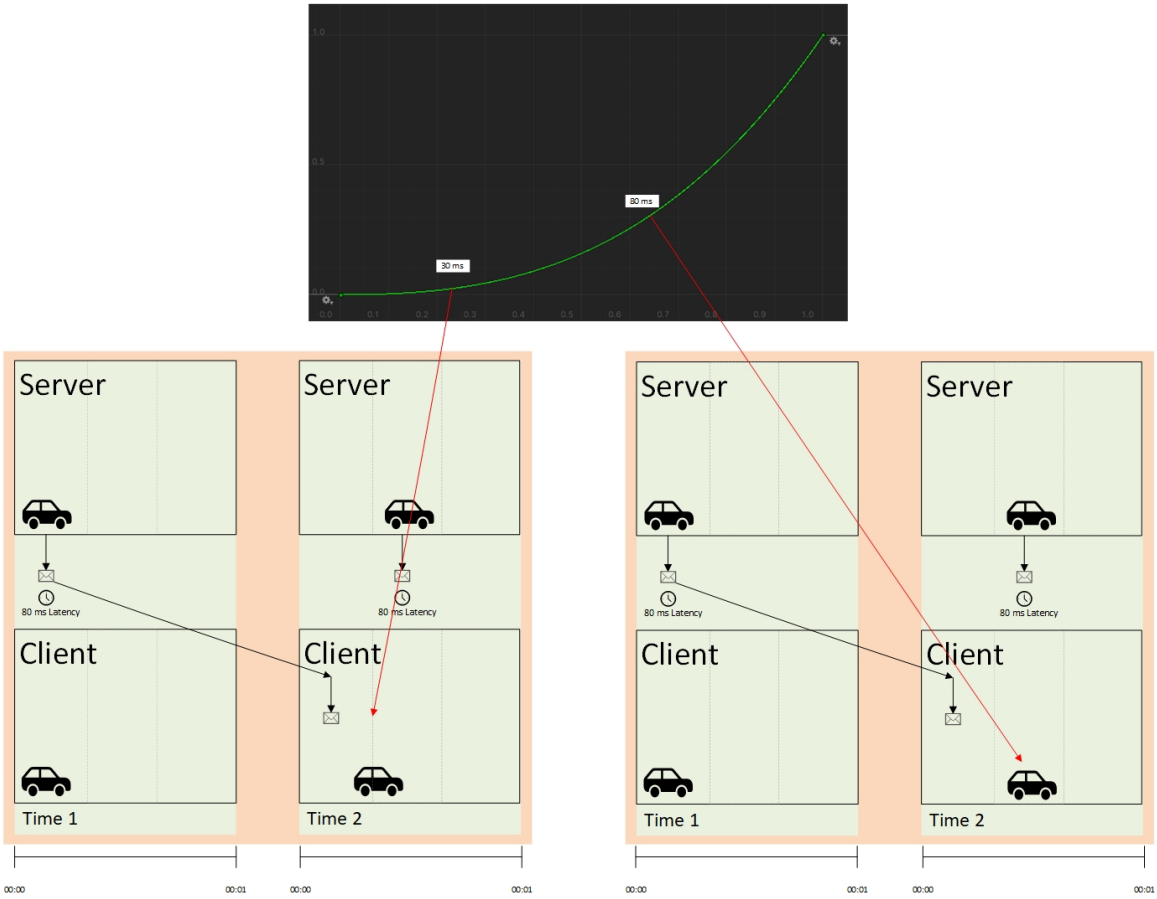
Latency is a challenge that has been a pursuit in multiplayer games since its first appearance, the latency in short, makes thing appear different from different players depending on how much time event and information arrives on their local instance.

The following drawing illustrates a latency problem in a racing game :



In the example above the vehicle on the client is always one step behind the vehicle on the server side, this can generate a lot of conditions that may break some game logic, In the ideal universe, both vehicles should keep synchronized.

Latency is a problem that you can mitigate or leave with but is impossible to solve completely. The Latency Factor tells the engine how in front object shall appear depending on player latency.



The above example illustrates how the Latency Factor works according to latency to keep the player close to the real position event when latency delay packet to be delivered to the client, how much more latency, more in front vehicle shall appear.

Speed Factor

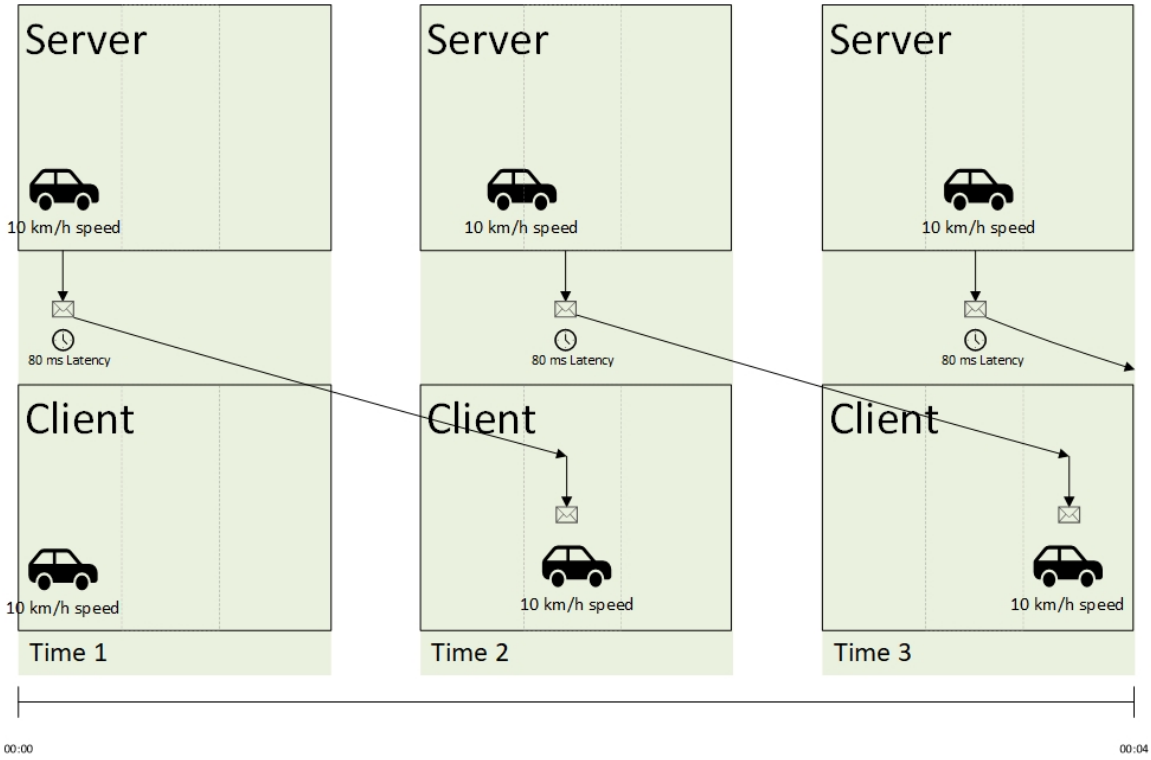
This topic explains how objects' speed affects how prediction shall calculate real object position and how the Speed Factor parameter can be used to mitigate this problem.

When an object is moving, the amount of traveled distance can vary according to its speed, this can be expressed by the physics formula [DISTANCE = SPEED x TIME].

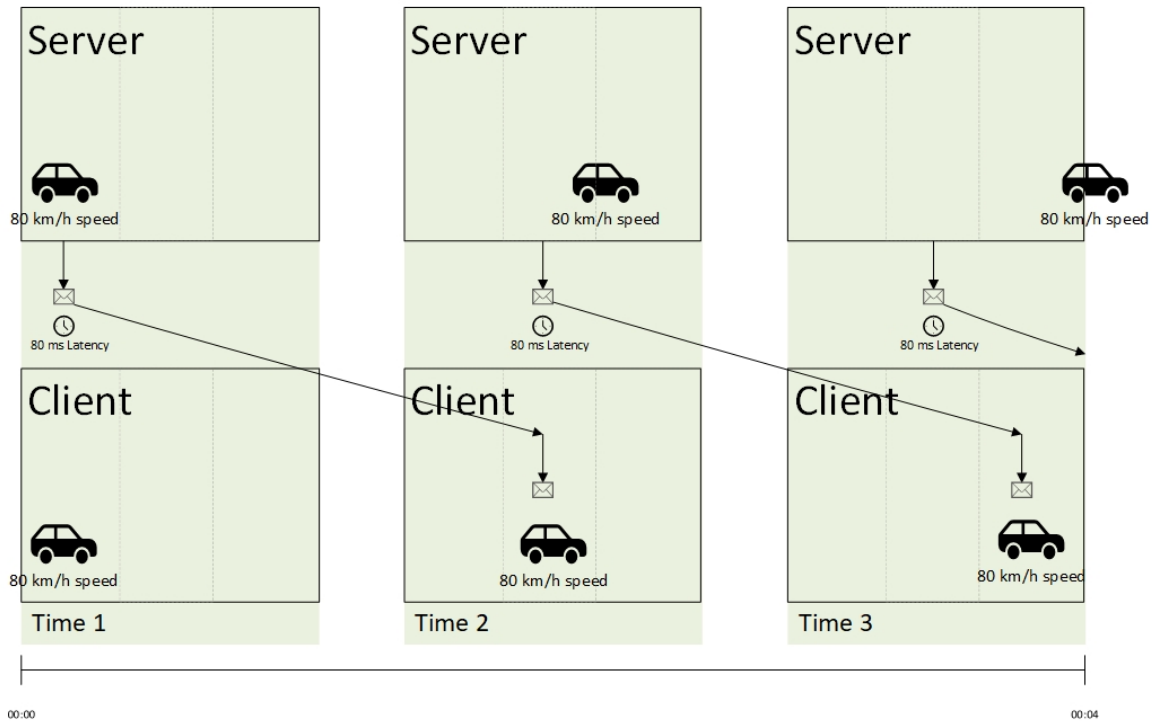
This means that much faster the object is moving, the more in front it should appear when latency increases.

Suppose that a vehicle is moving at 10 Km/h and you do not consider its speed during prediction, the result could be that you put the vehicle in front where the real appears on the server.

The following drawing(s) illustrates a latency problem in a racing game :

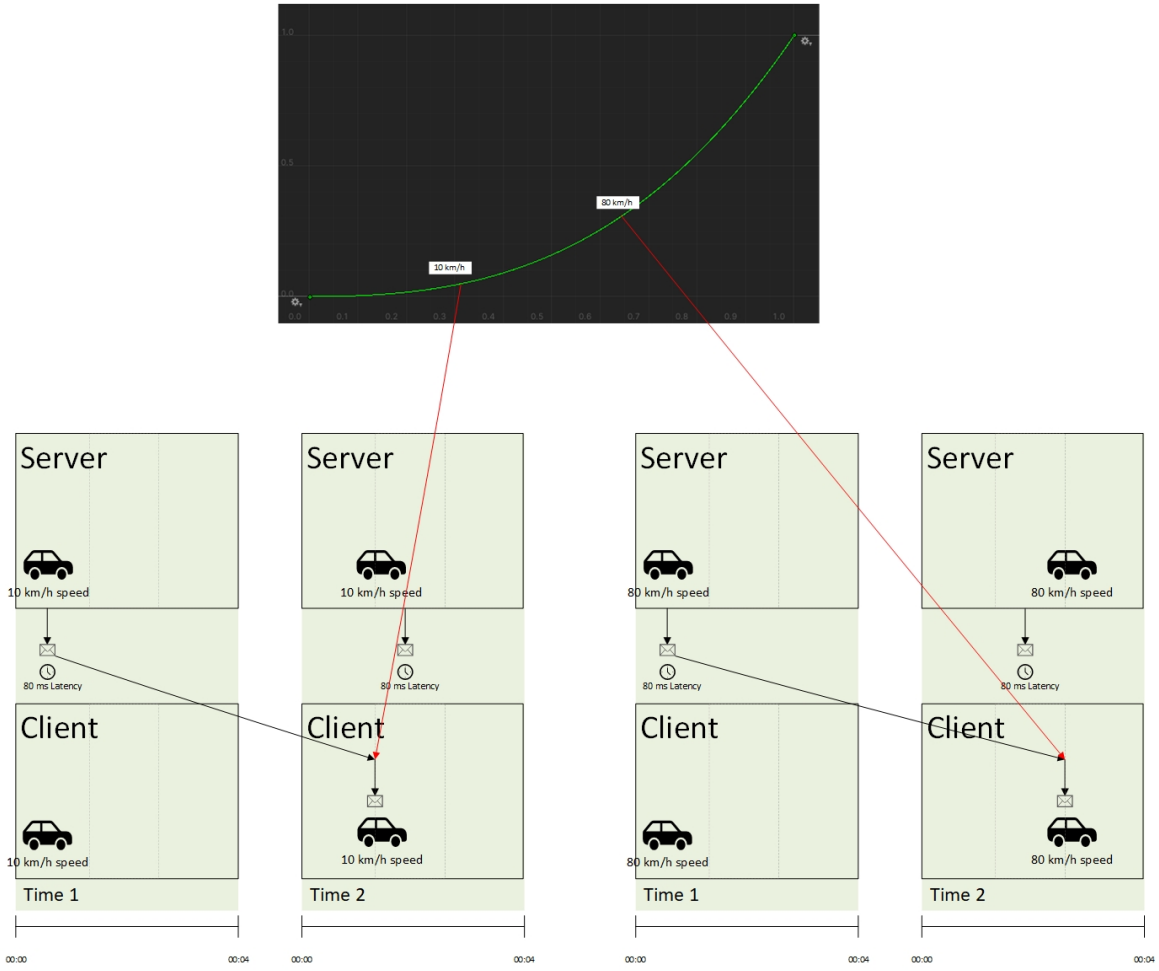


Now let's suppose that the same vehicle starts to move at 80km/h and you still do not consider his speed during prediction, the result could be that you put the vehicle in behind where the real appears on the server.



In both examples, the vehicle isn't synchronized with regard to his real position on the server due to the variation in vehicle speed.

Speed Factors mitigate this problem including object speed on prediction calculation. This means how much faster an object is moving, how much in front they shall appear when latency increases.

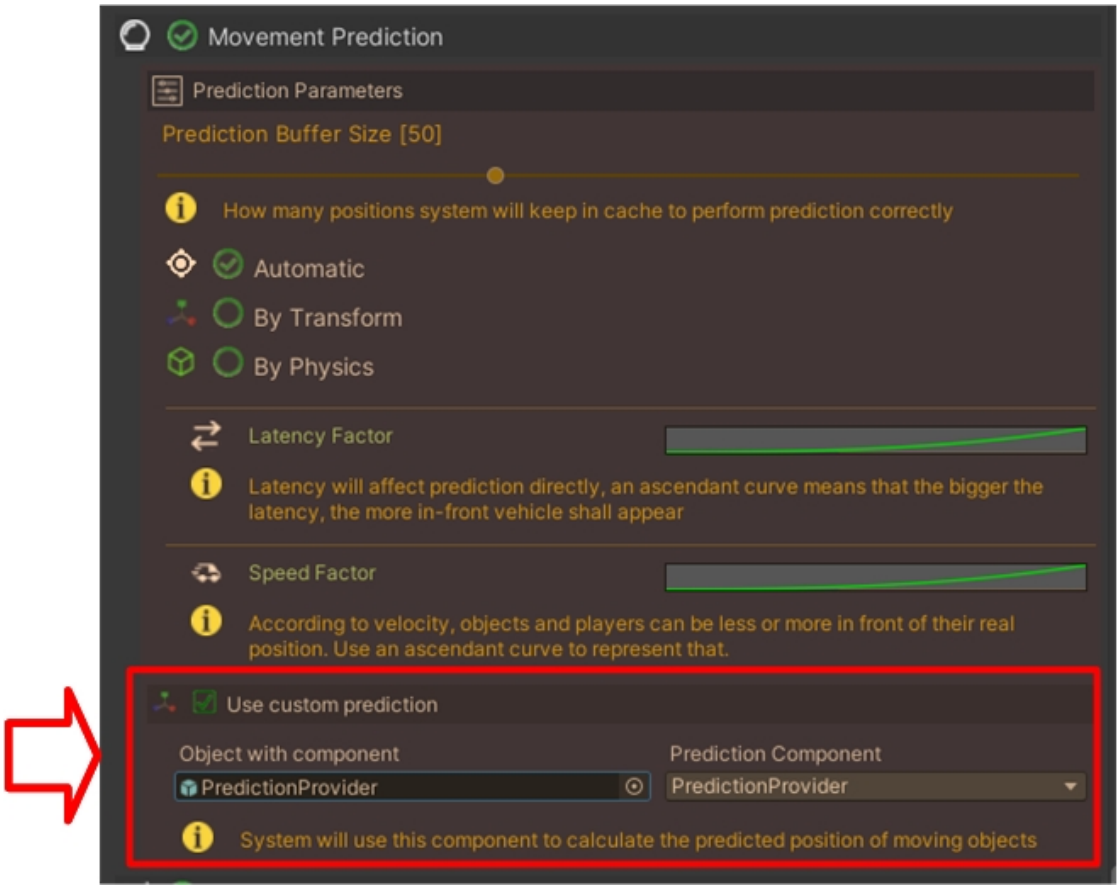


The above example illustrates how the Speed Factor works to mitigate this problem by adding object speed during calculations.

Custom Prediction

Embedded **Movement Prediction** provided by **ObjectNet** should work on all cases, nonetheless, **ObjectNet** provides an API to allow developers to create their own movement prediction and override the default prediction.

You can override the default prediction system by checking "**Use custom prediction**" option.

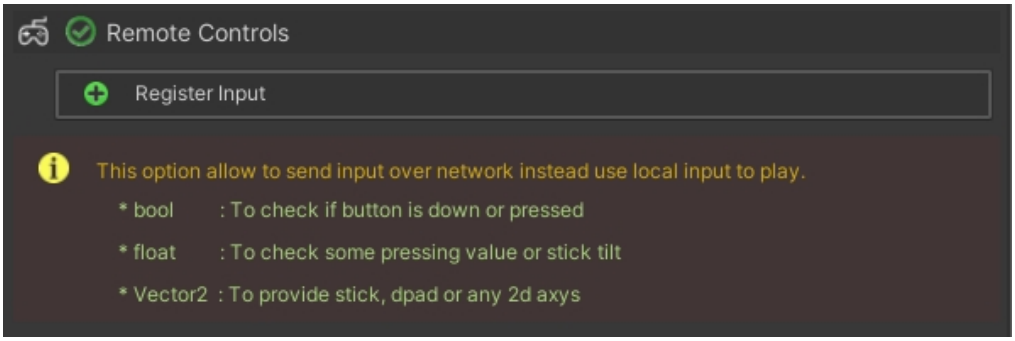


To create your own prediction system you need to create a provider script that implements the interface `IPrediction` (see Providers Scripts).

You can also check the example script `PredictionProvider` in the examples folder. This script has a skeleton implementation of a custom prediction system.

Remote Controls

Depending on the approach you decide to take on your game, you can send object data from clients but send client input controls only.



Question : But was it considered a Control?
Answer : Control is anything that the system can use to interact with the game, such as Keyboard, GamePad, Joystick, etc...

Remote controls can be synchronized in two ways, by using NetworkManager or in case of script inherits

from NetworkBehaviour by code.

Synchronized using NetworkManager

When Remote Controls is enabled, a system with put all objects as passive (event local player) and your player will move only based on server position and actions.

This can be very useful if you are trying to isolate your game from any cheating or hacking, since the server will only process player inputs, instead of sending and receiving his position and events to the server.

To use **Remote Controls** you will need to take the following steps :

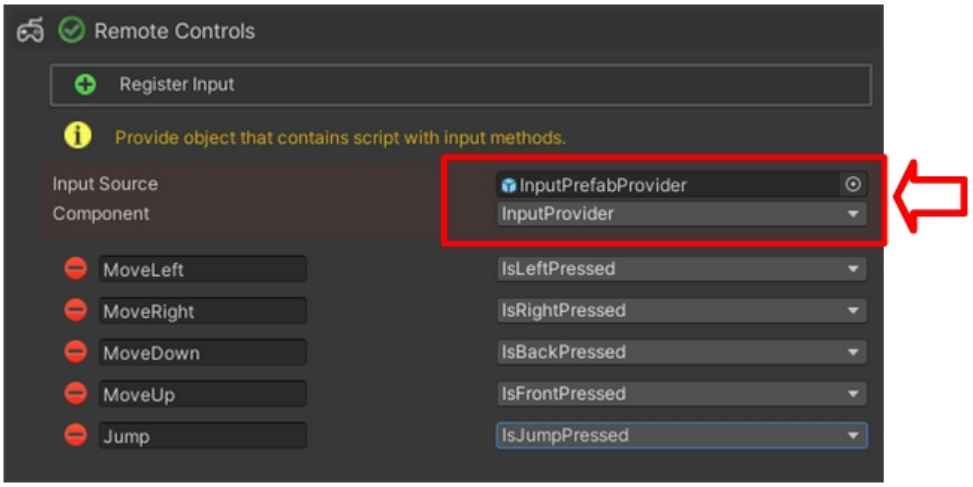
1. Create an script o implement how you with catch Inputs
   This script must implement `IInputProvider` interface.
   Each method of this script must return the state of input witch he is mapping, the following code is an example of how to implement those methods.

```
public bool IsJumpPressed() {
     return Input.GetKey(KeyCode.Space);
}

public bool IsFrontPressed() {
     return Input.GetKey(KeyCode.UpArrow);
}

public bool IsBackPressed() {
     return Input.GetKey(KeyCode.DownArrow);
}

public bool IsLeftPressed() {
     return Input.GetKey(KeyCode.LeftArrow);
}

public bool IsRightPressed() {
     return Input.GetKey(KeyCode.RightArrow);
}

public Vector2 GetMovement() {
     return new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));
}
```

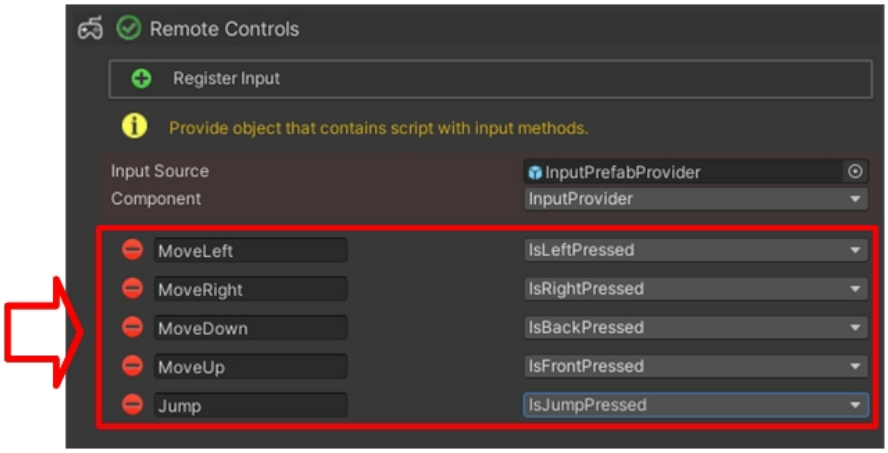
   Each method maps one input, and the possible return values shall be only **bool**, **float** and **Vector2**.

   You can also check the example script Input Provider in the examples folder. This script is a full implementation of the Input Provider.

2. Assign prefab and component where script provider is attached



3. Register all possible controls on NetworkManager



Each method must have a unique alias to be identified by the system and sent through the network.

4. Get an instance of **ObjectNet** class "`NetworkObject`" from current **NetworkPrefab**.

```
this.network = this.GetComponent<NetworkObject>();
```

5. Instead of using the direct Unity Input Methods (or any other API) get input status, you need to use **ObjectNet** input methods.

```
this.network.GetInput<bool>("MoveLeft");
this.network.GetInput<bool>("MoveRight");
this.network.GetInput<bool>("Jump");
```

Synchronized using by code NetworkManager

Remote control can also be synchronized by code, this can be useful if you need to do some rebinding or procedural generation. The following piece of code shows how to synchronize Network Inputs by code.

```
public void Awake() {
```

```
            // To map controls
            this.RegisterInput<bool>("jump").OnEvaluate(() => { return
Input.GetKeyDown("space"); });
            this.RegisterInput<float>("lados").OnEvaluate(() => { return
Input.GetAxis("horizontal"); });
      }
```

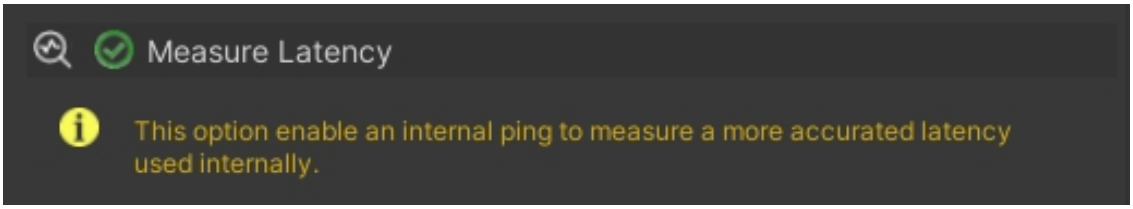
Each control needs to be mapped with the method which will return his current status.

Measure Latency

ObjectNet uses a lot of measurement systems to keep peers synchronized, some of those are provided by Operational System or Unity, nonetheless, ObjectNet has a custom measurement system to get more accurate information.

One of those measure is the latency. The latency is the time that a package takes for one peer to arrive at the server.

To enable internal latency measure you need to enable "**Measure Latency**" on **NetworkPrefab**.



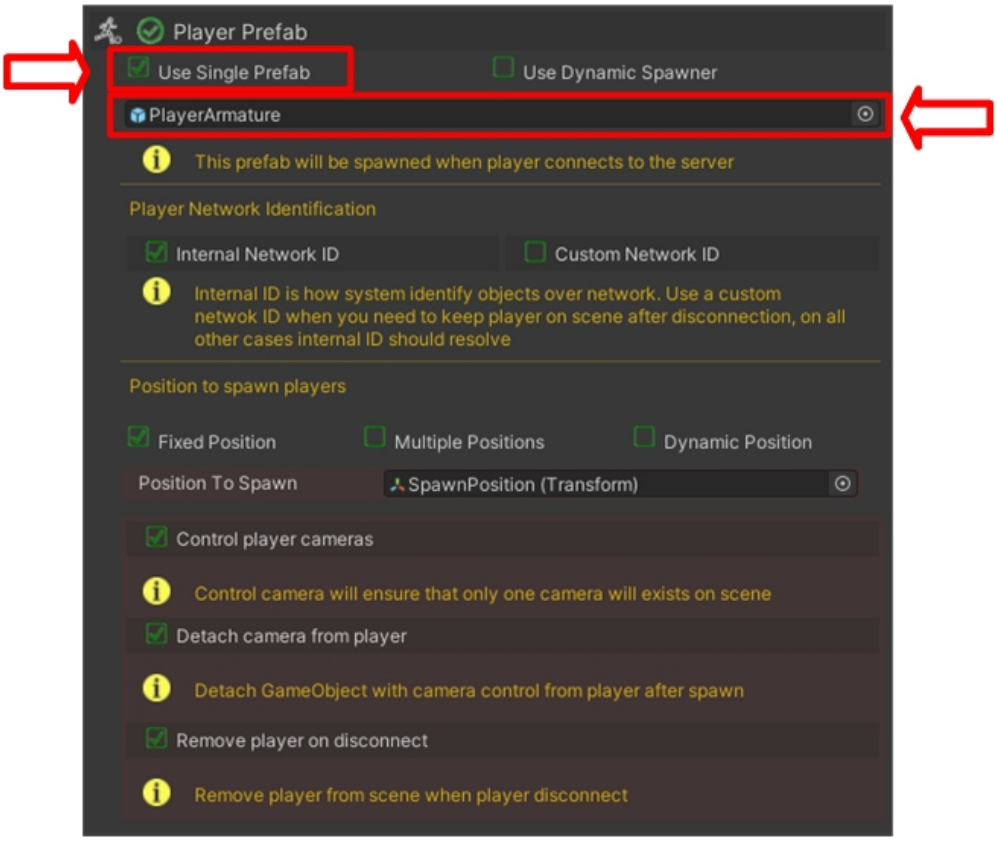
Player Prefab

In a MultiPlayer game, each player shall have his player avatar to play, ObjectNet provides the facility to control this object, and the left object manages his creation and destruction. This functionality is called Player Prefab.

Player Prefab is an internal process to control a player's avatar life-cycle.

Use Single prefab

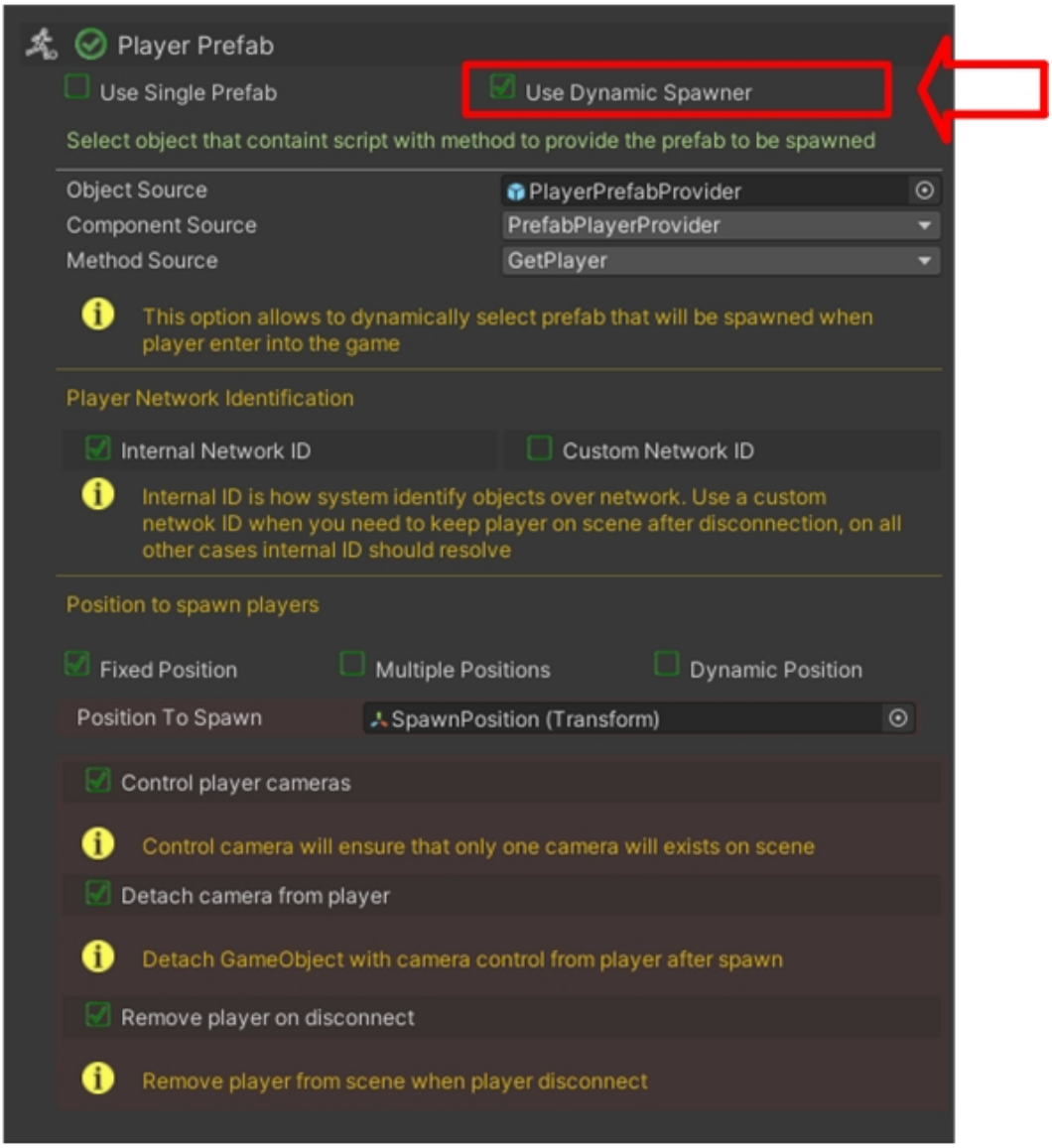
The Single Prefab option fixed the spawned prefab for all players who enter the game.



In the image above, every time a player enters a game the prefab "Player Armature" will be spawned and associated with this player.

Use Dynamic Spawner

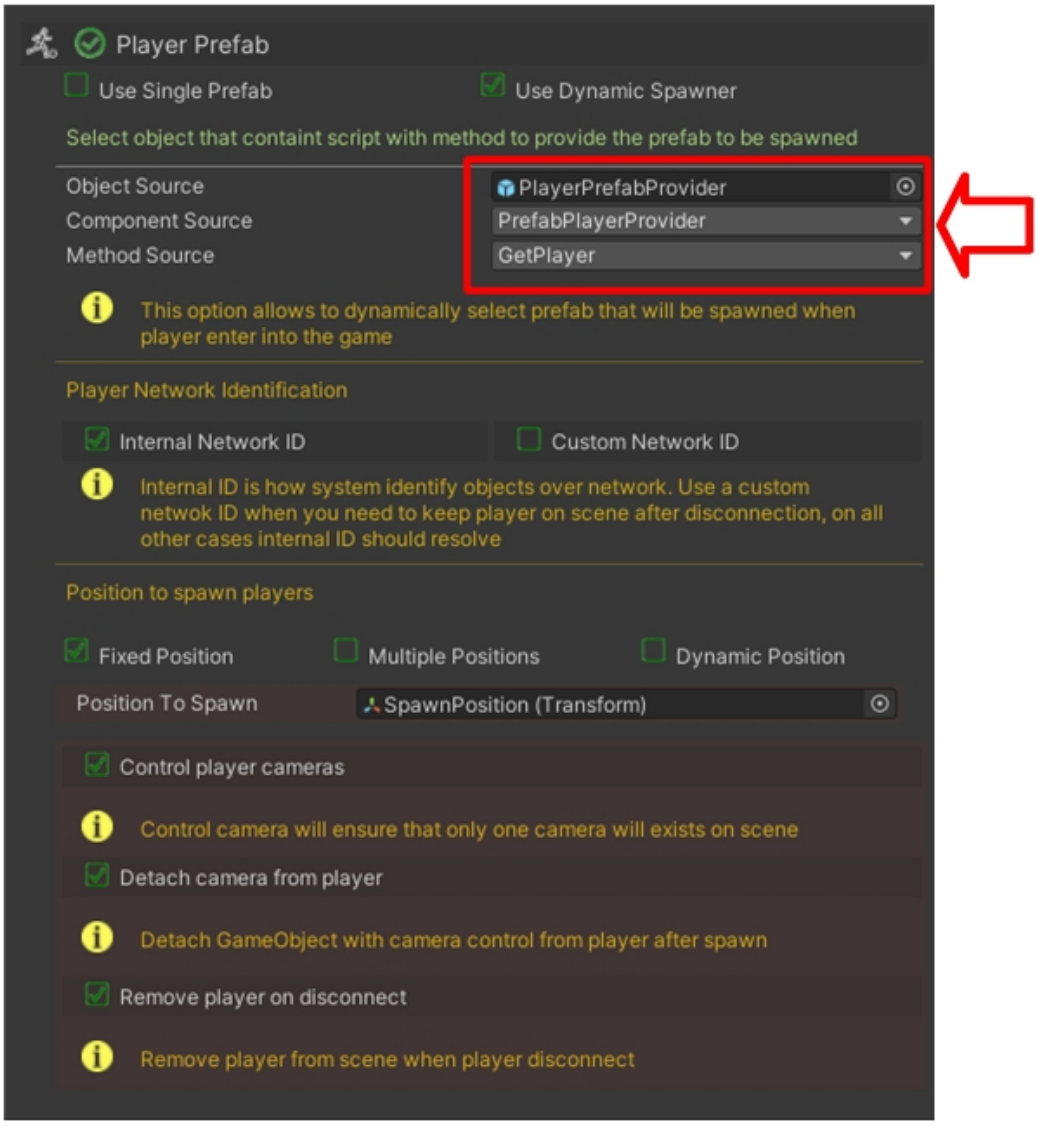
The Dynamic Spawner option allows to creation of a custom logic to decide which prefab will be spawned for each player.



This option allows to spawn prefab to each player individually and implement some type of skin feature.

Player Prefab Provider

The Dynamic Spawner option requires logic to decide which prefab will be spawned for each player.



In the image above, the "PlayerPrefabProvider" is the object with a component PrefabProvider which has a method called "GetPlayer"..

