Easy Code For Vivox

Version 2.0

Last Updated - 11/03/2022

Easy Code For Vivox

Future Documentation

More Documentation at
https://fullstackindie.gitbook.io/easy-code-for-vivox/intro/master

Bridging the gap between Solo/Indie developer teams and AAA games that implement Voice or Text Chat

Supports Unity Game Engine

This Asset only supports

- Windows
- Android

This asset may work with IOS, Mac, Linux, or Consoles. Implement at your own discretion. Source is included so you can modify it.

Vivox Voice and Text Chat is a separate free asset and **does support all major platforms and consoles.**

PlayMaker is a *separate* ***paid Asset*** on the **Unity Asset Store**. PlayMaker is only required if you want to use the PlayMaker Extension part of this asset. If not, you don't have to import it the PlayMaker part and are free to delete it without messing up core functionality. At the moment the Playmaker Extension is in **alpha** and not included in this asset and **may be a separate asset** for the Unity Asset store in the future.

Dependencies

This asset is a simple **API** to interact with the **Vivox Unity SDK** which is available in the **Unity Asset Store** as **Vivox Voice and Text Chat** or in the **Unity Gaming Services Dashboard** after you sign up for Vivox and create a project. This asset is built on top of **Vivox Voice and Text Chat** and will not work without it (You can also download the **SDK** through **Vivox's website**, and this asset will still work if that is your preferred method). You must create an account with Vivox at Login - Vivox Developer Portal and agree to Vivox's Terms or sign up with Unity Gaming Services and accept their Terms of Use before you can use their services. Unity Gaming Services(UGS) may require a credit/debit card to sign up and use their services.

WHERE'S THE CODE?

Example scripts are located **Assets/EasyCodeForVivox/Examples/Demo Scene Examples/** and are the highest abstraction away from the Vivox Unity SDK and the easiest scripts to get started with. **They are meant to be example scripts to show you how to inherit from EasyManager.cs.**

EasyManager.cs is a mid-level abstraction containing variable instances for the ***core Vivox Functionality*** scripts that located in **Assets/EasyCodeForVivox/Scripts/VivoxBackend**

Feel free to modify the code, redistribute, or sell. This asset is meant for the community and small teams and there will always be a free version that implements most if not all Vivox functionality.

Want To Customize This Asset

Assets/EasyCodeForVivox/EasyScripts/VivoxBackend folder contains all the scripts you need to implement Vivox Voice and Text Chat functionality. ***If you want to start learning Vivox from scratch or implement your own code*** look at the scripts in the Vivox Backend folder. The code they contain are very similar to the **Vivox Documentation** and it will be easier to implement Vivox from scratch after having working code examples to compare to the Vivox Documentation.

EasyCode uses **Zenject dependency injection** under the hood. It's basically injecting existing Singleton instances of classes instead of using the new keyword (to create new classes/objects in every class where I need a specific class) or having a static instance of the class.

It is **recommended** to inherit from **EasyManager.cs** or create your own version using the scripts in **VivoxBackend/** because all the boilerplate code has been written for you, you can just create custom wrapper methods around the default methods.

**** Don't change the name of EasyManager.cs or you risk breaking this asset and any future updates.** It's better to create a copy and rename it. Also change the namespace to a namespace that fits your project and just import EasyCode with a using statement

using EasyCodeForVivox;

Design Considerations & Opinionated Design Choices

- Check Out **Assets/EasyCodeForVivox/Demo Scenes** to test Vivox quickly and ***see if this is the right communications choice for your game/app or to see if this is the right asset for you.***
- **EasyCode** uses **Zenject dependency injection** internally so I can scale **EasyCode** without breaking projects in future releases, and to be friendly/compatible with teams working on big projects that already use dependency injection. Also, I am using **Project Context prefab** in Resources folder. If you have a **Project Context** as well it may conflict with **EasyCode** Project Context.
- The **3D demo scene** uses **NetCodeForGameObjects**. I was originally going to use ***Mirror Networking*** but they made a lot of breaking changes and at the time (Oct 2022) tutorials I followed/bought weren't updated, so I decided to adopt **NetCode** since it's part of Unity's Ecosystem and was finally production ready.
- Designed with some static variables to keep track of session (**EasySession.cs**), instead of Singleton design pattern, for persistent session data across scenes.
- Uses **Reflection for Dynamic events**. It is the slower option, but I don't know how to use Mono.Cecil or C# Dynamic Methods with IL Emit/OpCodes (maybe one day).
- At the time of this writing namespace **VivoxAccessToken** located at **Assets/EasyCodeForVivox/Plugins/VivoxAccessToken.dll** is basically a direct copy of how to implement Vivox Access Token. I didn't change method names so it would be easier to reference the docs.

