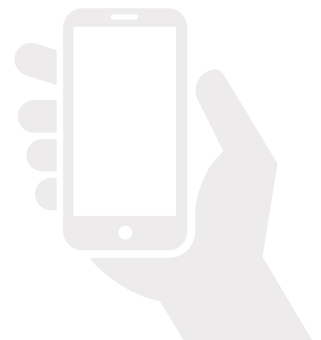




**VOICE  
CLIENT & SERVER  
SYSTEM  
FOR UNITY**

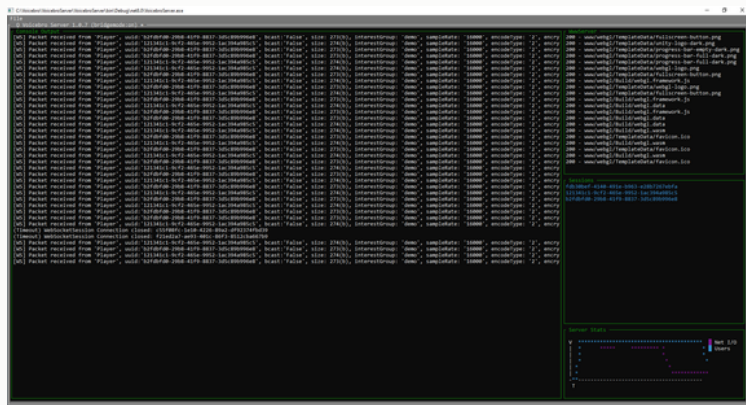


# TABLE OF CONTENTS

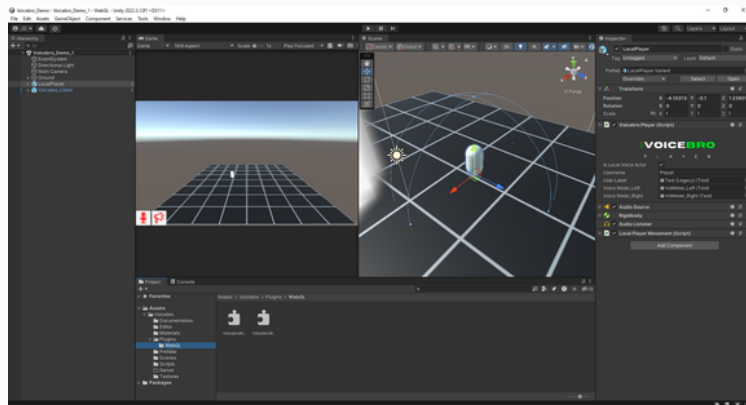
TABLE OF CONTENTS	1
SYSTEM OVERVIEW	2
SERVER	3-4
CLIENT	5-6
WEBGL & WEBSOCKETS	7
SSL CERTS	8
CUSTOM SERIALIZATIONS	9
FILES	10
FAQ	11

# SYSTEM OVERVIEW

Voicebro is a versatile voice solution for Unity projects that lets you run your own servers without strict licensing mechanisms, that can run on the internet and closed intranet systems. It supports client-side cryptography, WebGL and many other features.



Voicebro Server Console - Win build



Unity 2022.3.12f1 Demo Project

Voicebro consists of two main components:

- A) Server software (binaries & source code)
- B) Unity client plugin & scripts

All you have to do to get started is:

- 1) Import the Voicebro asset or UnityPackage into your Unity project
- 2) Open the demo scene provided with Voicebro
- 3) Click on the Voicebro\_Client object in the scene and set the hostname or IP address that the server is running on (default port for server is UDP 5056 for UDP mode, please adjust server firewalls as needed) -- or leave it set to our demo server for testing
- 4) Build the unity project OR hit play -- run on two or more machines or more than one instance of your build, editor, etc. to test multiuser functionality

NOTE: A demo server is provided to customers of this product at [virtualworldsystems.com](https://virtualworldsystems.com), but it is not guaranteed to be available at all times nor is it meant for heavy production usage. You will need to compile and run your own Voicebro server using the provided zipped visual studio 2022 project on a VPS or a physical server somewhere.

## Why run my own voice server?

Cost & savings - You will now be running your voice network for your game/project at the cost of a VPS, Droplet, or for intranet only projects you would run it on a server locally inside of the network. If you are still using third party providers for other networking in parallel to Voicebro, at least the cost of your voice services will be at cost and minimal

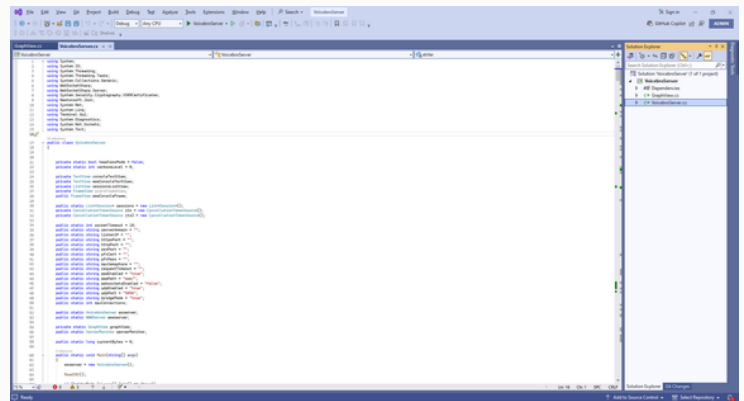
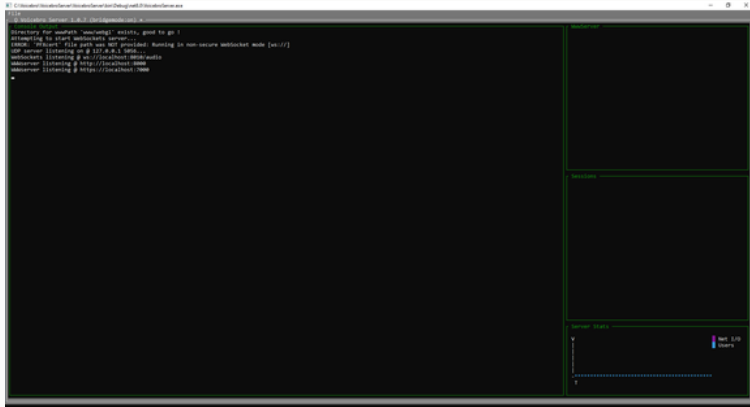
Encryption - Some projects in certain niche's of professional Unity project development such as Education, Healthcare, Government, etc require that encryption is strong and privacy is fully protected. Voicebro uses client side only cryptography, so only the users that know the same key/iv/interest group can successfully hear/speak to each other.