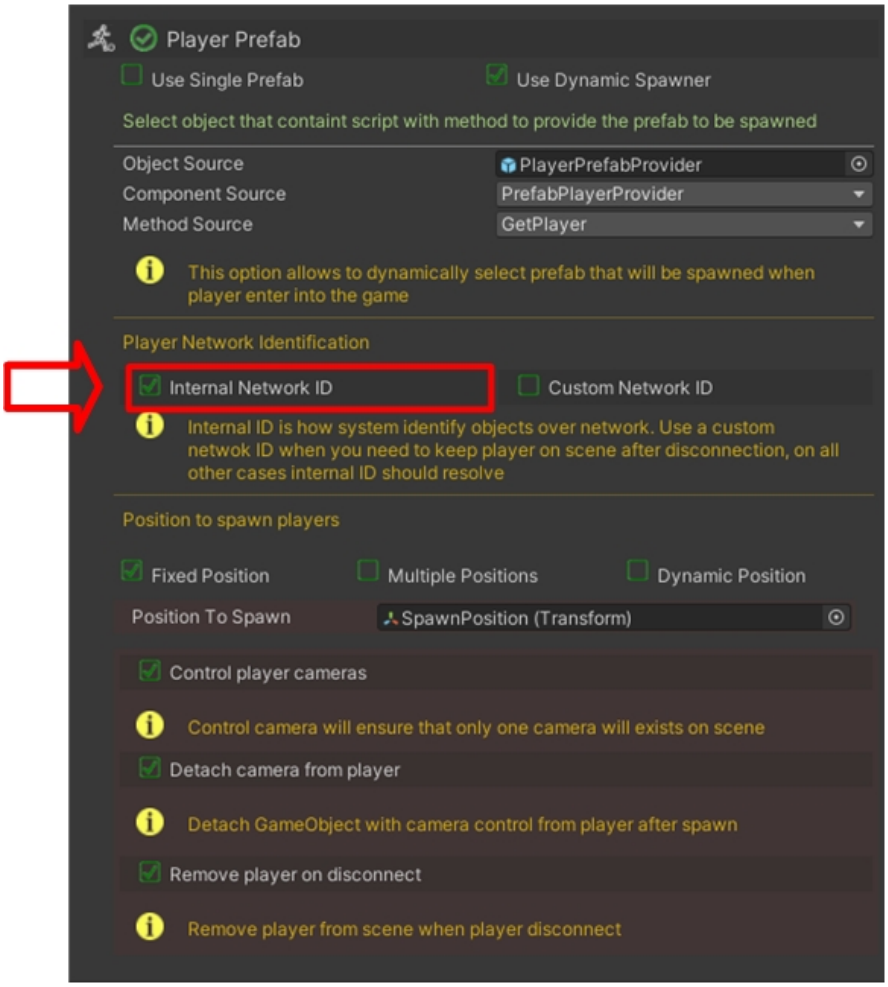
To create a Dynamic Spawner provider you will need to create a script to provide a player prefab to be spawned, this script shall implement the interface `IInformationProvider` to return the player prefab to be spawned (see Providers Scripts).

You can also check the example script `PrefabPlayerProvider` in the examples folder. This script has a full implementation of how to provide player prefab to be spawned.

Internal Network ID

On ObjectNet each network object must have a unique ID, this ID is the way objects are identified over the network..

This ID could be controlled internally by the system, the option "Internal Network ID" generates a sequential unique ID for each player created.



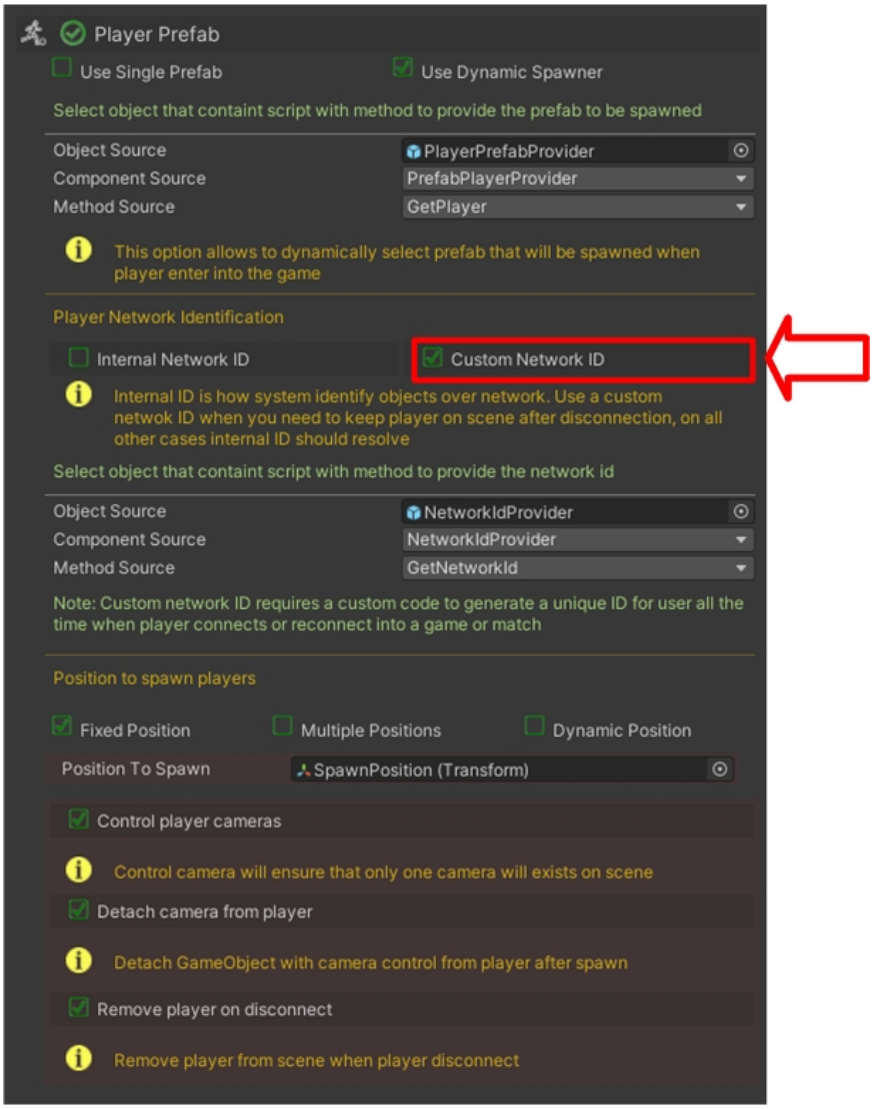
For a casual game where the player is not identified this is a safe time option, since you don't need to be worried about player identification.

Custom Network ID

ObjectNet provides a facility to identify a spawned player with a unique state-full ID.

This is pretty useful if you wish to reuse a player session in case of any disconnection. Associated a unique ID to the player allows bringing the player back when a player is reconnected at the same match.

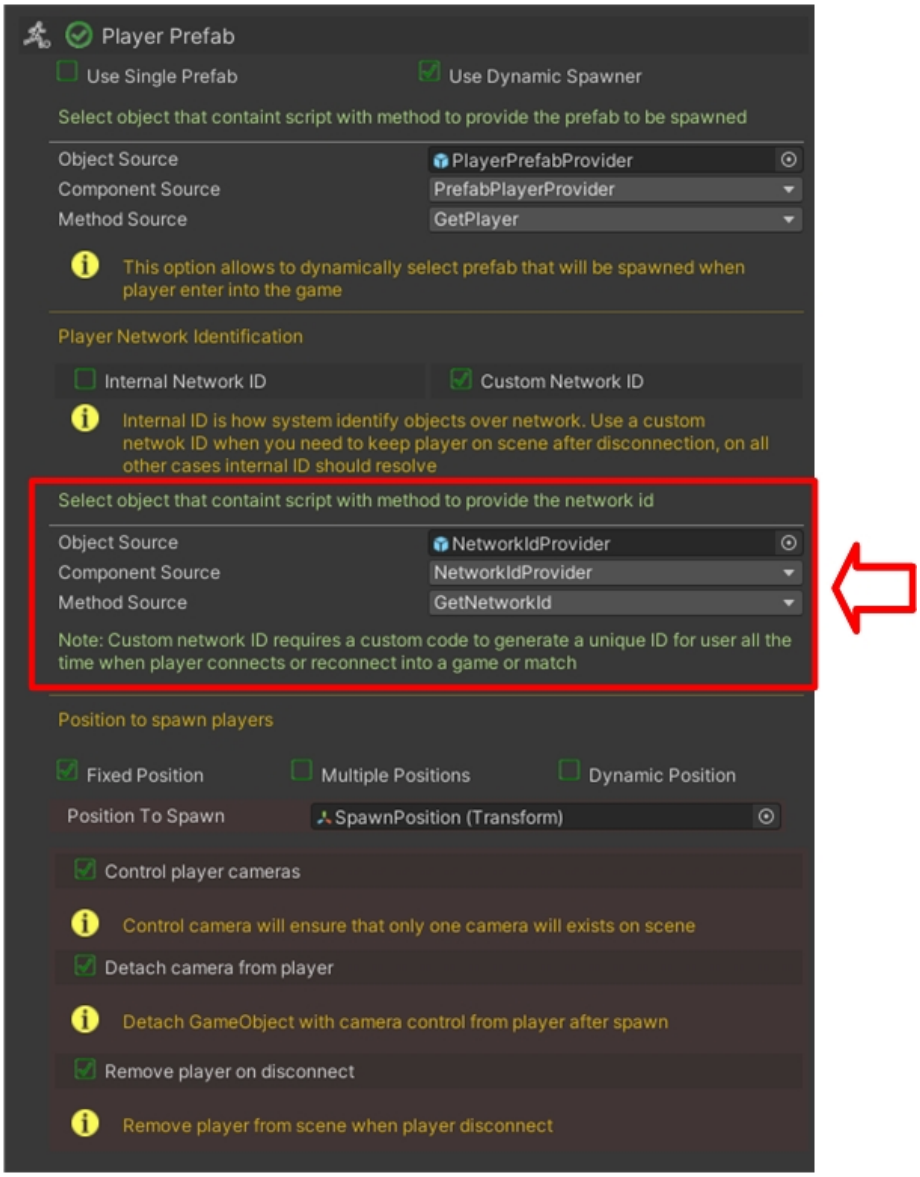
This option can be enabled by checking the "Custom Network ID" option.



This ID shall originate from a state full source such as a Database, PlayFab, Steam, or any other.

Network ID Provider

The Custom Network ID option requires a logic to return the Network ID to associate with the player.



In the image above, the "NetworkIdProvider" is the object with a component NetworkIdProvider which has a method called "GetNetworkId".

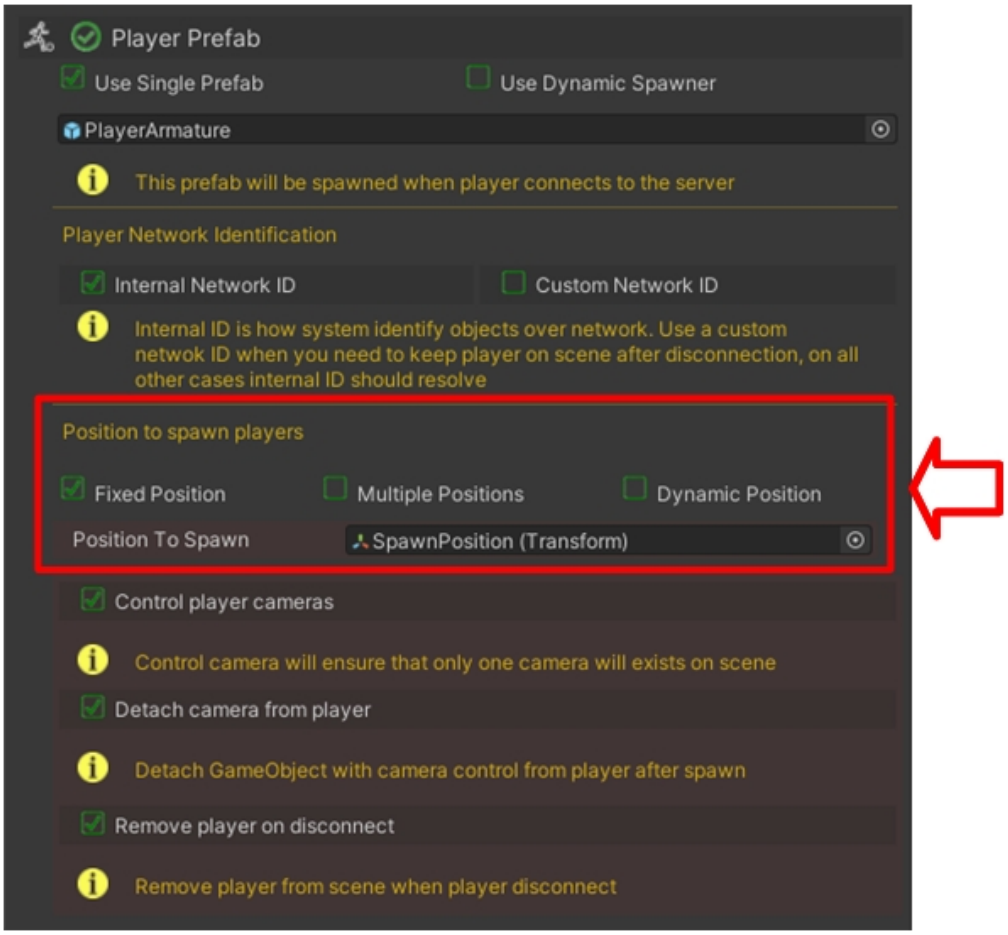
To create a NetworkIdProvider provider you will need to create a script to provide a custom ID, this script shall implement the interface `IInformationProvider` to return the ID to be associated with the player prefab to be spawned (see Providers Scripts).

You can also check the example script `NetworkIdProvider` in the examples folder. This script has a full implementation of how to provide custom IDs.

Position to Spawn

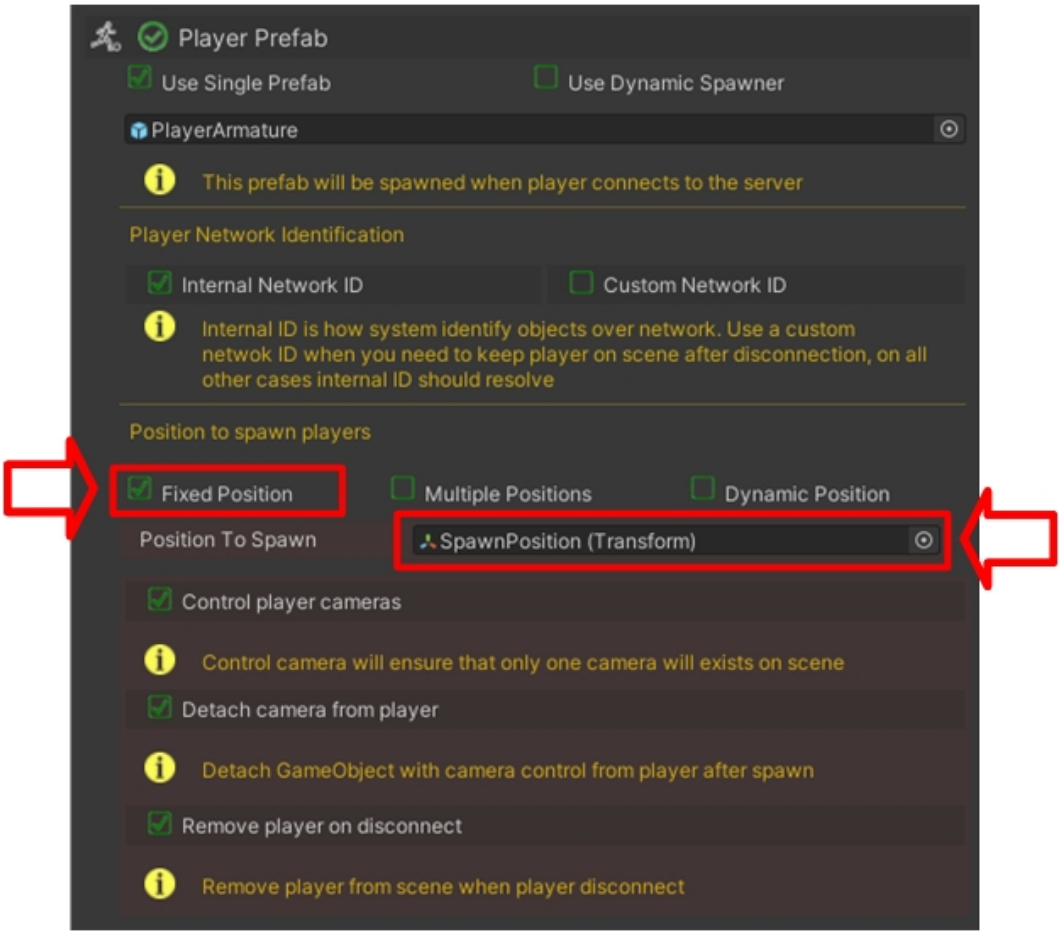
When some player is spawned, the system shall decide the position where this player will be spawned.

ObjectNet provides a facility to decide where the system will spawn players.



Fixed Position

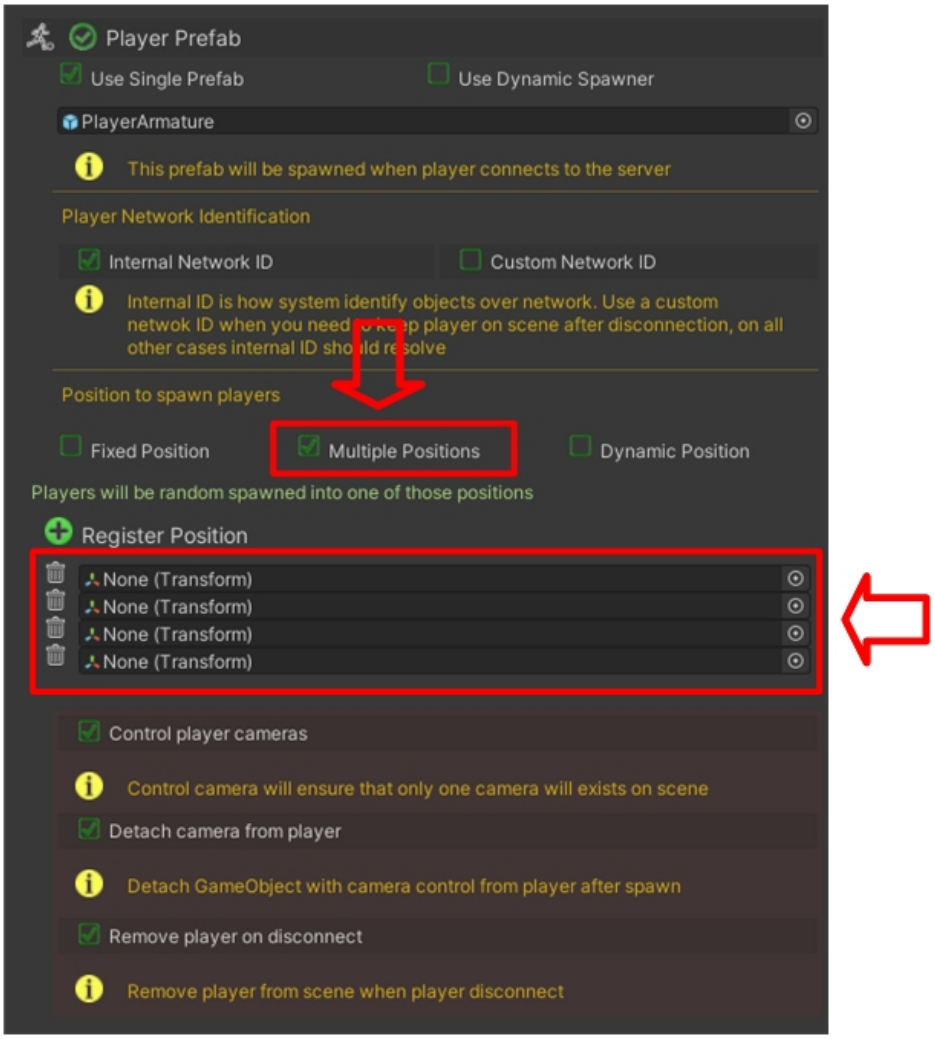
Fixed Position do what his name suggests, spawn all players on a specific position..



On the field "Position to Spawn" a transform must be provided, all players will be spawned on this exact position.

Multiple Positions

Multiple Positions allow to spawn players into different positions according to a provided list of possible positions.



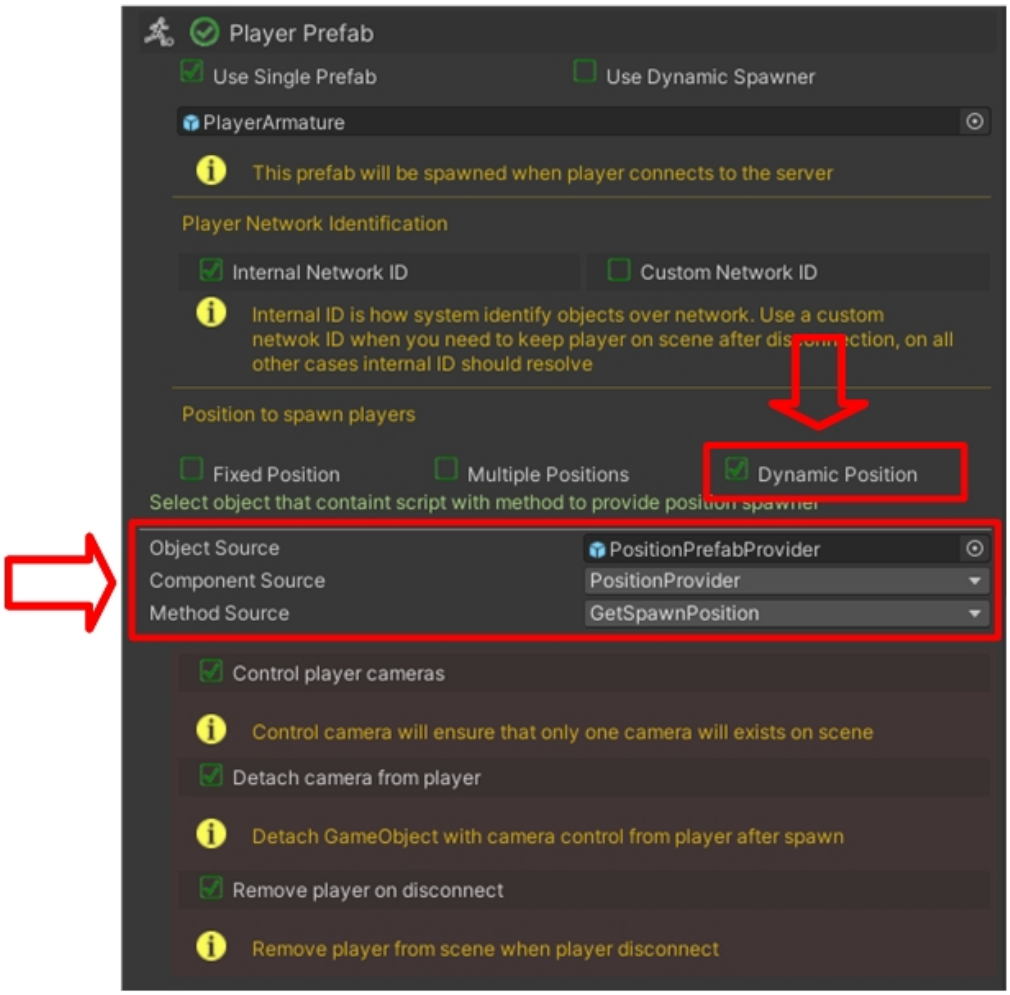
The system will perform a random rounding robin algorithm to select the next position to spawn, which means that the next player will not be spawned at the same position as the previous one.

When all positions are used, the system will restart his list and start to spawn from scratch.

This ensures that no one player will be close to another and all possible position has players spawned.

Dynamic Position

Dynamic Position allows to spawn players into positions according to some custom logic.



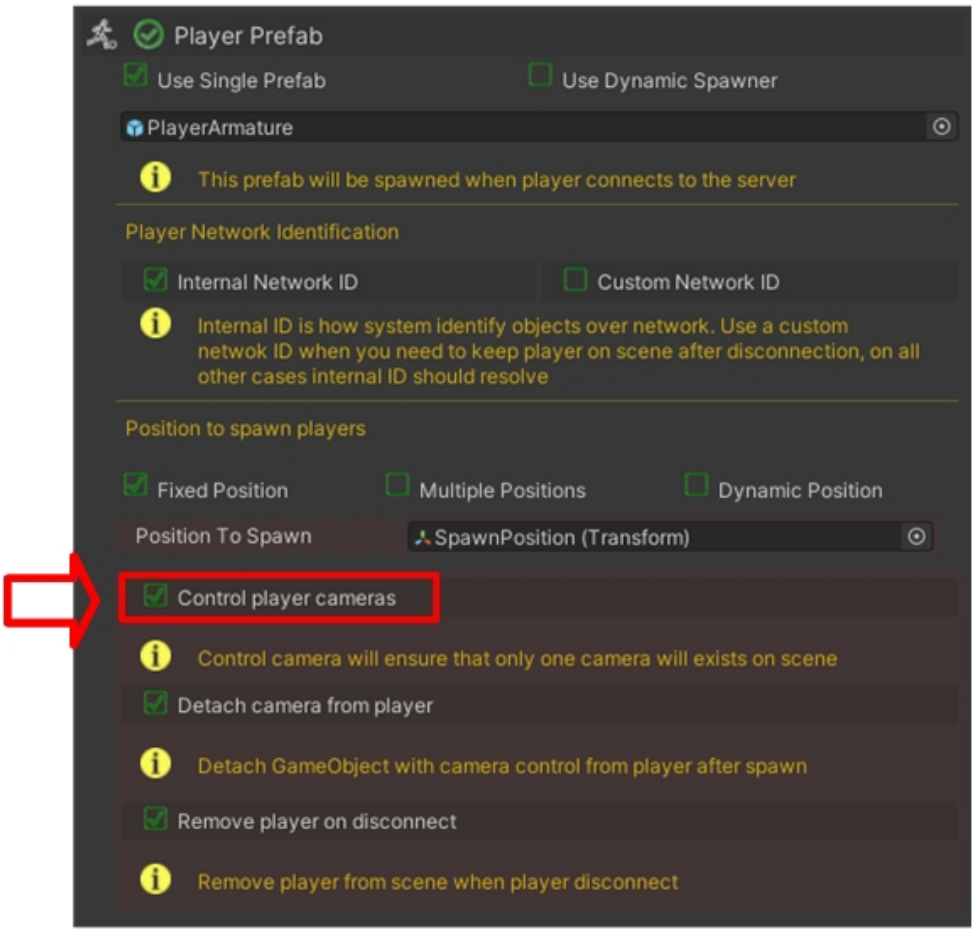
In the image above, the "PositionPrefabProvider" is the object with a component PositionProvider which has a method called "GetSpawnPosition".

To create a Dynamic Position provider you will need to create a script to provide the position to spawn the player, this script shall implement the interface `IInformationProvider` to return the position to use when the prefab was spawned (see Providers Scripts).

You can also check the example script `PositionProvider` in the examples folder. This script has a full implementation of how to provide positions.

Control player cameras

The option Control Player Cameras grand to ObjectNet the responsibility to control all cameras attached to the players.

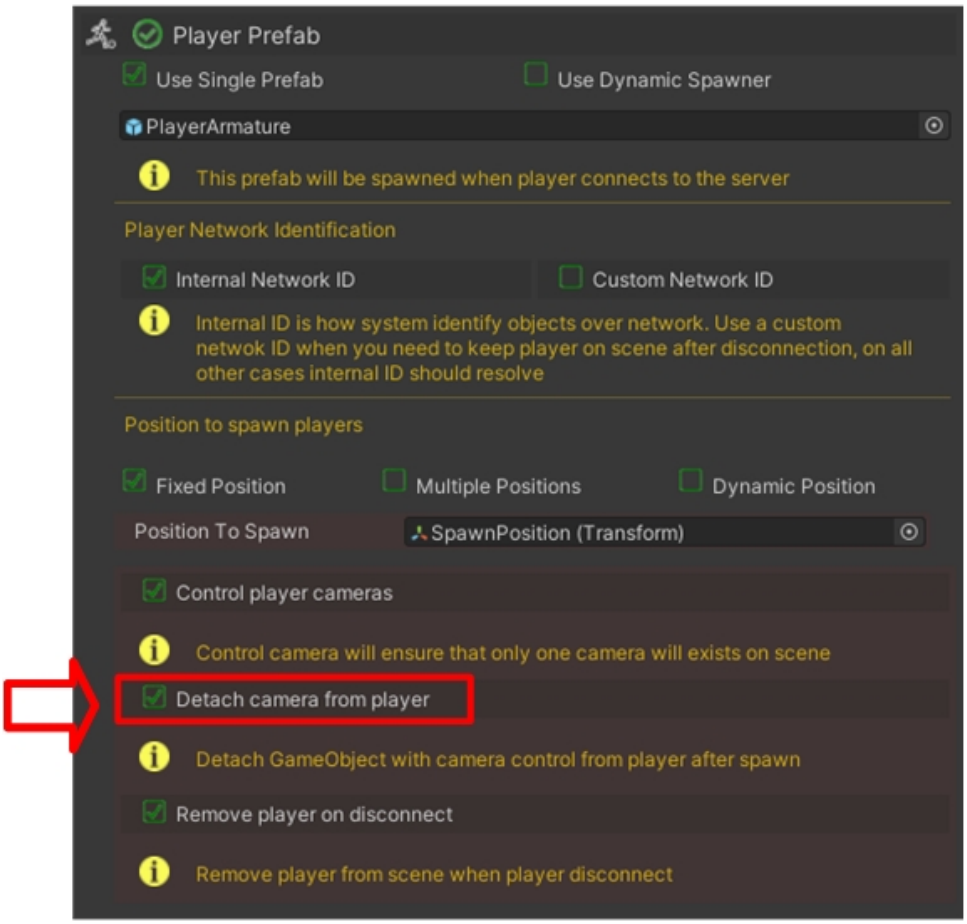


ObjectNet will ensure that only one camera is working on the system, when another player is spawned, ObjectNet will try to find the camera associated with the local player owner of the current instance of the game, and make this camera the only camera working on the scene.

This option is pretty useful to avoid mistakes when handling multiple cameras inside each player's prefab.

Detach camera from player

The option Detach camera from the player will remove the camera from the spawned player and put it at the root of the scene hierarchy.



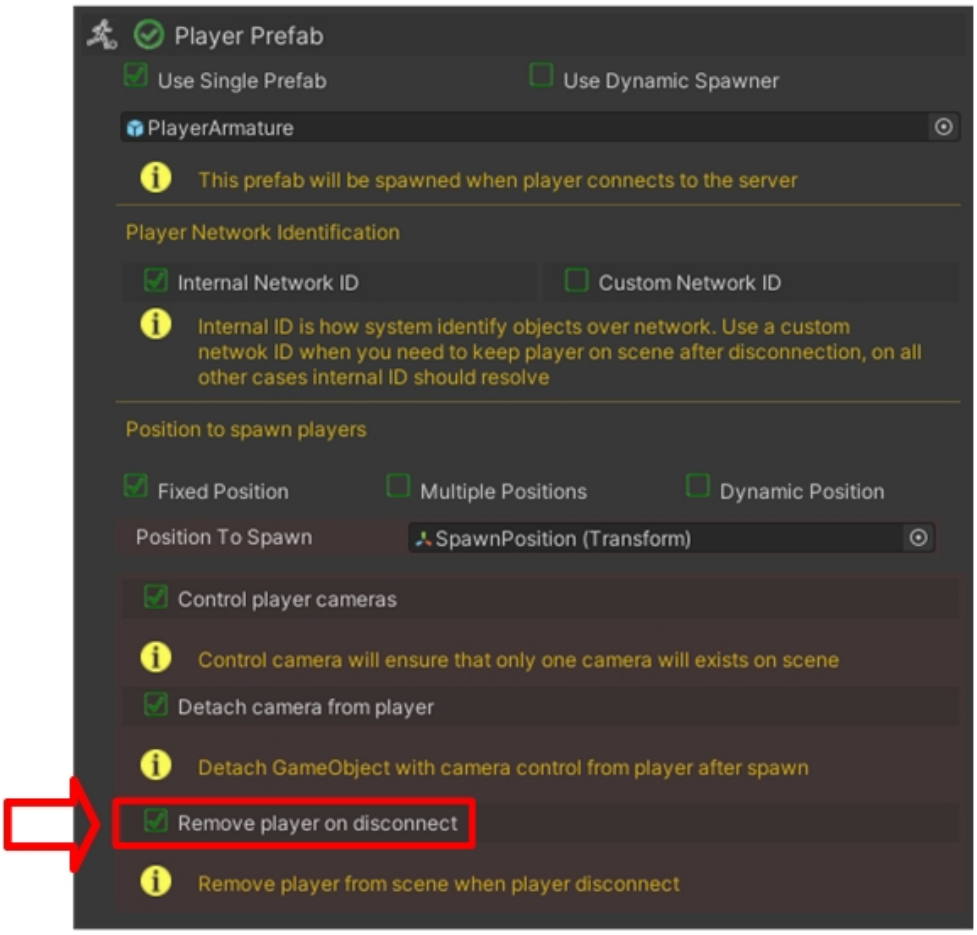
When the player camera is placed inside NetworkPrefab for convenience reasons but after spawned the same camera has some damping algorithm who not work properly due to the fact of being inside Player.

This option is useful because after spawned, the camera will be removed from the player object and placed at the top of the scene hierarchy.

Remove player on disconnect

ObjectNet controls all object's life cycles, this means that ObjectNet can detect when an object needs to be destroyed.

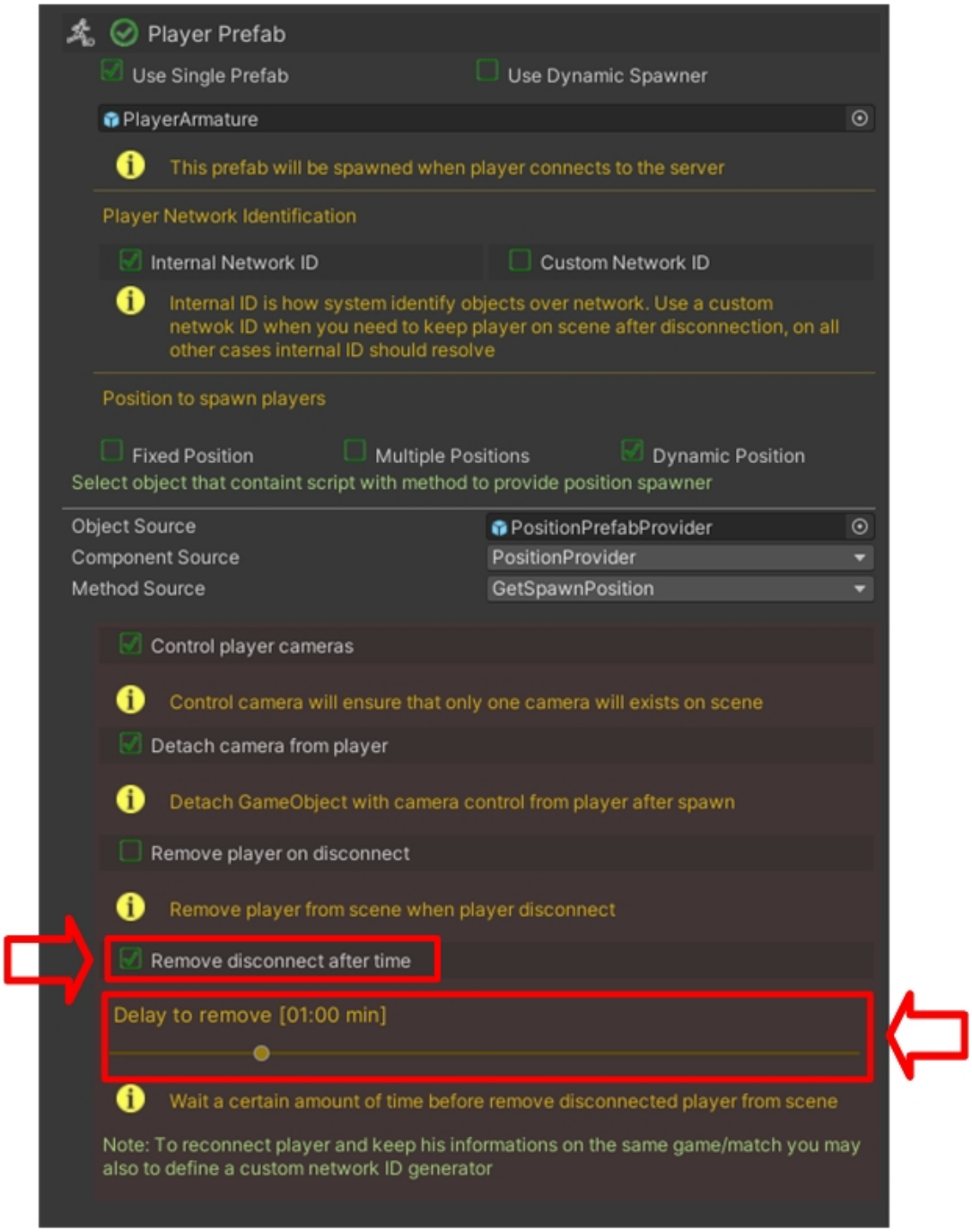
The option "Remove player on disconnect" will destroy the player's game object when the player disconnects from the server.



ObjectNet allows to destroy the player after a certain amount of time if disconnection is detected, the option "Remove disconnected after time" enables this feature.

If this option is not flagged, an extra option shall appear to allow to destruction player after a certain amount of time.

This option must be used in case a game allows them to play back on the game after disconnection, when combined with "Custom Network ID" the disconnected player may back to the game and restore the status that they had before disconnected.

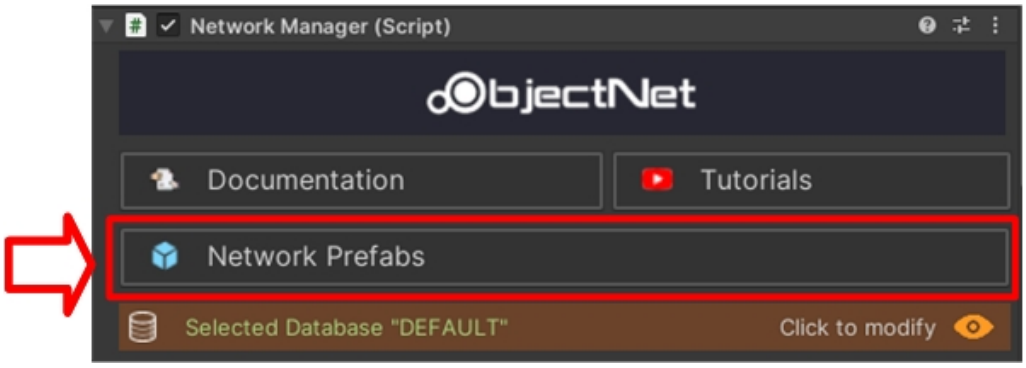


Note: If both options "Remove player on disconnect" and "Remove disconnected after time" were not flagged, the player object will keep on the scene while the match is not finished or the game is running.

