EuNet C# (.NET, .NET Core, Unity)

Build-Debug passing Build-Release passing downloads 1.4k release v1.1.16

Easy Unity Network (EuNet) is a network solution for multiplayer games.

Supports Server-Client, Peer to Peer communication using TCP, UDP, and RUDP protocols.

In the case of P2P (Peer to Peer), supports hole punching and tries to communicate directly as much as possible, and if it is impossible, automatically relayed through the server.

Great for developing Action MORPG, MOBA, Channel Based MMORPG, Casual Multiplayer Game (e.g. League of Legends, Among Us, Kart Rider, Diablo, etc.).

Produced based on .Net Standard 2.0, multiplatform supported(Windows, Linux, Android, iOS, etc.), and is optimized for .Net Core-based servers and Unity3D-based clients.

See EuNet-Starter for an example

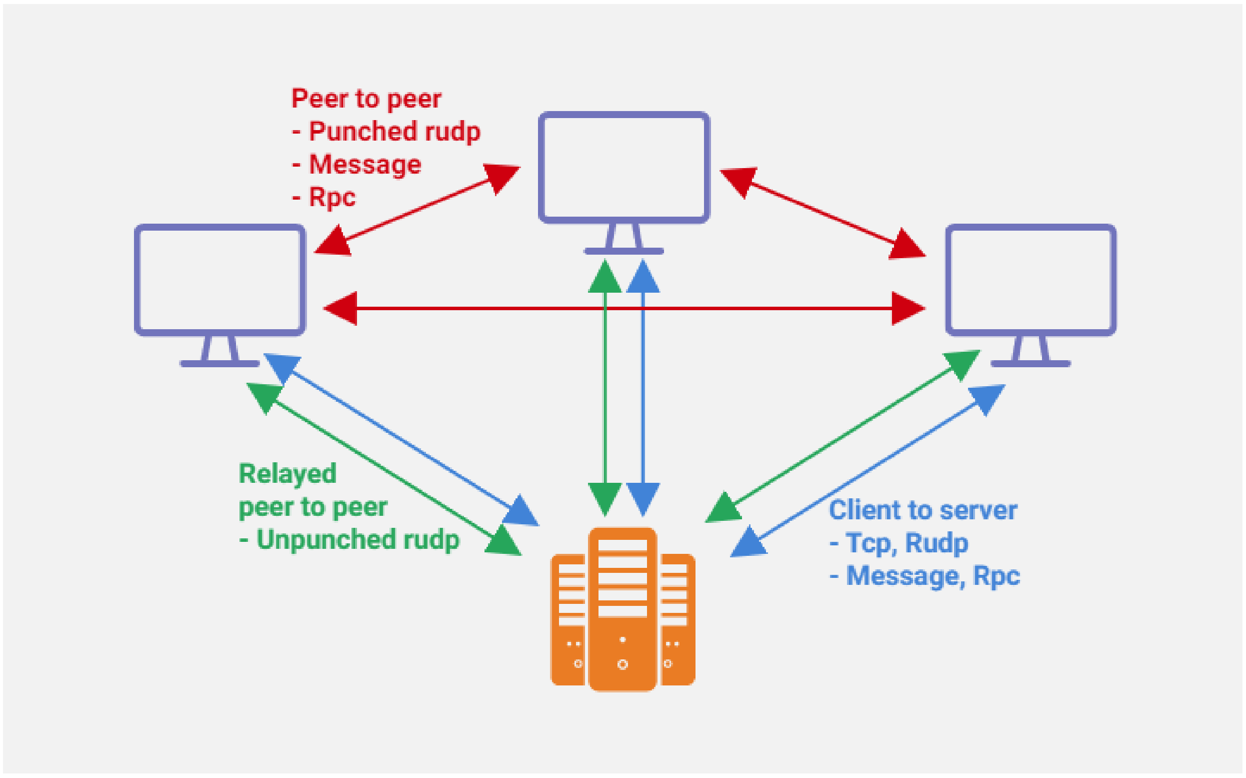


Table of Contents

Features

- Fast network communication
  - High speed communication using multi-thread
  - Fast allocation using pooling buffer
- Supported channels
  - TCP
  - Unreliable UDP
  - Reliable Ordered UDP
  - Reliable Unordered UDP
  - Reliable Sequenced UDP
  - Sequenced UDP
- Supported communication
  - Client to Server
  - Peer to Peer
    - Hole Punching
    - Relay (Auto Switching)
- RPC (Remote Procedure Call)
- Fast packet serializer (Partial using MessagePack for C#)
- Custom Compiler(EuNetCodeGenerator) for fast serializing and RPC
- Automatic MTU detection
- Automatic fragmentation of large UDP packets
- Automatic merging small packets
- Unity3D support
- Supported platforms
  - Windows / Mac / Linux (.Net Core)
  - Android (Unity)
  - iOS (Unity)