Compatibility & Ease of use - We have included plug 'n play functionality for Voicebro to make life easy for developers already or planning on building projects with Netcode, NetcodePlus, Mirror, Photon Fusion or PUN2. You can also make your own custom integration directly in the VoicebroPlayer script.

WebGL!?!? - Yep, Voicebro now supports WebGL by offering it's own custom Microphone and WebSockets JSLIB implementations that the client script works directly with to get the job done when running in WebGL builds. WebSockets mode also works in the Editor as well as all other build types. UDP mode is still great when you don't need WebGL builds, as it does not require you to configure any SSL certificate or anything.

# SERVER OVERVIEW

Voicebro has it's own server, and it needs to run somewhere such as a public VPS, or a physical computer inside your LAN. Your game client builds should all have the hostname/IP address & port of the same server plugged into the VoicebroClient script in your build/project.



To use your own server, you will need to compile the server yourself in Visual Studio 2022. Simply open the visual studio project after extracting the server source zip file outside of your Unity project.

The server source code is located inside the file  
Assets\Voicebro\Server\VoicebroServer\_Source.zip file

We have some command line flags and parameters that can be passed into then when launching:

Start in headless mode (text lines only, no text based GUI): VoicebroServer.exe -h

Set verbose level (value range is 0 to 2, 0 is default)  
Example with verbose level set to 1:  
VoicebroServer.exe -v=1

NOTE: Please only use verbose level 0 for production usage, level 1 will show details about each voice packet received, and level 2 shows retransmission debug info's and can get very spammy very quickly and affect performance negatively;

Both options at once, best choice for production usage: VoicebroServer.exe -h -v=0

NOTE: Performance will be dependent on the hardware and network abilities & limitations of the server itself, but because the server project is made and compiled directly in visual studio .NET as a C# cross-platform console app, the performance is actually quite decent with low overhead. It will be up to you to decide how and where you want to launch/shutdown copies of the voice server and how to point your game-client to them appropriately per scene, world, player group, etc. -- be it per port or via interest groups on one instance of the server. It usually makes sense to use 'interest groups' for the concept of 'rooms' when overall you have lower daily active user counts, or run multiple voice servers on different ports and/or hosts for larger daily active user counts.

When making your own builds of the VoicebroServer, here are some helpful PowerShell commands. After doing a clean/build etc, you can publish your compiled server binary and include all runtime files all into a single binary file, examples for windows & linux platforms:

```
dotnet publish -c Release -r win-x64 --self-contained -p:PublishSingleFile=true
dotnet publish -c Release -r linux-x64 --self-contained -p:PublishSingleFile=true
```

If you are running your VoicebroServer on Linux such as an Ubuntu VPS, Elastic Compute Cloud, Droplet, etc - you will need to install LibSSL and .NET Runtime SDK for the binary to launch properly:

```
sudo apt-get install -y dotnet-runtime-7.0
sudo apt-get install -y dotnet-runtime-8.0
(You can change the Framework version to 7.0/8.0/etc if you prefer in the VoicebroServer.csproj file directly)
```

LibSSL installation:  
sudo apt-get install libssl1.0.0

Make server binary file executable: chmod +x VoicebroServer

If you are looking for a good way to keep your Voicebro and other game servers running in the background and manage them on Linux, you might give 'tmux' a try, please take a look at [tmuxcheatsheet.com](https://tmuxcheatsheet.com)

```
tmux new-session -d -t 0
tmux attach -t 0
Ctrl+B+D to detach and keep things running in there
```

# SERVER CONFIG

The server project will need its environment setup

```
root@ubuntu:/voicebro/voicebroserver1.0.7# ls -l
total 63420
-rwxr-xr-x 1 root root 64927506 Jan 18 16:07 VoicebroServer
-rw-r--r-- 1 root root 319 Jan 18 19:29 config.ini
-rw-r--r-- 1 root root 6435 Jan 17 08:45 vws.pfx
```

Main (C:) > Voicebro > VoicebroServer > VoicebroServer > bin > Release > net5.0 > win-x64 > publish				
Name	Date modified	Type	Size	
www	1/19/2025 12:51 AM	File folder		
clrcompression	4/13/2022 9:42 PM	DLL File	754 KB	
cljit	4/13/2022 9:42 PM	DLL File	1,245 KB	
config	1/18/2025 4:33 PM	Configuration sett...	1 KB	
coreclr	4/13/2022 9:43 PM	DLL File	5,083 KB	
mscorlibcore	4/13/2022 9:43 PM	DLL File	1,035 KB	
VoicebroServer	1/19/2025 12:50 AM	Application	54,397 KB	
VoicebroServer	1/19/2025 12:50 AM	PDB File	19 KB	

The server project once built, will need a config.ini in the same directory you put the binary in, and it needs to explicitly contain certain entries, even if some are left blank -- without config.ini, the server will not startup correctly. Here is what the config.ini file should look like by default:

```
serverdomain=localhost
listenip=127.0.0.1
httpsport=7000
httpport=8000
wssport=8010
pfxcert=
pfxpass=
maxsemaphore=15
requesttimeout=15
wwwenabled=true
wwwpath=www/webgl
udpenabled=true
udpport=5056
websocketsenabled=true
bridgemode=true
```

Let's go over all of the config.ini options...

serverdomain (string hostname) -- The Fully Qualified Domain Name of the server, this is mostly used for WebGL, if you are testing locally, a value of 'localhost' without the quotes is perfect; If you are running the voice server in 'Secure WS' mode on a public server, you need to set this to the domainname value that the SSL certificate is for -- which should also match the domain you will be serving the WebGL applet off of.