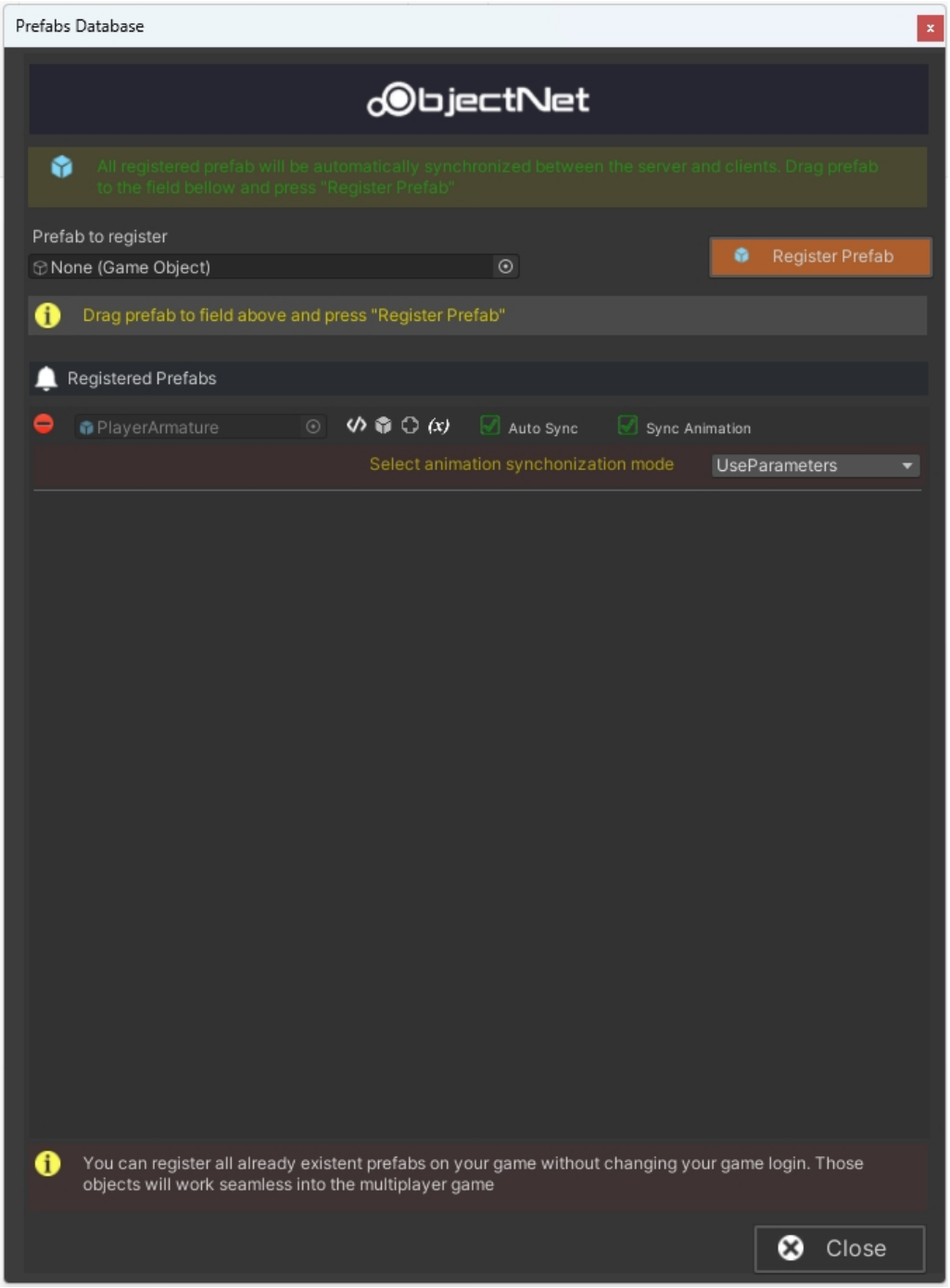
Network Prefabs

Network Prefabs is one fundamental aspect of ObjectNet.

ObjectNet controls all Network Object life-cycle internally, this means that ObjectNet catches and monitors all object creation and destruction.



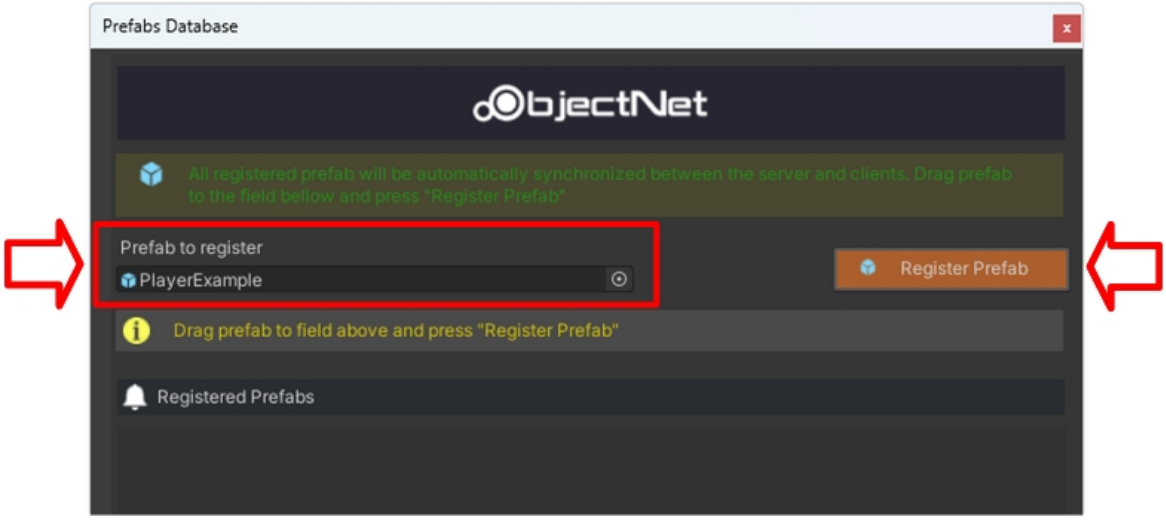
When clicking on "Network Prefab" button a new window shall appear.



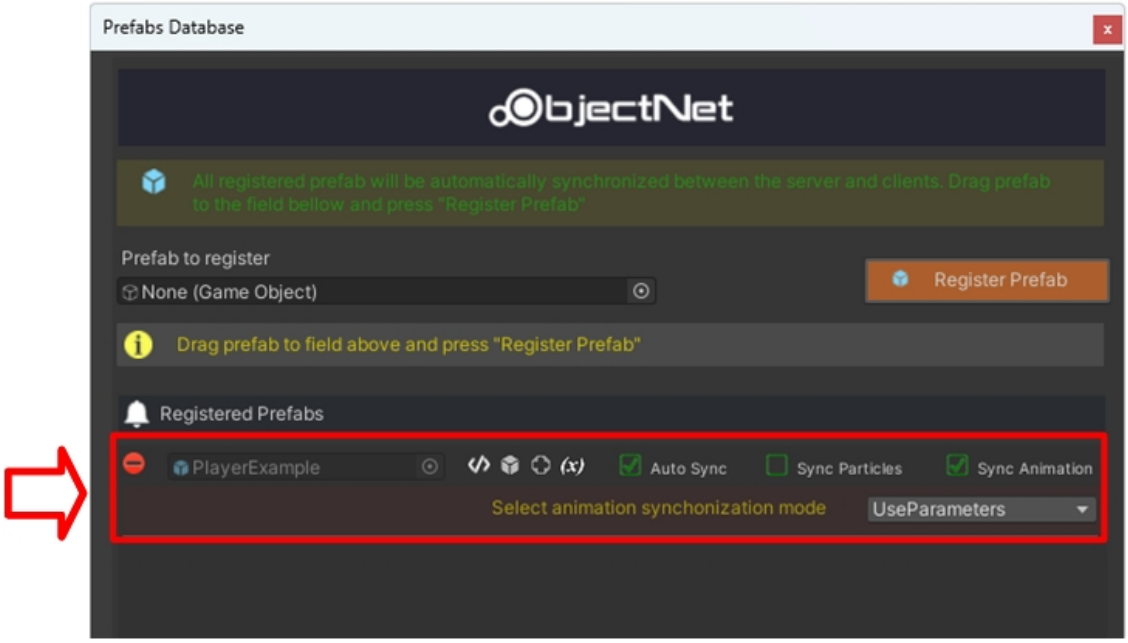
Registering Prefab

Network Prefabs works like a prefabs database, this means that all prefabs must be registered on this database.

1. Drag the prefab to the "Prefab to register" field and click on "Register Prefab".



2. The prefab shall be added to the list of registered prefabs



All objects that need to be spawned to all players must be included on this list, this will allow ObjectNet to automatically detect when the object was spawned and destroy and create it on all instances of the game.

Network Prefabs has a bunch of options to keep objects updated and synchronized over the network.

Note : Any player prefab must be registered as NetworkPrefab, otherwise prefab will not be spawned on the client instance (see Registering Prefab).

Synchronization Options

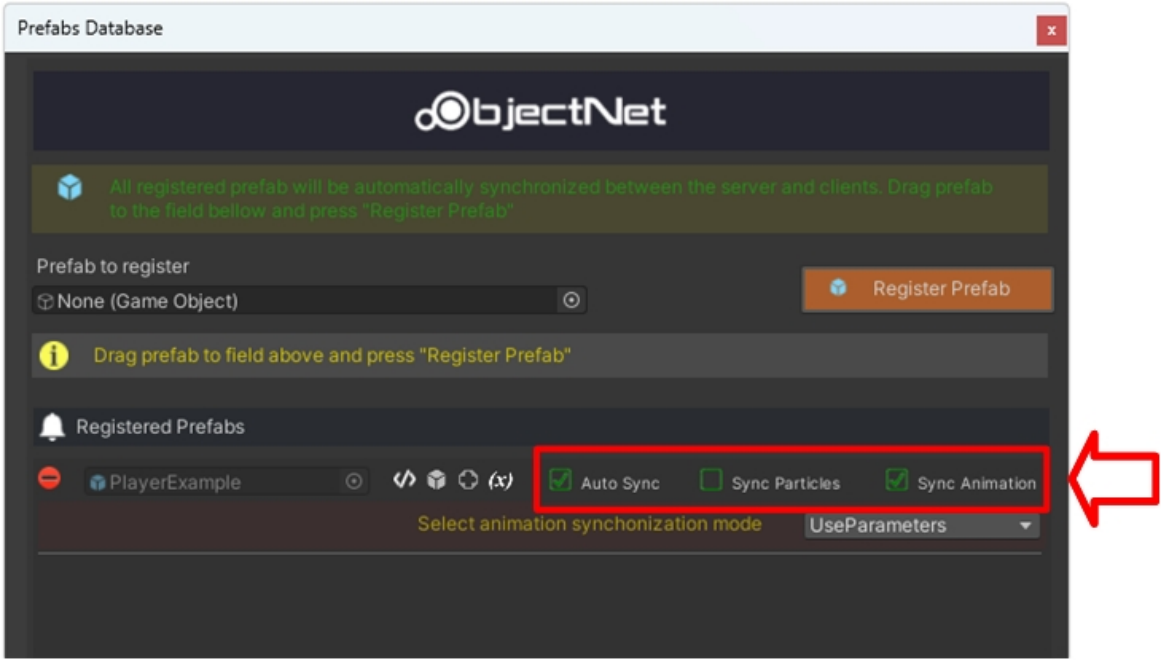
Network Prefab is a component handled by ObjectNet and its main purpose is to keep objects up to date regard to all network peers.

ObjectNet provides the facility to automatically update some object aspects and behaviors by checking each option on each prefab.

1. **Auto Sync**
   - This option will keep the Position, Rotation, and Scale of the object synchronized on all network peers

2. **Sync Particles**
   - This option will keep the particle system synchronized on all peers, when the particle is started or stopped on an active object, all network peers will be notified to do the same. The `RateOverTime` and `RateOverDistance` will also be synchronized.

3. **Sync Animation**
   - ObjectNet also synchronizes animations automatically on network prefabs. Animation can be synchronized using 3 different modes:
     - Using Controller : This will try to use an animation controller to keep animations synchronized.
     - Using Parameters : This will try to use animation parameters to keep animations synchronized (recommended).
     - Manually : On this option, animations must be played and stopped manually by code using the `NetworkAnimationController` animation API (see Manual animation).

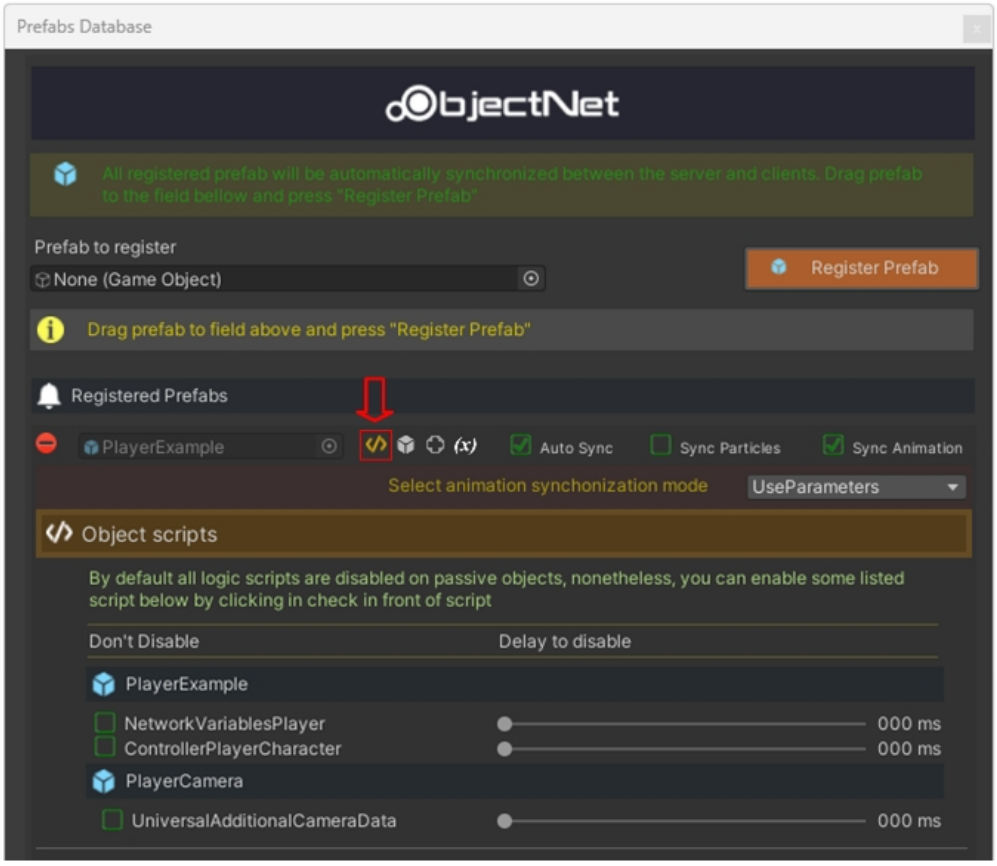


Prefab Scripts

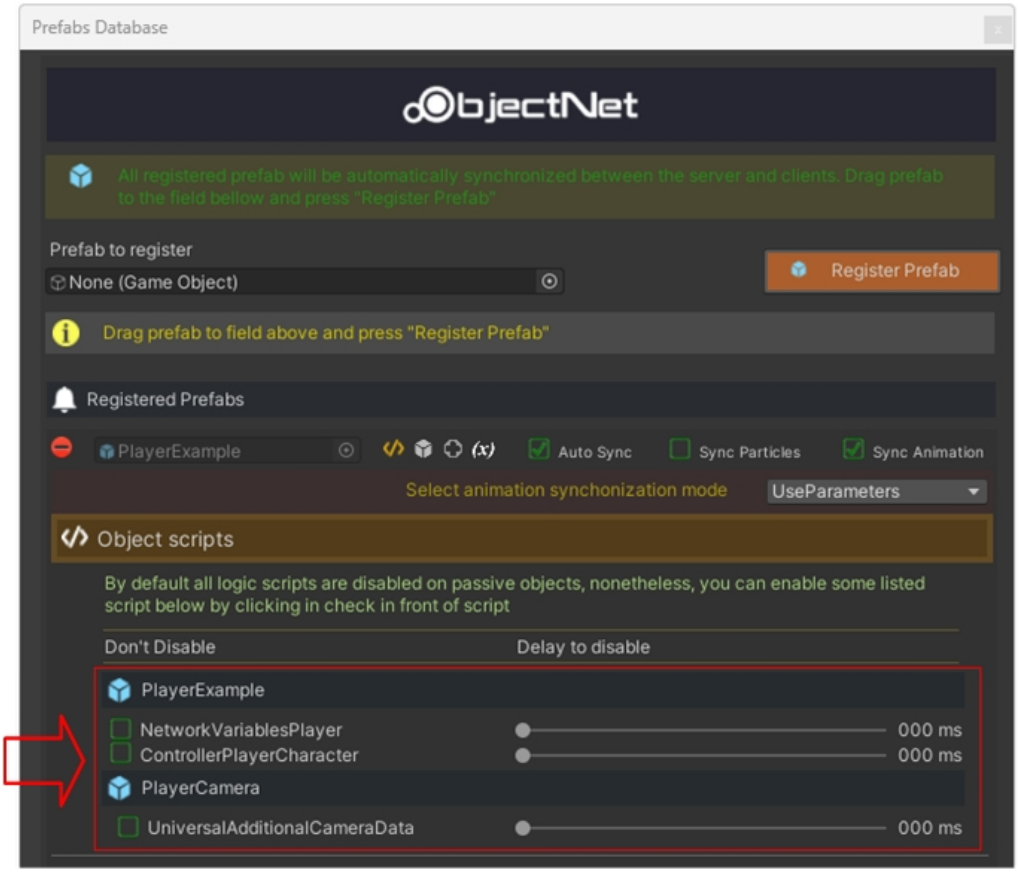
Network Prefabs like any game object may have scripts and components to control their behavior.

ObjectNet automatically disables all scripts when the game object is in passive mode since this object will be controlled remotely by the server, nonetheless, in certain circumstances some scripts shall be activated.

Scripts can be enabled/disabled by clicking on the script's icon.



A list of all scripts on each game object (including children) will appear, and you can enable/disable any script.



By flagging the check box you can enable the script to run on passive instances.

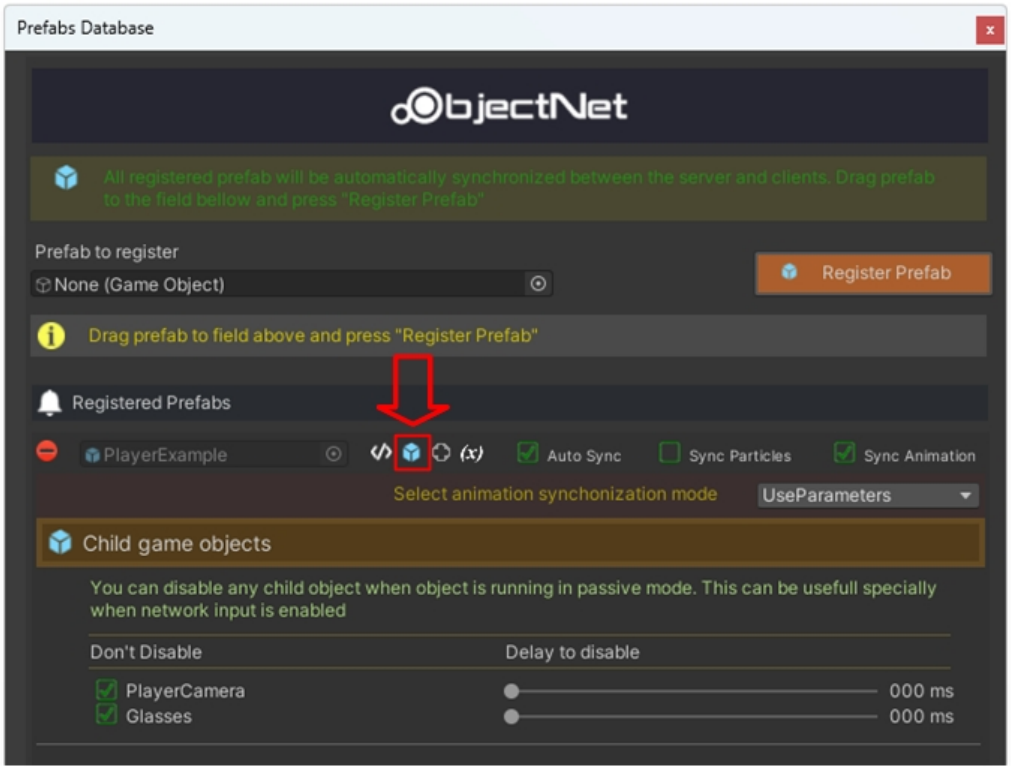
In case any script needs to be executed before being disabled (if some data was collected in **OnAwake** on **OnStart** methods) you shall keep the checkbox unchecked and use the slider to tell to ObjectNet to disable this script only after a certain amount of time.

Child Objects

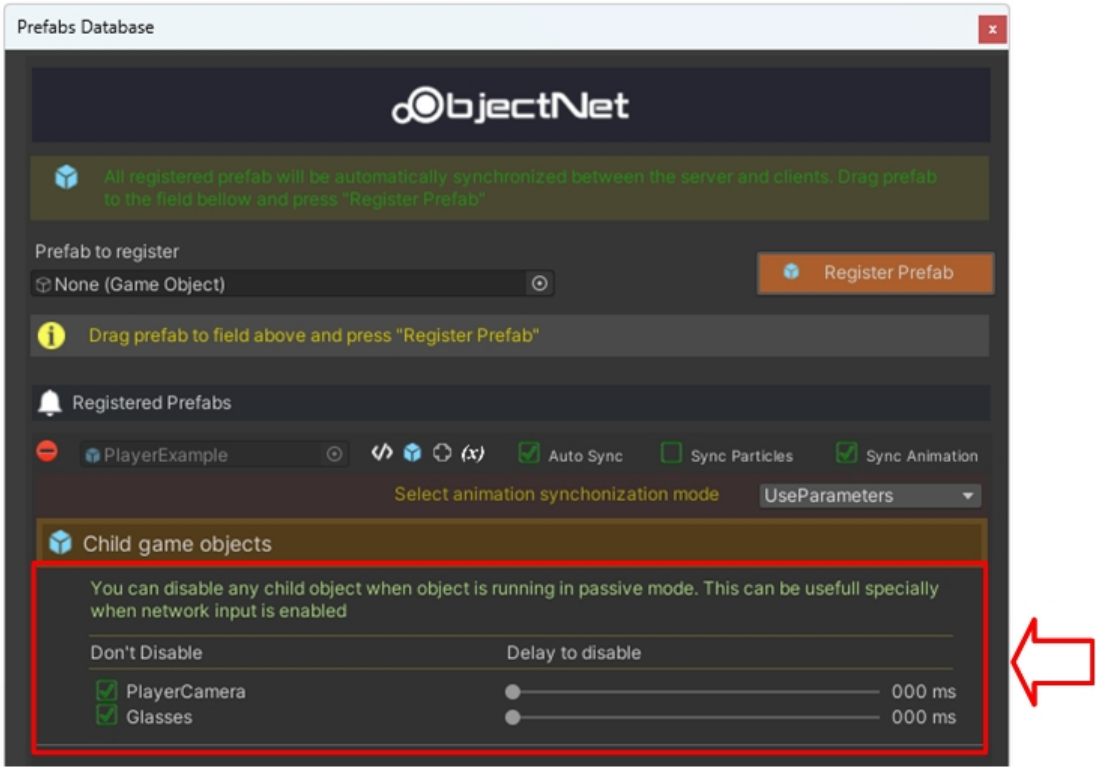
Network Prefabs can be composed of many objects, always when the root game object has child objects.

ObjectNet provides the facility to disable some child objects when NetworkPrefab is running in passive mode.

Child Objects can be child objects by clicking on the script's icon.



A list of all child objects will appear, and you can enable/disable any object.



By flagging the check box you can enable objects on passive instances.

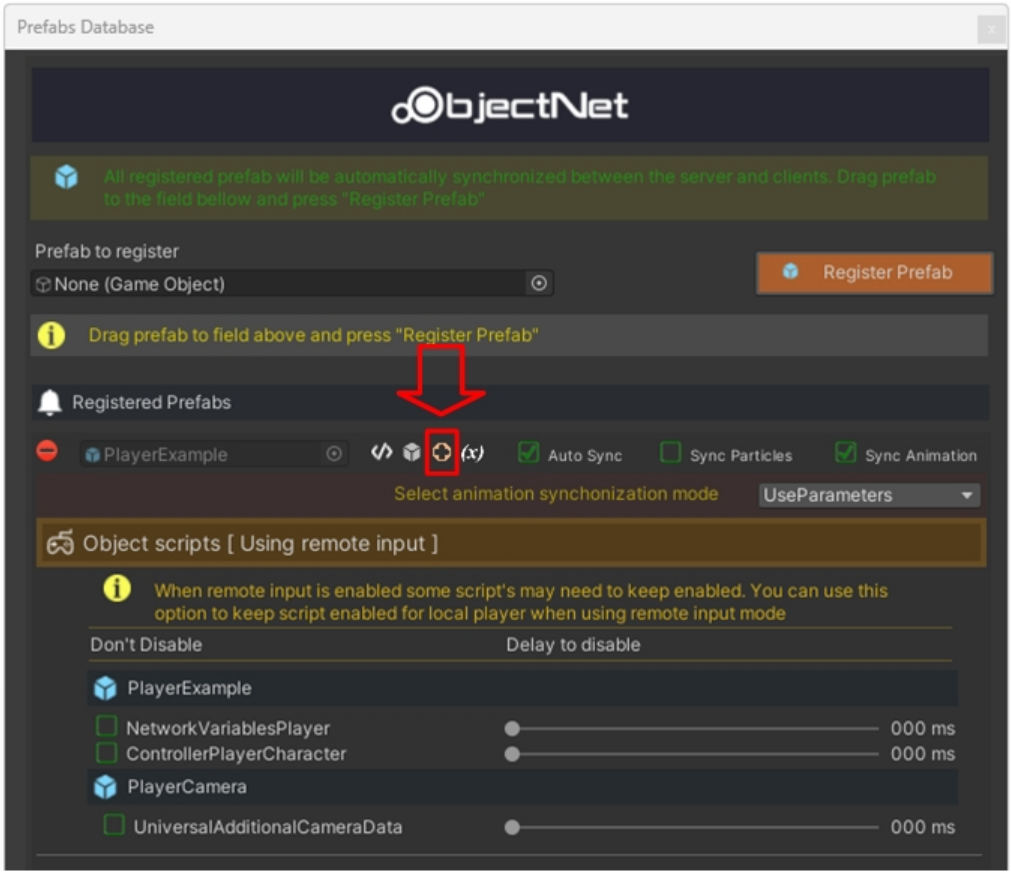
In case any child needs to be executed before being disabled (if some data was collected in OnAwake on OnStart methods attached to this object) you shall keep the checkbox unchecked and use the slider to tell to ObjectNet to disable only after a certain amount of time.

Prefab Scripts (Remote Input)

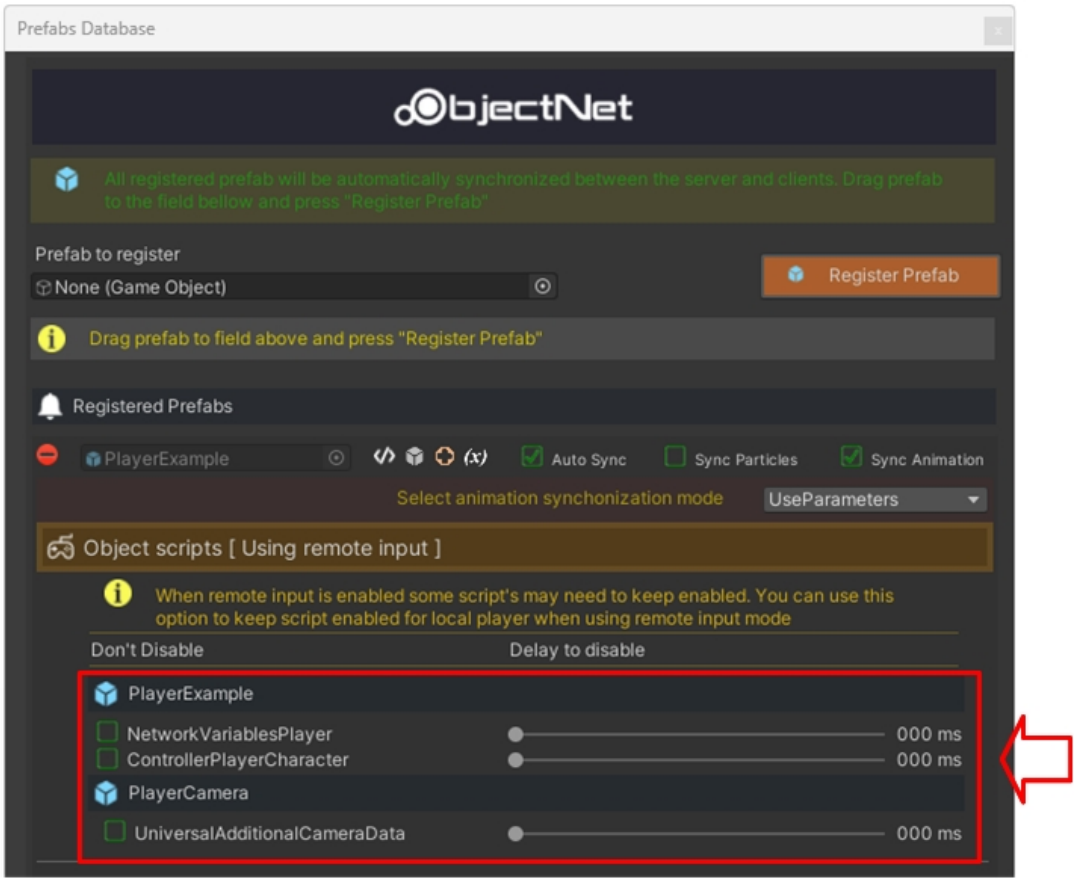
When Remote Input is enabled, Network Prefabs can behave differently from his normal behavior since the local player is running his player prefab on passive mode and sending input only.

ObjectNet automatically disables all scripts when the game object is in passive mode, although in this specific case, some script needs to be executed, one example is the script that executes Input Checks (for example), otherwise Network Inputs will not be sent to the server.

Script over Remote input can be enabled/disabled by clicking on an icon.



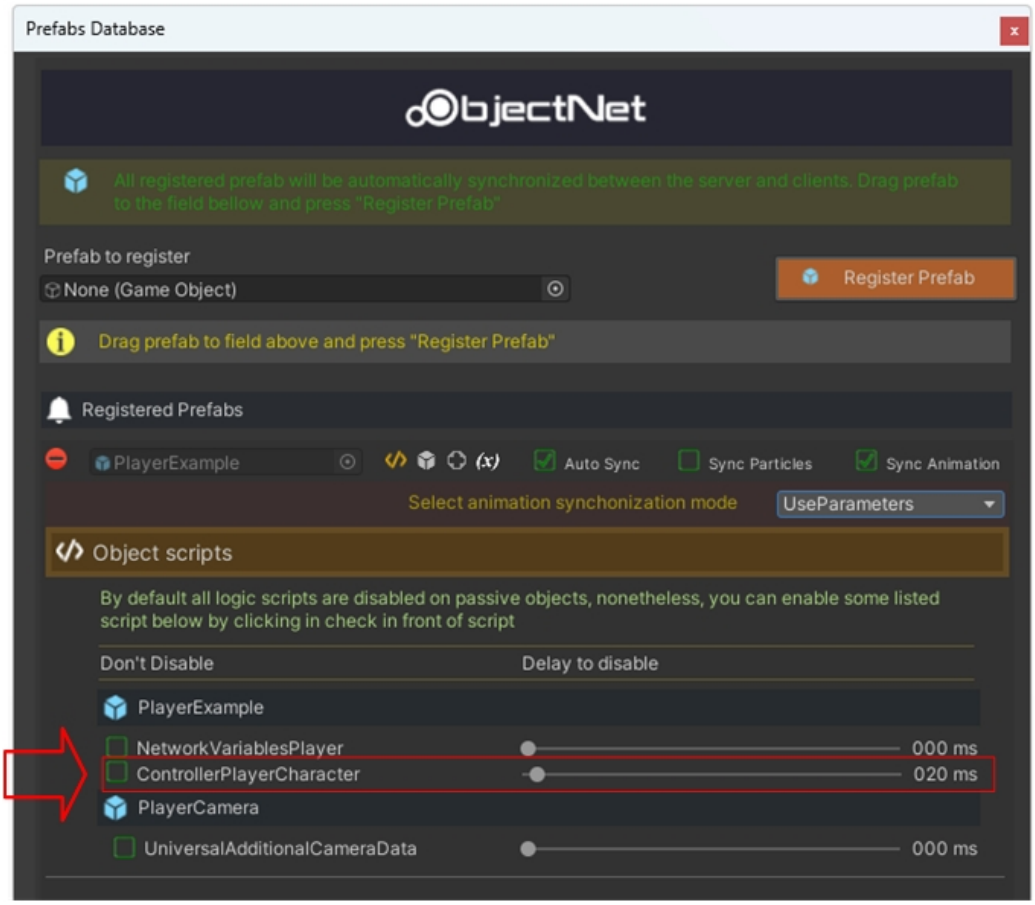
A list of all scripts on each game object (including children) will appear, and you can enable/disable any script.



By flagging the check box you can enable the script to run on passive instances.

In case any script needs to be executed before being disabled (if some data was collected in OnAwake on OnStart methods) you shall keep the checkbox unchecked and use the slider to tell to ObjectNet to disable this script only after a certain amount of time.

In the example below, the script "ControllerPlayerCharacter" will only be disabled after 20 ms of object creation.

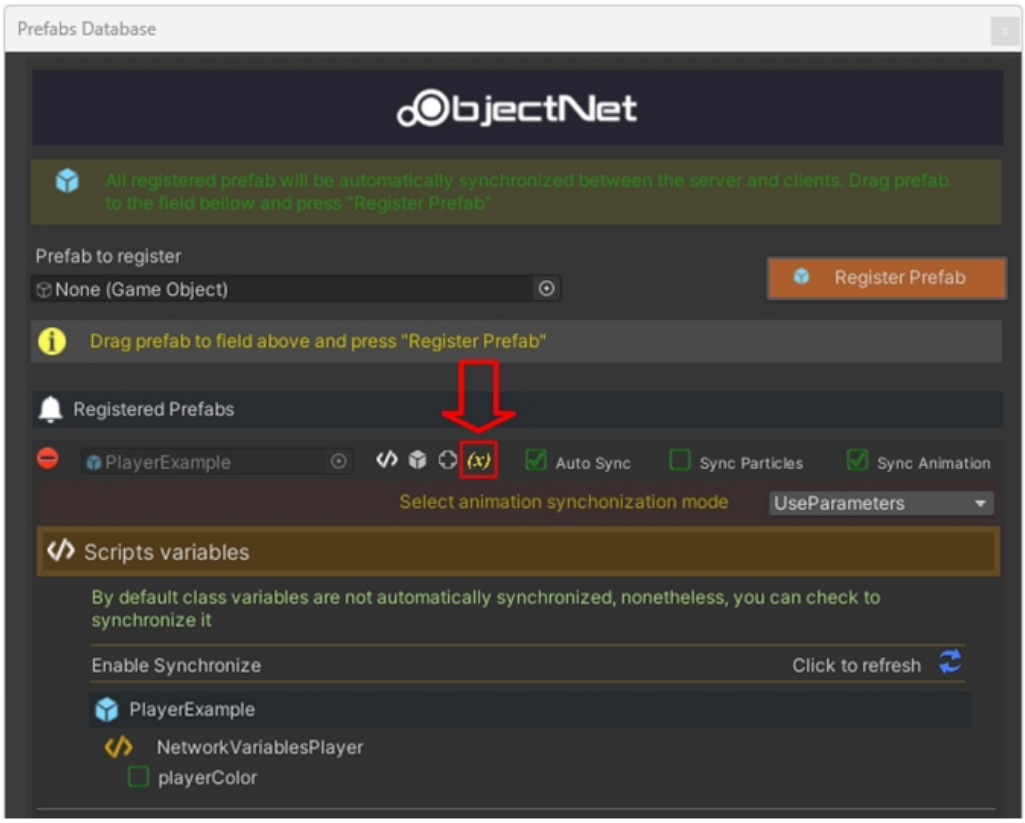


Script Variables

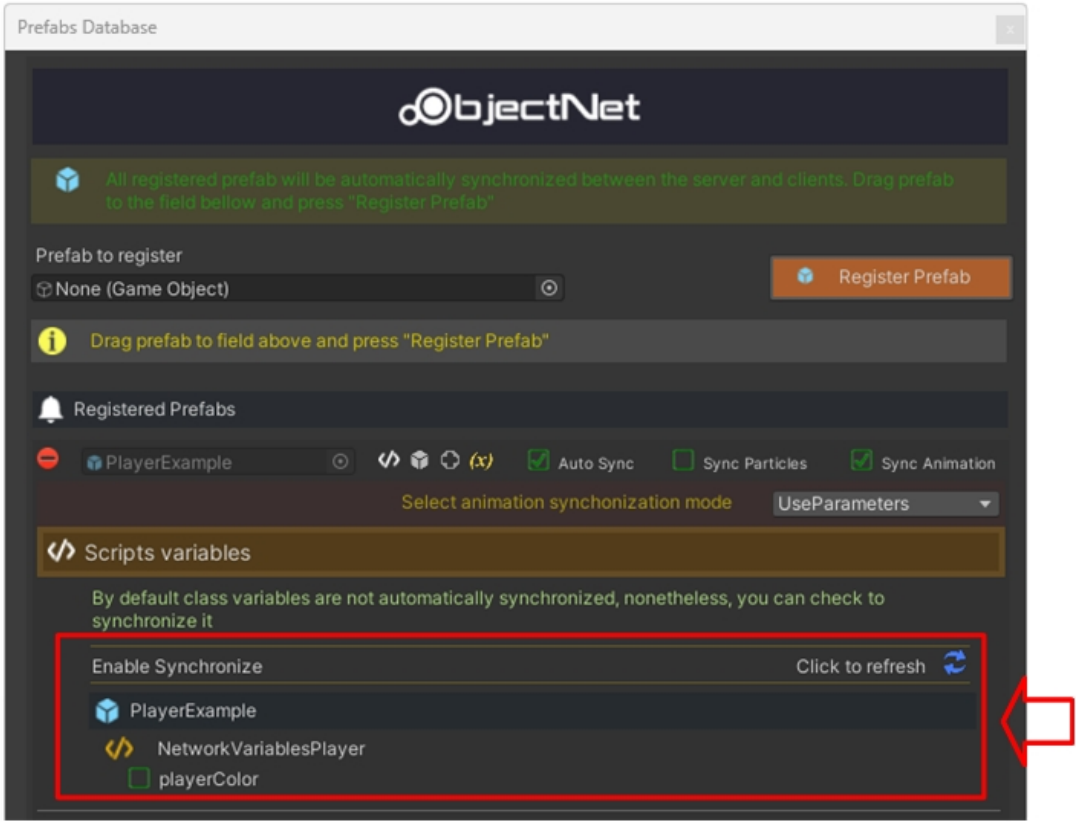
ObjectNet provides the facility to automatically synchronize variables across the network. This can be very useful to avoid a lot of events to keep objects updated.

