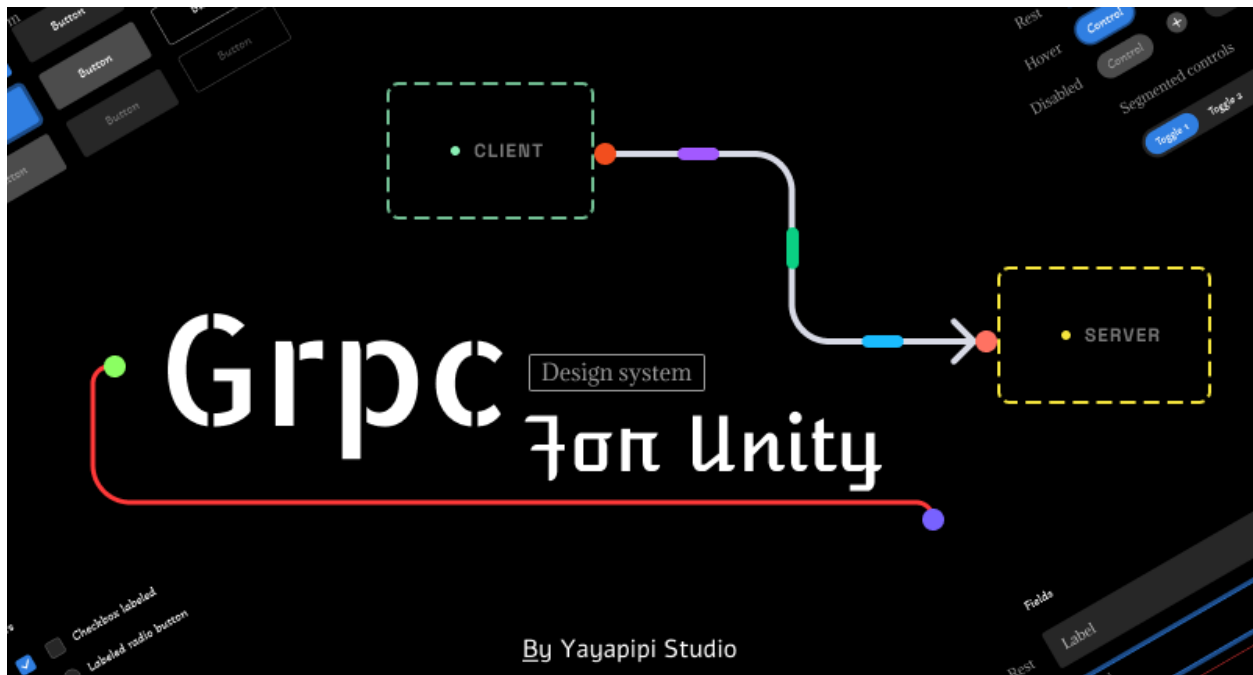




Grpc For Unity Documentation One Page

Hi, welcome to grpc for unity asset documentation.



Link: [Online Documentation](#) | [Discord](#)