listenip (ipv4 address)-- The IP we want to bind our listeners to for UDP and WebSockets mode, 127.0.0.1 works great for local testing, but otherwise needs to be the public ipv4 address of the server it is running on

httpport (int) / httpsport (int) -- This is the HTTP and HTTPS port the 'built in webserver' can use for local testing purposes only, but please do not use the built in web server for production purposes

wssport (int) -- This is the WebSockets port we want to listen on, please leave this at its default value of 8010 for getting started

pfxcert (string path)-- The full filepath to your SSL certificate that matches the domain name set in 'serverdomain' field value; It cannot be a CRT file, it must be converted to a PFX format; It can be a filename that is expected to be in the same folder as the compiled server binary, OR a full file path disk:\folder\file.pfx etc

pfxpass (string) -- The password set on the PFX file when converted/generated, if no password was set when converting it, simply leave this field blank

maxsemaphore (int) -- The maximum number of threads the wwwserver can use at any given time

requesttimeout (int) -- This is for wwwserver, the amount of seconds before a request is given up on if not finished sending

wwwenabled (true/false) -- this enables the local web server for testing purposes only; when enabled, it expects a folder to exist at 'wwwpath', and when it exists and this option is enabled, it will easily serve files out of it on port 8000 (http), which comes in quite handy for local/dev testing when building WebGL game applets. You can supply a direct path to a folder for wwwPath, OR a subfolder path, ex: wwwpath=www/webgl is expecting a folder named www inside the folder where our server binary is, and another sub folder named 'webgl' inside of that -- this is where it will serve our built Unity WebGL applet from -- and with this config, the URL would be http://localhost:8000/webgl -- **Please use a heavy duty web server in your actual production environment for serving your WebGL applet and webpage files, such as Apache or NGINX, not this applet-testing wwwserver -- please set this value to false for production environment !**

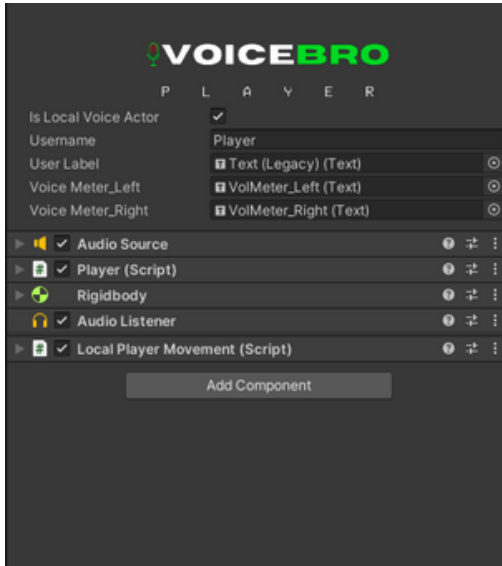
udpenabled (true/false) / udpport (int) -- Enable or disable UDP mode, and the port to use for that mode, please leave at default value when possible

websocketsenabled (true/false) -- Enable or disable WebSocket mode

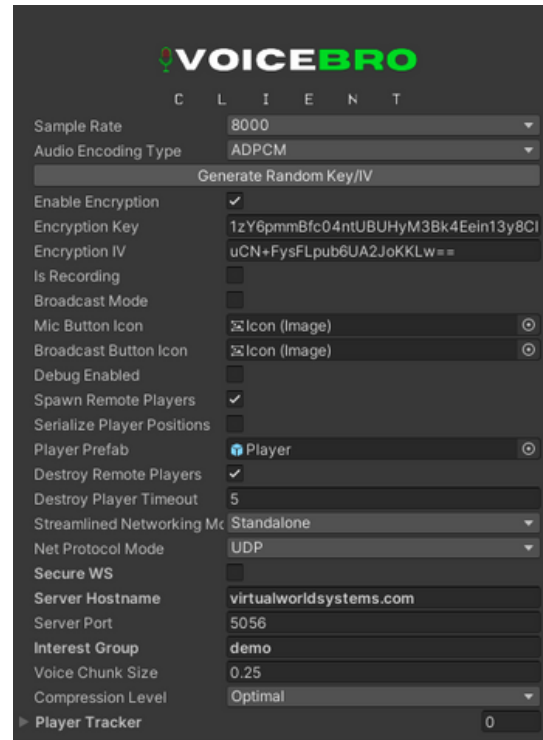
bridgemode (true/false) -- When set to true, the server will relay packets/connections and serialize players across both UDP and WebSockets mode for use cases that might have more than one build type where one type might prefer to be in UDP mode and additionally has a WebGL client or other need to also offer WebSockets mode

# CLIENT

The Voicebro client portion runs inside your Unity project. It consists of two main scripts: VoicebroClient.cs and VoicebroPlayer.cs; There are also two editor scripts, as well as the VoicebroCore.dll cross platform plugin.



Voicebro PLAYER script



Voicebro CLIENT script

## The VoicebroPlayer script

Each script contains editor tooltips that explain what each value is used for:

**isLocalPlayer:** This value needs to be set to true before the script is enabled/activated/started on the local player prefab at runtime, and disabled on the spawnable player prefab itself

**username/uuid:** For visual debug purposes only, do not change at runtime; To change the local users name at runtime, please set it via `VoicebroClient.instance.username` via your other scripts before connecting.

**UserLabel:** Slot in a Text component if you want to show usernames of each player serialized thru Voicebro on it

**VoiceMeter\_Left:** Slot in a Text component if you want to use them for displaying voice level meters of each player (left)

**VoiceMeter\_Right:** Slot in a Text component if you want to use them for displaying voice level meters of each player (right)

## Overview

This script goes on your player prefab, `isLocalPlayer` must be set to true when it is spawned or ahead of time if already placed in the scene, and before the VoicebroPlayer script is activated, etc. It should be set to false by default in your prefab directly. The provided demo-scene has and is expecting the LocalPlayer prefab variant directly in the scene with it checked. Voicebro does not provide a demo option for spawning/destroying the *local* player, only *remote* players.