VivoxAccessToken may be changed in the future since later Unity Versions provide a NewtonSoft Json package in the Unity Package Manager which can be used to serialize instead of the JavaScriptWebSerializer currently used. This would mean one less Plugin/dll to maintain separately, and source code would be included in EasyCode

- Designed **EasyManager.cs** to be **inherited from**. You **cannot** reference it using **GetComponent<EasyManager>(); to gain access to its methods**
- Balance of Ease of Use and Performance (My computer is pretty performant so results may vary. **Only when using Dynamic events should you worry about performance**)
- Designed with scaling in mind but if you think your game is going to have over 10, 000 players or need an enterprise solution I would look at **Assets/EasyCodeForVivox/Scripts/VivoxBackend** and **EasyManager.cs** And implement your own solution based on the code in this asset and **Vivox documentation**. Especially if you don't like my naming conventions or design 😄

Things to Consider

- **EasyCode** uses **Zenject** dependency injection internally. Zenject supports **IL2CPP** but be aware there may be issues if using **IL2CPP.**
- Although it's not necessary, for almost all game types that will implement Vivox you will need a networking solution such as the following. **Unity's NetCode for GameObjects, FishNet, Mirror Networking, Mirage, Photon PUN 2/Bolt/Fusion, DarkRift**.
- Even after you have chosen a networking solution, to get your game approved by Vivox for production status (out of sandbox mode) your players need to get authorization tokens (used for secure voice communications and keeps your Vivox credentials out of client / game / app code) from a dedicated server or possibly use cloud lambda functions (tested it with Amazon Lambda and it works) to receive proper Vivox Access Tokens. You can choose to keep your player count under < 5000 player limit and in sandbox mode, but it still poses a security risk and not recommended. Also limits your game from being successful.
- With all this in mind this asset does not provide networking code or solve the problem of getting Vivox Access Tokens (VAT's) from a dedicated server. This asset does provide production code methods and examples but still poses the problem of requesting VAT's directly from the client. One solution is to have a custom game server request the VAT's from Vivox's servers and then pass the VAT to the client so they can connect securely.
- Unity now provides easy to setup dedicated game servers called Multiplay so you can stay within the Unity Eco-System. You can also use AWS EC2(Azure and Google provide similar options) or AWS Game Lift. These options can be more expensive and more difficult to setup and maintain in the long run. Do your own research
- Also, at the time of this writing Vivox Presence Feature does not support access to information on any players connected to their server (currently buggy). ****Update** Presence Feature will likely get deprecated in favor of Unity Friends**.
- This makes it borderline impossible to send direct messages to players because players will have no knowledge of who is currently online. Your players would have to actually know each other or meet on a forum or Discord community and exchange usernames to be able to communicate with each other.
- With this in mind if you are using a dedicated server, you can implement your own functionality. If you're not using a networking solution (or don't want to implement your own system) then I would recommend using one of the following multiplayer friend/leaderboard integrations. There may be better options or for your game so do your research.

Getting Started

Open the **EasyCodeForVivox** folder and check out the Demo Scene to see how easy it is to start using Vivox Voice Chat. If you open the **Scripts** folder, you will find all the code necessary to get started with **EasyCodeForVivox.**

The scripts in **EasyCodeForVivox/Examples/Demo Scene Examples/** are the main scripts used in the demo scenes if you want to see all the methods you will need to access most of Vivox's functionality.