▼ Grpc For Unity

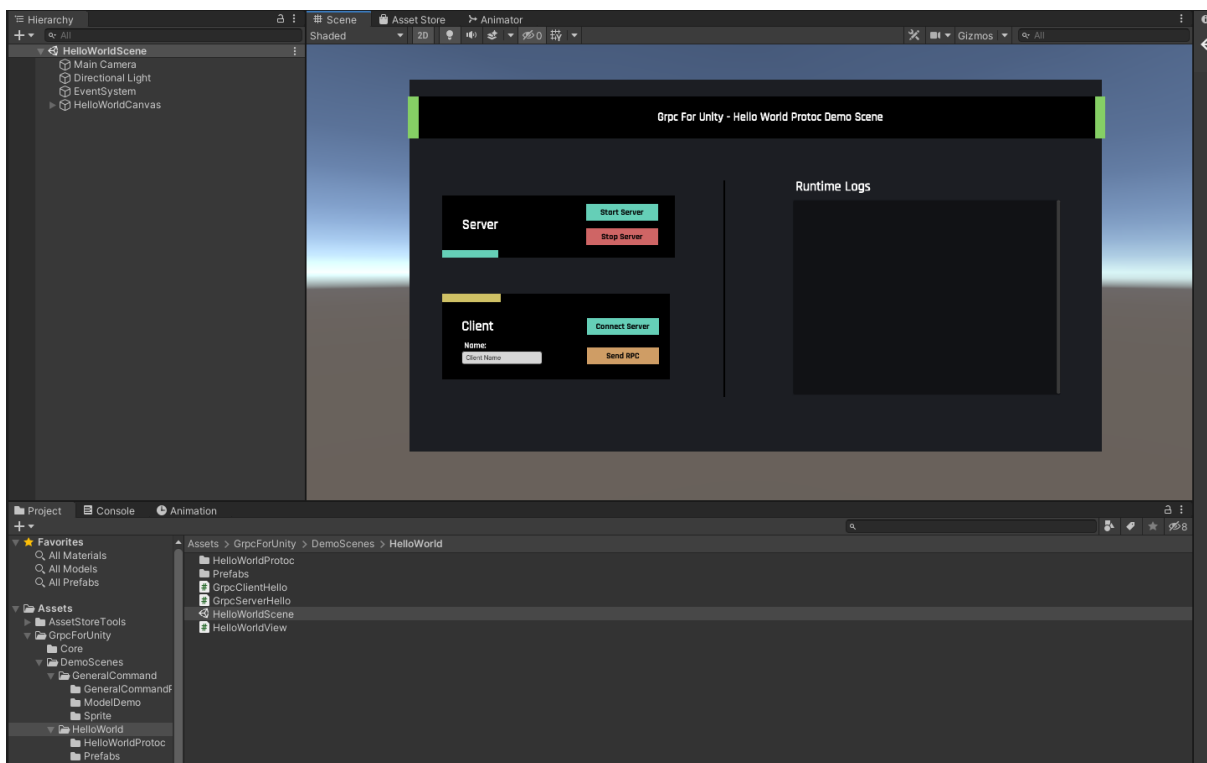
Grpc for unity is a asset store plugin to help you quickly implement grpc inside Unity, save lot of time to download and install plenty of complex dll enviroment. Grpc For Unity include the features below:

- Easy To Implement Grpc Server and Client
- Clean and Easy To Understand the Core Code

- No need to install Grpc Dll Enviroment (We do it for you!)
- Hello World Demo Scene Included
- More Advanced General Command Demo Scene Included
- Built-In One Click Grpc Protoc Generator Inside Unity Editor

Getting Start :

This page will help you understand how to use Grpc for unity step by step and all the script in inside the HelloWorld demoscene.



But Before Start:

1. Define your .proto file. For example:

```
syntax = "proto3";

package helloworld;
```

```

service Greeter{
  rpc SayHello(HelloRequest) returns (HelloReply){}
}

message HelloRequest{
  string name =1;
}

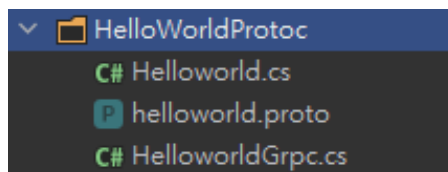
message HelloReply{
  string message=1;
}

```

For more information about protobuf, please check official website <https://developers.google.com/protocol-buffers/docs/csharputorial>.

2. Place your .proto file inside Unity folder, and use our built-in tool to compile protobuf class. For more info about protobuf compilation tool, please check [here](#).

It should contains the .proto file, and 2 generated c# class. Note We already pre-compile the example helloworld.proto class for you.