Network Variables are synchronized from Active to Passive instance, this means that a variable modified on the Active instance will be sent over the network and applied on all Passive instances.

Network Variables can be enabled/disabled by clicking on the variables icon.



A list of all variables on the main NetworkObject component shall appear and you can enable/disable any script.



By flagging the check box you can enable variable synchronization across the network.

Note: Network Variables is available only for scripts inherited from NetworkBehavior, if the script was not

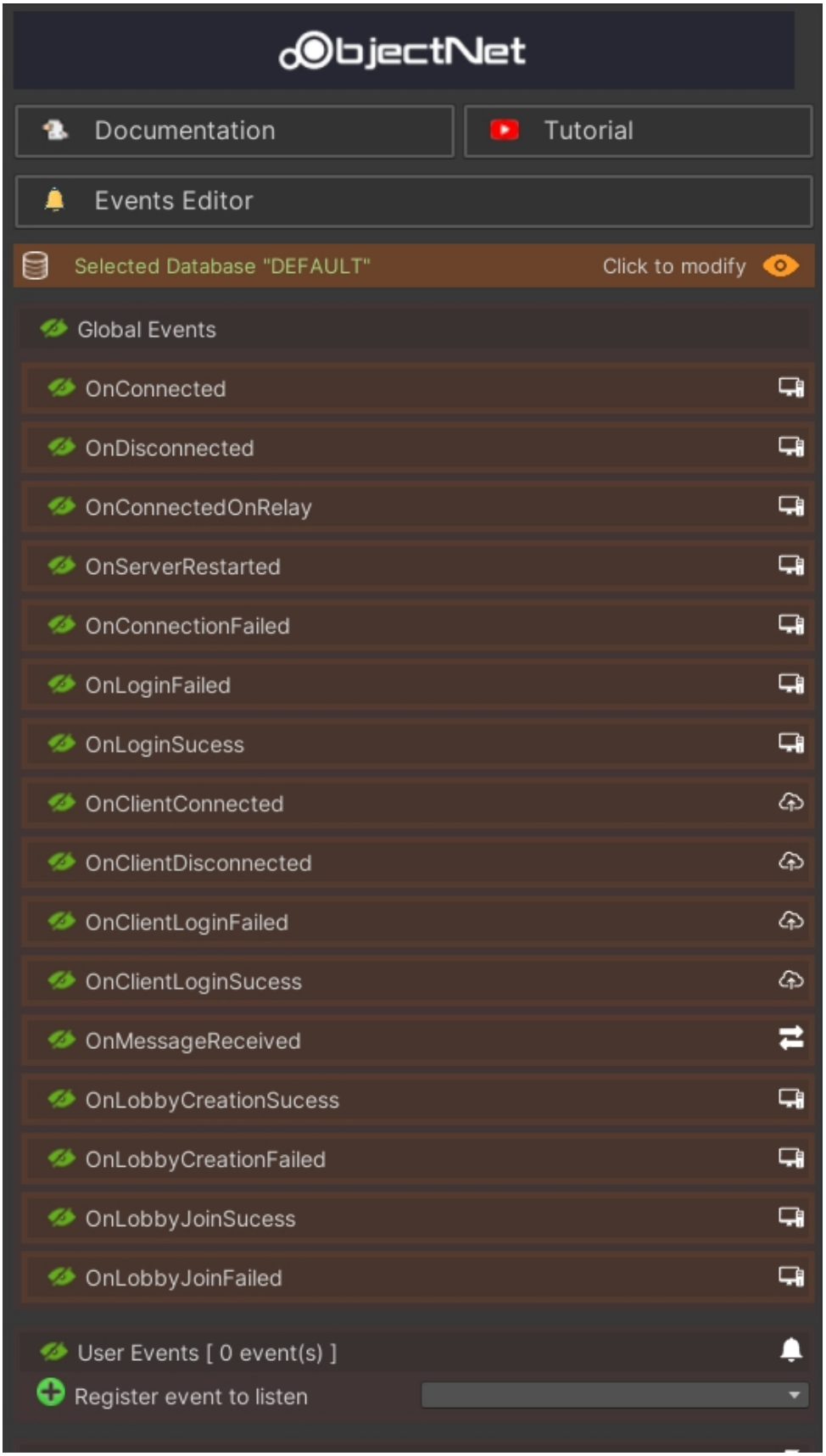
inherited from NetworkBehavior is not possible to synchronize variable values (see Network Variables).

Network Events Manager

ObjectNet provide a embedded system to make easy to developer to handle with events.

NetworkEventsManager allow user to manage events of two types.

- Global Events
- Custom Events



Global Events

ObjectNet considers a Global Event, all events related to internal engine control.

NetworkEventsManager allows the user to catch those events and implement any custom logic.

Currently, NetworkEventsManager is allowed to listen to the following global events :

- **OnServerStarted**
  - **Event triggered when server initialize his connection and start to listen for clients.**
- **OnConnected**
  - **Event triggered when the client successfully connects to the server.**
- **OnDisconnected**
  - **Event triggered when the client disconnects from the server.**
- **OnConnectedOnRelay**
  - **Event triggered when the client establishes a connection with a relay server.**
- **OnServerRestarted**
  - **Event triggered when the client detects that the server has restarted.**
- **OnConnectionFailed**
  - **Event triggered when the client fails to connect to the server.**
- **OnLoginFailed**
  - **Event triggered when the client fails to log in to the server.**
- **OnLoginSucess**
  - **Event triggered when the client successfully logs in to the server.**
- **OnClientConnected**
  - **Event triggered when a new client connects to the server.**
- **OnClientDisconnected**
  - **Event triggered when a client disconnects from the server.**
- **OnClientLoginFailed**
  - **Event triggered when a client attempts to log in with invalid credentials.**
- **OnClientLoginSucess**
  - **Event triggered when a client successfully logs in to the server.**
- **OnMessageReceived**
  - **Event triggered when a message is received by either the client or the server.**
- **OnLobbyCreationSucess**
  - **Event triggered when the client successfully creates a lobby.**
- **OnLobbyCreationFailed**
  - **Event triggered when the client fails to create a lobby.**
- **OnLobbyJoinSucess**
  - **Event triggered when the client successfully joins a lobby.**
- **OnLobbyJoinFailed**
  - **Event triggered when the client fails to join a lobby.**

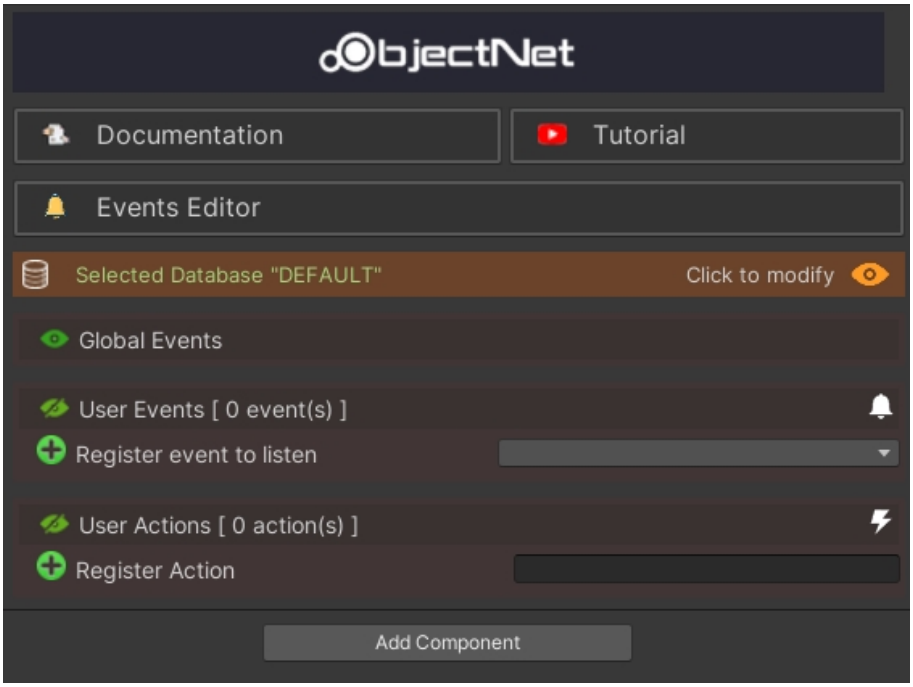
Full detailed information can be found on API documents.

You can find a full example of how to implement global events listener on "**15 - Global Events**" scene.

Custom Events

ObjectNet allows developers to create, listen, and send custom events managed by NetworkEventsManager.

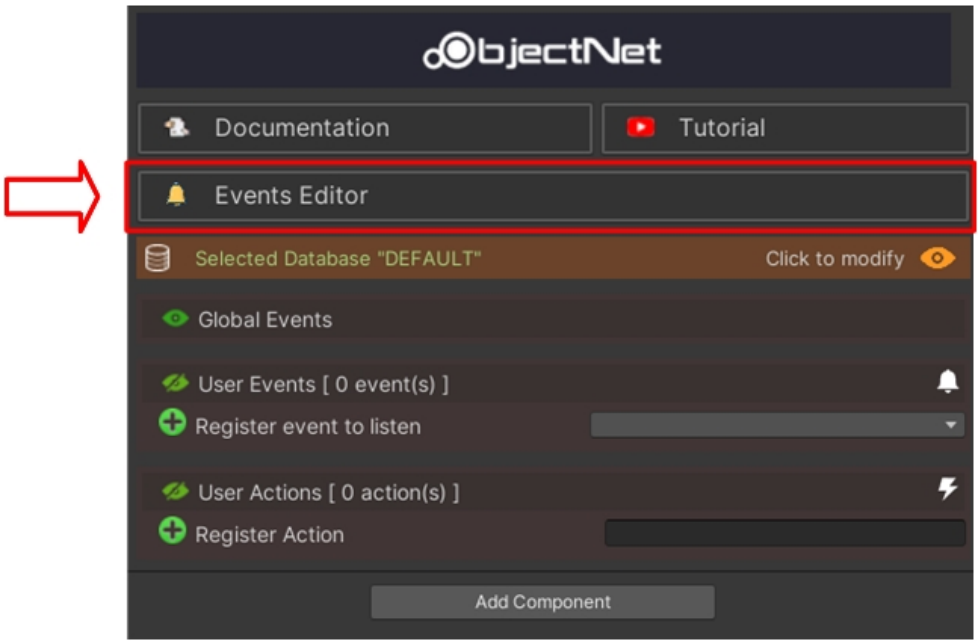
NetworkEventsManager allows users to create customized events to catch the specific events of their game.



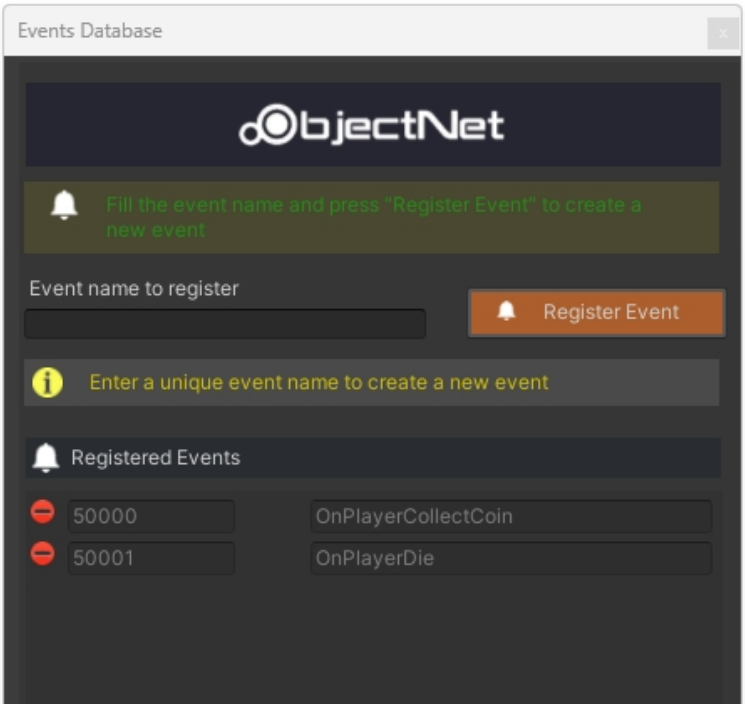
You can find a full example of how to implement custom events on "**16 - Global Events**" scene.

Registering Event

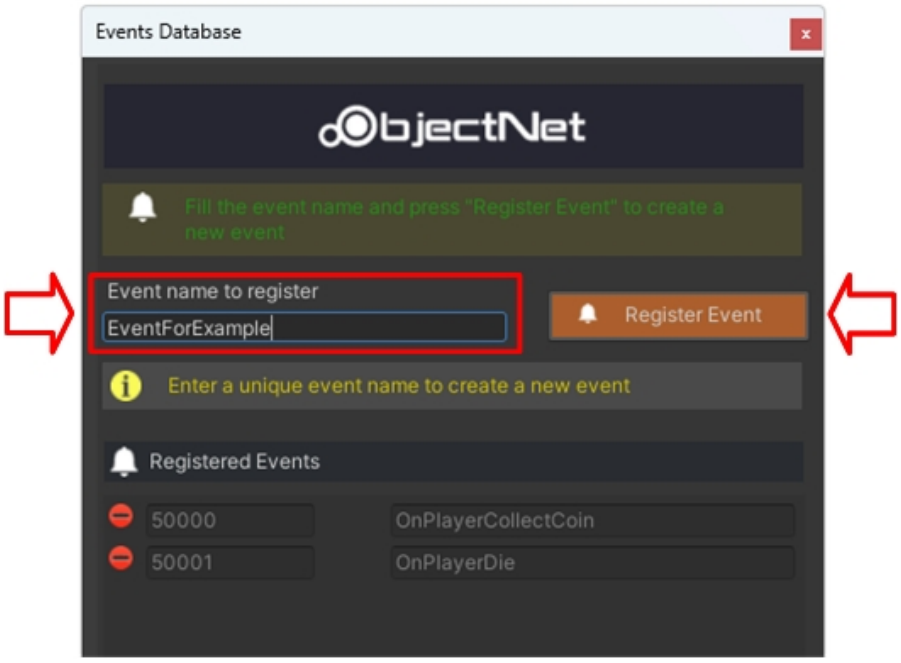
To register a new event you need to select **NetworkEventsManager** and click on **Event Editor** button.



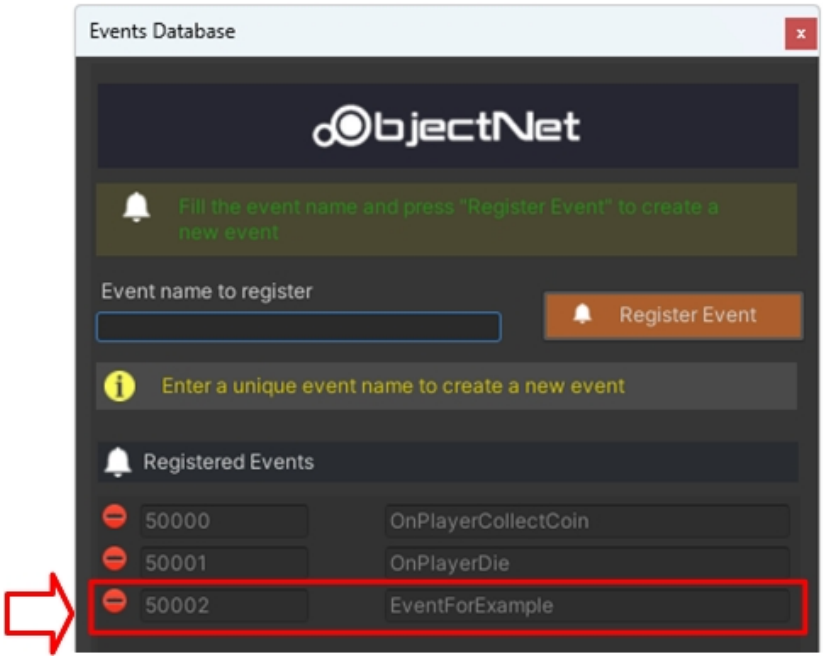
A new **Events Database** window shall appear :



Fill the event name and press "**Register Event**"



A new event shall appear on events list.

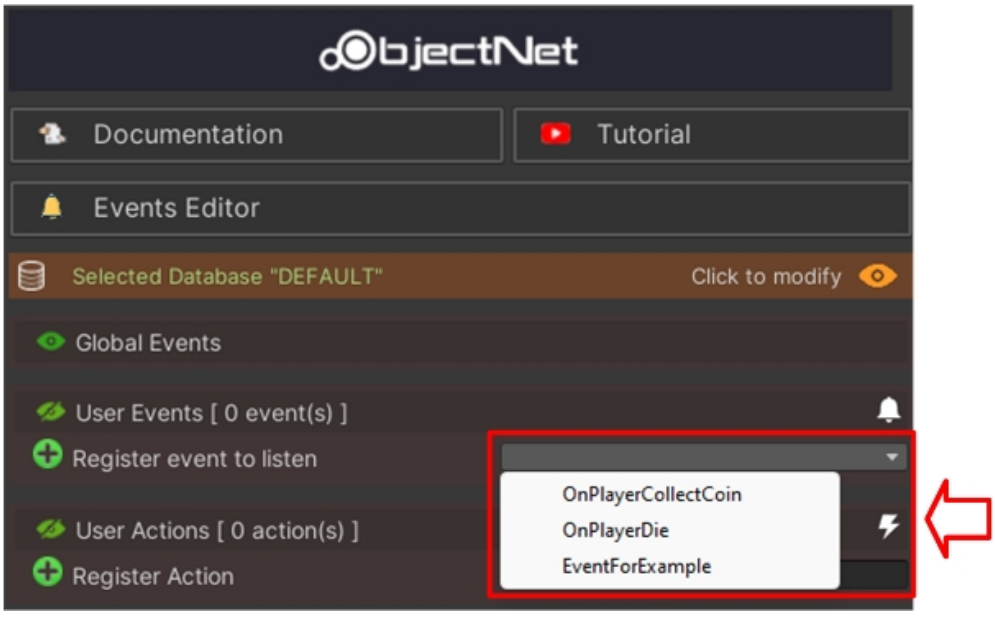


Each event will have its own unique code, this code is an Integer that is used to identify events over the network.

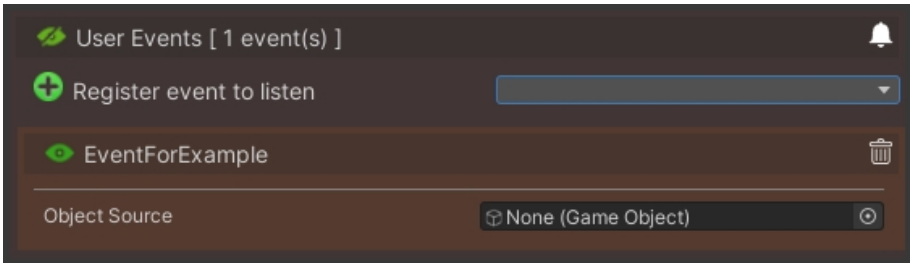
User Event

User Event is the way to NetworkEventsManager allows the user to listen when some custom event arrives.

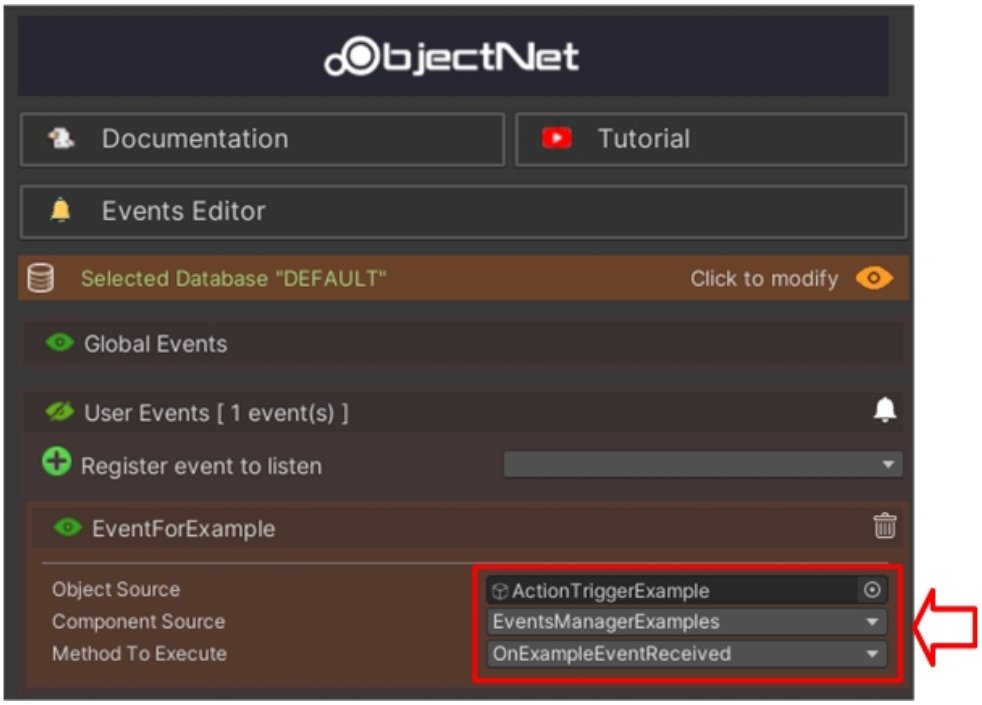
Select the event that you wish to listen to and click on "Register event to listen".



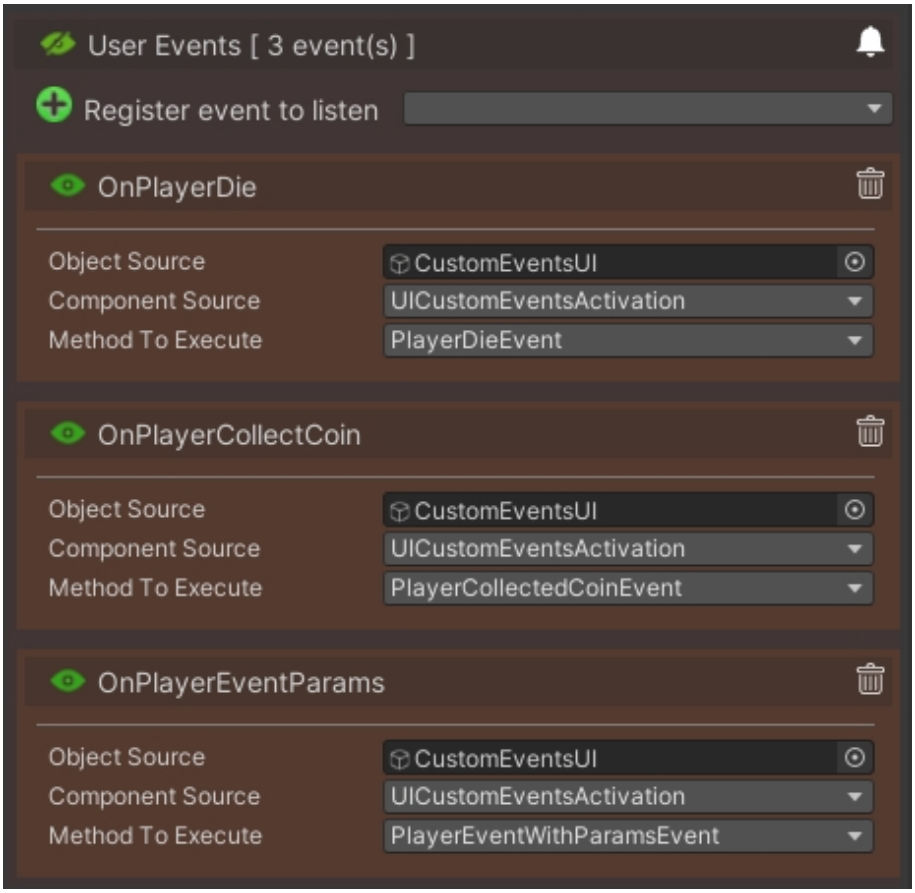
The selected event will be registered.



Now you need to select the action to take when this event arrives.



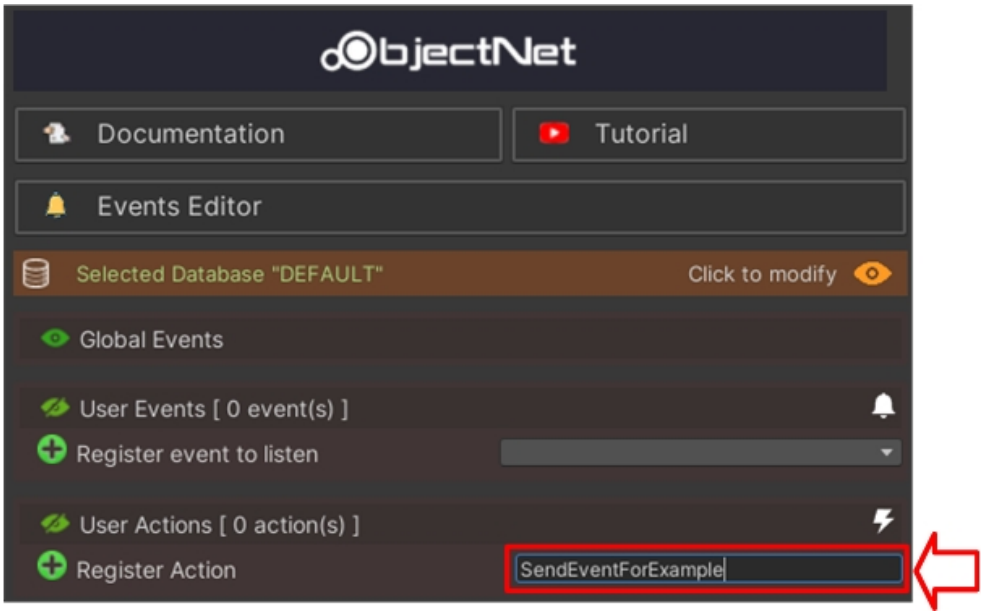
You can find a full example of how to implement custom events on "**16 - Custom Events**" scene. In this scene, you will find scripts and objects to clarify how to implement your own custom events.



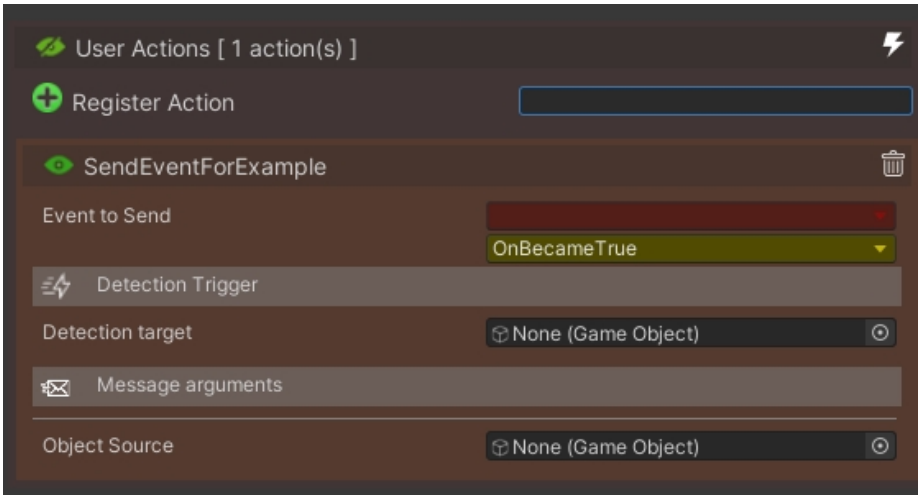
User Actions

User Action is how NetworkEventsManager allows the user to send vents when some status is reached.

Fill in the action name and click on "Register Action" button.

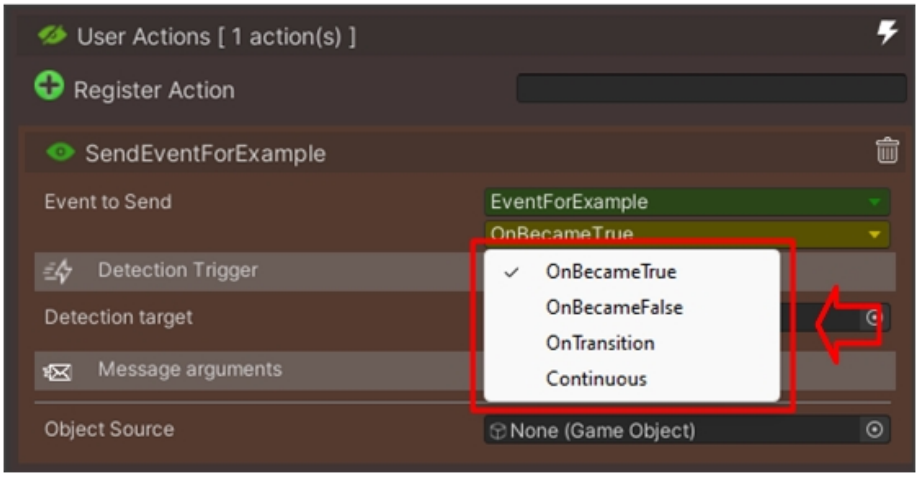


The action will be registered on the User Actions list.

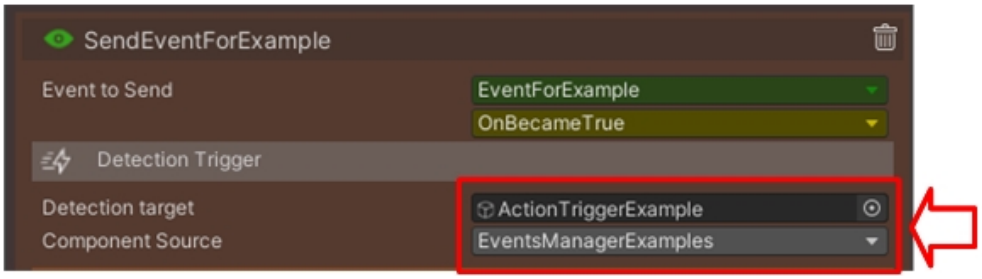


The first to do is select which event you wish to send when action is triggered.

The next step is to decide when you wish to trigger this event, the possible values can be found on `ActionExecutionMode` (see API).

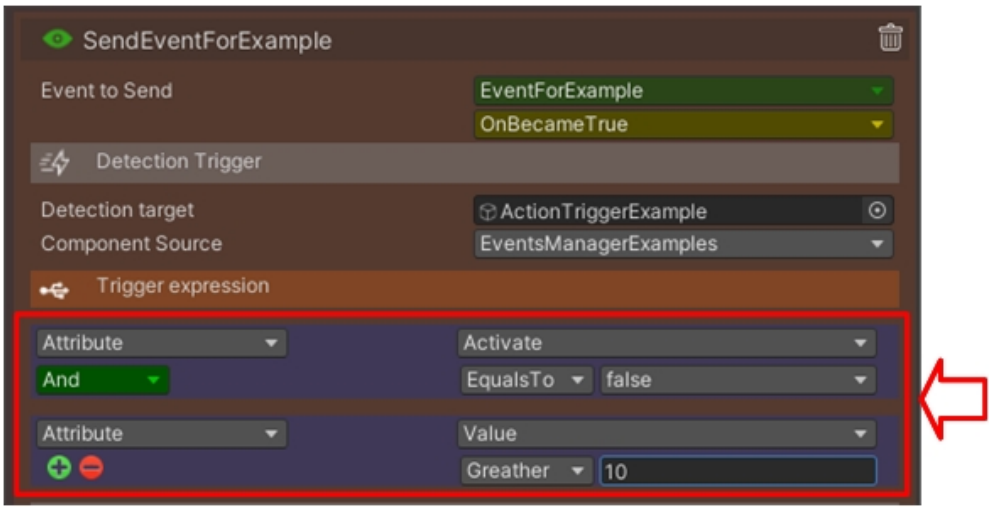


Now you need to assign the object and component to watch, this component must be a component that exists in scene .

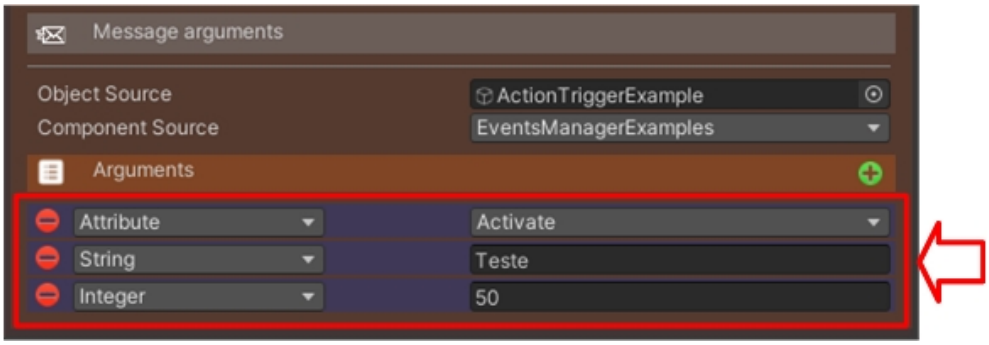


Now you to fill the expression that will be evaluated to check if this condition matches with the selected on `ActionExecutionMode`.

On this part you can select component (script) attributes or function witch return allowed types. The system will evaluate this condition on each execution frame to check if condition is reached.

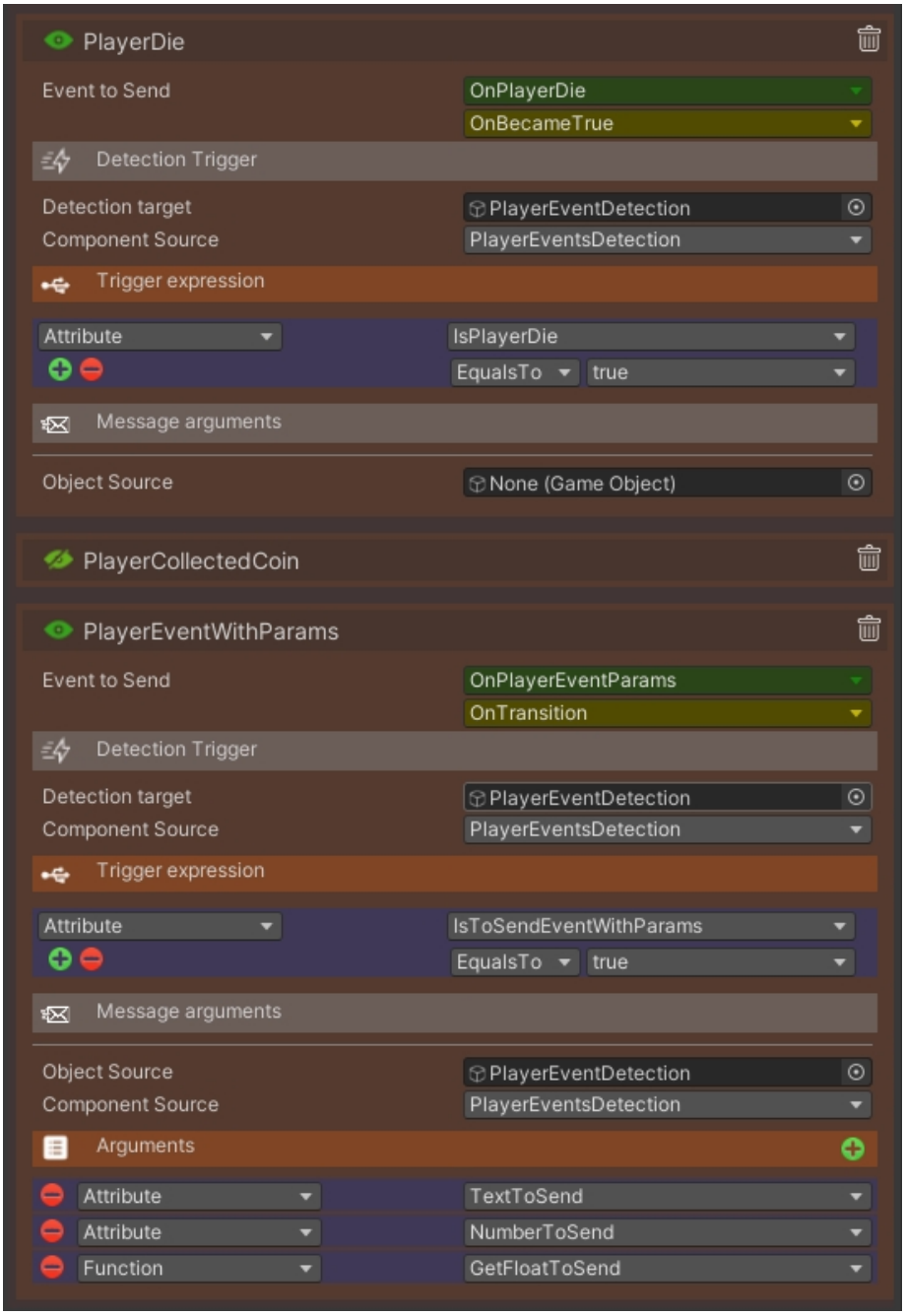


The last step (when applicable) is define witch values will be included on message when the selected event was send.