Channels

Channels	Transmission guarantee	Not duplicate	Order guarantee

Channels	Transmission guarantee	Not duplicate	Order guarantee
TCP	✓	✓	✓
Unreliable UDP	✗	✗	✗
Reliable Ordered UDP	✓	✓	✓
Reliable Unordered UDP	✓	✓	✗
Reliable Sequenced UDP	✓ (Last order)	✓	✓
Sequenced UDP	✗	✓	✓

Installation

We need three projects

- Common project (.Net Standard 2.0)
  - Server, Client common use
  - Generate code using EuNetCodeGenerator
- Server project (.Net Core)
- Client project (Unity3D)

See EuNet-Starter for an example

Common project

- Create .Net Standard 2.0 based project.
- Install nuget package.

```
PM> Install-Package EuNet.CodeGenerator.Templates
```

- Rebuild project.
- If you look at the project, `CodeGen/EuNet.Rpc.CodeGen.cs` file was created.

Server project (.net core)

- First install the nuget package.

```
PM> Install-Package EuNet
```

- Add common project to reference

```
Solution Explorer -> [User Project] -> References -> Add Reference -> [Add
Common project]
```

- Write server code. Server Code Sample
- Write session code. Session Code Sample

Client project (Unity3D)

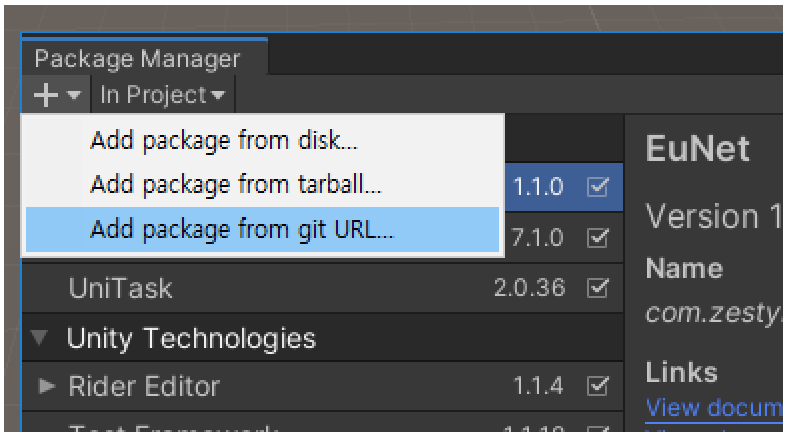
Install via git URL

After Unity 2019.3.4f1, Unity 2020.1a21, that support path query parameter of git package. You can add package from UPM (Unity Package Manager)

```
https://github.com/zestylife/EuNet.git?
path=src/EuNet.Unity/Assets/Plugins/EuNet
```

If you want to add a specific release version, add `#version` after the url. ex) version 1.1.13

```
https://github.com/zestylife/EuNet.git?
path=src/EuNet.Unity/Assets/Plugins/EuNet#1.1.13
```



Install via package file

- Install the unity-package. Download here

Rpc Sample

Common project

```
// Declaring login rpc interface
public interface ILoginRpc : IRpc
{
    Task<int> Login(string id, ISession session);
    Task<UserInfo> GetUserInfo();
}
// Generate Rpc code using EuNetCodeGenerator and use it in server and client
```

Server project (.Net Core)

```
// User session class inherits Rpc Interface (ILoginRpc)
public partial class UserSession : ILoginRpc
{
    private UserInfo _userInfo = new UserInfo();

    // Implement Rpc Method that client calls
    public Task<int> Login(string id, ISession session)
    {
        if (id == "AuthedId")
            return Task<int>.FromResult(0);

        return Task<int>.FromResult(1);
    }

    // Implement Rpc Method that client calls
    public Task<UserInfo> GetUserInfo()
    {
        // Set user information
        _userInfo.Name = "abc";

        return Task<UserInfo>.FromResult(_userInfo);
    }
}
```