Grpc Server Base:

1. Create a new c# class and inherit GrpcServerBase.

```

public class GrpcServerHello : GrpcServerBase
{
    [0+1 usages] [yayapipi]
    public override List<ServerServiceDefinition> BindServerServices()
    {
        return new List<ServerServiceDefinition>
        {
            Greeter.BindService(new HelloWorldHandler()),
        };
    }
}

```

2. You need to bind the proto service with your custom class, for example:

```
public class HelloWorldHandler : Greeter.GreeterBase
{
    0+2 usages yayapi *
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        // Your Custom Function To-do when receive request
        RuntimeLogView.SpawnLogs(text: "Server Receive: " + request.Name);
        return Task.FromResult(new HelloReply { Message = "Hello " + request.Name });
    }
}
```

Grpc Client Base:

1. Create a new c# class and inherient GrpcClientBase.

```
public class GrpcClientHello : GrpcClientBase
{
    0+4 usages yayapi
    public override object SendGrpc<TRequest>(TRequest info)
    {
        var client = new Greeter.GreeterClient(GrpcClientChannel);
        var reply = client.SayHello(info as HelloRequest);
        return reply;
    }
}
```

SendGrpc() is the function that need to implement, which send the Rpc info to the server.

SayHello() - The function name you define in .proto file.

TRequest - The generic request you define in .protoc file which is the [Hello Request](#).

You are now ready to **GO**:

1. Create a new c# monobehavior script and declare the grpc variable:

```
private IGrpcServer GrpcServer = new GrpcServerHello();  
private IGrpcClient GrpcClient = new GrpcClientHello();
```

Then just use it, for example:

IGrpc Server API:

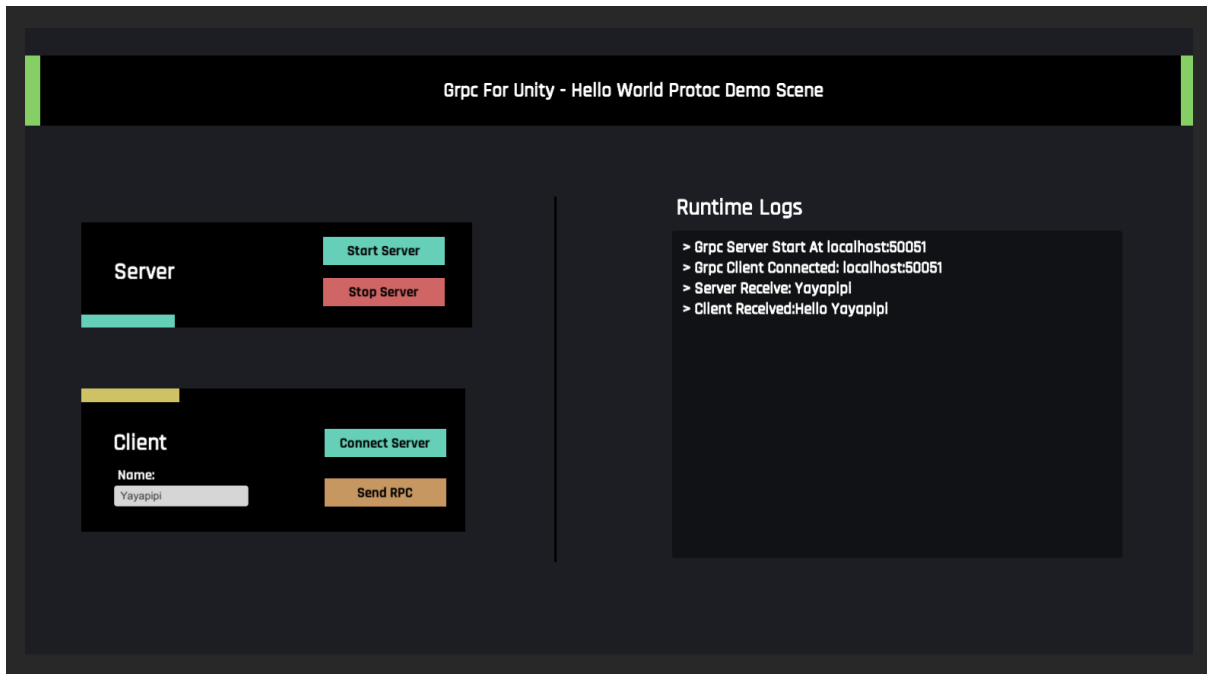
- Start Grpc Server: `GrpcServer.StartServer();`
- Stop Grpc Server: `GrpcServer.StopServer();`