You should be spawning your players via your games backend/server-client system or cloud networking service directly for complete control of all the various serializations your players will need still, This is only a voice system. Extra serialization's sent within voice chunks only show up once per interval which is every half a second by default (`voiceChunkSize`), please see the 'Custom Serializations' section for more information on how to serialize more variables per player via voice packets directly

## The VoicebroClient script

The most important thing we need to understand about it is that there should only be one object in the scene with the VoicebroClient script on it, but that object+script should exist once in any scene that you want the users to have voice communications working in. Feel free to disable this object in the scene by default and enable the gameObject or script when you want the local user to actually try to connect to the voice server initially.

Please also make sure your encryption key/iv matches for all users connecting to the same server/port across all clients, as well as the compression setting. Interest groups can be changed on the fly to 'change rooms'

### Script properties

Each script contains editor tooltips that explain what each value is used for:

sampleRate: The sampling rate the local player encodes their mic data at, can be unique per player and changed on demand from other scripts via VoicebroClient.instance.sampleRate

AudioEncodingType: Choose from PCM (Uncompressed),  $\mu$ -law, or ADPCM compression to control bandwidth usage

enableEncryption: When enabled, voice packets are encrypted & decrypted on the client side -- requires a KEY/IV to be generated using the provided button when enabled & the key needs to be the same by all receiving parties

encryptionKey: Encryption Key -- please generate with button above, and ensure the key&iv match on all connected clients

encryptionIV: Encryption Initialization Vector -- please generate with button above, and ensure the key&iv match on all connected clients

isRecording: When true, the local player will transmit audio from the systems default microphone/input device. This value can easily be changed via VoicebroClient.instance.isRecording via your own scripts

broadcastMode: When enabled, the local player will be heard by all other players in the interest group a.k.a. 'channel', regardless of their world position (SpatialBlend = 2D), when disabled the local player will only be heard when someone is near them.

MicButtonIcon: Please slot in your UI button's image for the Mic/Mute button (optional)

BroadcastButtonIcon: Please slot in your UI button's image for the Broadcast button (optional)

debugEnabled: When enabled, verbose debug entries will be created in the console

spawnRemotePlayers: When enabled, Voicebro client will spawn remote players when voice packets are received without a player gameObject already existing in the Scene & PlayerTracker; This does NOT spawn the localPlayer -- it needs to be put in the scene manually when this mode is enabled

serializePlayerPositions: When enabled, remote player positions will be serialized through Voicebro (once every interval, based on voiceChunk size)

playerPrefab: The player prefab to use when spawnRemotePlayers is enabled

destroyRemotePlayers: Destroy remote players when they have timed out, based on the Voicebro system (for use with spawnRemotePlayers enabled)

destroyPlayerTimeout: The timeout (in seconds) that Voicebro decides a player has left/stopped talking to the server for use when destroyRemotePlayers is enabled

NetProtocolMode: UDP or WebSockets, generally UDP mode is better for larger number of voices/players, but WebSockets mode is required for WebGL builds. If you try to build your game for WebGL with Voicebro and UDP mode enabled, the build will fail, and you must switch this mode to WebSockets and get that setup. See 'WebSockets & WebGL' section for more info on this topic.

SecureWS: When enabled, if NetProtocolMode is set to WebSockets, this will cause the client to use the wss:// protocol, when disabled it will use the ws:// non-secure protocol, which is only good for working with 'localhost'

serverHostname: The hostname or IP address of the remote voice server

serverPort: The remote port of the voice server

interestGroup: The voice 'channel' that the local player should belong to, can be changed on the fly at runtime -- make this unique for each scene or 'room' in your project

voiceChunkSize: The size (in seconds) of each voice chunk/frame -- recommended value is 0.2 to 0.5

compressionLevel: Compression level of voice data -- must be consistent for all players in interest group/server

### Streamlined Networking Mode property

If you are using a common player networking system and want to run Voicebro in parallel with it, you can import the SDK for that system into your project first and then select it from the StreamlinedNetworkingMode dropdown on your VoicebroClient script in the scene. This will set a custom scripting symbol in your project that tells Voicebro how to serialize the Voice Actor's UUID and username across all game clients so it can cross reference them to the actual player prefabs in the scene at runtime and know who's AudioSource to spit out what signals thru. If you have this setting set to the default 'Standalone' mode, that is meant for when you want Voicebro to spawn/destroy/serialize player prefabs based on voice packets -- in standalone mode you need to make sure searchForRemotePlayers, spawnRemotePlayers, SerializePlayerPositions, and destroyRemotePlayers are all set to **true**, and in any other mode they are all set to **false**.

NOTE: Voicebro does not tunnel voice data through these other networking systems or providers, this simply allows it to work seamlessly with other player networking systems your project might already be or planning to use. Obviously, some of these are not good for offline/intranet only projects, but others are like Netcode/NetcodePlus.

Please take notice of the empty #ifdef blocks in VoicebroPlayer.cs titled 'VOICEBRO\_CUSTOM', these are left intentionally blank with a comment in each one that shows you exactly where you would need to add code similar to each's neighbor, to make your own custom connector for any other player networking system we did not provide one for.



# WEBSOCKETS & WEBGL

If you want to build WebGL applets in Unity that use Voicebro, you will need to setup the server to allow WebSockets mode, but you will also need to configure your VoicebroClient in your Unity WebGL project accordingly too. Here are some examples

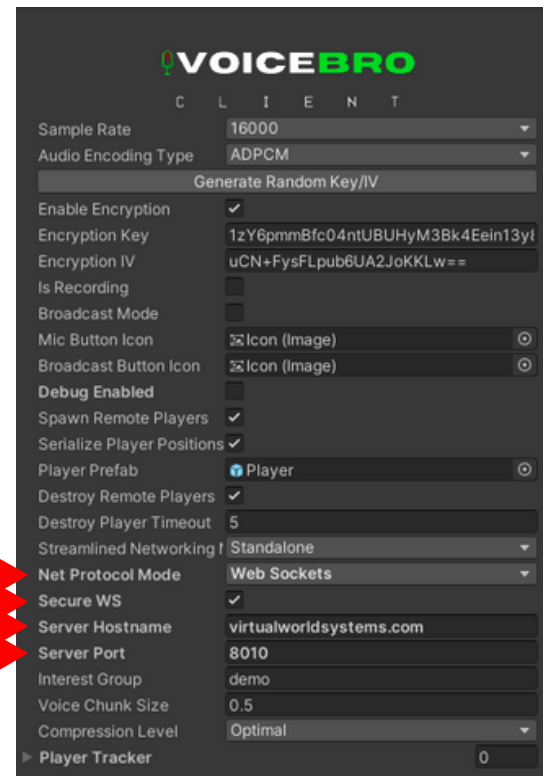
This is what the config.ini and VoicebroClient settings should look like for running on the actual production webserver, obviously with your server's details instead. You will need to build your WebGL game, then upload it to your webserver -- I usually zip it and send it over then just extract it in it's place where Apache serves it out of.