Those values will be included on event when **ObjectNet** send it over network, and must be read when message arrives.

You can found a full example about how to implement custom events on "**16 - Custom Events**" scene. On this scene you will find scripts and objects to clarify how implement you own custom events.

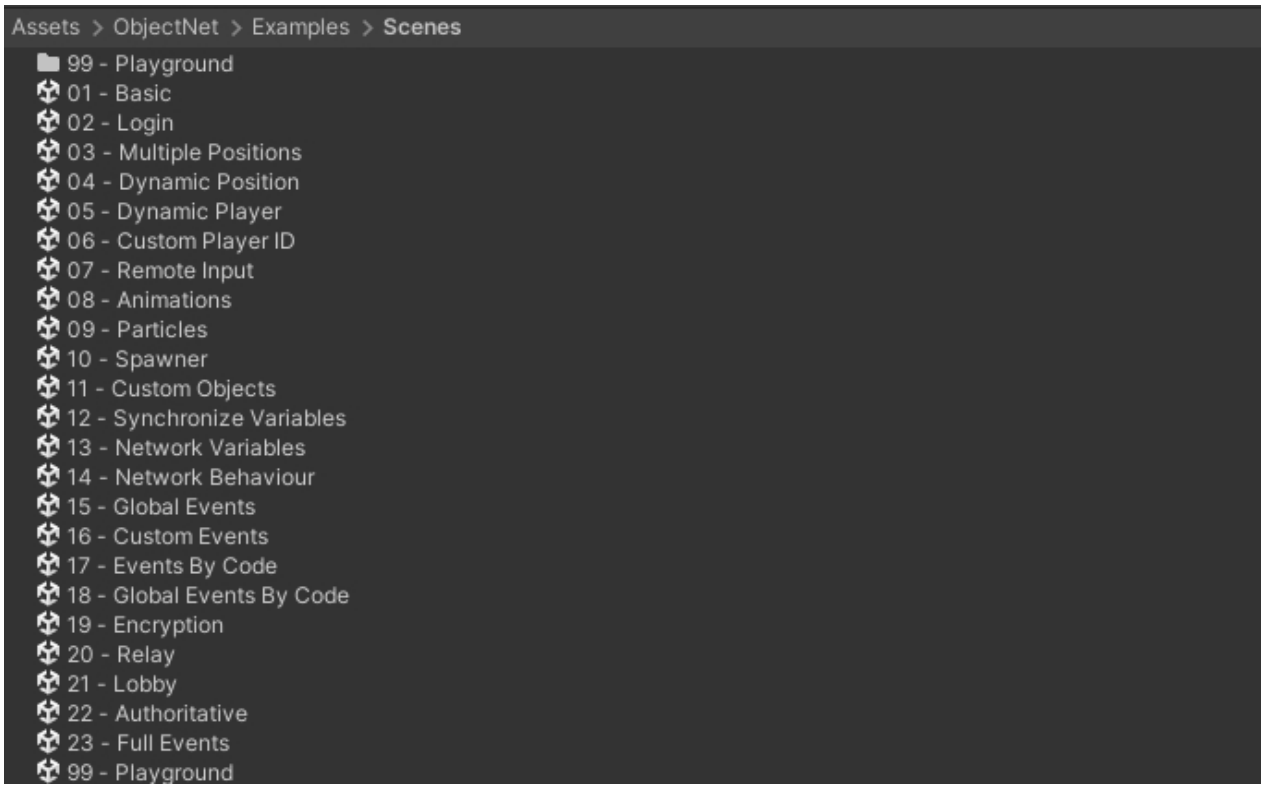


Example Scenes

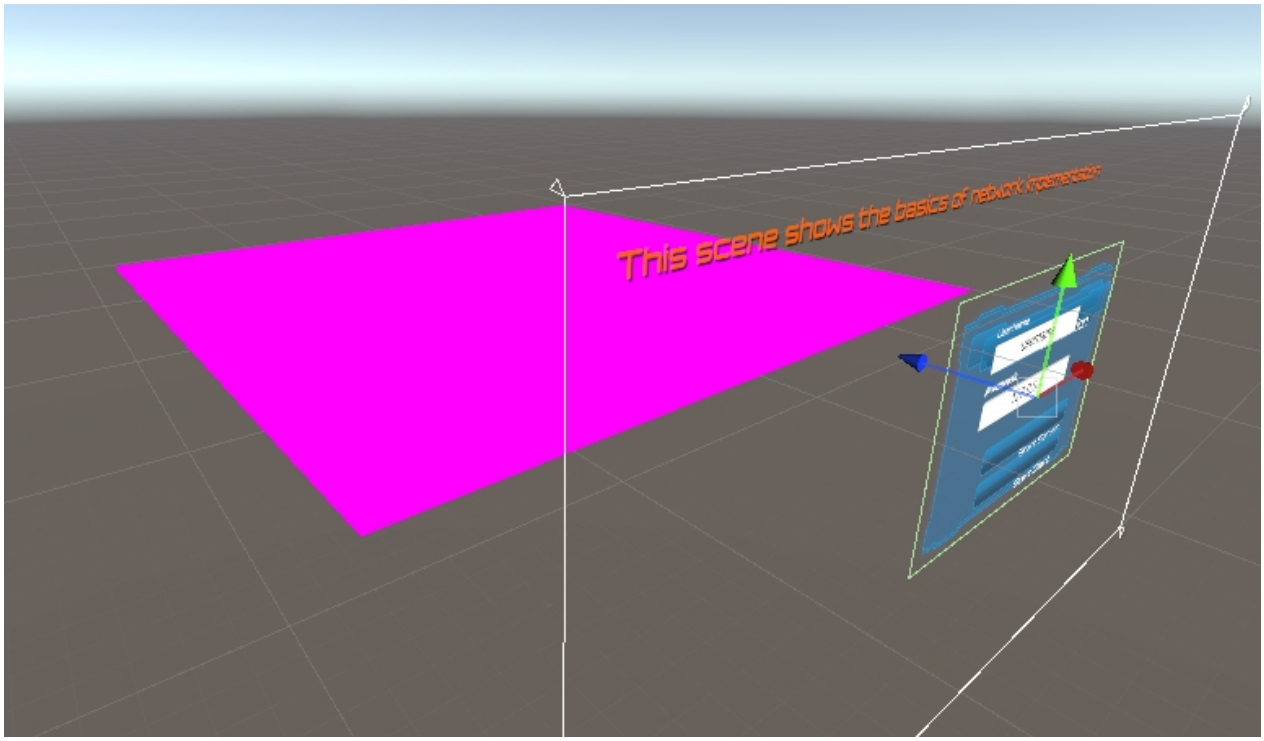
ObjectNet provides a bunch of examples of scenes to be used as guidelines.

You can use those scenes only as guidelines or as a multiplayer project's starting point.

Each scene was created to explain one aspect of **ObjectNet**, nonetheless, some scenes can cover more than one feature.



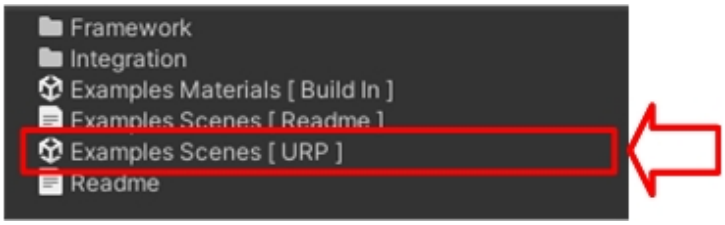
⚠ **Note:** If examples scenes appear in punk color you need to convert your materials (see).

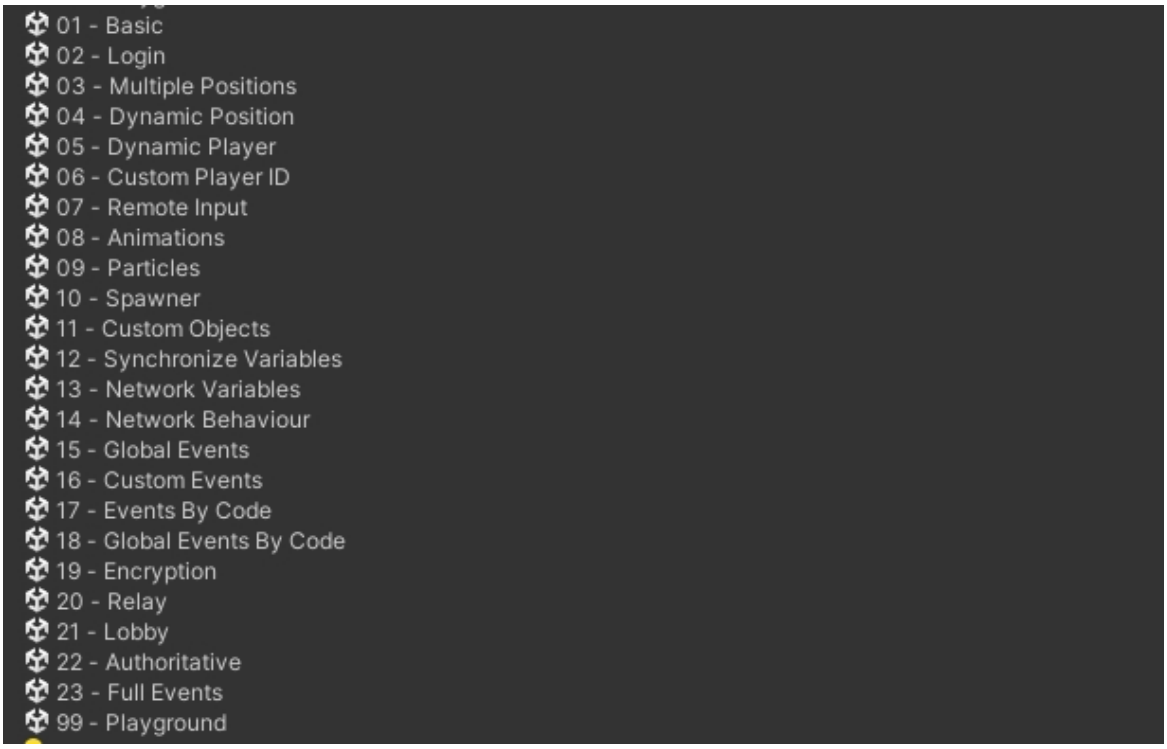


Importing Examples

ObjectNet provides many example scenes, importing "**Examples Scenes [URP]**" extra package will

import all 24 ObjectNet examples to be used as reference guide.



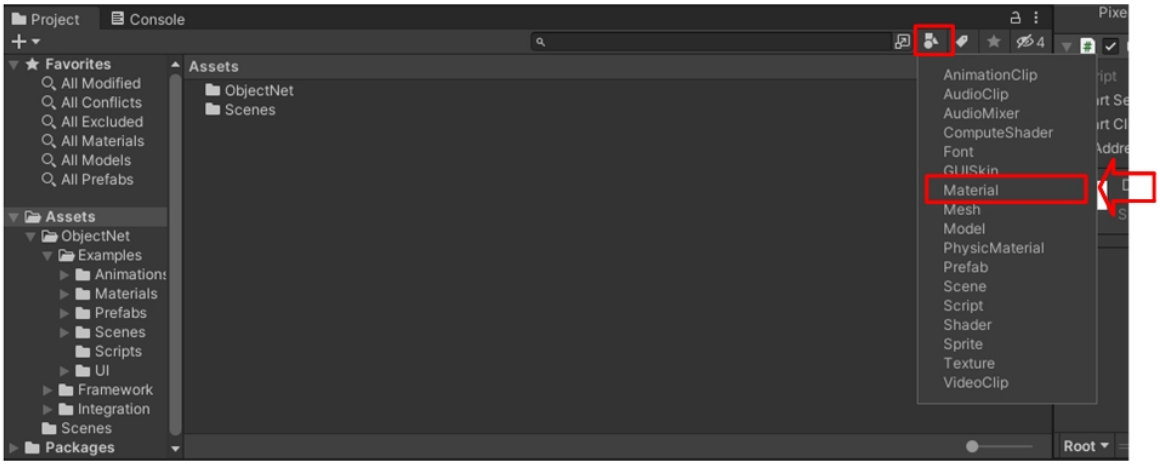


Converting Materials

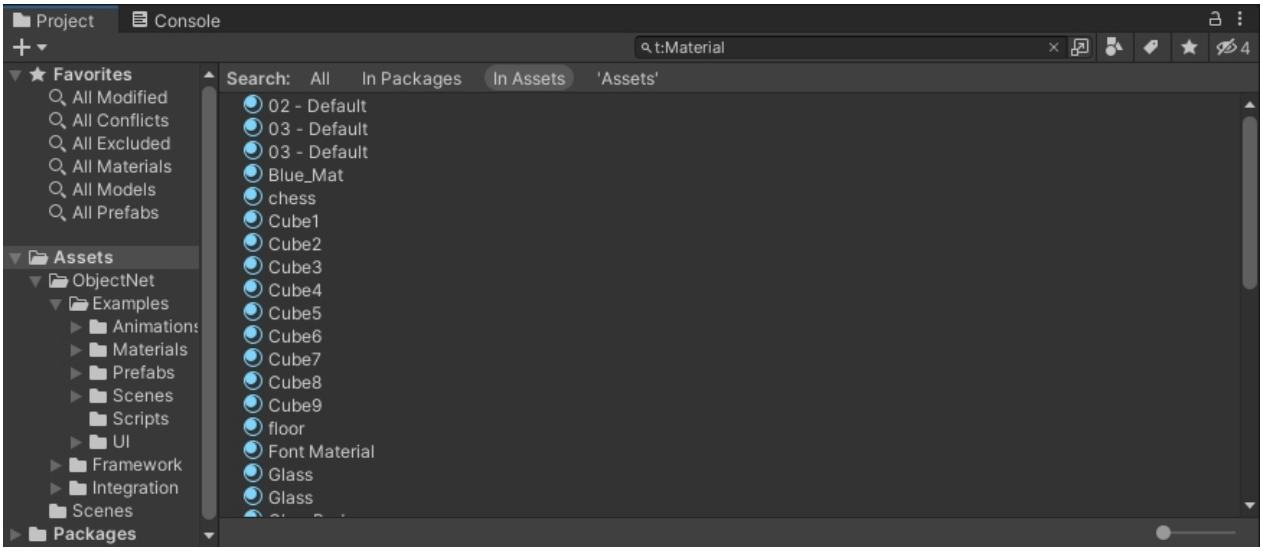
ObjectNet provides many example scenes using URP as the default Render Pipeline, nonetheless, you may be using another pipeline, this section explains how to change materials to be correctly rendered.

The following steps explain how to convert materials manually, to see how to do this automatically check the end of this section.

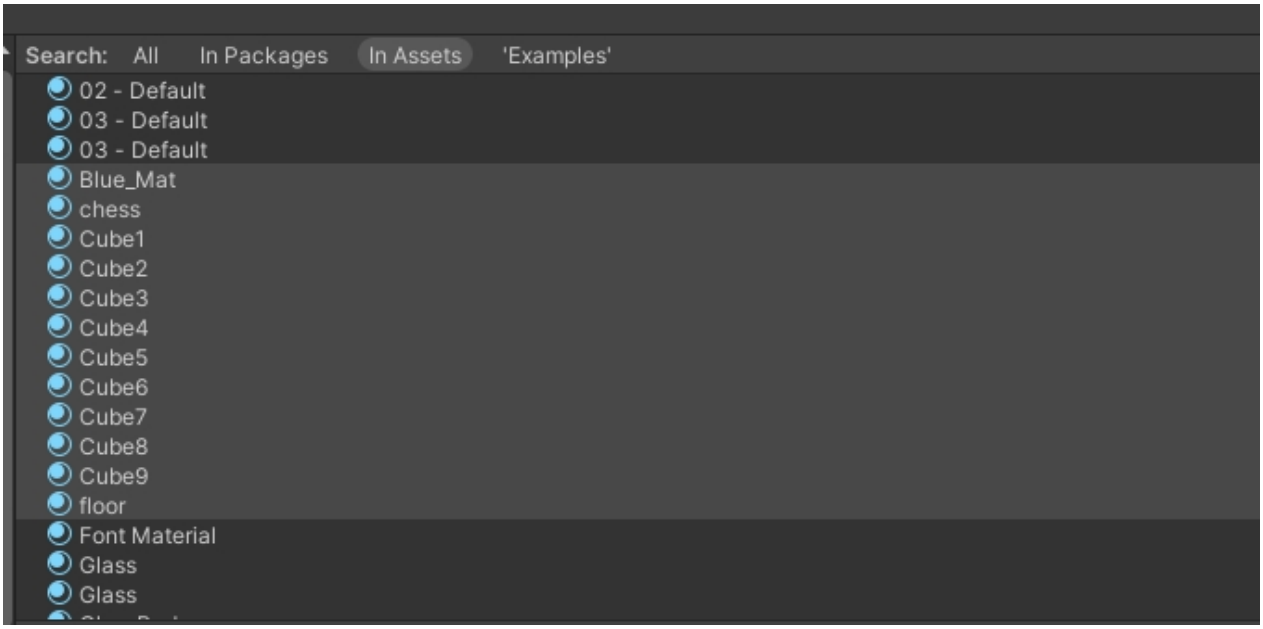
1. On project explorer click on "Search by type" option and select Material option.



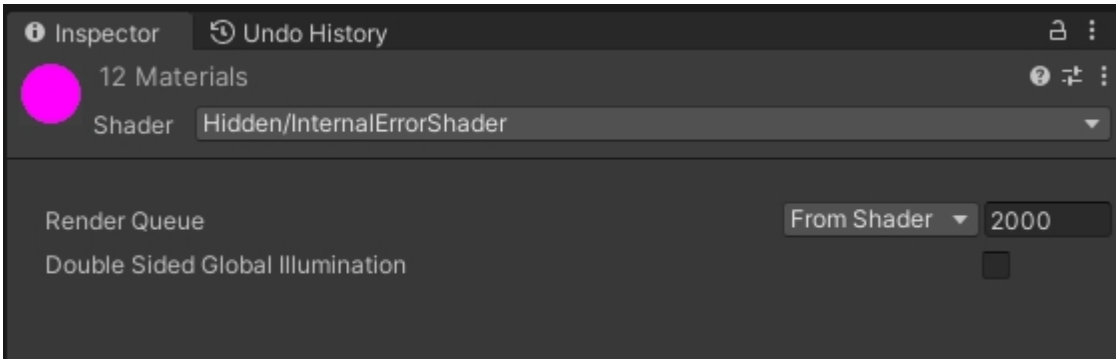
2. This shall list all materials on your project



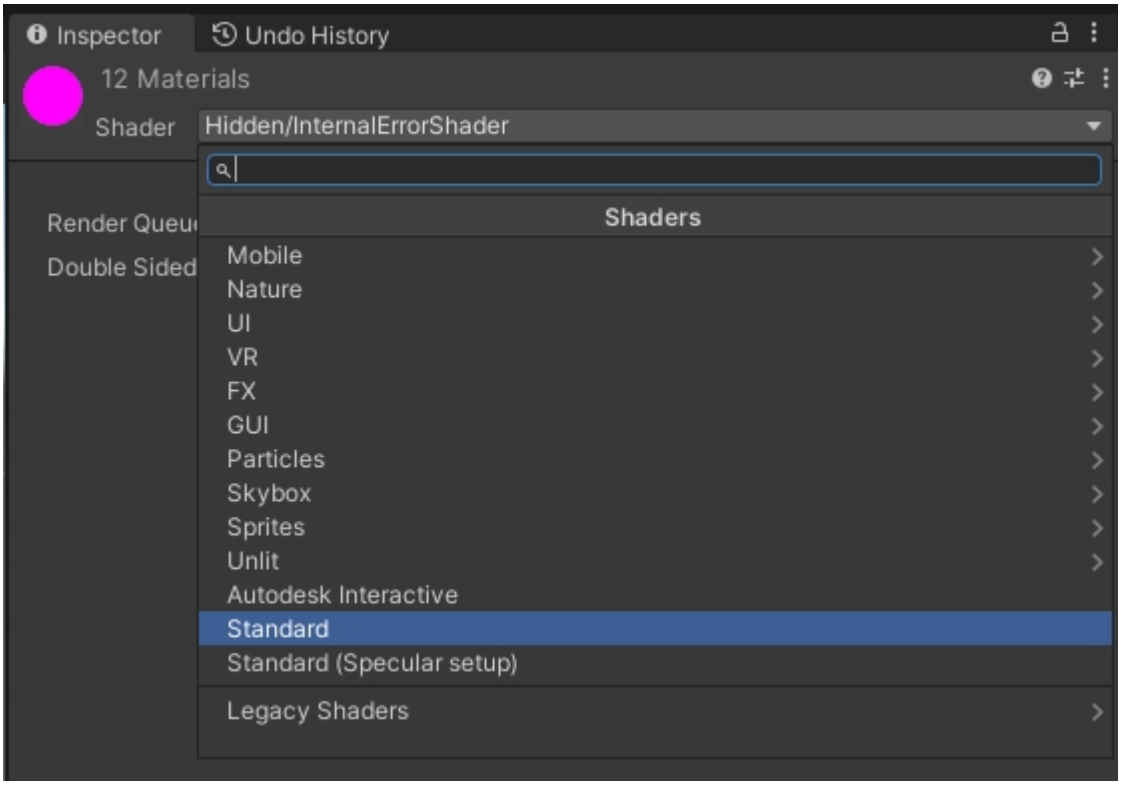
3. Select materials on the Examples folder of ObjectNet.



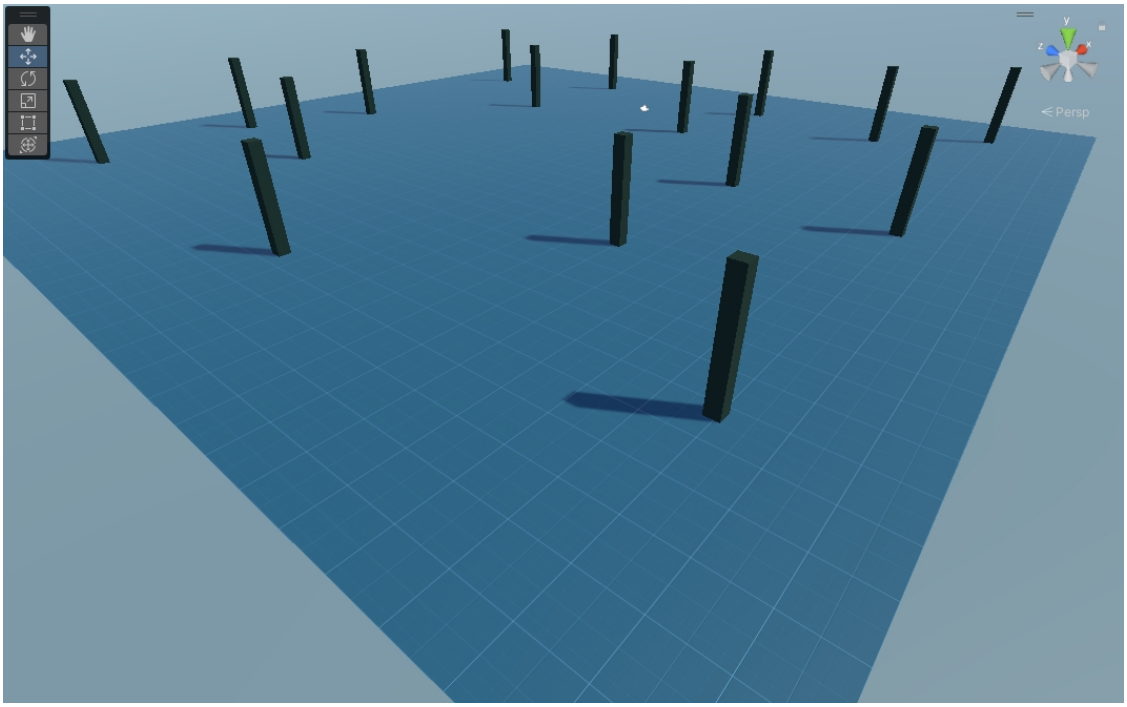
4. On the inspector an error must be displayed showing InternalErrorShader



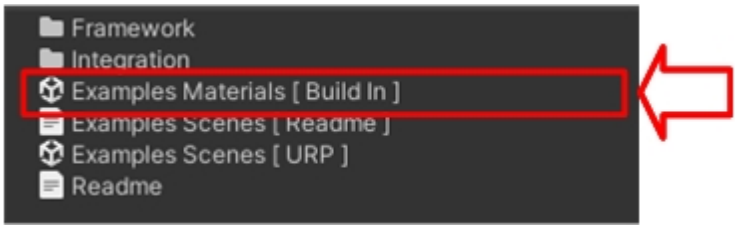
5. Click on combo box and select Standard shader option



6. Do this for all materials with same error and all scenes shall be correctly rendered



ObjectsNet *provide an extra package with all materials into build in format, importing this package will convert all materials used by object net examples into build in format.*

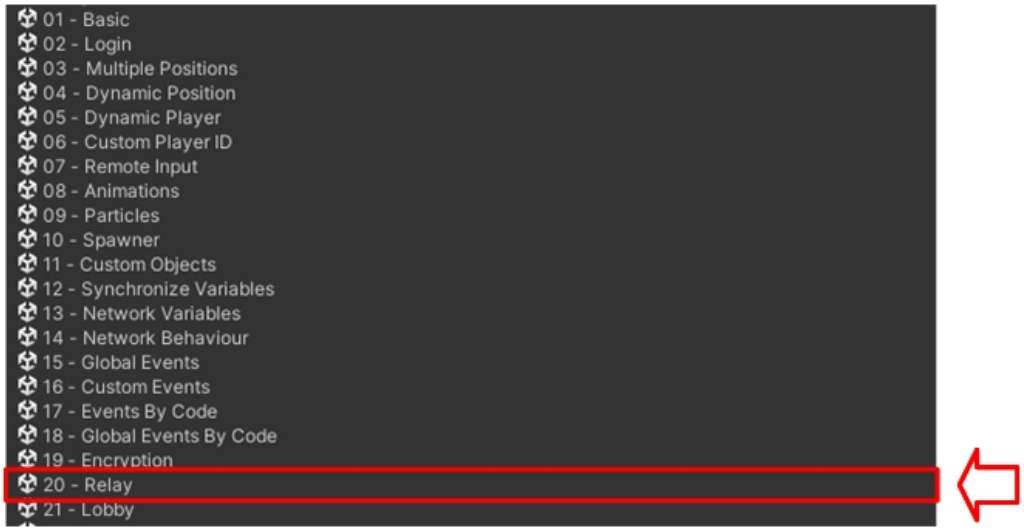


Relay Example

ObjectNet allows to use game architecture on Relay mode (see Relay Mode).

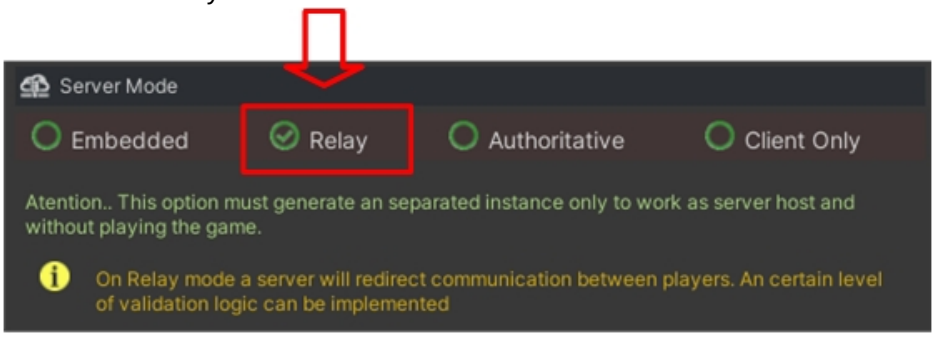
Building Server

Open scene "20 - Relay" under examples scene folder.

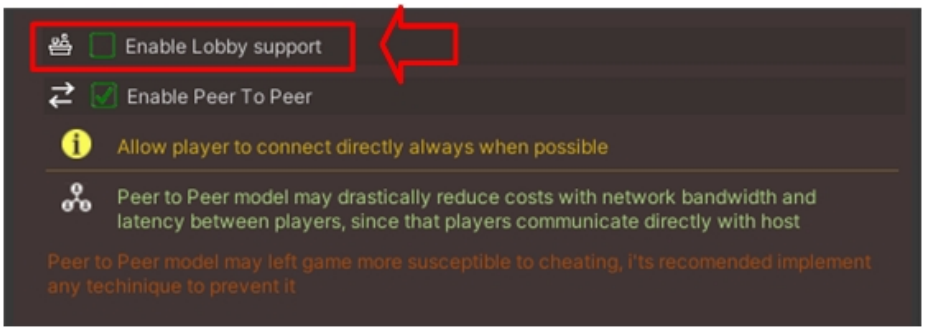


Ensure that following option if correctly, if not, change it.

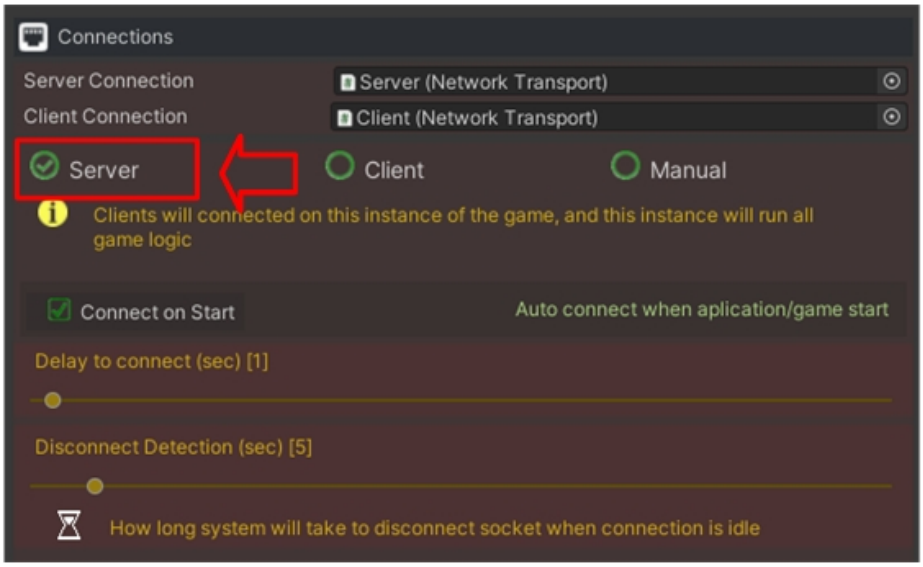
1. Server mode is in Relay mode.



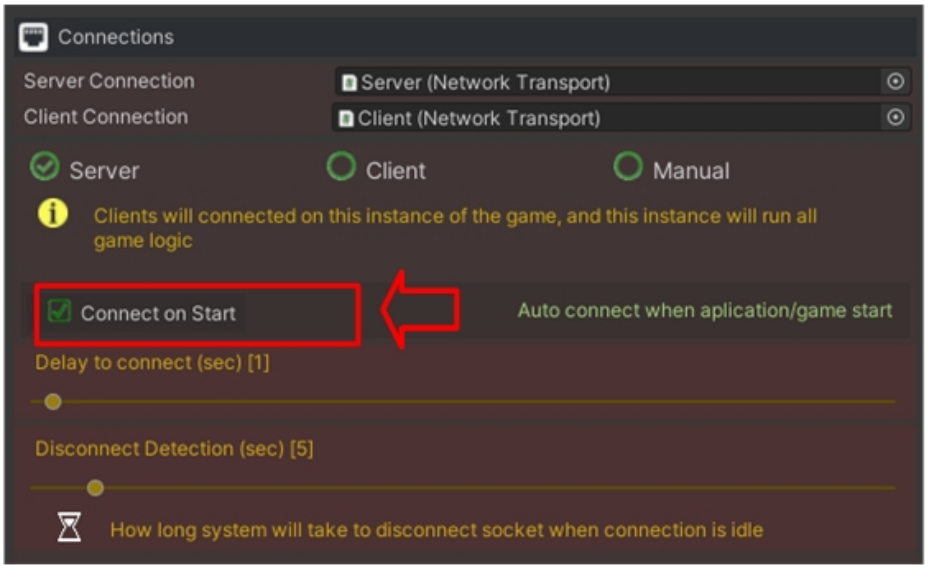
2. "Enable Lobby support" in unchecked



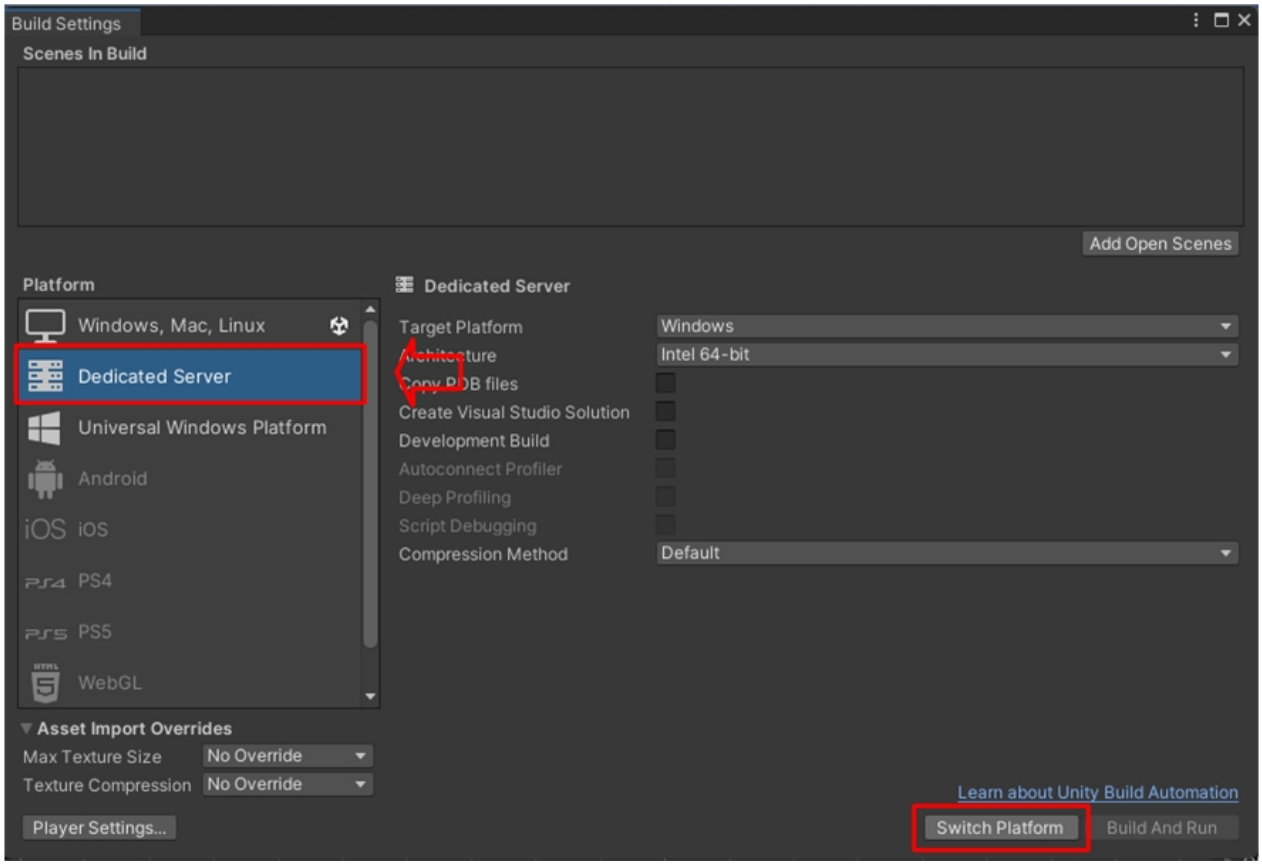
3. Server connection type shall be selected



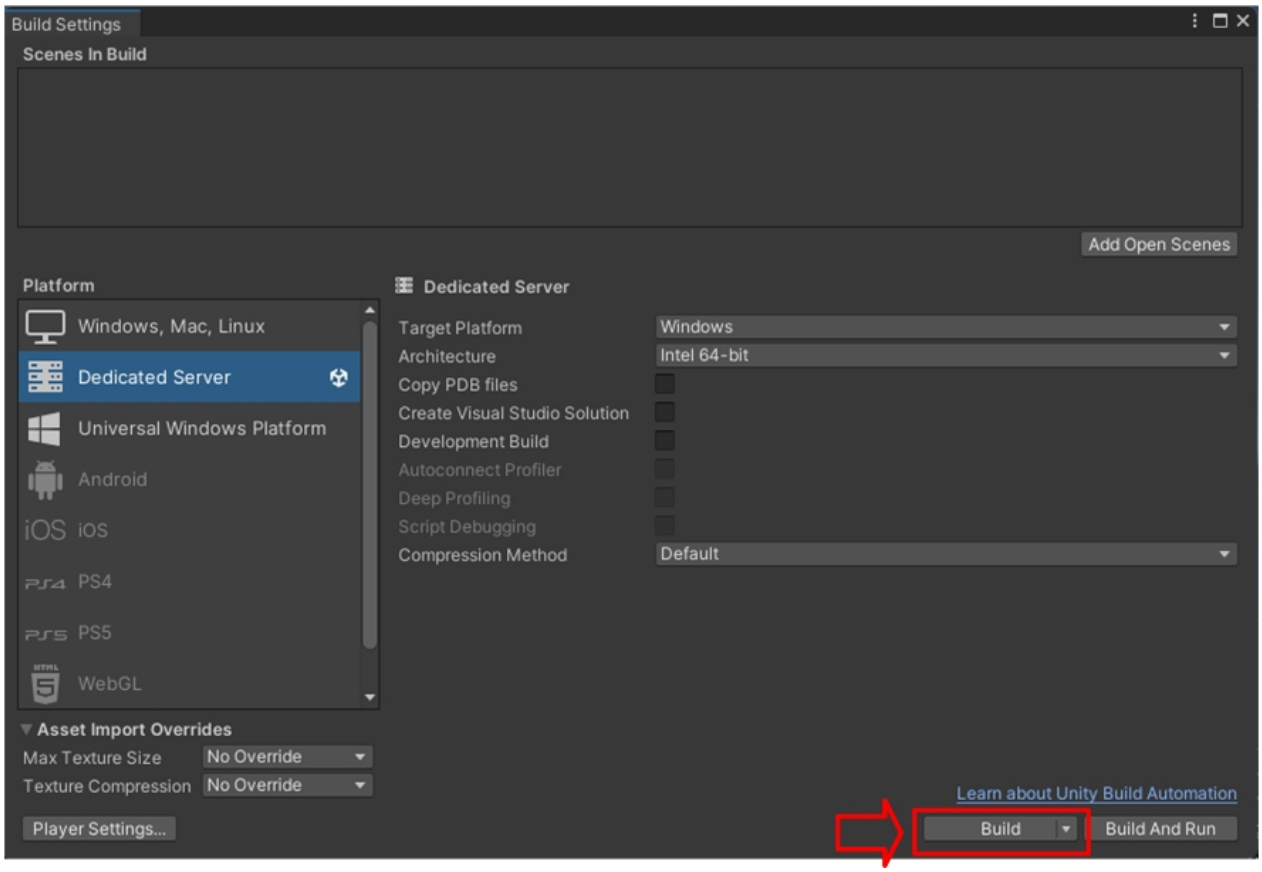
4. "Connect on Start" is checked (This will ensure that server will start after launched)