IGrpc Client API:

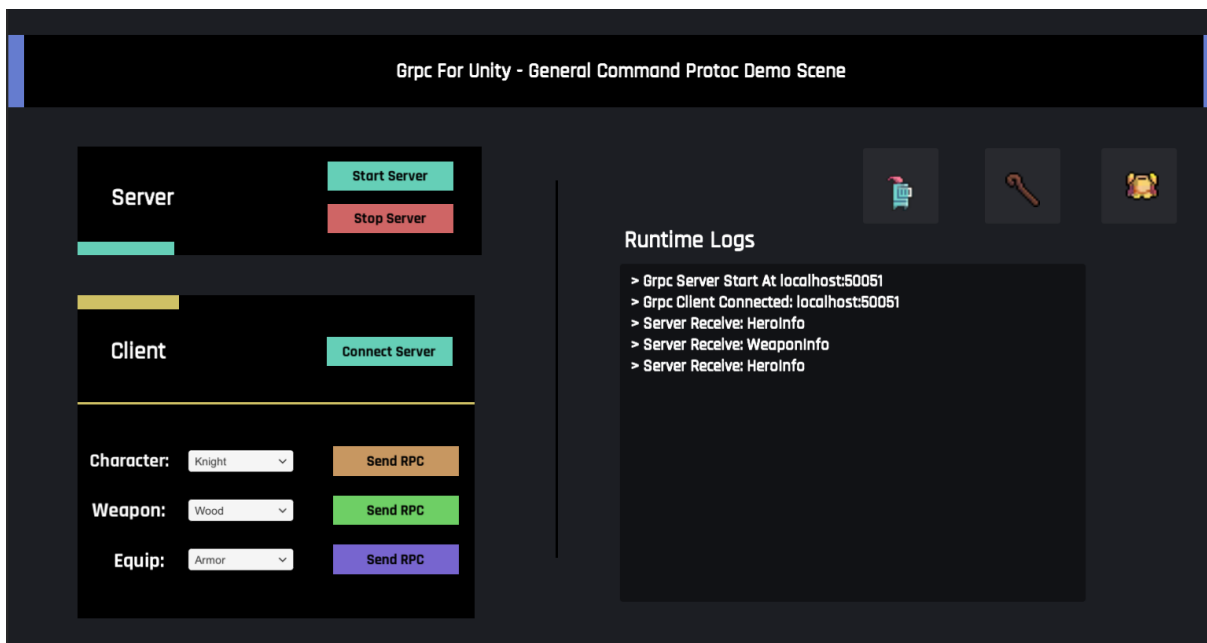
- Connect Grpc Server: `GrpcClient.ConnectServer(GrpcServer.ip, GrpcServer.port);`
- Send Rpc to Server:

```
private void SendHelloToServer()  
{  
    var replyObj = GrpcClient.SendGrpc(new HelloRequest()  
    {  
        Name = ClientName.text  
    });  
  
    if (replyObj is HelloReply reply)  
        Debug.Log("Client Received: " + reply.Message);  
}
```

Try the demo scene to check the result:



General Cmd Demo Scene:



We provide another demo scene (**GeneralCmdScene.unity**) which allow you to send different class and command to the server.

The General Command Proto:

```
syntax = "proto3";

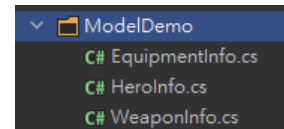
package GeneralCommand;

service CommandClass{
  rpc SendCommand(CmdRequest) returns (CmdReply){}
}

message CmdRequest{
  string CmdName =1;
  string CmdData =2;
}

message CmdReply{
  string CmdName =1;
  string CmdReplyData =2;
}
```

The core idea is passing the rpc through cmd name (class name) and cmd data (serialize with json). Then convert to any c# class and no need to define too many .proto function. For example, we define 3 different type of class structure and it allow to transfer by using Grpc.