Notice that SecureWS is enabled on the VoicebroClient, this because we are using a PFXCert (SSL cert) on the server and so Secure WS mode needs to be enabled in the Unity WebGL build too.

Notice wwwenabled is set to false, this is because here we are serving our WebGL applets folder via Apache2 for production, so we don't need the built-in applet-testing webserver to be enabled. If it even matters, we could also disable UDP mode here too, if we only had a WebGL build for our game then we really would have no need for the UDP server to be enabled, nor would we need bridgemode enabled -- but each use case / scenario will be unique so the options are here and available to customize. With that being said, if it is needed, bridge mode enabled is still totally fine for production use.

```
serverdomain=virtualworldsystems.com
listenip=123.45.67.89
httpsport=7000
httpport=8000
wssport=8010
pfxcert=/sslfiles/vws.pfx
pfxpass=
maxsemaphore=15
requesttimeout=15
wwwenabled=false
wwwpath=
udpenabled=true
udpport=5056
websocketsenabled=true
bridgemode=true
```

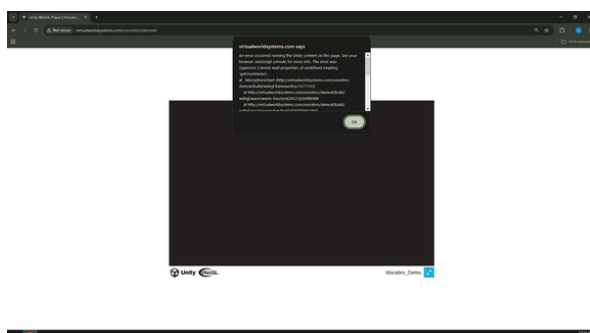
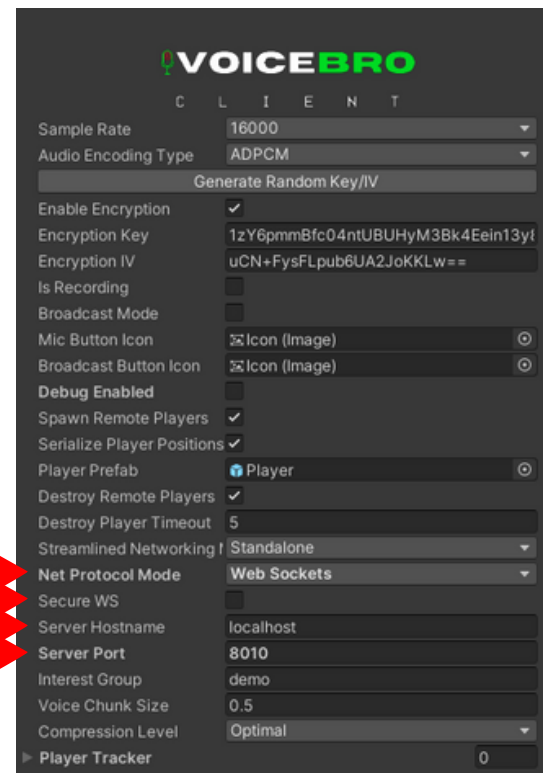
Server config.ini example  
(production)



This is what the config.ini and VoicebroClient settings should look like by default, for running locally. Take note, that when using WebSockets without a SSL certificate (PFXCert), only the hostname 'localhost' will allow actual data to transmit thru the WebSockets connection. If you use any other hostname, all browsers require you to use an SSL certificate that matches the domain name the WebGL applet is served off of, in order to send actual messages thru the WebSockets. However, for 'localhost', this works in your local browser without a SSL cert, but you must use the HTTP (not HTTPS) url to the test webserver, ex: http://localhost:8000/ .. and SecureWS must be turned off on the VoicebroClient in the WebGL build -- then you'll be serving your WebGL build out of the www/ folder -- so build it and put in in there!

```
serverdomain=localhost
listenip=127.0.0.1
httpsport=7000
httpport=8000
wssport=8010
pfxcert=
pfxpass=
maxsemaphore=15
requesttimeout=15
wwwenabled=true
wwwpath=www
udpenabled=true
udpport=5056
websocketsenabled=true
bridgemode=true
```

Server config.ini example  
(for local testing)



A screenshot of a JS alert box error message that is seen when trying to load the Unity WebGL applet over a HTTP connection instead of an HTTPS version of the URL, when the SecureWS option was enabled on the VoicebroClient in the scene when making the Unity WebGL build.