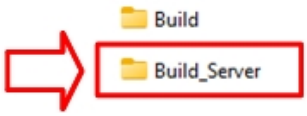
5. Now you need to switch you project to "Dedicated Server Mode" (see Unity Dedicated Server)



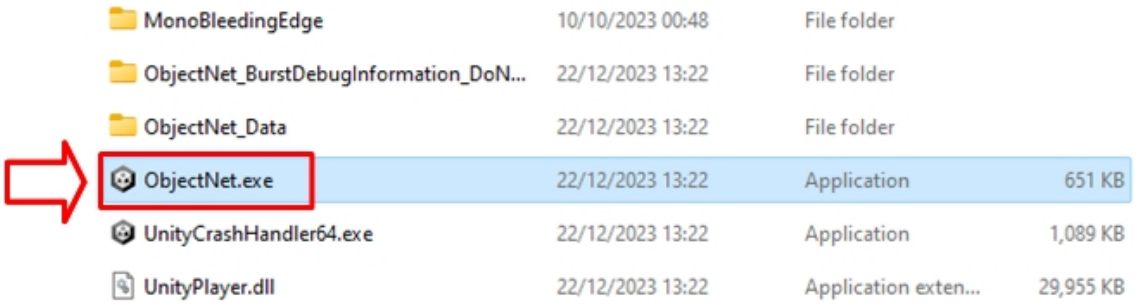
6. After finished a "Build" option shall appear



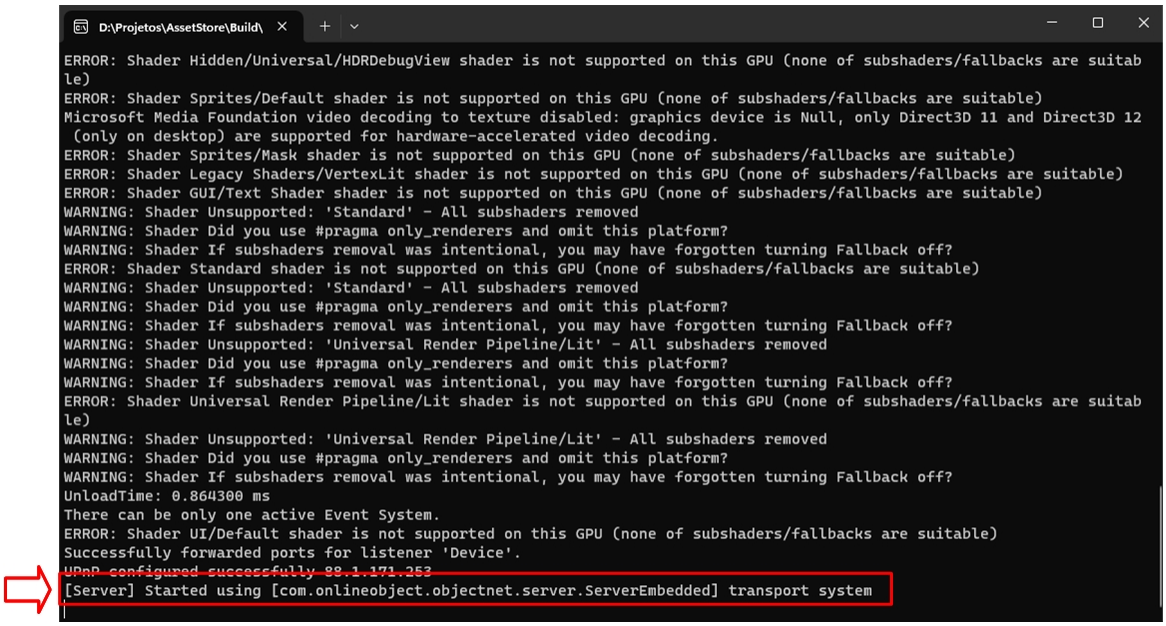
7. Build server mode into a separated folder (It's important separate Game build from Server build)



8. No run the server build



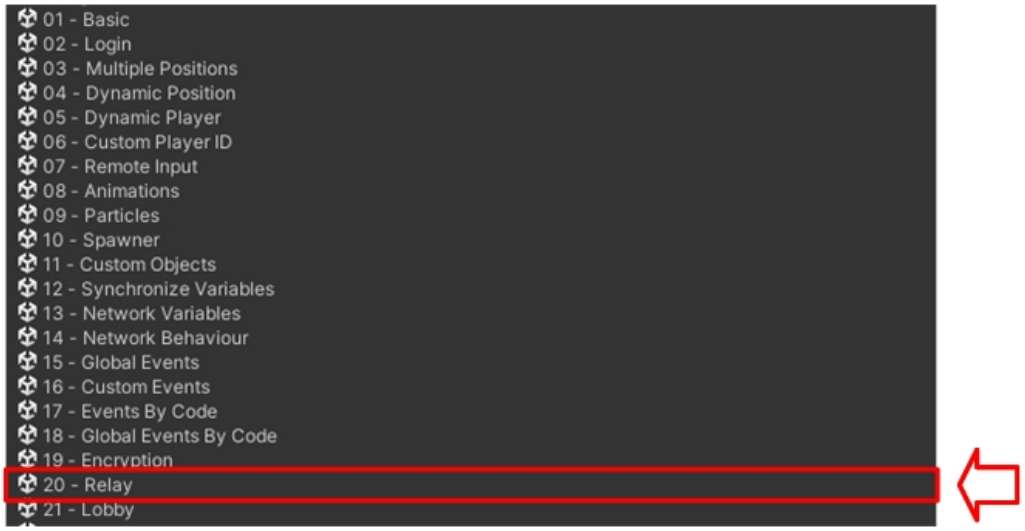
9. Server should be run without any problem



The log shall show the server up and running and the current user transport system.

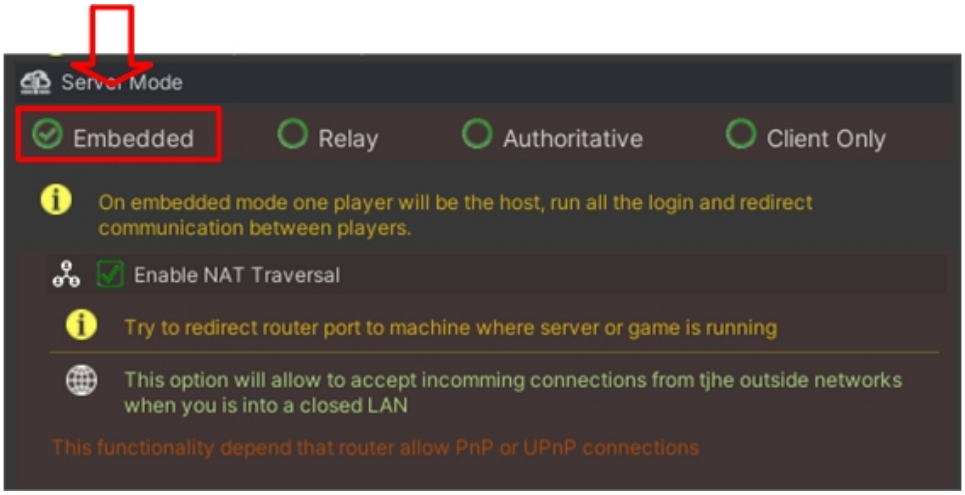
Building Client

Open scene "20 - Relay" under examples scene folder.

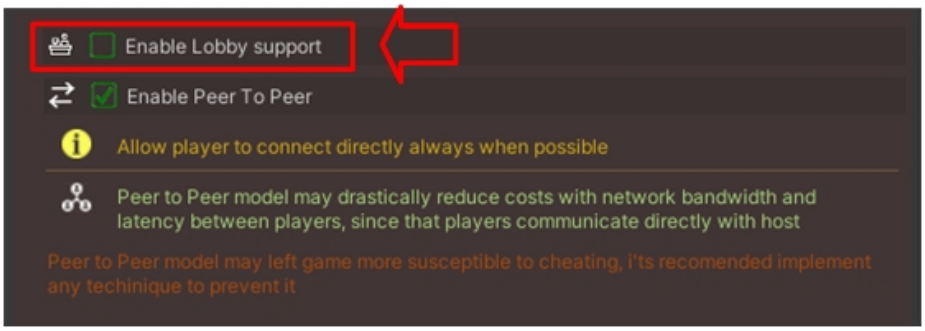


Ensure that following option if correctly, if not, change it.

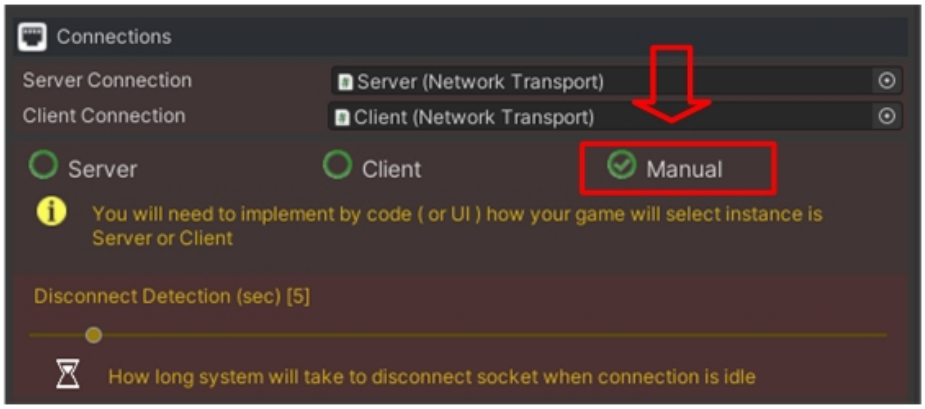
1. "Embedded" or "Client Only" must be selected. On this example we will select "Embedded".



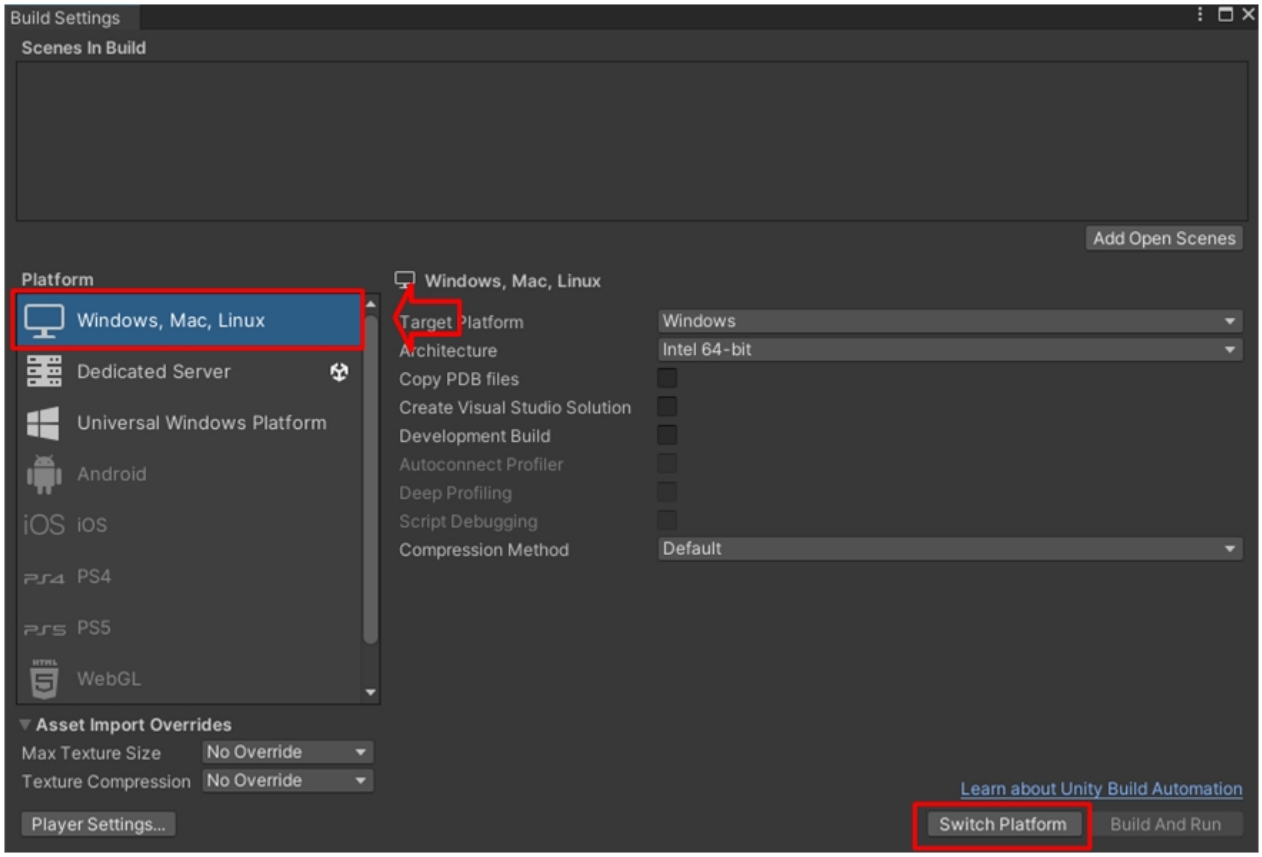
2. "Enable Lobby support" in unchecked



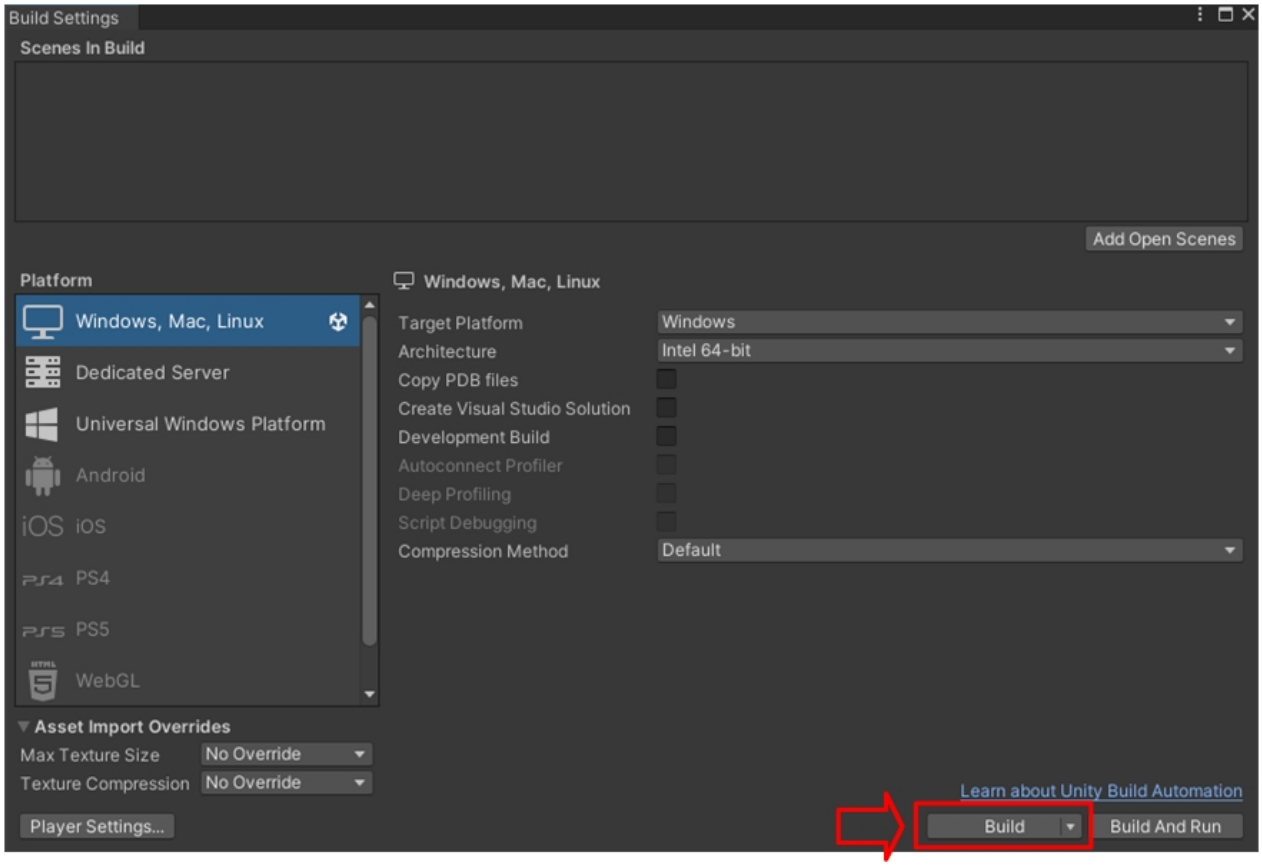
3. "Client" or "Manual" connection type shall be flagged



4. Now you need to switch you project to the selected platform target, on this case we going to use "Windows, Mac, Linux"

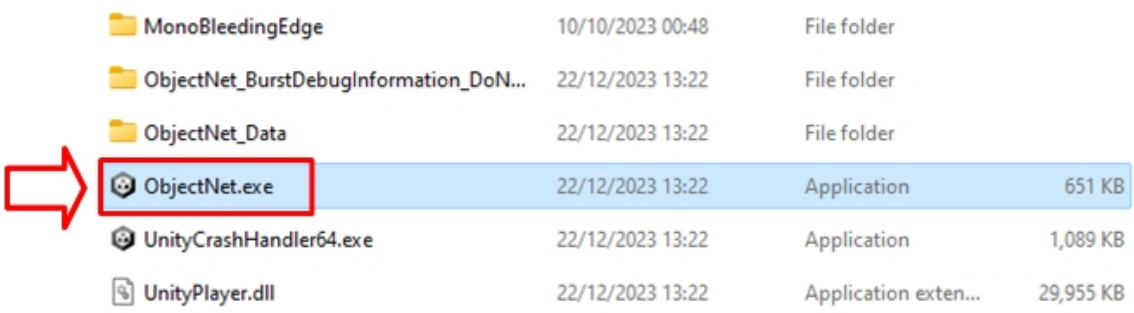


5. After finished a "Build" option shall appear



6. Build server mode into a separated folder (It's important separate Game build from Server build)

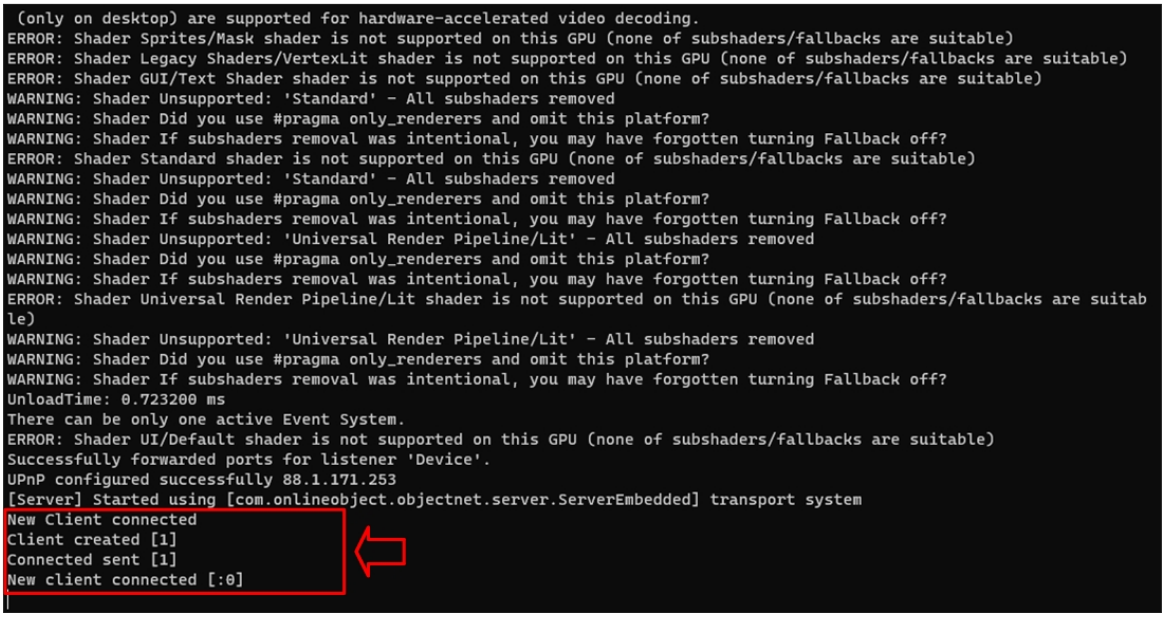
7. No run the compiled game



8. The game shall start

9. Player must be spawned on scene

10. Player will be connected at Server

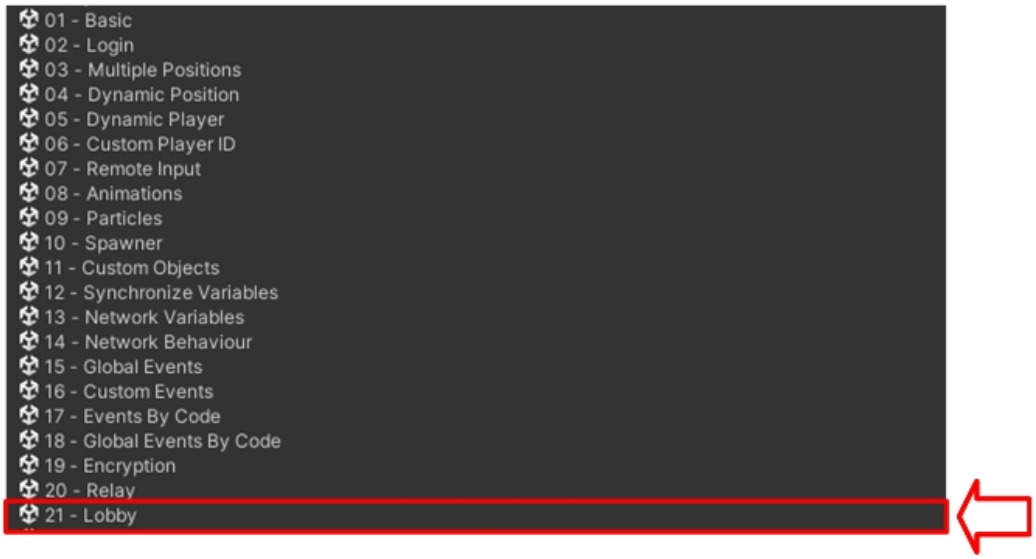


Lobby Example

ObjectNet allows to use game architecture on Relay mode allowing player to create and enter on Lobbies (see Lobby System).

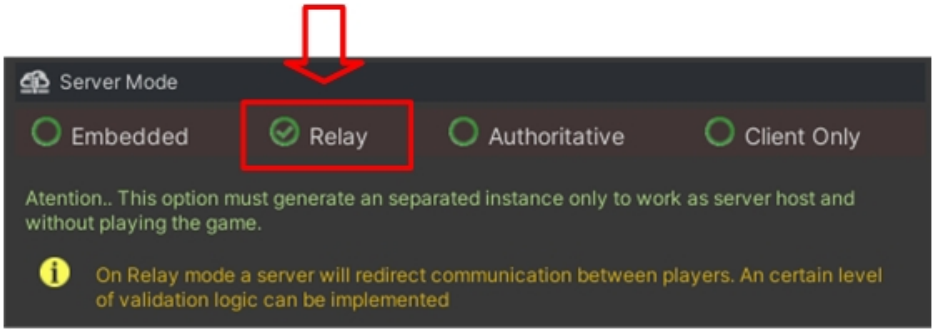
Building Server

Open scene "21 - Lobby" under examples scene folder.

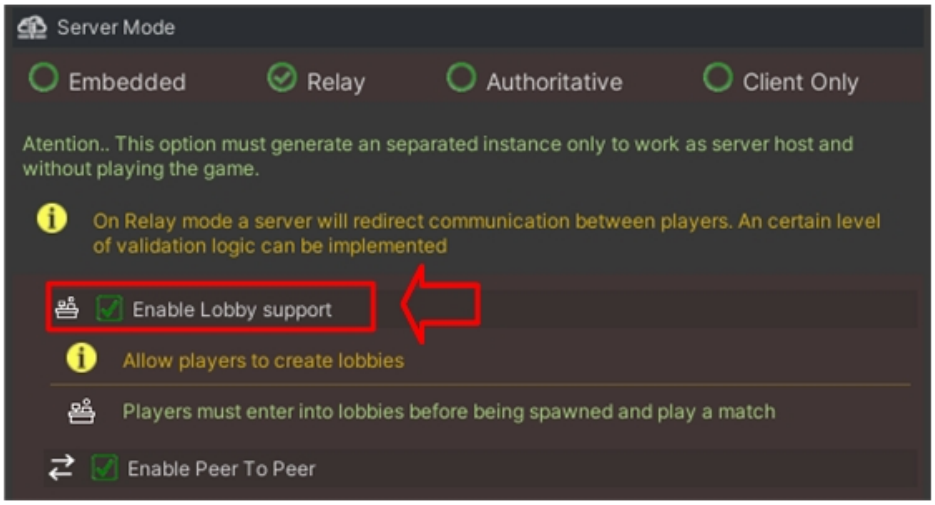


Ensure that following option if correctly, if not, change it.

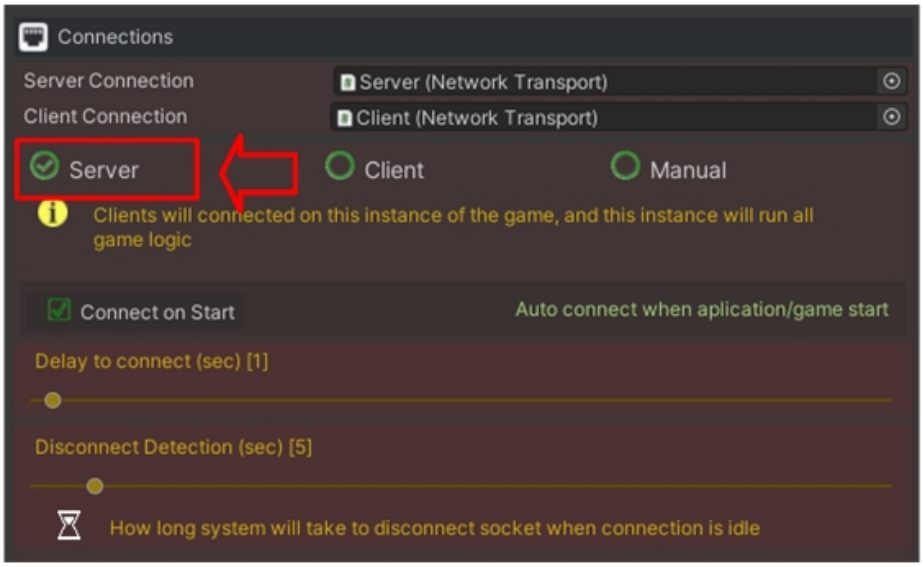
1. Server mode is in Relay mode.



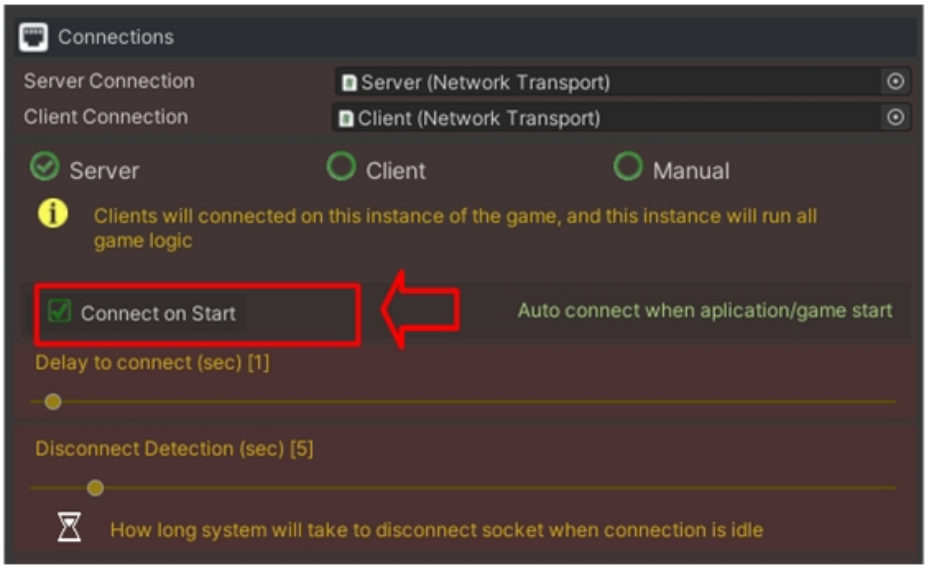
2. "Enable Lobby support" in checked



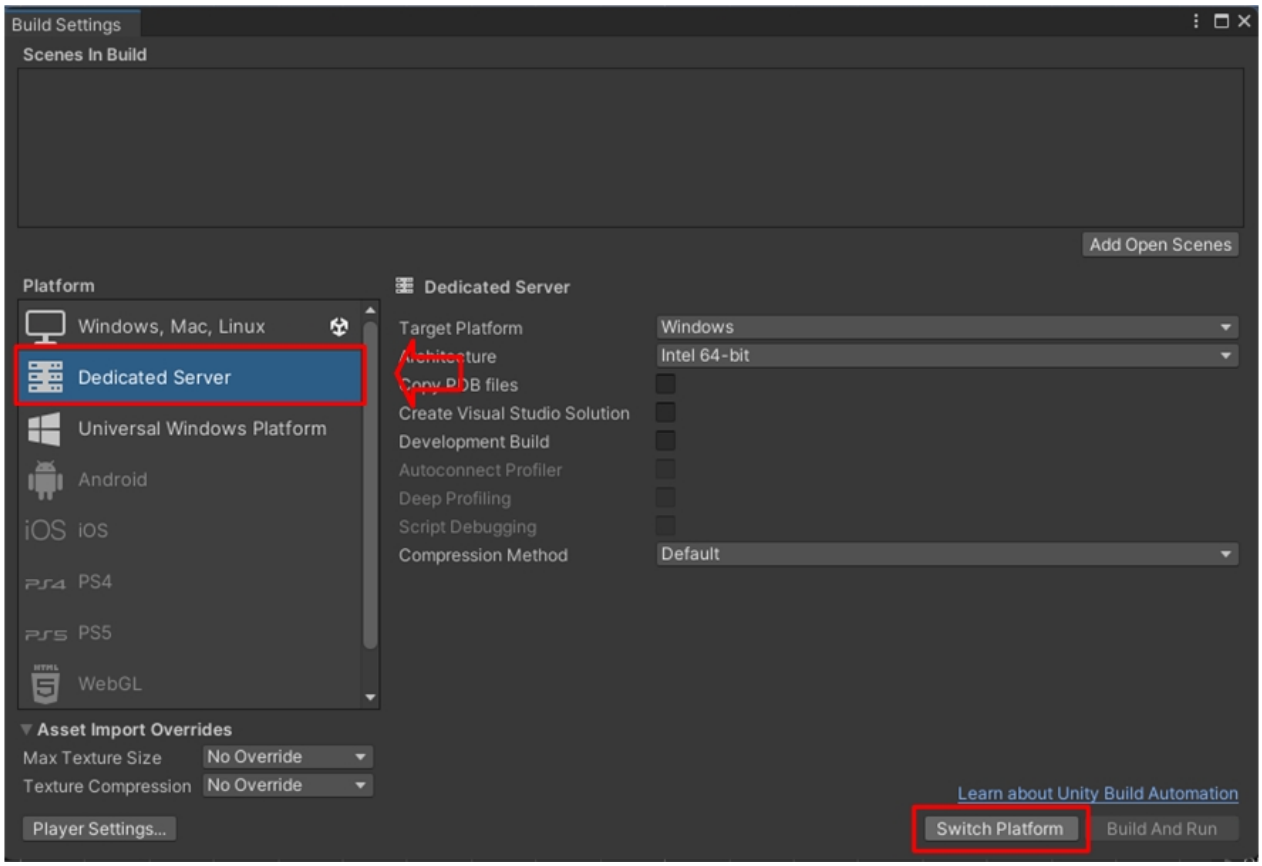
3. Server connection type shall be selected



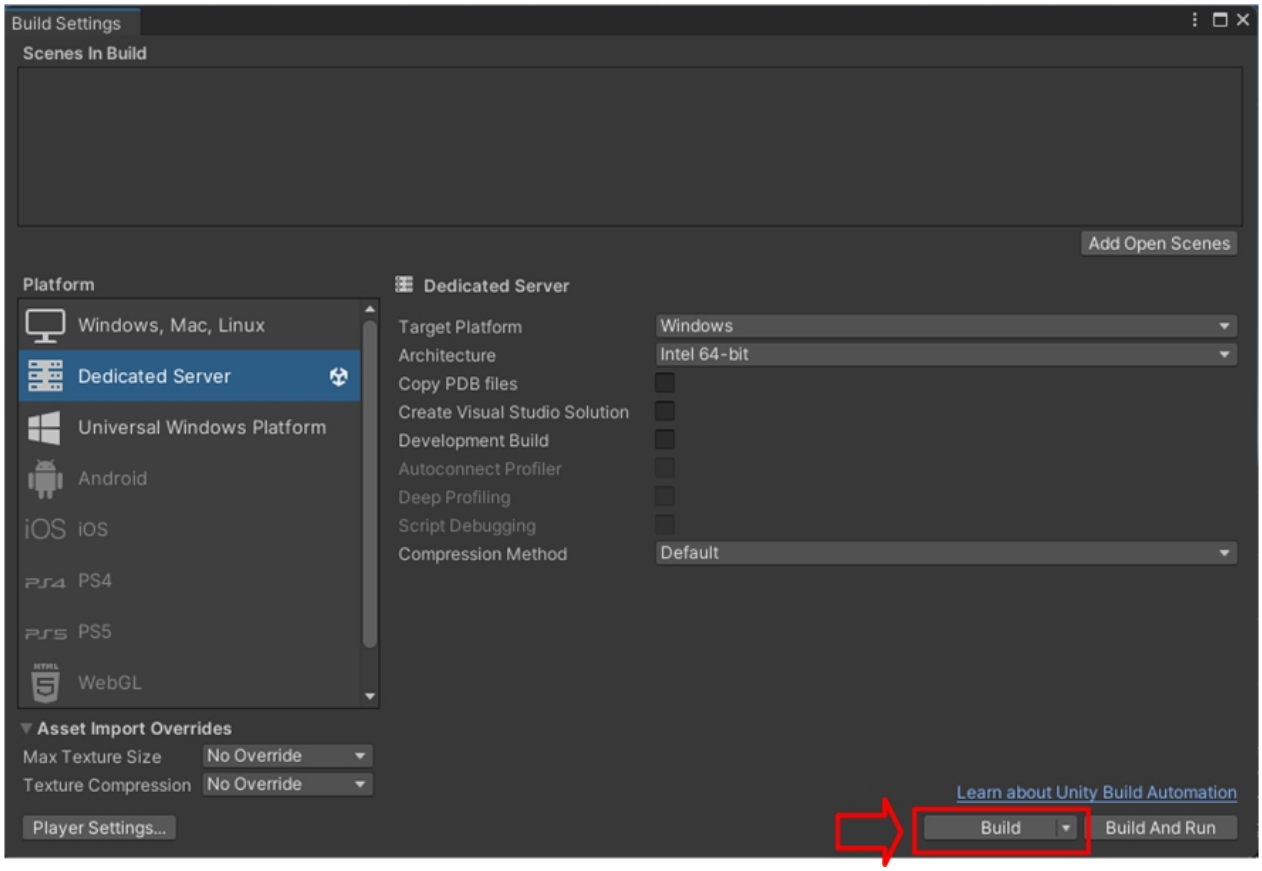
4. "Connect on Start" is checked (This will ensure that server will start after launched)



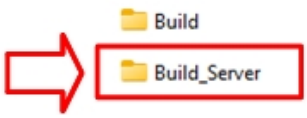
5. Now you need to switch you project to "Dedicated Server Mode" (see [Unity Dedicated Server]())



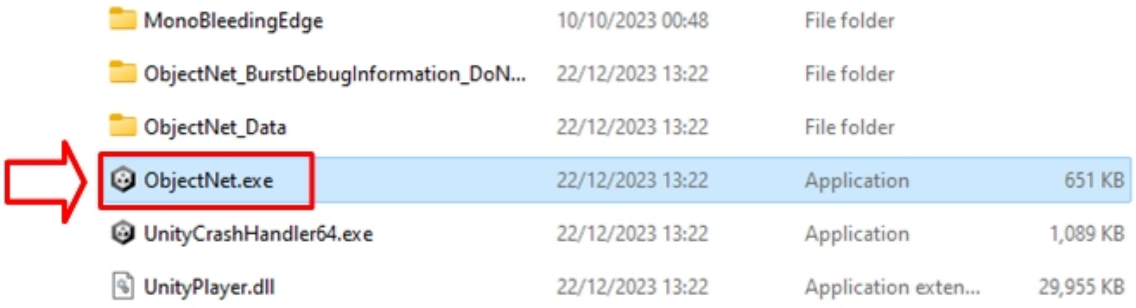
6. After finished a "Build" option shall appear



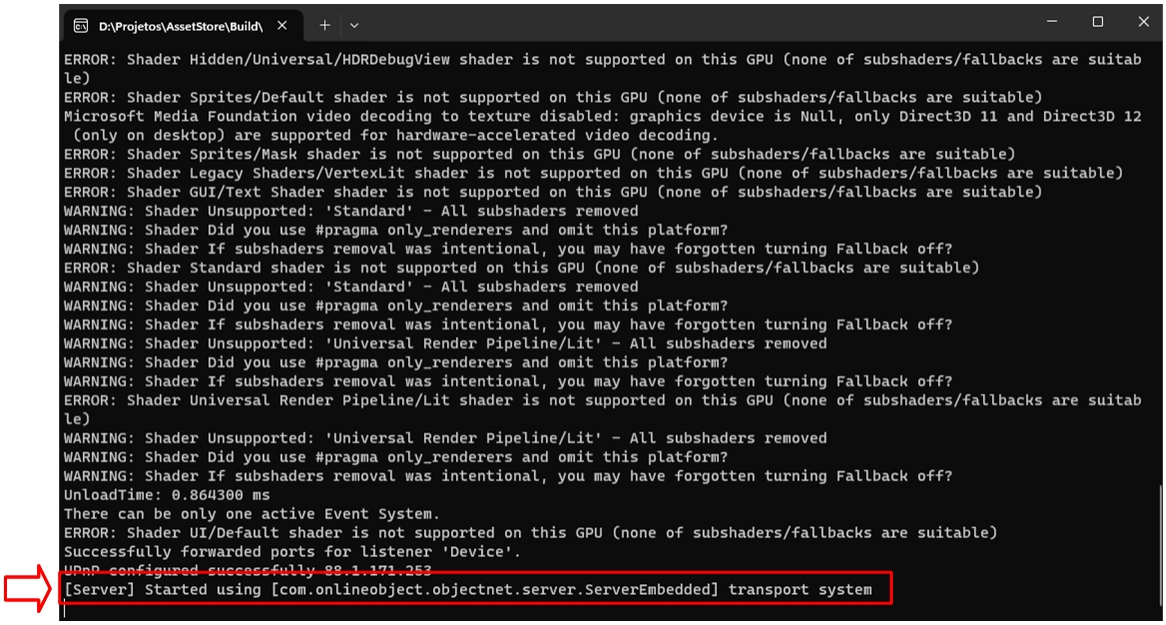
7. Build server mode into a separated folder (It's important separate Game build from Server build)



8. No run the server build



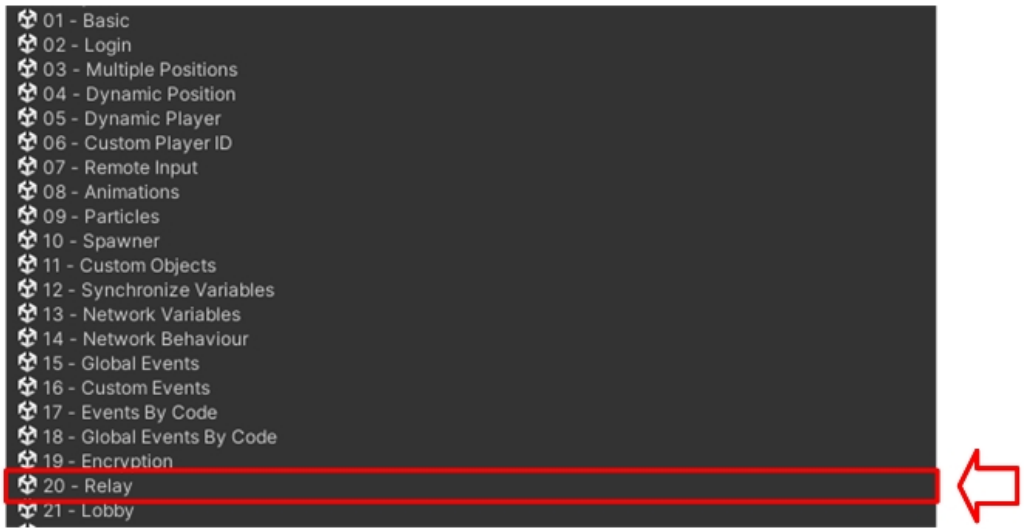
9. Server should be run without any problem



The log shall show the server up and running and the current user transport system.