Client project (Unity3D)

```
private async UniTaskVoid ConnectAsync()
{
    var client = NetClientGlobal.Instance.Client;

    // Trying to connect. Timeout is 10 seconds.
    var result = await client.ConnectAsync(TimeSpan.FromSeconds(10));

    if(result == true)
    {
        // Create an object for calling login Rpc
        LoginRpc loginRpc = new LoginRpc(client);

        // Call the server's login function (UserSession.Login)
        var loginResult = await loginRpc.Login("AuthedId", null);
```

```
            Debug.Log($"Login Result : {loginResult}");
            if (loginResult != 0)
                return;

            // Call the server's get user information function (UserSession.GetUserIr
            var userInfo = await loginRpc.GetUserInfo();
            Debug.Log($"UserName : {userInfo.Name}");
            // UserName : abc
        }
        else
        {
            // Fail to connect
            Debug.LogError("Fail to connect server");
        }
    }
```

Quick Start

Serialize

Object serialization is required to use Rpc. There are two ways to serialize objects.

Using Auto-Generated formmater

Declaring the `NetDataObject` Attribute makes the class serializable. All public objects are serialized. Declaring the `[IgnoreMember]` Attribute does not serialize it.

```
[NetDataObject]
public class DataClass
{
    // Serializable
    public int Int;

    // Serializable
    public int Property { get; set; }

    // Ignore
    public int PropertyOnlyGet { get; }

    // Ignore
    private int IntPrivate;

    // Ignore
    protected int IntProtected;

    // Ignore
    [IgnoreMember]
    public int IgnoreInt;
    // Ignore
    [IgnoreMember]
    public int IgnoreProperty { get; set; }
}
```

Manualy serialize

Implement serialization manually by inheriting `INetSerializable` . You have to code, but it's the fastest and most flexible.

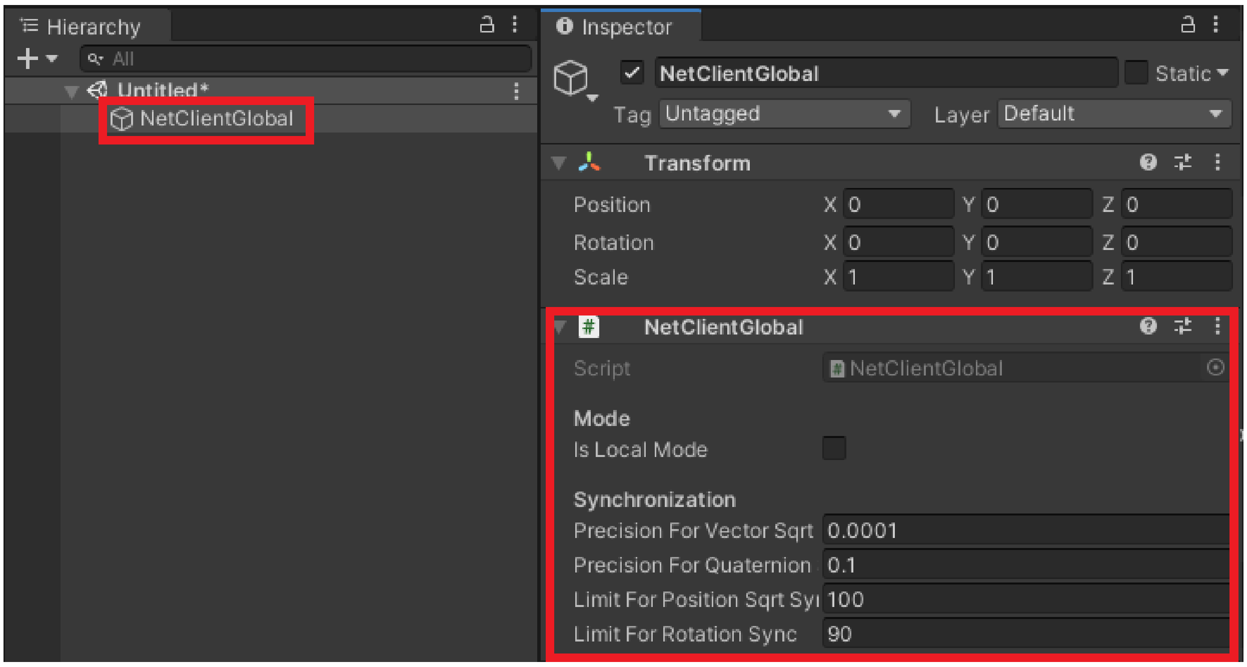
```
public class InterfaceSerializeClass : INetSerializable
{
    public int Value;
    public string Name;

    public void Serialize(NetDataWriter writer)
    {
        writer.Write(Value);
        writer.Write(Name);
    }

    public void Deserialize(NetDataReader reader)
    {
        Value = reader.ReadInt32();
        Name = reader.ReadString();
    }
}
```

Unity3D

- Special object NetView is supported, and synchronization and Rpc communication between NetViews are possible.
  (NetClientP2pBehaviour required. Peer to Peer only)
- Supported global settings with NetClientGlobal object. (Singleton)
- Supported for communication to the server (NetClientBehaviour, NetClientP2pBehaviour)
- Support NetClientP2pBehaviour only one.