# SSL CERTS

When using Secure WebSockets with VoicebroServer, you will need to convert your .CRT certificate to a .PFX file; PFX is the format required by the server .NET project, and .CRT is what is usually given and installed on a production webserver running Linux such as Apache or Nginx, so we usually need to convert it for our Voicebro WebSocket server's purposes.

```
# Our ssl cert files that our production webserver is already using (Apache, NGINX, etc) for example only - 'mysite.com'
```

```
root@localhost:/ssls/mysite# ls
mysite_com.ca-bundle  mysite_com.crt  mysite.csr  mysite.key
```

```
# Need to install OpenSSL first if you don't already have it on your server, this example is for Ubuntu Linux
```

```
root@localhost:/ssls/mysite# sudo apt update
root@localhost:/ssls/mysite# sudo apt install openssl
```

```
# Use OpenSSL to convert our .crt file to a .pfx file for use with our compiled VoicebroServer for linux
```

```
root@localhost:/ssls/mysite# openssl pkcs12 -export -out mynewcert.pfx -inkey vws.key -in mysite_com.crt -certfile mysite_com.ca-bundle
```

```
Enter Export Password: (I left it blank)
Verifying - Enter Export Password:
```

```
# Now we have our PFX cert file ready for use with Voicebro
```

```
root@localhost:/ssls/mysite# ls
mysite_com.ca-bundle  mysite_com.crt  mysite.csr  mysite.key  mynewcert.pfx
```

```
# Now that we have our pfx file ready, we can point pfxcert value in our VoicebroServer's config.ini file directly to it; Notice I left # pfxpass blank, because I left it blank when converting the CRT file to a PFX file using openssl:
```

```
root@localhost:/voicebroserver1.0.7# pico config.ini
```

```
serverdomain=mysite.com
listenip=123.45.67.89
httpsport=7000
httpport=8000
wssport=8010
pfxcert=/ssls/mysite/mynewcert.pfx
pfxpass=
maxsemaphore=15
requesttimeout=15
wwwenabled=false
wwwpath=
udpenabled=true
udpport=5056
websocketsenabled=true
bridgemode=true
```

```
# If all is setup correctly, when we startup the server, it will show our WebSocketServer status, the part highlighted in green below shows # the PFX was loaded in correctly, it will show errors instead if are any issues occur while loading the PFX cert:
```

```
root@localhost:/voicebroserver1.0.7# ./VoicebroServer -h -v=0
```

```
Server Domain: mysite.com
Listen IP: 123.45.67.89
WWW Enabled: false
UDP Enabled: true
UDP Port: 5056
WebSockets Enabled: true
WSS Port: 8010
PFX Cert: /ssls/mysite/mynewcert.pfx
PFX Pass:
Max Semaphore: 15
Request Timeout: 15
_ Voicebro Server 1.0.7 (bridgemode:on) _
```

```
Attempting to start WebSockets server...
```

```
WebSocketServer SSL PFX: /ssls/mysite/mynewcert.pfx
WebSockets listening @ wss://mysite.com:8010/audio
UDP server listening on @ 123.45.67.89 5056...
WebSocket connection opened: 7d7857c2c44f41418df98ed6b8bbef97
WebSocket connection opened: 17a23d8d292e487f843881e10cebef42
```

# CUSTOM SERIALIZATIONS

You might of noticed, as of version 1.0.7, a new script named 'Player' is provided, and included on both the LocalPlayer and Player prefabs now. This script is meant to be modified, and it allows the developer using it (you) to serialize a custom set of variables from within this script only, directly thru the voice packets, sometimes allowing us to not use any other networking besides Voicebro, which very much comes in handy when with simple WebGL/Metaverse related projects.

All we have to do to start serializing a new variable for each player local & remote, is add the variable to the top of this script, marked it using the [SerializableVariable] tag, and setup what to do with the data when its received for each remote player in ReceiveData(), and how to refresh the data for the local player right before we send out each serialization in GetPackedData()

By default, this script is doing extra serializations for position, rotation, and animator state (if animator exists)

The reason we do position in here (and have SerializePlayerPositions disabled on our VoicebroClient in the scene) is so we can lerp from position to position, as when using SerializePlayerPositions its a direct change from old to new, which only occurs once every 0.5s or so, depending on the VoiceChunk size being used. So we have it disabled, and do lerping here so remote player movements look more realistic.

We serialize the players rotation here too, because Voicebro doesn't serialize player rotations, so this is a nice extra to have in here.

Feel free to add your own !

ExtraData serializations are received for RP's via PlayReceivedAudio() in VoicebroClient.cs  
ExtraData serializations are sent for LP via Update() in VoicebroPlayer.cs

```
using UnityEngine;
using Voicebro;
[SerializableVariable] [0] 6 references
public class Player : VariableSerializer
{
    // Here is where you can add more custom variables to serialize, anything marked with [SerializableVariable] in this script on each player will get serialized
    // This script makes use of the extraData slot on the AudioData packets, this was introduced in Voicebro v1.0.7 release

    [SerializableVariable] public int AnimState;
    [SerializableVariable] private Quaternion playerRotation;

    // Make sure SerializePlayerPositions on the VoicebroClient object/script in the scene is set to
    // false -- since we are doing custom player position serializations thru here via extraData binary packets
    [SerializableVariable] private Vector3 playerPosition;

    3 references
    public string myData { get; set; } = ""; //do not modify, this is where out binary data is stored for each serialization
    @ Unity Message 1 0 references
    public void Update()
    {
        // This allows us to 'lerp' each players new position for smooth movement, but we only need to do this for remote players
        if (GetComponent<VoicebroPlayer>() != null)
        {
            if (GetComponent<VoicebroPlayer>().IsLocalVoiceActor == false)
            {
                transform.position = Vector3.Lerp(transform.position, newTransformPosition, LerpTime);
            }
        }

        public float LerpTime = 0.25f;
        private Vector3 newTransformPosition;
        1 reference
        public void ReceiveData()
        {
            UnpackSerializableVariables(myData);

            // Below here, is where you can 'do something' with received values for each serialized variable defined at the top of this script, when they are
            // received for each remote player

            transform.rotation = playerRotation;
            newTransformPosition = playerPosition;

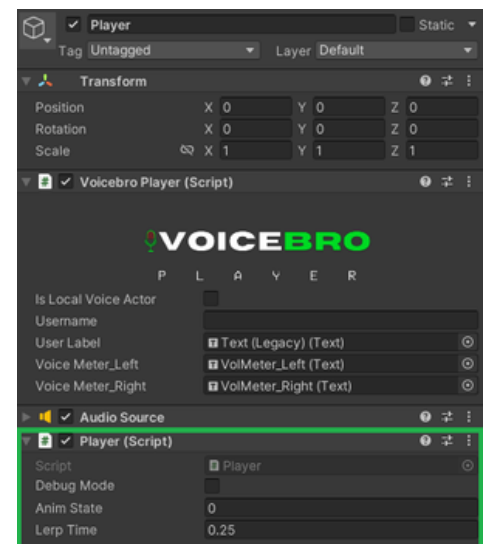
            if (GetComponentInChildren<Animator>() != null)
            {
                GetComponentInChildren<Animator>().SetInteger("AnimState", this.AnimState);
            }
        }

        1 reference
        public string GetPackedData()
        {
            // Here is how we 'grab' the data for each thing we are serializing from each local player, from various places, which
            // occurs each time before we serialize out the local players variables for use with extraData

            playerRotation = transform.rotation;
            playerPosition = transform.position;

            if (GetComponentInChildren<Animator>() != null)
            {
                this.AnimState = GetComponentInChildren<Animator>().GetInteger("AnimState");
            }

            string packed = PackSerializableVariables();
            return packed;
        }
    }
}
```