Then we using the JsonUtility Tool to Send Different Rpc Data:

```
private void SendCharacterRpc()
{
    GrpcClient.SendGrpc(new CmdRequest
    {
        CmdName = nameof(HeroInfo),
        CmdData = JsonUtility.ToJson(new HeroInfo
        {
            Name = CharacterDropdown.options[CharacterDropdown.value].ToString(),
            Hp = 100,
            Type = (HeroType)CharacterDropdown.value
        })
    });
}
```

```
private void SendWeaponRpc()
{
    GrpcClient.SendGrpc(new CmdRequest
```

```

    {
        CmdName = nameof(WeaponInfo),
        CmdData = JsonUtility.ToJson(new WeaponInfo
        {
            Name = WeaponDropdown.options[WeaponDropdown.value].ToString(),
            Damage = 1000,
            Type = (WeaponType)WeaponDropdown.value
        })
    });
}

```

```

private void SendEquipRpc()
{
    GrpcClient.SendGrpc(new CmdRequest
    {
        CmdName = nameof(EquipmentInfo),
        CmdData = JsonUtility.ToJson(new EquipmentInfo
        {
            Name = EquipDropdown.options[EquipDropdown.value].ToString(),
            Value = 500,
            Type = (EquipmentType)EquipDropdown.value
        })
    });
}

```

Then we can convert back the json data to custom class when receive Rpc call.

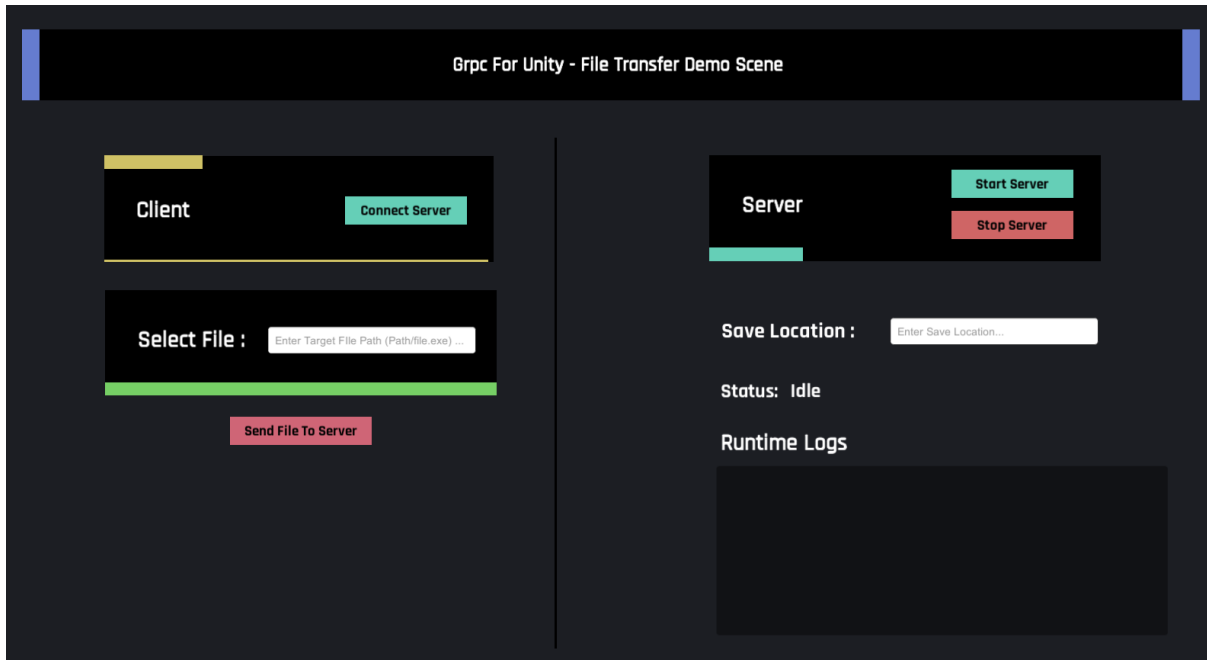
```

var heroInfo = JsonUtility.FromJson<HeroInfo>(data);
var weaponInfo = JsonUtility.FromJson<WeaponInfo>(data);
var equipmentInfo = JsonUtility.FromJson<EquipmentInfo>(data);

```



Local File Transfer Demo Scene:



We provide a demo file transfer scene, which allows you to transfer file locally from client to server.