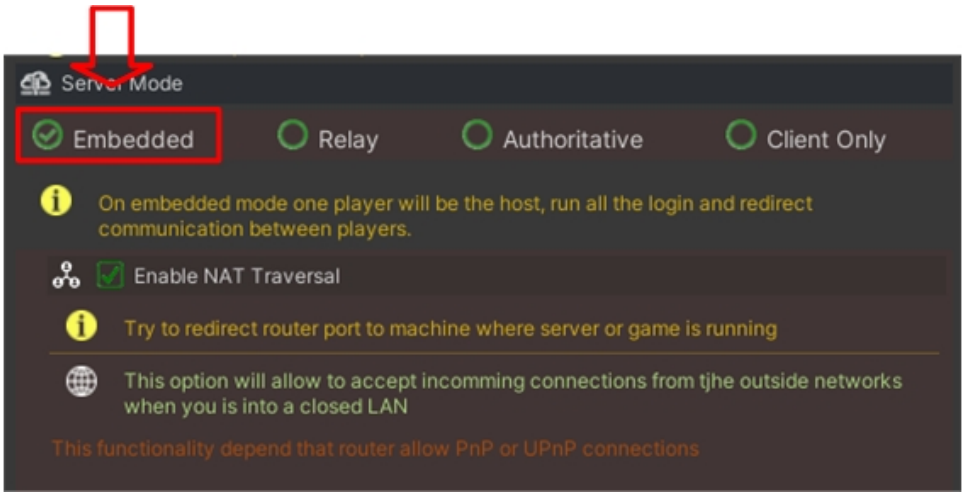
Building Client

Open scene "21 - Lobby" under examples scene folder.

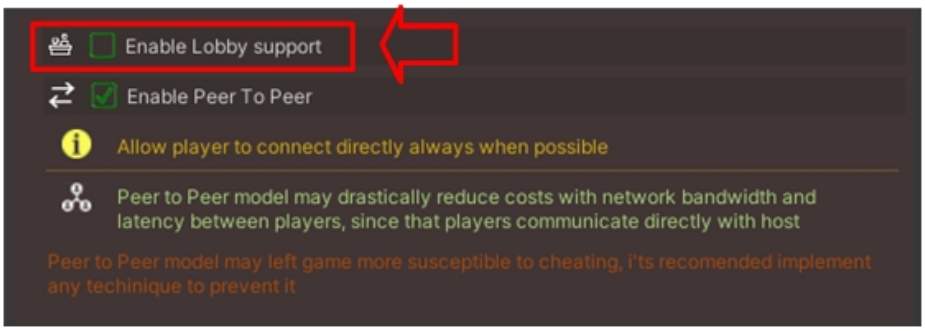


Ensure that following option if correctly, if not, change it.

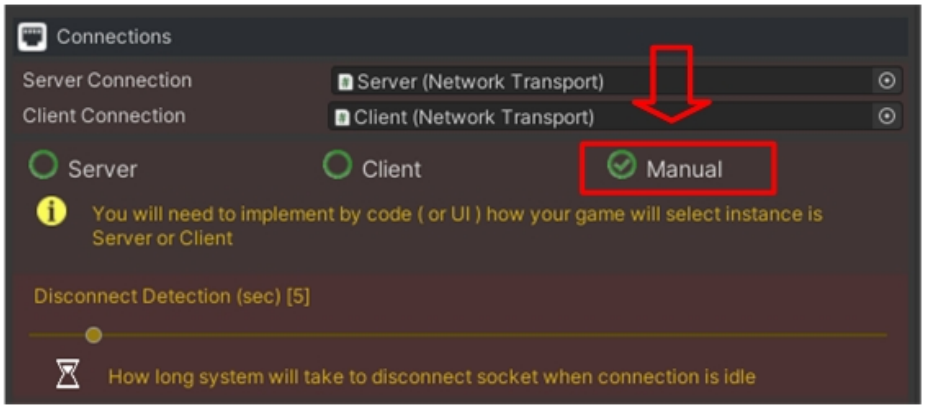
1. "Embedded" or "Client Only" must be selected. On this example we will select "Embedded".



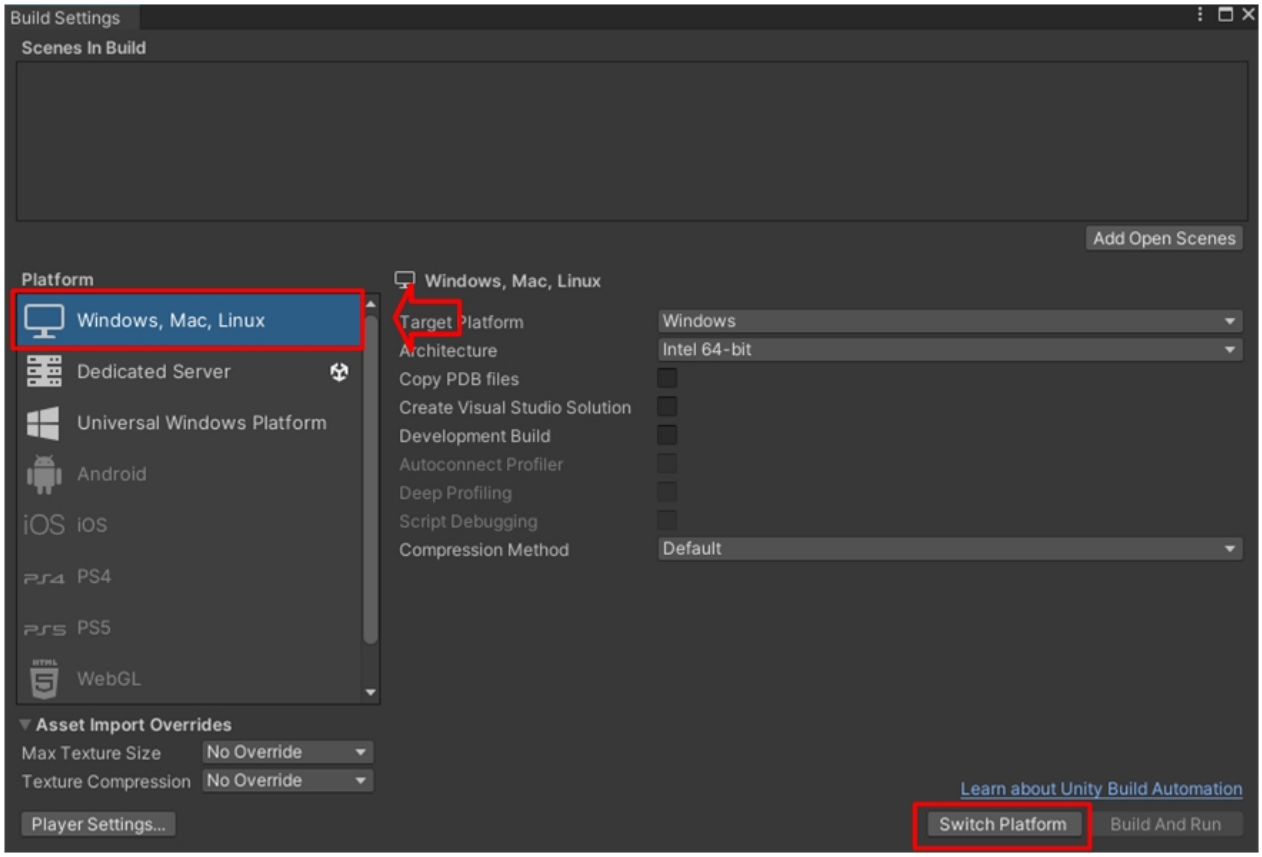
2. "Enable Lobby support" in unchecked



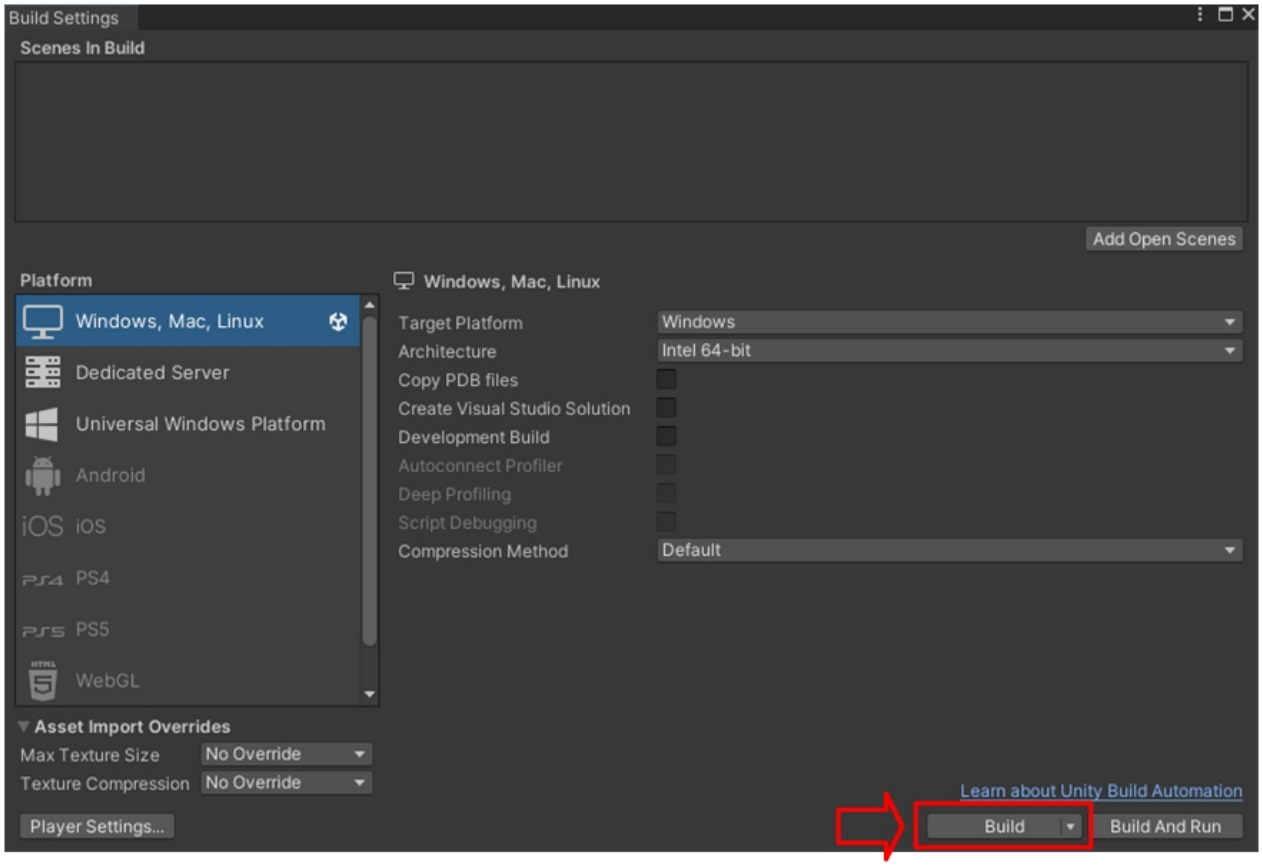
3. "Client" or "Manual" connection type shall be flagged



4. Now you need to switch you project to the selected platform target, on this case we going to use "Windows, Mac, Linux"

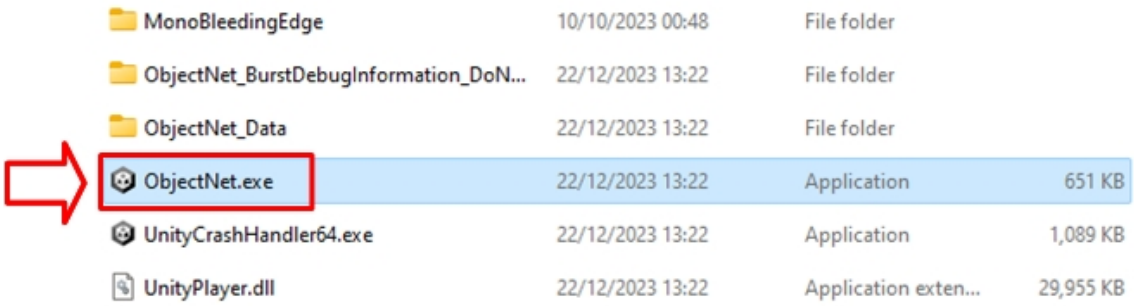


5. After finished a "Build" option shall appear



6. Build server mode into a separated folder (It's important separate Game build from Server build)

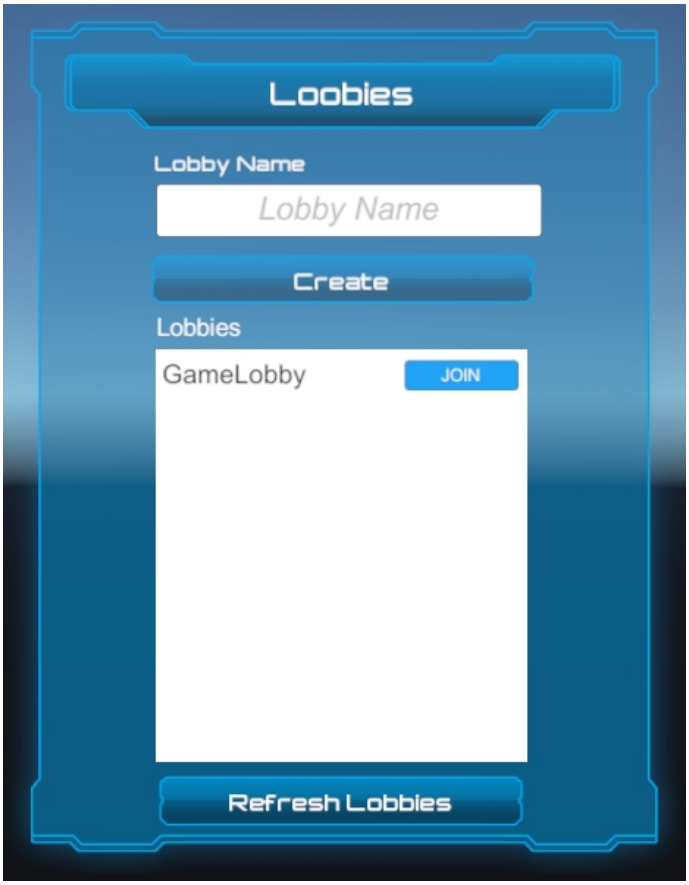
7. No run the compiled game



8. The game shall start

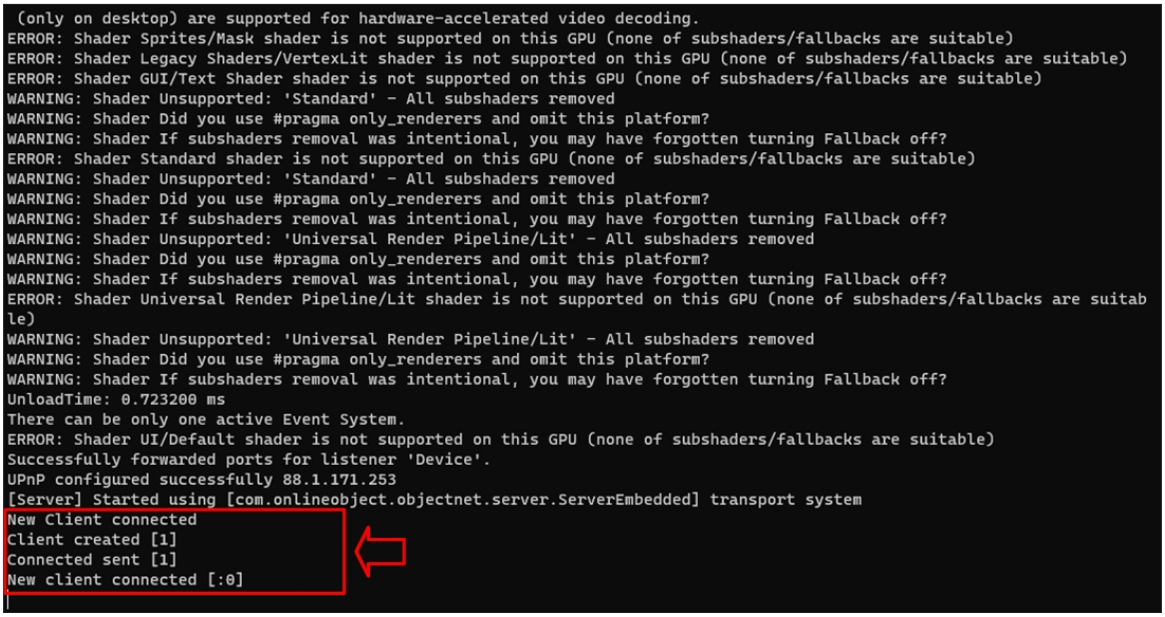
9. A window shall appear to allow to create or enter into a lobby

10. If you wish to list current lobbies, you need to click on "Refresh Lobbies" and then click into join on the selected lobby



11. Player must be spawned on scene

12. Player will be connected at Server

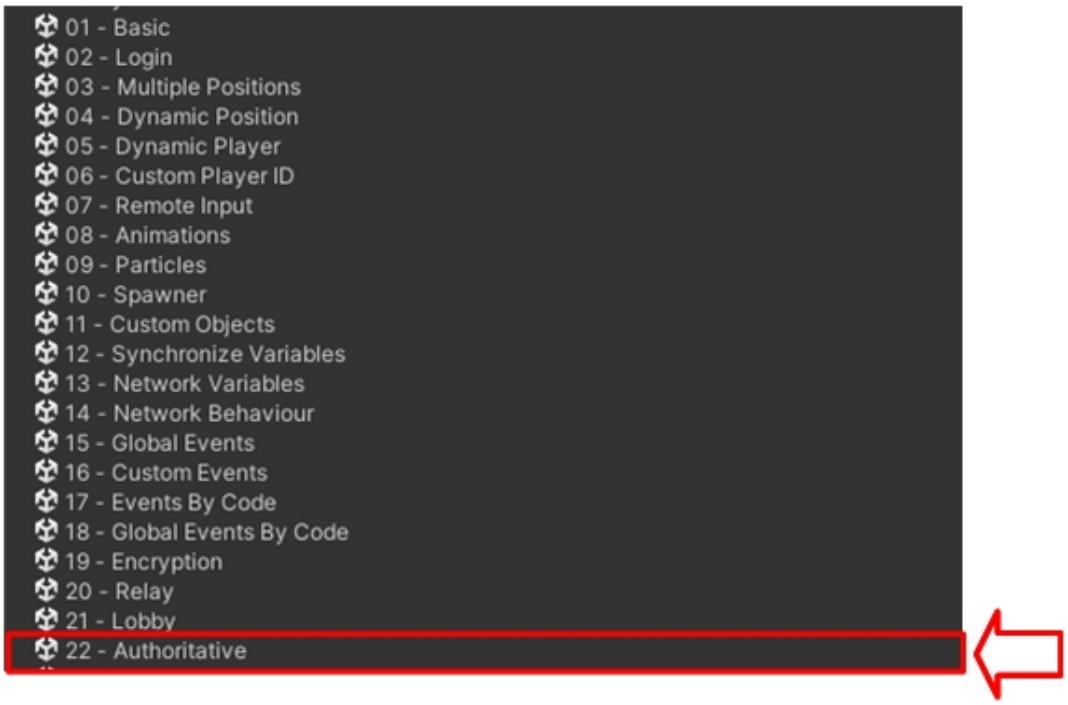


Authoritative Example

ObjectNet allows to use game architecture on Authoritative mode (see Authoritative).

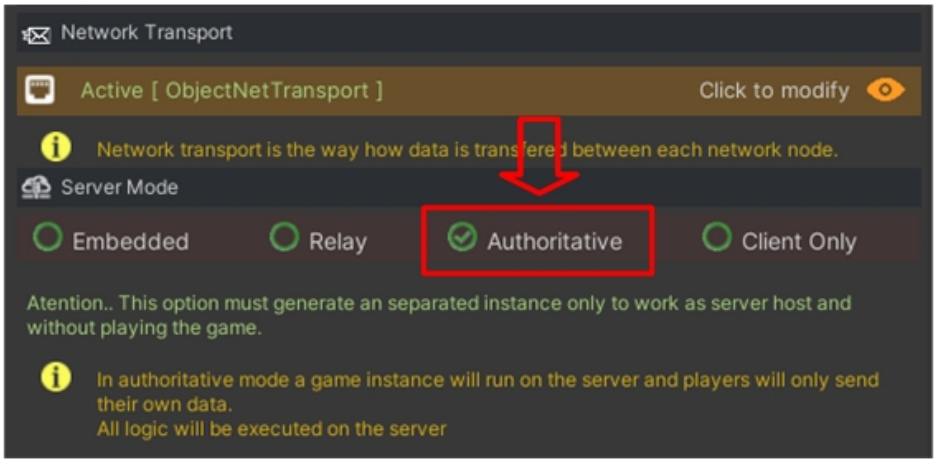
Building Server

Open scene "22 - Authoritative" under examples scene folder.

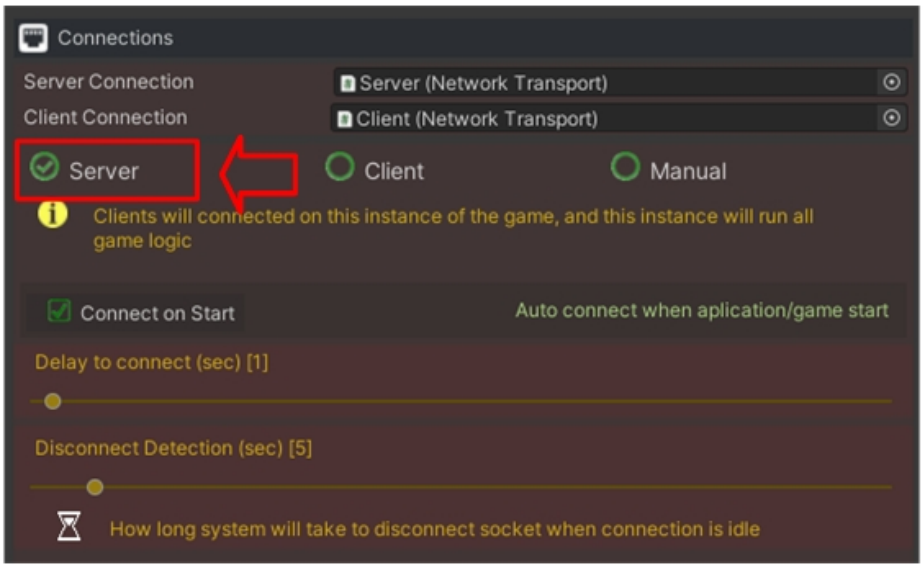


Ensure that following option if correctly, if not, change it.

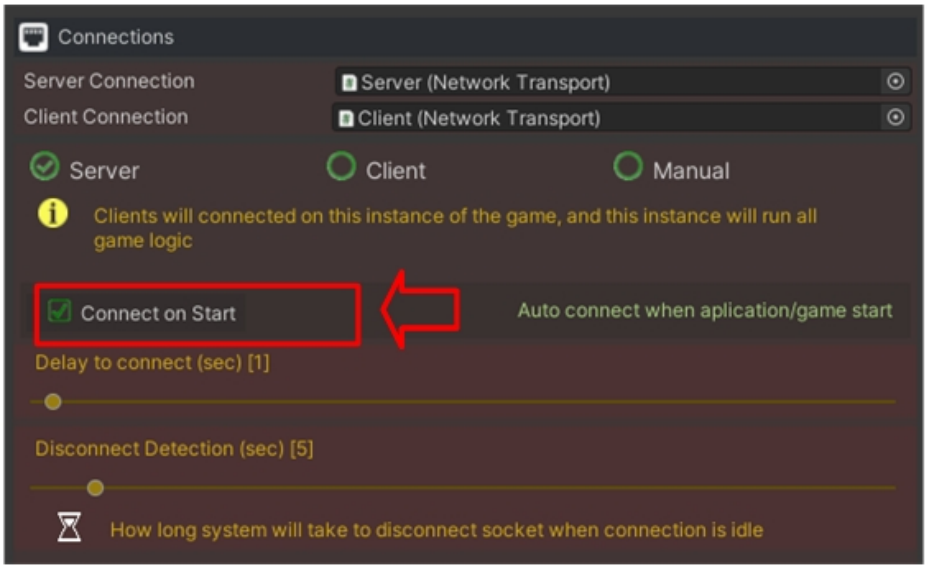
1. Server mode is in Authoritative mode.



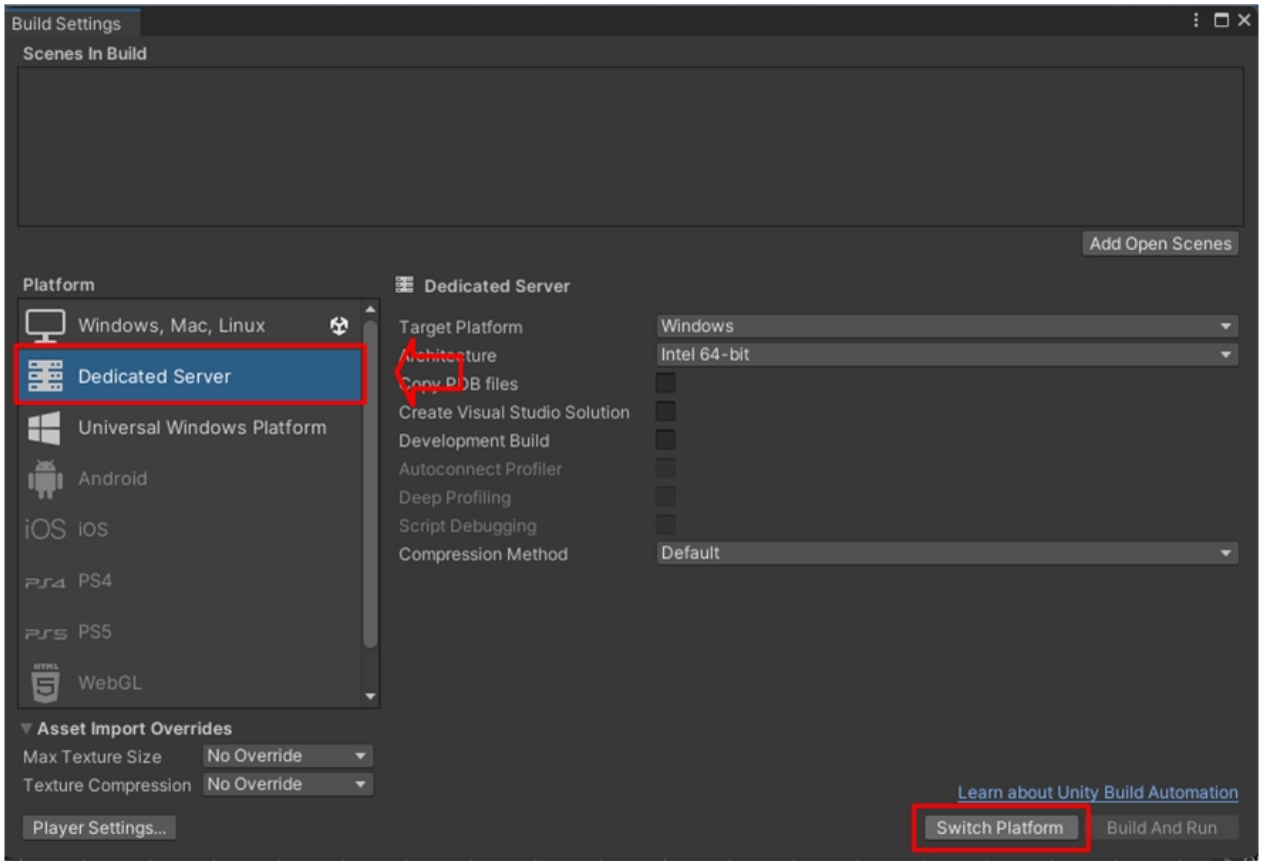
2. Server connection type shall be selected



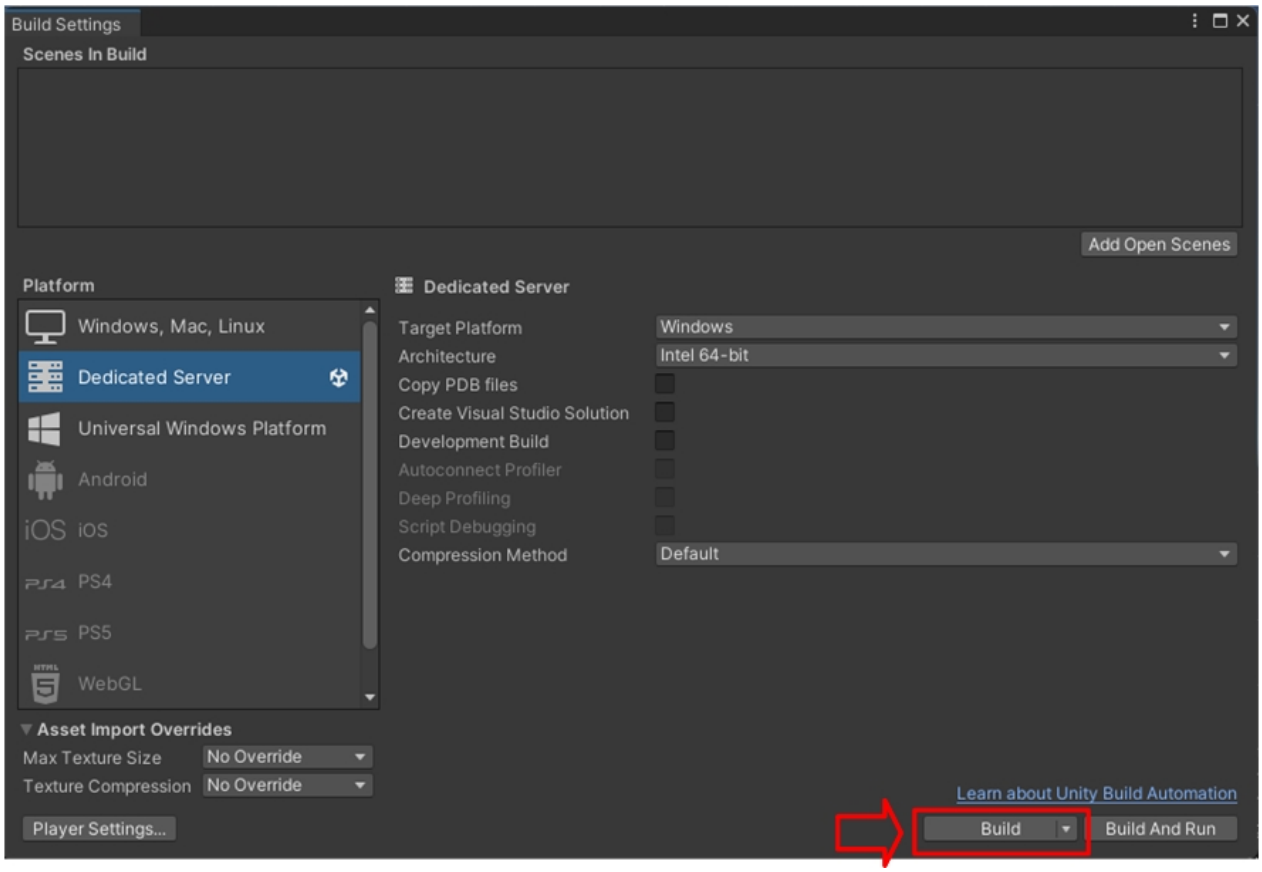
3. "Connect on Start" is checked (This will ensure that server will start after launched)



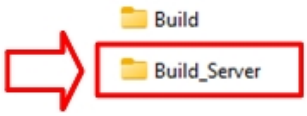
4. Now you need to switch you project to "Dedicated Server Mode" (see Unity Dedicated Server)



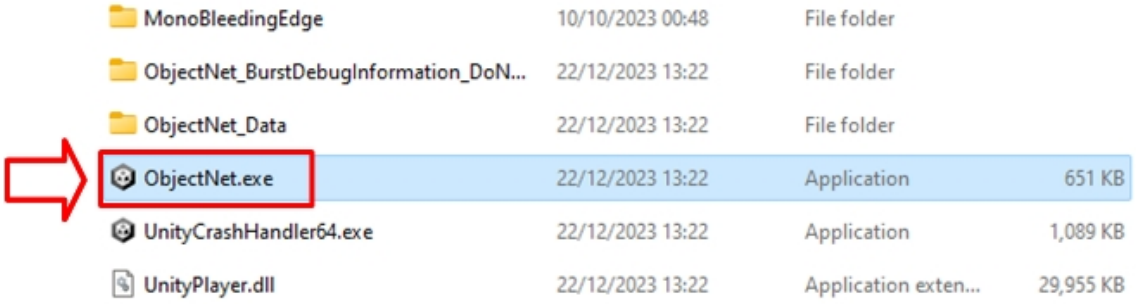
5. After finished a "Build" option shall appear



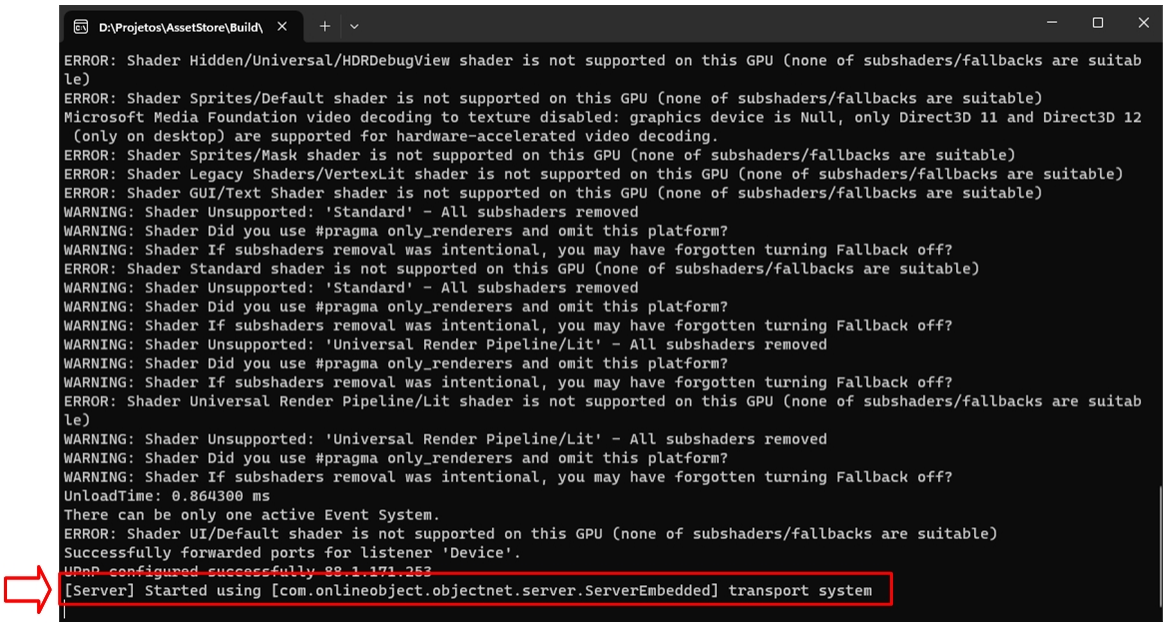
6. Build server mode into a separated folder (It's important separate Game build from Server build)



7. No run the server build



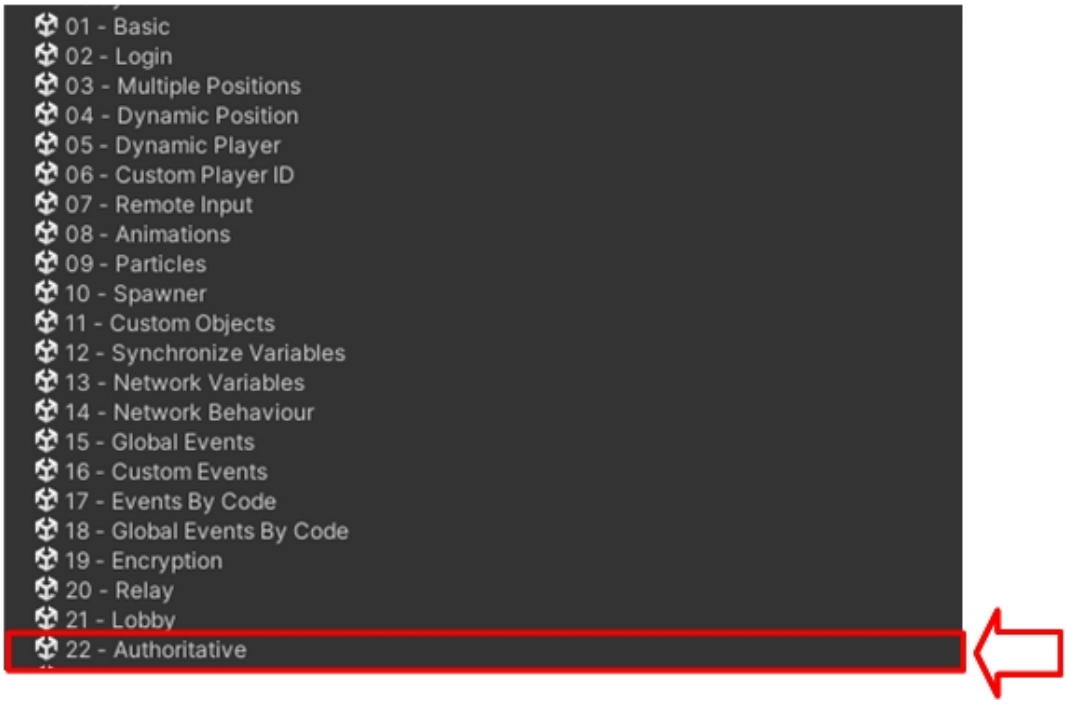
8. Server should be run without any problem



The log shall show the server up and running and the current user transport system.