Assets/EasyCodeForVivox/Scripts/EasyBackend/EasyManager.cs is necessary for **EasyCodeForVivox** to work and **you need to inherit from EasyManager to access its methods**. This is only required for any scene that needs to handle Vivox Voice or Text Chat functionality.

It is recommended that the derived class that inherits **EasyManager** includes **DontDestroyOnLoad(this);** in the **Awake()** method. **Voice and Text communications** will carry across scenes (**because session state/data is stored in a static class called EasySession**) but without an **EasyManager.cs** script you cannot interact with **Vivox** to leave a channel or send a channel message, etc.

You can use **Zenject** dependency Injection (*comes with EasyCodeForVivox in the Plugins folder*) to inject any class you need located in **Assets/EasyCodeForVivox/Scripts/VivoxBackend/** which is where the core **Vivox** related functionality lies that **EasyCode** uses.

Setting Up Your Vivox Credentials

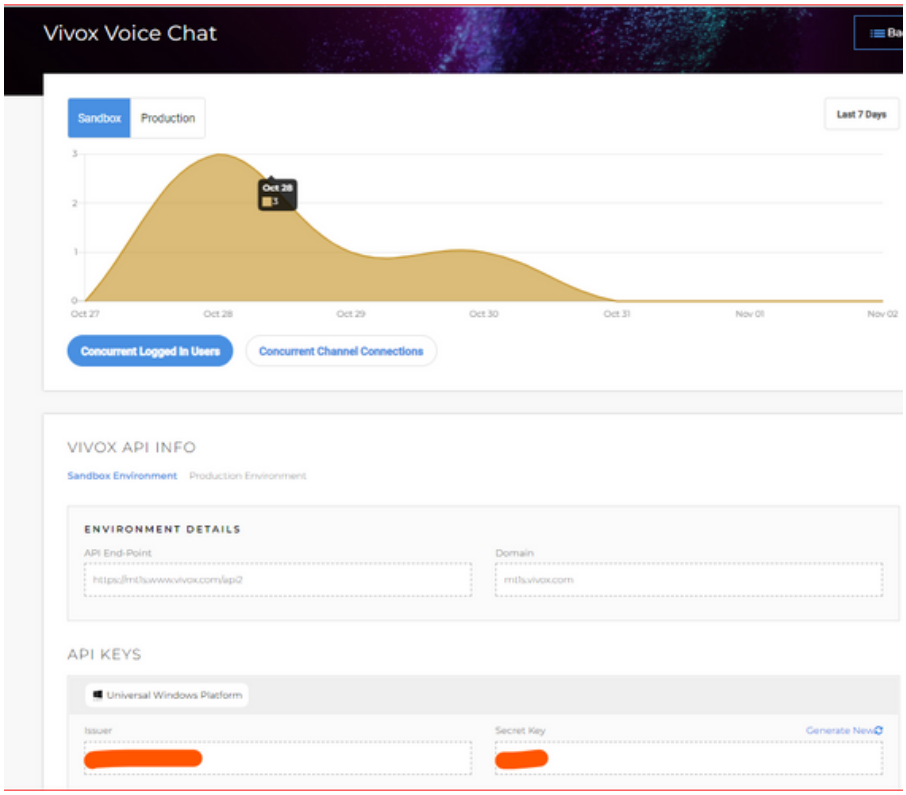
In your script that inherits from **EasyManager**, you must hardcode your credentials. This is used for development purposes. Vivox recommends not hardcoding your credentials in your application. You can implement custom logic to retrieve your credentials at runtime from your game server. You can also use **Unity's Remote Config** or **Cloud Code** to get credentials at runtime if you don't have a dedicated game server such as **Unity's Multiplay, AWS EC2** instance (or other cloud providers VM's), or self-hosted home server

Your credentials should be assigned variables located inside of the static class EasySession