Settings

- Add an empty GameObject to the first run Scene and add a NetClientGlobal component.
  Only one should be made globally.


Hierarchy

Untitled*

NetClientGlobal

Inspector

NetClientGlobal

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

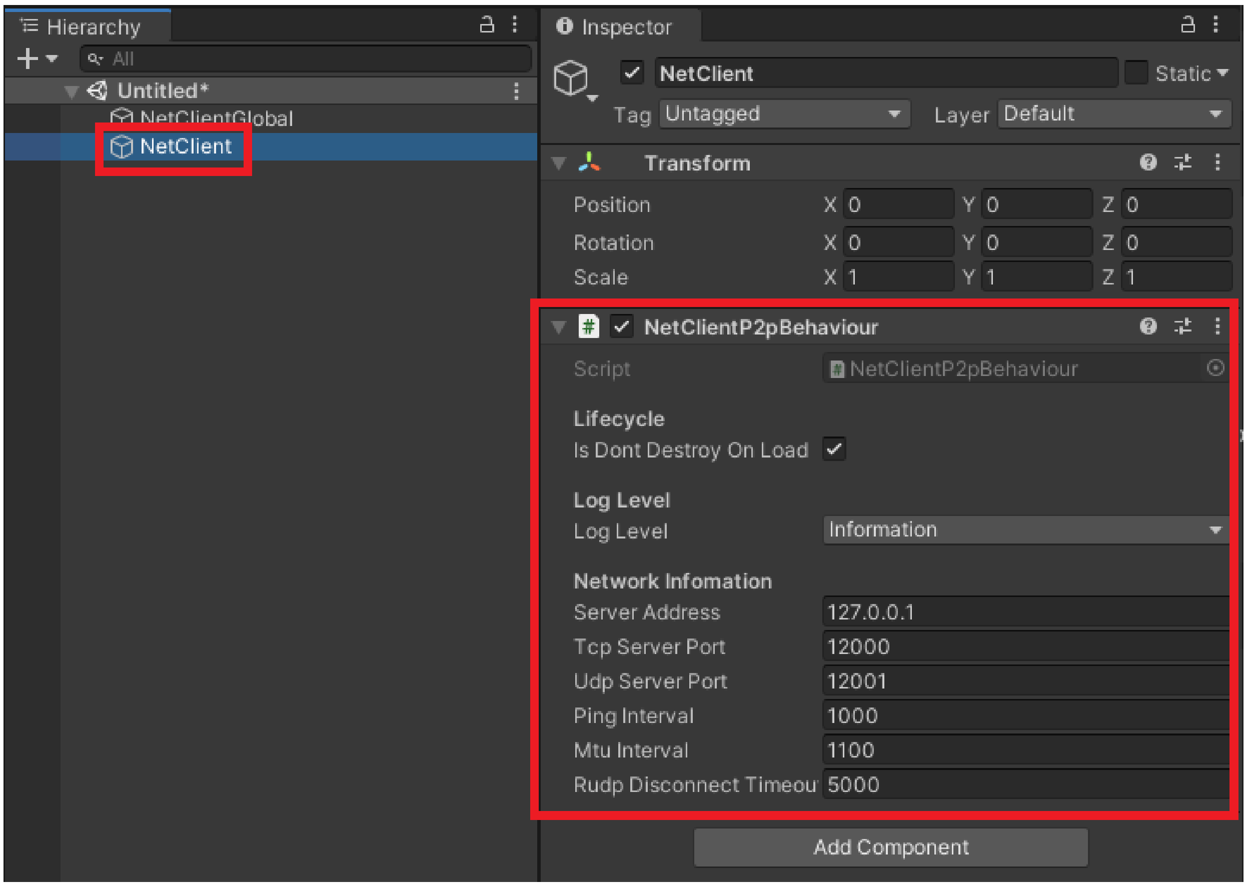
NetClientGlobal

Script NetClientGlobal

Mode
Is Local Mode

Synchronization
Precision For Vector Sqrt 0.0001
Precision For Quaternion 0.1
Limit For Position Sqrt Syı 100
Limit For Rotation Sync 90

- Add NetClientP2pBehaviour component for communicate to one server (including P2p).
  Modify the options as needed.