Based on the Grpc Limitation, it only allows transfer 4Mb with each message call, but luckily, we can stream the large file through Grpc bi-direction stream protoc.

The File Transfer Command Proto:

```
syntax = "proto3";

package fileproto;

service FileTransfer{
  rpc SendFile(stream FileSendInfo) returns (stream FileReceivedReply){}
}

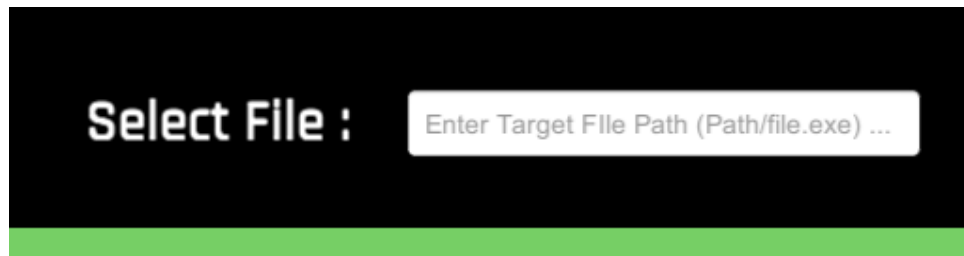
message FileSendInfo{
  bytes FileData = 1;
}

message FileReceivedReply{
  float Progress = 1;
}
```

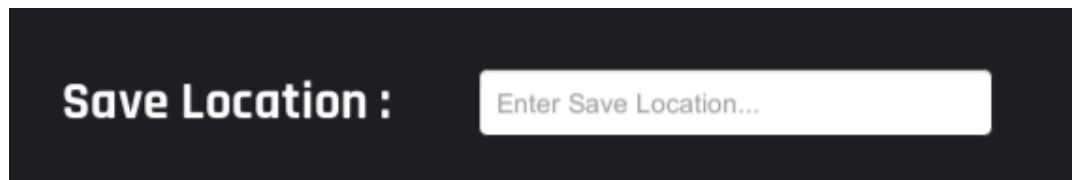
Note : We use bi-direction stream for client and stream to send file bytes[] and trace progress from server.

▼ How To Use:

1. Enter the Target File Location at the Select File Input Field; (Need to include file extension.)

A dark-themed UI element with the text "Select File :" in white. To its right is a white rectangular input field containing the placeholder text "Enter Target File Path (Path/file.exe) ...". A green horizontal bar is visible at the bottom of the dark area.

2. Enter the Save File Location.

A dark-themed UI element with the text "Save Location :" in white. To its right is a white rectangular input field containing the placeholder text "Enter Save Location...".

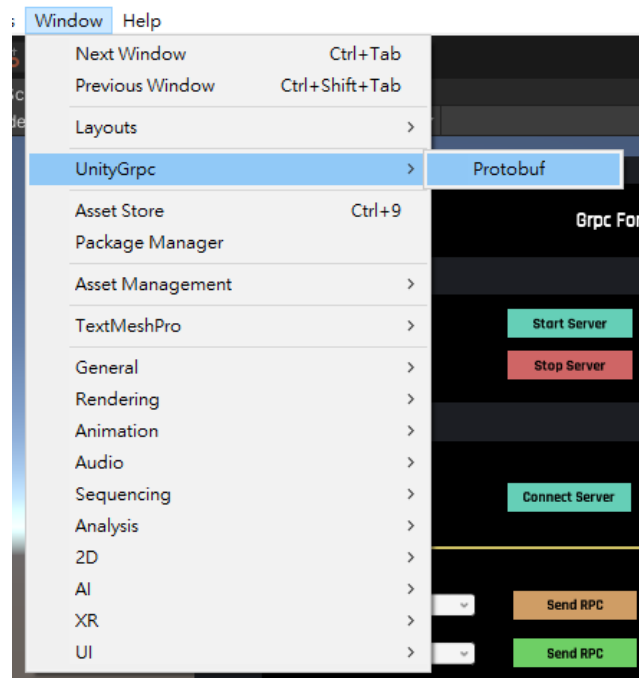
3. Start Grpc Server and Connect It.
4. Click Send File To Server!