# FILES

Description of each script/file in the Voicebro UnityPackage

Documentation/Voicebro Documentation 1.0.7.pdf -- this manual

Editor/VoicebroClientEditor.cs -- Unity editor script that makes VoicebroClient.cs look nice in the inspector  
Editor/VoicebroPlayerEditor.cs -- Unity editor script that makes VoicebroPlayer.cs look nice in the inspector

Materials/Grid.mat -- floor material  
Materials/PlayerCapsule.mat -- player capsule material

Plugins/VoicebroCore.dll -- Cross platform library that contains mostly encoding/decoding and encryption/decryption functions, will be included within builds

Plugins/WebGL/VoicebroMicrophone.jslib -- custom logic that works directly with VoicebroMicrophone.cs to allow usage of the microphones in WebGL  
Plugins/WebGL/VoicebroWebSockets.jslib -- custom WebGL logic for WebSockets

Prefabs/LocalPlayer Variant.prefab -- the local player prefab that gets put directly in the scene, when using 'Standalone Mode' (built in basic serializations thru Voicebro packets)  
Prefabs/Player.prefab -- the remote player prefab that get's slotted into VoicebroPlayer client in the scene  
Prefabs/Voicebro\_Client.prefab -- VoicebroClient to put in 'live scene' where voice is to occur, with it's default values

Scenes/Voicebro\_Demo\_1.unity -- the default demo scene for Voicebro

Scripts/CustomErrorLogger.cs - a script that shows all runtime errors as an overlay using GUI, handy for debugging WebGL builds  
Scripts/LocalPlayerMovement.cs -- basic local player script that allows player movement via mouse and vertical/horizontal inputs  
Scripts/Player.cs - script that can (optionally) be added to your local + remote player prefabs, that allows for extra serializations per player  
Scripts/VariableSerializer.cs - the core logic for use with Player.cs  
Scripts/VoicebroMicrophone.cs -- script that lets Unity C# talk to VoicebroMicrophone.jslib  
Scripts/VoicebroClient.cs -- this should be on an object in any scene you want voice to work in, where live players are  
Scripts/VoicebroPlayer.cs -- this script goes on player prefabs at the root, +AudioSource, +AudioListener for LocalPlayer only  
Scripts/VoicebroUDPClient.cs -- the UDP mode script for client connection  
Scripts/VoicebroWebSocketClient.cs -- the unity c# script that talks to the custom jslib VoicebroWebSockets.jslib script in WebGL builds

Server/VoicebroServer\_Source.zip -- The C# .NET project sourcecode for the Voicebro Server -- please extract this outside of the Unity project, and open the VS Solution file with Visual Studio 2022 w/ admin privs to get started

Textures/Grid.png -- used for the floor material in demo scene  
Textures/Microphone.png -- used for the mic on/off toggle UI button  
Textures/Megaphone.png -- used for the 'broadcast' on/off toggle UI button

# FAQ

## Frequently Asked Questions

Q: How many copies of the server am I allowed to run?

A: As many as you want, as many that your servers/resources can handle, there is no strict or enforced license system or subscriptions with Voicebro

Q: How many users can be in a 'voice room' or 'interest group' at once?

A: There is no hard limit, if you have a large amount of users, you may want to consider adding in a hard-limit in your game logic directly or use player-distance culling where you destroy players that are too far away. The answer is 'whatever the servers AND each players hardware can handle'

Q: From my own unity C# script, how can I make the local player muted/unmuted?

A: Add the 'using Voicebro;' directive at top of script, then do:  
VoicebroClient.instance.isRecording = true; //or false

Q: From my own unity C# script, how can I make the local player toggle broadcasting mode?:

A: VoicebroClient.instance.broadcastMode = true; //or false

Q: My online game project doesn't have the local player already in the scene, it gets spawned at runtime by some other networking system at runtime, how do I handle this?

A: Then you don't want the 'Standalone' option for the StreamlinedNetworkMode value on your VoicebroClient script in the scene, but rather it needs to match whatever system that is, and if it isn't listed you will need to make your own custom connector in the VoicebroPlayer.cs script's ifdef sections marked 'VOICEBRO\_CUSTOM'

Q: Can this voice system be used for projects with strict compliance requirements?

A: It supports custom encryption, closed system / offline / intranet usage, no license checks or phoning home to a third party server ever, and the only code that could be considered 'hidden' is inside VoicebroCore.dll, which can still be fully inspected via tools like ILSpy or DotPeek, it is not obfuscated -- you can even move the code from this directly into your unity project if you really want, but I think performance of this code is possibly better in an external library. With all that being said I think so yes, but please check your projects exact requirements yourself to be sure.

Q: Do I need to open/forward ports on my VPS, Droplet, ECC etc for VoicebroServer?

A: Yes, whatever ports (defaults are 5056 for UDP mode and 8081 for WebSockets mode) your VoicebroServer listens on need to be opened on a VPS/Droplet/ECC, or unblocked on firewall and port forwarding setup if the server device is behind a router.

Q: Will this voice system work with WebGL + the 3rd party networking system listed below?

A: Yes! if any of the StreamlinedNetworkMode providers (for example, Photon Fusion) allow their system to work within WebGL builds by utilizing WebSockets, for those yes Voicebro will still work in parallel with them by using the StreamlinedNetworkMode presets.

Q: With the various options for StreamlinedNetworkMode, what components from each networking SDK does VoicebroPlayer expect to be on the same transform as it (root of player prefab) at runtime for each mode? Can you also give extended info about each mode as well?