Hierarchy

Untitled*

NetClientGlobal

NetClient

Inspector

NetClient

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

NetClientP2pBehaviour

Script NetClientP2pBehaviour

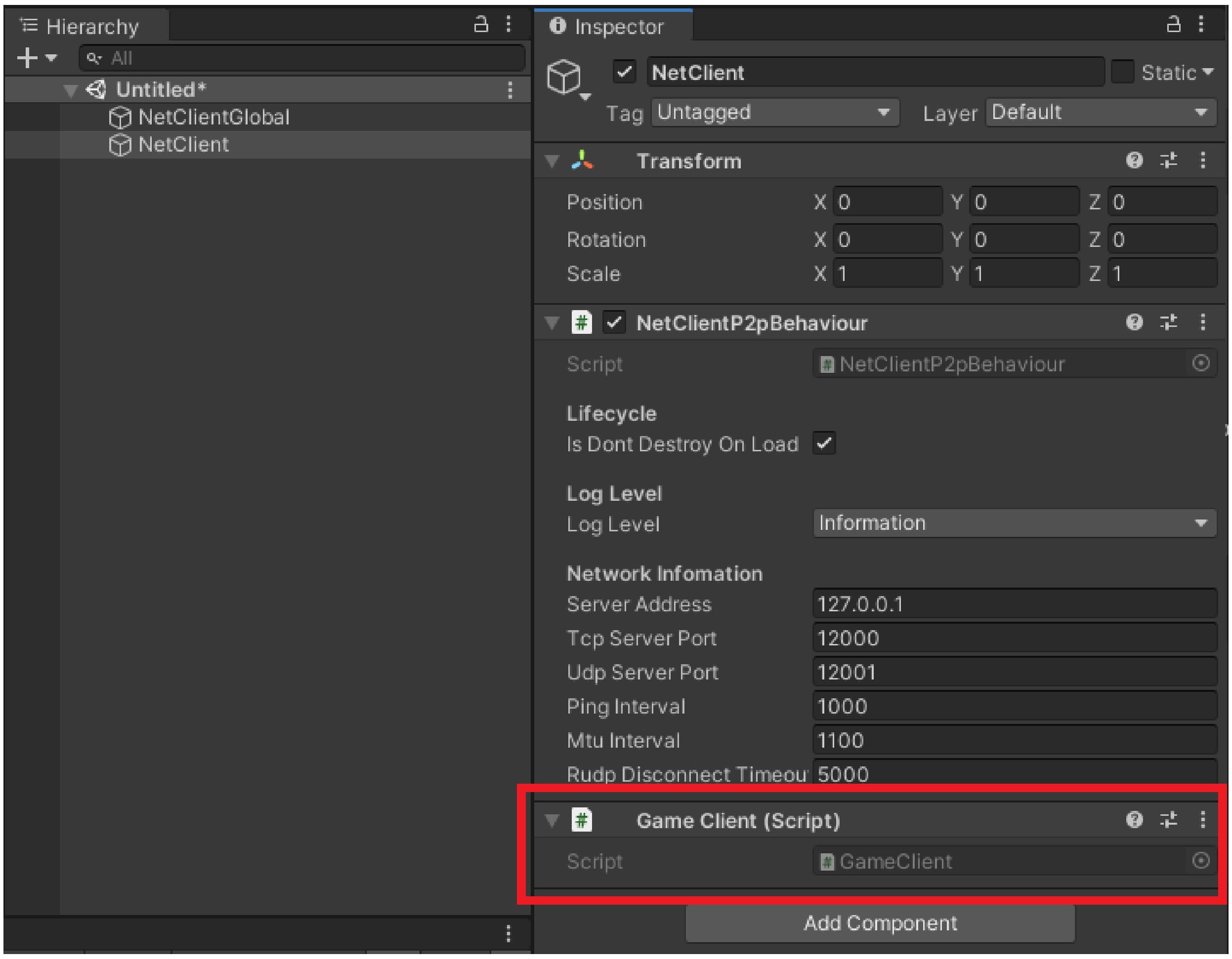
Lifecycle
Is Dont Destroy On Load ✓

Log Level
Log Level Information

Network Infomation
Server Address 127.0.0.1
Tcp Server Port 12000
Udp Server Port 12001
Ping Interval 1000
Mtu Interval 1100
Rudp Disconnect Timeou 5000

Add Component

- Add user component to receive and process events.