Send File To Server

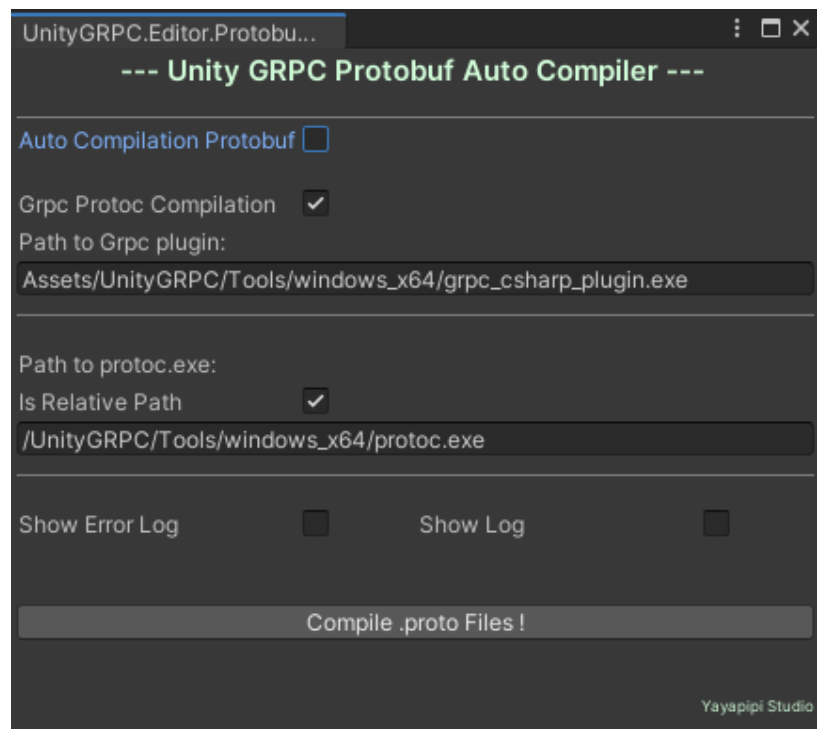


Built-in ProtoBuf Generator :

1. We have a built-in Grpc Protobuf generator under **Window/UnityGrpc/Protobuf**.



2. By clicking it, you will open the Protoc Generator Window Below:



How To Use:

1. Download The Grpc Protobuf Generator Tools:
 - a. Github Website <https://github.com/protocolbuffers/protobuf>
 - b. Or Our Pre-download Version
<https://drive.google.com/file/d/16beDi2EayAqhAGMWMXJg1eKJwvrqMeMP/view?usp=sharing>
2. Enter The Absolute Path For Grpc **Plugin.exe** and **Protoc.exe**
3. Put your .proto file inside the Unity Assets folder.
4. Click '**Compile .proto Files!**' Button.
5. It will scan all the .proto files inside Unity folder, and start generate the Grpc Class.
6. If **Auto Compilation Protobuf** is enabled, it will auto compile the .proto files.

? FAQ :

▼ Ui Not Show When Receive Rpc Call.

Since the ui only available operate in main thread, it is not the main thread when receive rpc call receive. To solve this, you can use the <https://github.com/PimDeWitte/UnityMainThreadDispatcher> to enqueue the function back to main thread. For example:

```
UnityMainThreadDispatcher.Instance().Enqueue(() =>
{
    _instance.WeaponImg.sprite = _instance.WeaponSprite[(int)weaponInfo.Type];
});
```