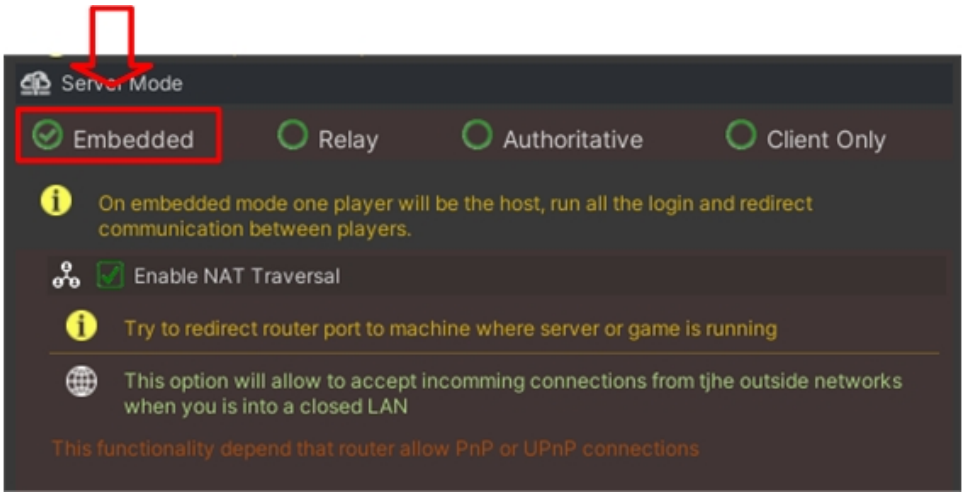
Building Client

Open scene "22 - Authoritative" under examples scene folder.

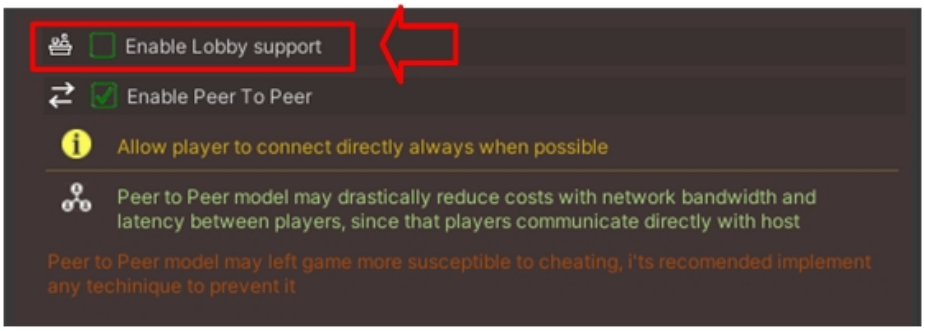


Ensure that following option if correctly, if not, change it.

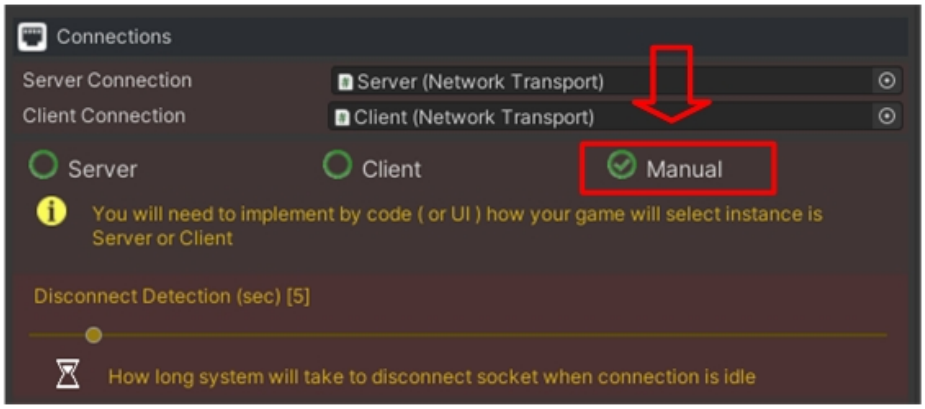
1. "Embedded" or "Client Only" must be selected. On this example we will select "Embedded".



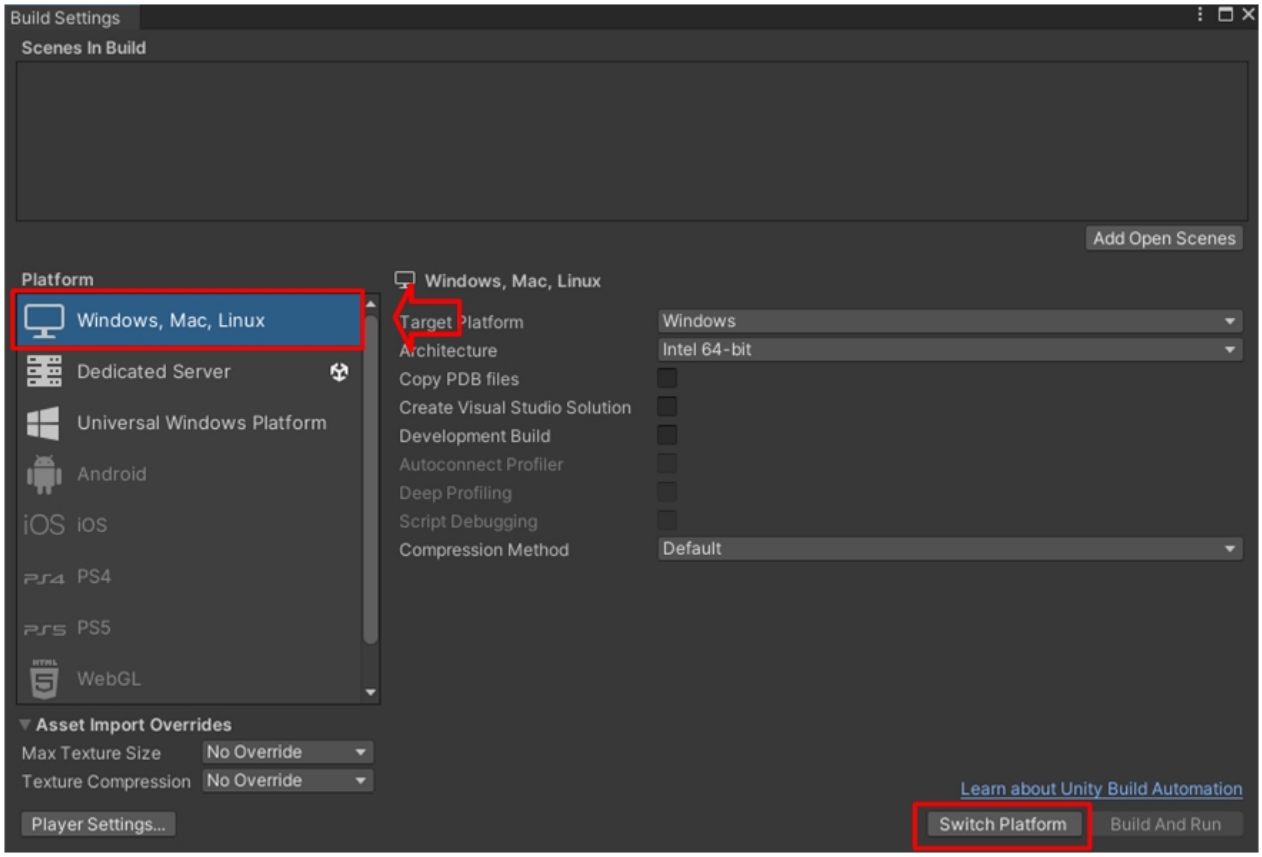
2. "Enable Lobby support" in unchecked



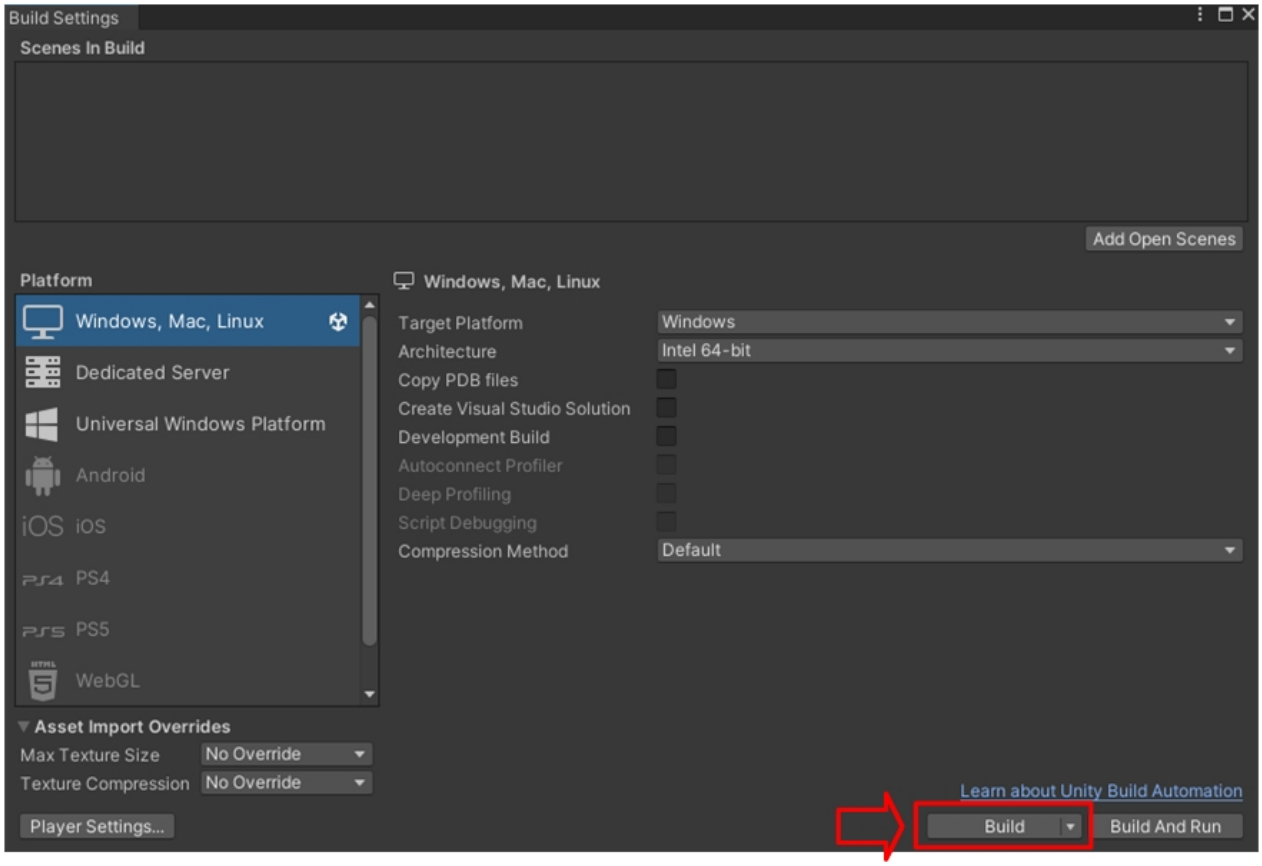
3. "Client" or "Manual" connection type shall be flagged



4. Now you need to switch you project to the selected platform target, on this case we going to use "Windows, Mac, Linux"

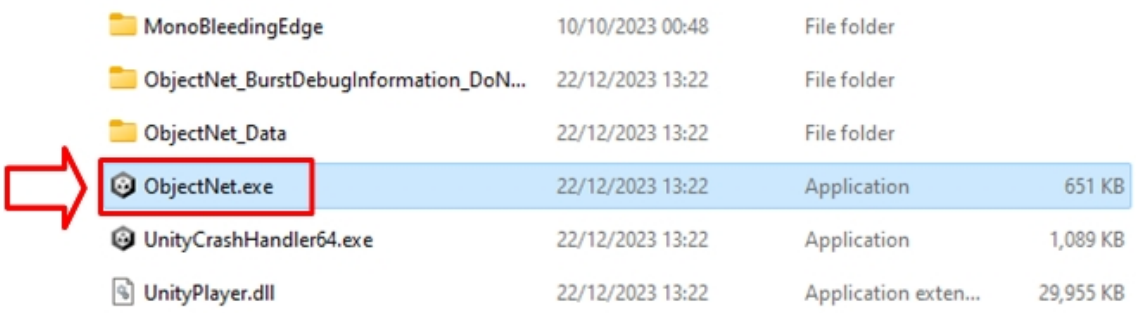


5. After finished a "Build" option shall appear



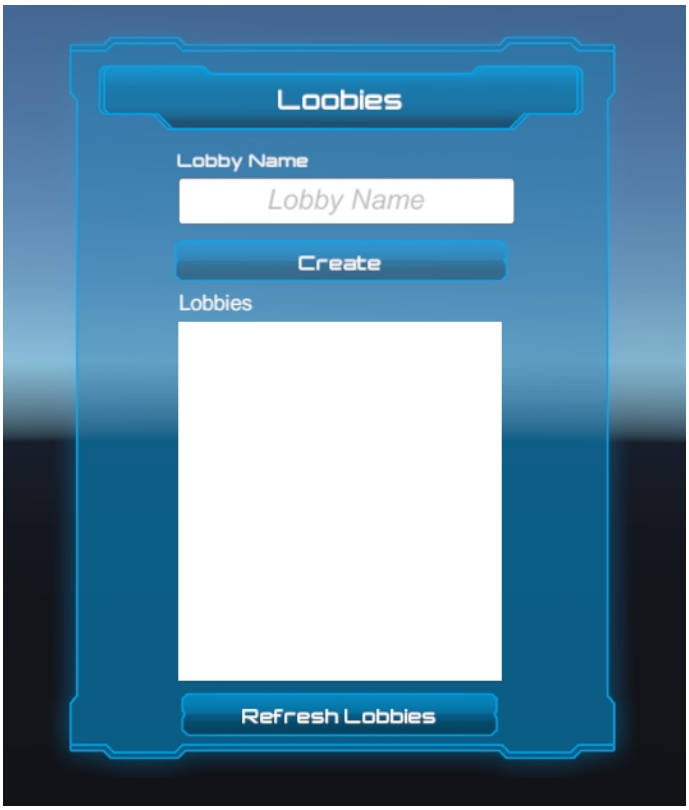
6. Build server mode into a separated folder (It's important separate Game build from Server build)

7. No run the compiled game

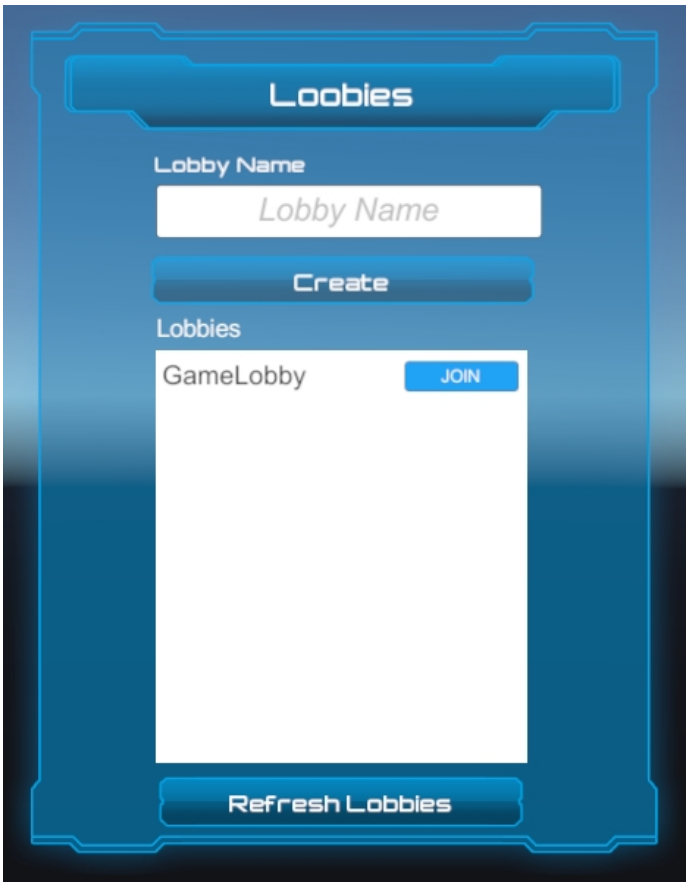


8. The game shall start

9. A window shall appear to allow to create or enter into a lobby



10. If you wish to list current lobbies, you need to click on "Refresh Lobbies" and then click into join on the selected lobby



11. Player must be spawned on scene

12. Player will be connected at Server

```
 (only on desktop) are supported for hardware-accelerated video decoding.
ERROR: Shader Sprites/Mask shader is not supported on this GPU (none of subshaders/fallbacks are suitable)
ERROR: Shader Legacy Shaders/VertexLit shader is not supported on this GPU (none of subshaders/fallbacks are suitable)
ERROR: Shader GUI/Text Shader shader is not supported on this GPU (none of subshaders/fallbacks are suitable)
WARNING: Shader Unsupported: 'Standard' - All subshaders removed
WARNING: Shader Did you use #pragma only_renderers and omit this platform?
WARNING: Shader If subshaders removal was intentional, you may have forgotten turning Fallback off?
ERROR: Shader Standard shader is not supported on this GPU (none of subshaders/fallbacks are suitable)
WARNING: Shader Unsupported: 'Standard' - All subshaders removed
WARNING: Shader Did you use #pragma only_renderers and omit this platform?
WARNING: Shader If subshaders removal was intentional, you may have forgotten turning Fallback off?
WARNING: Shader Unsupported: 'Universal Render Pipeline/Lit' - All subshaders removed
WARNING: Shader Did you use #pragma only_renderers and omit this platform?
WARNING: Shader If subshaders removal was intentional, you may have forgotten turning Fallback off?
ERROR: Shader Universal Render Pipeline/Lit shader is not supported on this GPU (none of subshaders/fallbacks are suitab
le)
WARNING: Shader Unsupported: 'Universal Render Pipeline/Lit' - All subshaders removed
WARNING: Shader Did you use #pragma only_renderers and omit this platform?
WARNING: Shader If subshaders removal was intentional, you may have forgotten turning Fallback off?
UnloadTime: 0.723200 ms
There can be only one active Event System.
ERROR: Shader UI/Default shader is not supported on this GPU (none of subshaders/fallbacks are suitable)
Successfully forwarded ports for listener 'Device'.
UPnP configured successfully 88.1.171.253
[Server] Started using [com.onlineobject.objectnet.server.ServerEmbedded] transport system
New Client connected
Client created [1]
Connected sent [1]
New client connected [:0]
```

Programing

ObjectNet provides a lot of functionality to make it possible to integrate your game without the need to make deep changes, nonetheless, to extract the maximum of ObjectNet and implement advanced features, you may need to create a script or convert some classes.

This section covers the main topics about how to integrate your game with ObjectNet when code changes

are required**.**

Events

ObjectNet provides a full API to allow the developer to send and listen to events by code.

That event can be global or local. Global Events can be listened to or sent by any part of your code, on the other hand, Local Events can only be sent or caught by the object who sends and listens to this event.

This section will cover Global Event, the Object Event is covered on NetworkBehaviour section.

Sending Events

This section covers how to send events over the network.

First, you need to create a unique identifier for this event, this identifier shall be a positive integer number.

```
      const int CUSTOM_EVENT = 10000;
```

Note: Is highly recommenced to group events to avoid two events using the same code.

Events provide the possibility to send data, for example, in an event to restart a player position you may wish to send position information in the message. ObjectNet allows us to do this by using DataWritter.

This code illustrates how to send an event over the network, you can include data on the message to be received at the target destination.

```
      using (DataStream writer = new DataStream()) {
            writer.Write(this.targetPosition); // Vector3
            writer.Write(this.remaingLife); // float
            writer.Write(this.money); // int
            // Send event
            NetworkManager.Instance().Send(CUSTOM_EVENT, writer,
DeliveryMode.Reliable);
      }
```

You can include the following data types on the event message.

```
      int
      uint
      long
      ulong
      short
      ushort
      float
      double
      byte
```

```
byte[]
string
char
char[]
bool
Vector3
Color
```

Listening Events

This section covers how to Listen to events over the network.

First, you need to create or use a unique identifier for this event, this identifier shall be a positive integer number.

```
const int CUSTOM_EVENT = 10000;
```

Note: Is highly recommenced to group events to avoid two events using the same code.

You can listen to events sent by other network peers and execute some action based on this event, this process is called by ObjectNet a Listener on an event.

This code illustrates how to listen to some network events, read data on these events, and take some action.

```
public void Start() {
      NetworkManager.Events.RegisterEvent(CUSTOM_EVENT,
this.OnCustomEventReceived);
}

private void OnCustomEventReceived(IDataStream reader) {
      Vector3 receivedTargetPosition      = reader.Read<Vector3>();
      float   remaningLife                = reader.Read<float>();
      int     money                       = reader.Read<int>();
      // Update player with received information
      this.UpdatePlayer(receivedPosition, remaningLife, money);

}
```

On this piece of code, the **CUSTOM_EVENT** is registered to execute "**OnCustomEventReceived**" arrives.

The code of **OnCustomEventReceived** the first extract parameter in the same order that was sent. With this information, some method must be called to update an object, change status, create or destroy items, and another code to make your multiplayer game work.

NetworkBehaviour

ObjectNet provides a special class that can be extended to provide a bunch of events and methods, this class is NetworkBehaviour.

The NetworkBehaviour is a class that replaces Unity MonoBehaviour class and provides a lot of embedded methods and events.

NetworkBehavour can be used even if your game doesn't implement MultiPlayer features, this means if your game must work on the single-player mode you can keep this inheritance without affecting your single-player mode behavior.

This is pretty useful since the developer can create your game to run in Single-player mode and multi-player while keeping the same code base.

This section will cover the main aspects of NetworkBehaviour.

To extend your MonoBehaviour script from NetworkBehaviour it's just change inheritance.

```
      public class MyGameScript : NetworkBehaviour {

            // Your class code

      }
```

Once inherited, you can implement some special method to handle your multiplayer game, the following script will illustrate the main methods:

OnNetworkStarted : This event is called once when Network is available to this object.

```
      /// <summary>
      /// This event is called when Network features starts
      ///
      /// Note : This event is controlled internally by framework and ensure
that
      ///        network is up and running
      /// </summary>
      public override void OnNetworkStarted() {
            // Here you can do any that you need after ensure that Network is
initialized
      }
```

ActiveAwake : This event is the ***Awake*** execute only when object is running on Active Mode.

```
      /// <summary>
      /// Active Awake is executed when objects on Active mode Awake
      /// </summary>
      public void ActiveAwake() {
      }
```

PassiveAwake : This event is the ***Awake*** execute only when object is running on Passive Mode.

```
      /// <summary>
      /// Active Awake is executed when objects on Passive mode Awake
      /// </summary>
      public void PassiveAwake() {
```

```
    }
```

ActiveStart : This event is the ***Start*** execute only when object is running on Active Mode.

```
    /// <summary>
    /// Active Start is executed when objects on Active mode Start
    /// </summary>
    public void ActiveStart() {
    }
```

PassiveStart : This event is the ***Start*** execute only when object is running on Passive Mode.

```
    /// <summary>
    /// Passive Start is executed when objects on Passive mode Start
    /// </summary>
    public void PassiveStart() {
    }
```

ActiveUpdate : This event is the ***Update*** execute only when object is running on Active Mode.

```
    /// <summary>
    /// Active Update is executed every frame for Active object
    /// </summary>
    public void ActiveUpdate() {
    }
```

PassiveUpdate : This event is the ***Update*** execute only when object is running on Passive Mode.

```
    /// <summary>
    /// Passive Update is executed every frame for Passive object
    /// </summary>
    public void PassiveUpdate() {
    }
```

ActiveFixedUpdate : This event is the ***FixedUpdate*** execute only when object is running on Active Mode.

```
    /// <summary>
    /// Active FixedUpdate is executed every physics frame for Active object
    /// </summary>
    public void ActiveFixedUpdate() {
    }
```

PassiveFixedUpdate : This event is the ***FixedUpdate*** execute only when object is running on Passive Mode.

```
    /// <summary>
```

```
      /// Passive FixedUpdate is executed every physics frame for Passive
object
      /// </summary>
      public void PassiveFixedUpdate() {
      }
```

ActiveLateUpdate : This event is the ***LateUpdate*** execute only when object is running on Active Mode.

```
      /// <summary>
      /// Active LateUpdate is executed every frame after Update for Active
object
      /// </summary>
      public void ActiveLateUpdate() {
      }
```

PassiveLateUpdate : This event is the ***LateUpdate*** execute only when object is running on Passive Mode.

```
      /// <summary>
      /// Passive LateUpdate is executed every frame after Update for Passive
object
      /// </summary>
      public void PassiveLateUpdate() {
      }
```

Sending Events

This section covers how to send events over the network from a specific NetworkObject.

First, you need to create a unique identifier for this event, this identifier shall be a positive integer number.

```
      const int LOCAL_CUSTOM_EVENT = 20000;
```

Note: Is highly recommenced to group events to avoid two events using the same code.

Local events work exactly as Global Events, the main difference is that Object Event is sent having a NetworkObject as origin and will arrive on the same object in the client instance, this means that Object Event is a smart way to update or change a specific NetworkObject.

This code illustrates how to send an event from an object over the network, you can include data on the message to be received on the target destination.

```
      using (DataStream writer = new DataStream()) {
            writer.Write(this.transform.position); // Vector3
            writer.Write(this.objectStatus); // int
            writer.Write(this.errorCode); // uint
            // Send event
            // Is important to send using "this" because system will include
```

```
object information's during event send
            this.Send(LOCAL_CUSTOM_EVENT, writer, DeliveryMode.Reliable);
      }
```

Listening Events

This section covers how to Listen to events over the network.

First, you need to create or use a unique identifier for this event, this identifier shall be a positive integer number.

```
      const int LOCAL_CUSTOM_EVENT = 20000;
```

Note: Is highly recommenced to group events to avoid two events using the same code.

You can listen to events sent by other network peers and execute some action based on this event, this event will arrive specifically on the same object where the event will be raised on the server or another remote player.

This code illustrates how to listen to some network event from a NetworkObject, read data on this event, and take some action.

```
      public void Start() {
            this.RegisterEvent(LOCAL_CUSTOM_EVENT,
this.OnCustomEventReceivedOnObject);
      }

      private void OnCustomEventReceivedOnObject(IDataStream reader) {
            Vector3 receivedPosition      = reader.Read<Vector3>();
            float   receivedStatus        = reader.Read<float>();
            int     receivedErrorCode     = reader.Read<ushort>();
            // Update player with received information
            this.RaiseObjectError(receivedPosition, receivedStatus,
receivedErrorCode);
      }
```

On this piece of code, the **LOCAL_CUSTOM_EVENT** is registered to execute "**OnCustomEventReceivedOnObject**" arrives.

The code of **OnCustomEventReceivedOnObject** first extracts parameters in the same order they were sent. With this information, some method must be called on this object to update an object, change the status, create or destroy items, and another code to make your multiplayer game work.

Manual Animation

ObjectNet allows to manually animate objects instead using auto animation provided by Network Prefab (see Synchronization Options).

To animate objects using ObjectNet API you need to use the following code.

```
    this.Animation.Play("clipName");
```

This method will play animation locally and send animation event over network to be played on network instances

Important: This method is available only to classes inherited from NetworkBehaviour.

Note: You can do the same or create a custom synchronization by using your own event.

Playing Audio

ObjectNet allows to play audio clips, sound or event over network.

To play audio clips over network you need to use ObjectNet API you need to use the following code.

```
    this.Audio.Play(this.runningAudioClip);
```

This method will play the selected audio track locally and send audio clip information on an event over the network to be played on network instances.

Important: This method is available only to classes inherited from NetworkBehaviour.

Note: You can do the same or create a custom synchronization by using your own event.

Network Variables

See Network Variables

Custom Behaviour

ObjectNet provides the facility to create custom behavior and apply it to NetworkObjects (see Network Behaviour).

To implement your own Behavior you can check the API document on `NetworkEntity` class.

The following piece of code example of how to implement your behavior :

```
public struct DataToSync {
    public Vector3 positionValue;
    public Color color;
}

public class BehaviourExample : NetworkEntity<DataToSync, IDataStream> {

    private DataToSync currentValue;

    public BehaviourExample() : base() {
```

```
        }

        public BehaviourExample(INetworkElement networkObject) :
base(networkObject) {
        }

        public override void ComputeActive() {
            this.currentValue.positionValue =
this.GetNetworkObject().GetGameObject().transform.position;
            this.currentValue.color         =
this
.GetNetworkObject().GetGameObject().GetComponent
<Renderer>().materials[0].color;
        }

        public override void ComputePassive() {
            this.GetNetworkObject().GetGameObject().transform.position =
this.currentValue.positionValue;
            this
.GetNetworkObject().GetGameObject().GetComponent
<Renderer>().materials[0].color = this.currentValue.color;
        }

        public override DataToSync GetPassiveArguments() {
            return this.currentValue;
        }

        public override void SynchonizePassive(DataToSync data) {
            this.currentValue.color         = data.color;
            this.currentValue.positionValue = data.positionValue;
        }

        public override void SynchonizeActive(IDataStream writer) {
            writer.Write(this.currentValue.positionValue);
            writer.Write(this.currentValue.color);
        }

        public override void Extract(IDataStream reader) {
            this.currentValue.positionValue = reader.Read<Vector3>();
            this.currentValue.color         = reader.Read<Color>();
        }
    }
```

After your behavior is created you need to register it on all objects that you need to synchronize information.

```
    public void Awake() {
       this.GetNetworkElement().RegisterBehavior(new BehaviourExample());
    }
```

This code tells **ObjectNet** to include this behavior during the network synchronization procedure.

Custom DataStream

ObjectNet provides the facility to send and receive simple and complex structured over-network (see

DataStream).

The main idea of DataStream is to write and read data without the need to write on each place, you can use some DataStream and encapsulate how much data you need.

To implement your own DataStream you can check the API document on `NetworkEntity` class.

The following piece of code example how to implement your own DataStream :

```
    public class DataToSync {
        public Vector3 positionValue;
        public Color color;

        public DataToSync(Vector3 position, Color color) {
          this.positionValue = position;
          this.color = color;
        }
    }

    public class DataStreamExample : DataHandler<Vector3> {

        public override int Write(DataToSync data, ref byte[] buffer, ref int
offset) {
            int result = base.Write(data.positionValue, ref buffer, ref
offset, typeof(Vector3));
            result += base.Write(data.color, ref buffer, ref offset,
typeof(Color));
            return result;
        }

        public override DataToSync Read(byte[] buffer, ref int offset) {
            return new DataToSync(this.Read<Vector3>(buffer, ref offset),
                                  this.Read<Color>(buffer, ref offset));
        }
    }
```

After your data stream is created you need to register into stream factory to be visible to ObjectNet engine.

```
    DataStream.RegisterGlobalStream<DataToSync, DataStreamExample>();
```

This code tells to **ObjectNet** that every time that someone try to send or receive a "**DataToSync**" object, the "**DataStreamExample**" must be used.

Spawning Objects

ObjectNet allows to spawn objects over network, when those objects are spawned all network peers will be notified to spawn the same object.

You can spawn objects automatically by using NetworkPrefabs automatic detection (see Registering Prefab), this detection is made on the host or server only, and once detected this spawned object will be sprayed for all network peers.

On the server of host client, this spawn can be execute using standard Unity methods.

```
      GameObject.Instantiate(this.prefabToSpawn, this.positionToSpwan,
Quaternion.identity);
```

When this object is spawned, server or host will detect it and send spawn event over the network to all connected players.

Nonetheless, if you need to spawn object on Passive network clients or even to spawn on the same way on all parts of your game you need to use an ObjectNet special method.

```
      // Send event to spawn this prefab on server
      NetworkGameObject.NetworkInstantiate(this.projectile,
this.spawnPosition.position, rocketRotation);
```

You can also use this method on the server side, on this case, you have the option to get the spawned object as functions return.

```
      // Send event to spawn this prefab on server
      GameObject spawned =
NetworkGameObject.NetworkInstantiate(this.projectile,
this.spawnPosition.position, rocketRotation);
```

Note: If you spawn an object using unity standard spawn on the Passive instance of the game, those objects will no be spawned on the other network peers, The standard Unity spawn method only works if you execute it on the server or host player and only if the object was registered on NetworkPrefabs manager.

Network Methods

ObjectNet allows to execute script methods over network, this can be a powerful mechanism to keep you game synchronized and handle with aspects of your multiplayer game.

The network method can be executed mainly in three ways.

```
      // You class must have a method
      void MethodName() {
        ... Do something
      }

      // And in any part of you code at same class you can have
      this.NetworkExecute(this.MethodName);
```

ObjectNet will decide the best place to execute you code, if code will be executed on server, on client or in both.

You can also select if your method shall be executed on server only.

```
      // Execute method on server only
      this.NetworkExecuteOnServer(this.MethodName);
```

Finally you can select if your method shall be executed on client's.

```
// Execute method on all clients ( including me if I'm a client )
this.NetworkExecuteOnClient(this.MethodName);
```

ObjectNet also allow to execute methods with parameters if needed

```
// You class must have a method
void MethodName(int life, string name) {
   // Do something
}

// And in any part of you code at same class you can have
this.NetworkExecute<Int32, String>(this.MethodName, 10, "MyName");
```

Note: You can execute methods with up to 10 arguments.

API Reference

ObjectNet provides a full API Documentation Reference. You can access the official ObjectNet API here (ObjectNet API);