Hierarchy
Untitled*
NetClientGlobal
NetClient

Inspector
NetClient
Tag Untagged
Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

NetClientP2pBehaviour
Script NetClientP2pBehaviour

Lifecycle
Is Dont Destroy On Load ✓

Log Level
Log Level Information

Network Infomation
Server Address 127.0.0.1
Tcp Server Port 12000
Udp Server Port 12001
Ping Interval 1000
Mtu Interval 1100
Rudp Disconnect Timeou 5000

Game Client (Script)
Script GameClient

- `GameClient.cs` file

```
using Common.Resolvers;
using EuNet.Core;
using EuNet.Unity;
using System.Threading.Tasks;

public class GameClient : Singleton<GameClient>
{
    private NetClientP2pBehaviour _client;

    public NetClientP2p Client => _client.ClientP2p;

    protected override void Awake()
    {
        base.Awake();

        _client = GetComponent<NetClientP2pBehaviour>();

        Client.OnConnected = OnConnected;
        Client.OnClosed = OnClosed;
        Client.OnReceived = OnReceive;

        // Register automatically generated resolver.
        //CustomResolver.Register(GeneratedResolver.Instance);

        // If you generated RpcService, register it.
```

```
            //Client.AddRpcService(new GameScRpcService());
        }

        public Task<bool> ConnectAsync()
        {
            // Try to connect server. All functions can be accessed with Client insta
            return Client.ConnectAsync(TimeSpan.FromSeconds(10));
        }

        private void OnConnected()
        {
            // Connected
        }

        private void OnClosed()
        {
            // Disconnected

        private Task OnReceive(NetDataReader reader)
        {
            // Received data. No need to use when using RPC
            return Task.CompletedTask;
        }
    }
```

How to use Rpc

Rpc is service that can call remote procedures.
EuNet's Rpc is a function call service between the server and the client.
When you declare an interface that inherits the IRpc interface, calls and service codes are automatically generated.

- Create Rpc Interface in `Common` project.
- Build project.

```
using EuNet.Rpc;
using System.Threading.Tasks;

namespace Common
{
    // Inherit IRpc for Rpc
    public interface IGameCsRpc : IRpc
    {
        // Login Rpc
        Task<int> Login(string id);
    }
}
```

- In the `Server` project, register RpcService when creating a server .

```
_server.AddRpcService(new GameCsRpcServiceSession());
```

- In the `Server` project, `UserSession` class inherits from `IGameCsRpc` .

```
public partial class UserSession : IGameCsRpc
{
    public Task<int> Login(string id)
    {
        return Task.FromResult(0);
    }
}
```

- In the `Client` project, call Rpc.

```
// Rpc callable object
var rpc = new GameCsRpc(_client.Client, null, TimeSpan.FromSeconds(10));

// Call Rpc Login
var loginResult = await rpc.Login("MyId");
Debug.Log(loginResult);
```

How to use ViewRpc

ViewRpc is a technology that makes peer-to-peer communication between NetView Components as Rpc.
By adding a NetView Component to the GameObject, you can call functions of the same NetView Component (same ViewId) that exist on different clients.
For example, if you shoot a cannon from a red tank, the other user's red tank will also fire.
1:1 or 1:N call is possible, and in case of 1:N, return value can not be received.

IL2CPP issue (AOT)

Some platforms do not allow runtime code generation. Therefore, any managed code which depends upon just-in-time (JIT) compilation on the target device will fail. Instead, you need to compile all of the managed code ahead-of-time (AOT). Often, this distinction doesn't matter, but in a few specific cases, AOT platforms require additional consideration.

See more
https://docs.unity3d.com/2019.4/Documentation/Manual/ScriptingRestrictions.html

Serialization

There is a problem when serializing generic objects as AOT cannot generate code So, you need to provide a hint so that AOT can generate the code.

- Class for serialize (In `Common` project)

```
[NetDataObject]
public class DataClass
{
    public Tuple<int,string> TupleData;
    public Dictionary<int,string> DictionaryData;
}
```

- Hint function (In `Client` unity project)

```
private void UsedOnlyForAOTCodeGeneration()
{
    // Hints for using <int,string> in TupleFormatter<T,T>
    new TupleFormatter<int, string>();

    // Hints for using <int,string> in DictionaryFormatter<T,T>
    new DictionaryFormatter<int, string>();

    // Exception!
    throw new InvalidOperationException("This method is used for AOT code generat
}
```