A: Yes! See below:

Version	Scripting Symbol	VoicebroPlayer inherits from	Sister Transform Component	Unity Versions Tested+Pass
2.0.3	✓ VOICEBRO_FUSION	NetworkObject	NetworkObject	2022.3.12f1, 6000.0.12f1
2.46	✓ VOICEBRO_PUN2	MonoBehaviour, IPunObservable	PhotonView	2022.3.12f1, 6000.0.12f1
89.8.0	✓ VOICEBRO_MIRROR	NetworkBehaviour	NetworkIdentity	2022.3.12f1, 6000.0.12f1
	VOICEBRO_NETCODE	NetworkObject	NetworkObject	-----
NGO 1.4.0	✓ -----	-----	-----	2022.3.12f1
NGO 1.9.1	✓ -----	-----	-----	6000.0.12f1
	VOICEBRO_NETCODEPLUS	SNetworkPlayer	SNetworkPlayer+SNetworkObject	-----
1.0.3	✓ -----	-----	-----	2022.3.12f1
NGO 1.6.0	✓ -----	-----	-----	6000.0.12f1
NGO 1.9.1	✓ -----	-----	-----	6000.0.12f1
	VOICEBRO_STANDALONE	MonoBehaviour	N/A	2022.3.12f1, 2021.3.12f1,
1.0.7	✓ -----	-----	---	6000.0.12f1

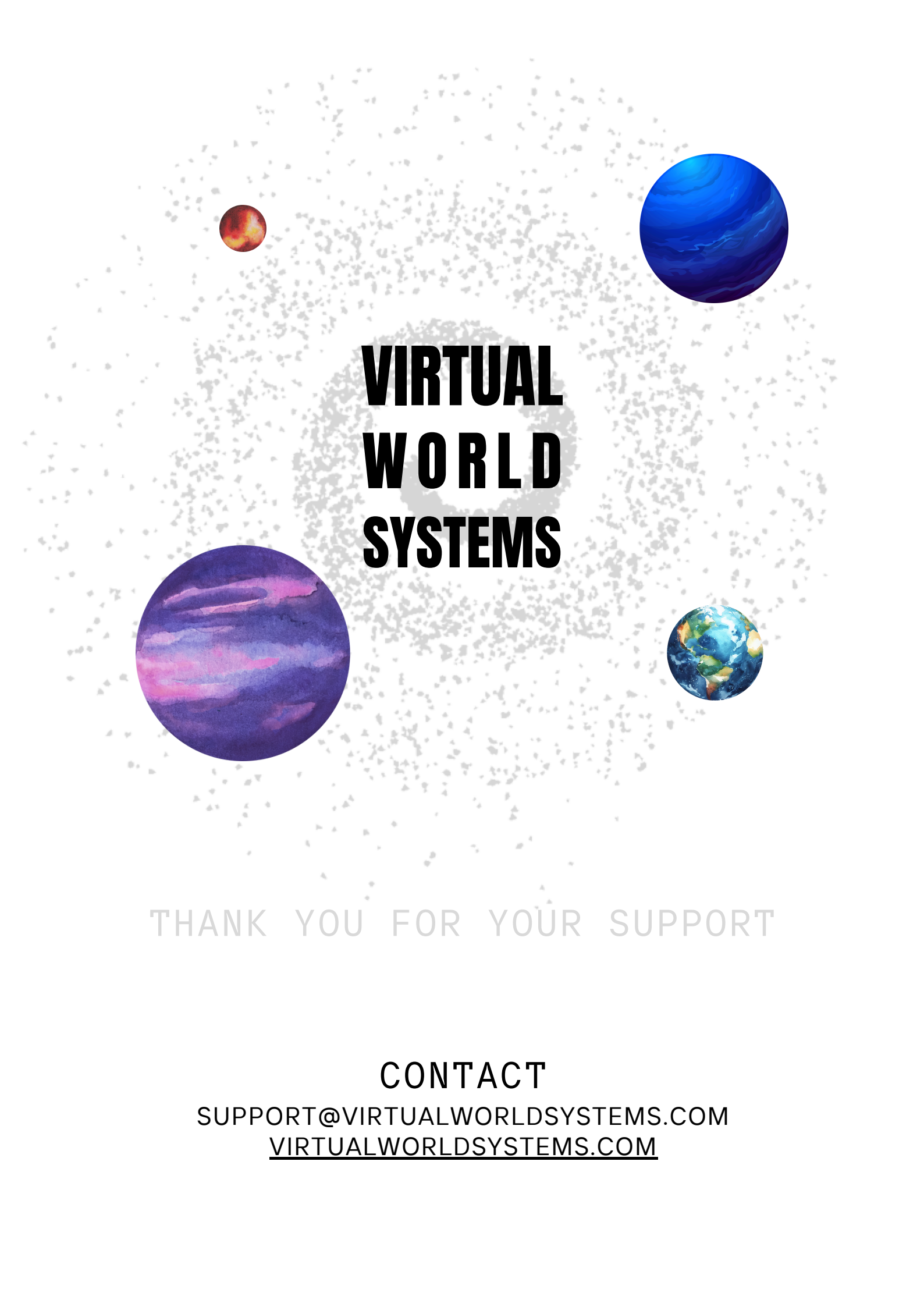
Q: Does Virtualworld Systems have a community discord server?

A: Yes! You are officially invited, [Here](#) is the link!

Q: What projects did you base each 'StreamlinedNetworkMode' off of and test with?

A: Special thanks to the creators of the following demo's for this purpose:

Photon Fusion SDK demo 'FusionDemoGameplayHost'  
Photon PUN2 SDK demo 'PUN Asteroids'  
Mirror SDK demo 'MirrorTanks'  
Netcode-Multiplayer demo by Rishav Nath Pati @ ConvAI  
NetcodePlus by Indie Marc 'Tanks', 'Puzzle' and 'Simple' demos



# **VIRTUAL WORLD SYSTEMS**

THANK YOU FOR YOUR SUPPORT

## **CONTACT**

SUPPORT@VIRTUALWORLDSYSTEMS.COM  
[VIRTUALWORLDSYSTEMS.COM](http://VIRTUALWORLDSYSTEMS.COM)