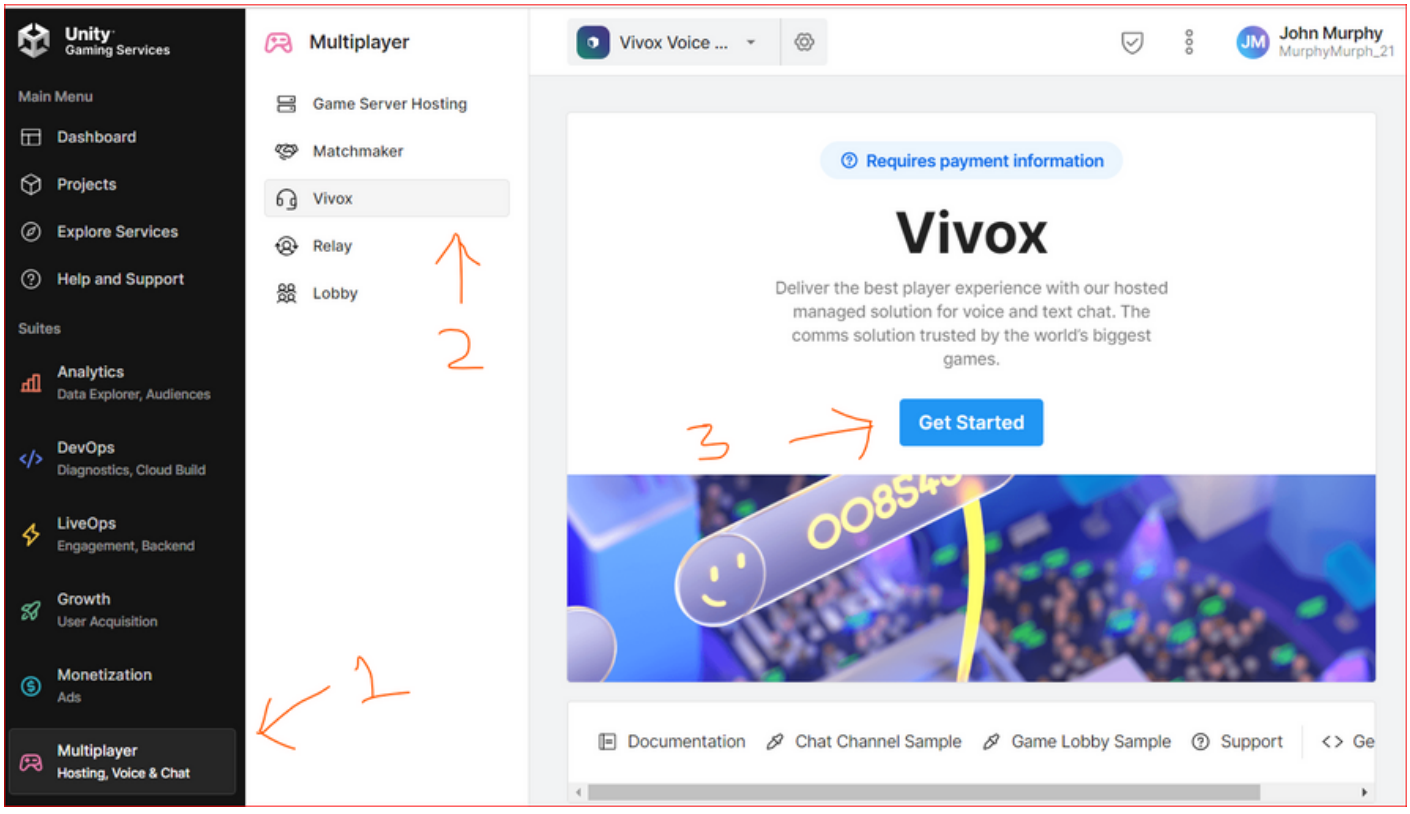
```
public class VivoxManager: EasyManager

    [SerializeField] string apiEndpoint;
    [SerializeField] string domain;
    [SerializeField] string issuer;
    [SerializeField] string secretKey;

    private void Awake()
    {
        EasySession.APIEndpoint = new Uri(apiEndpoint);
        EasySession.Domain = domain;
        EasySession.Issuer = issuer;
        EasySession.SecretKey = secretKey;
    }
```

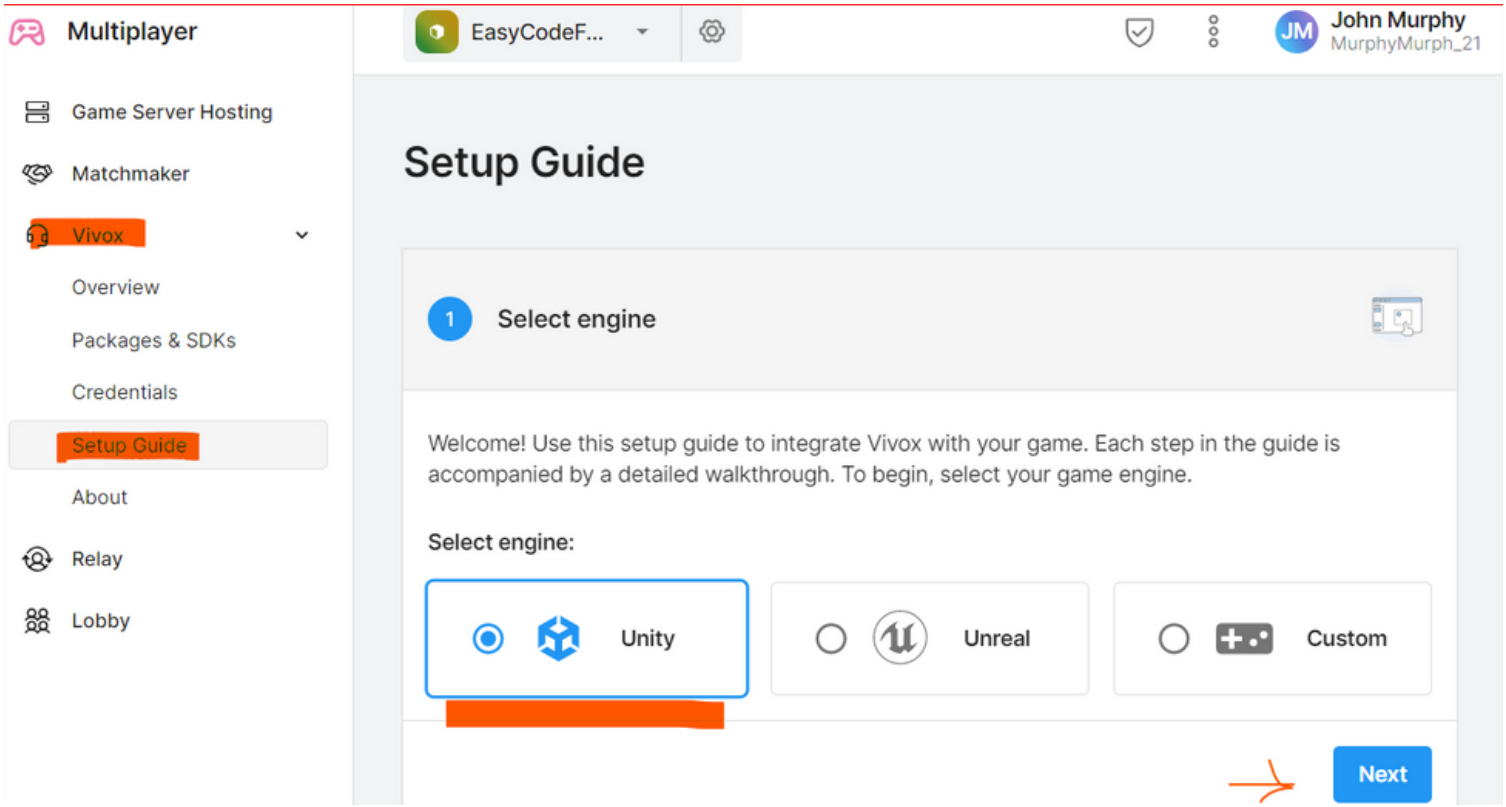
Use **Vivox Developer Portal** or **Unity Gaming Services (UGS) Dashboard** and create a project to get access to credentials or link your project inside the Unity Editor to be able to access credentials. Images below

Vivox Developer Portal - Credentials Dashboard

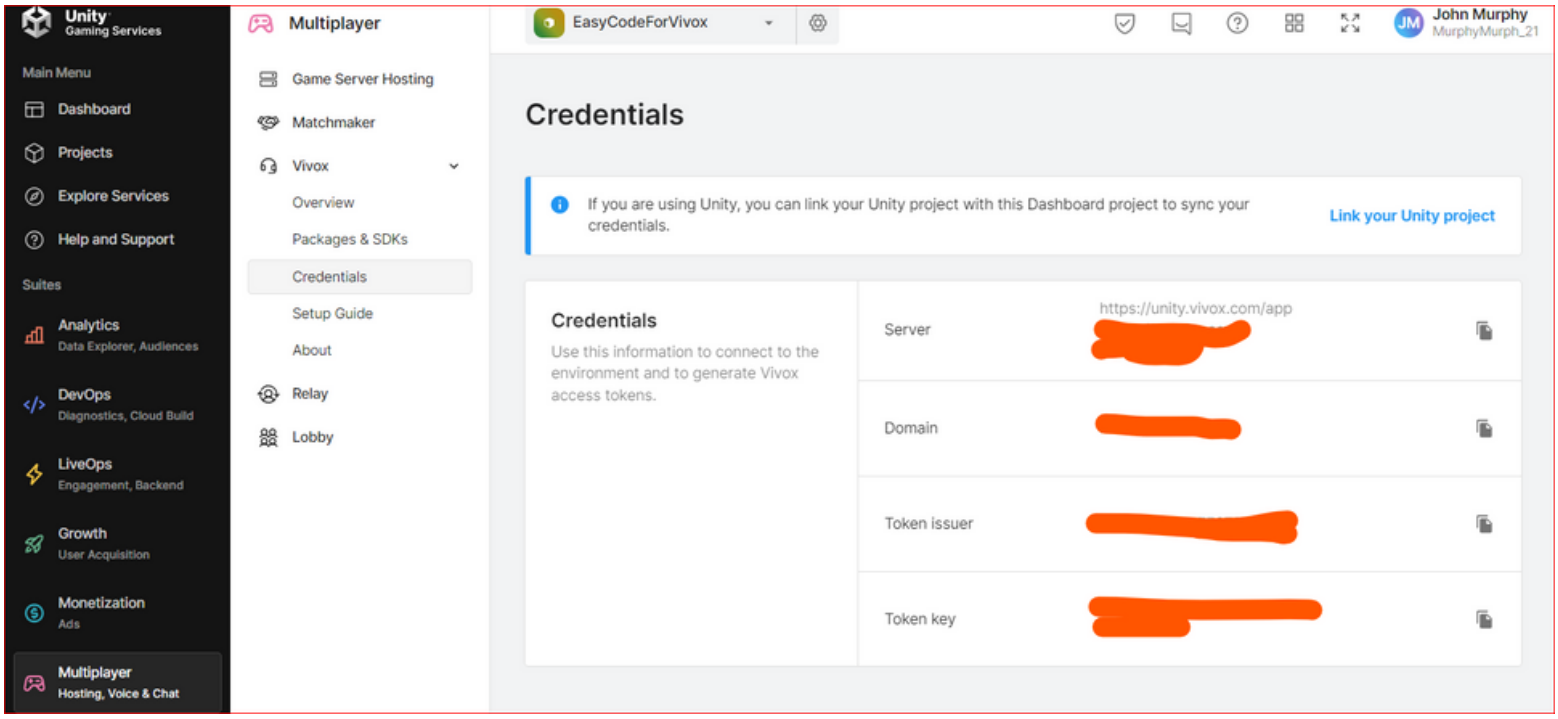


Unity Gaming Services Vivox Dashboard

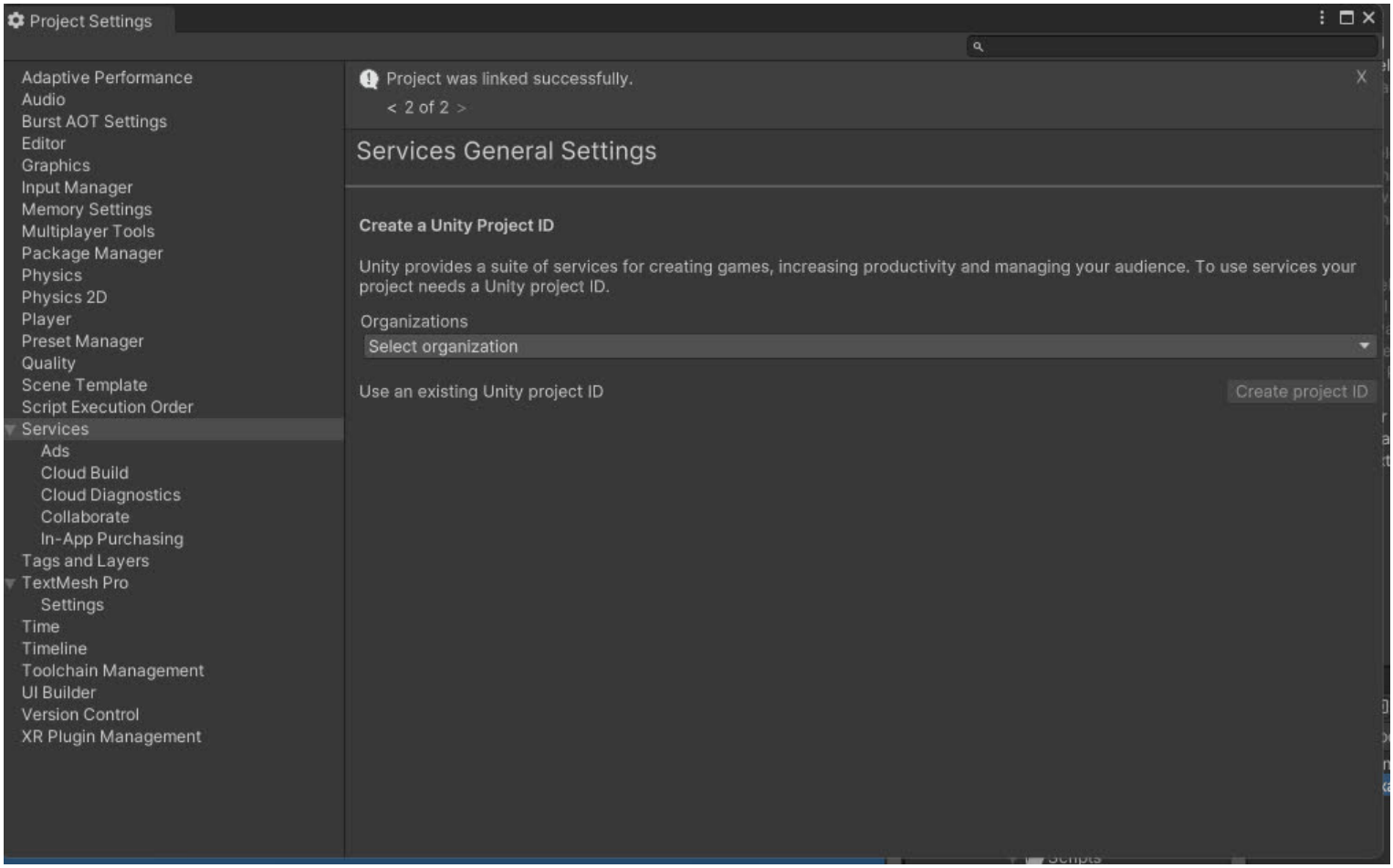




Get your credentials from UGS Dashboard in Vivox section



Setup using Existing Project in Unity that was created in the UGS Dashboard. If you hadn't created a project, choose **Create Project Id** and a new project will be created and will appear in UGS Dashboard



How to use Example scripts

Scripts located in **Assets/EasyCodeForVivox/Examples/Demo Scene Examples/** cover most functionality you will need for Vivox Voice Chat. You can edit these scripts and rename them without breaking any EasyCode core functionality. ** You may mess up the demo scenes from working if you edit these scripts tho **

You may notice **EasyChatExample** and **Easy3DExample** inherit from **EasyManager**. Because of this inheritance you don't need to add an EasyManager to the project or a GameObject. The **[SerializedField]** properties are only applicable to the Demo Scene and serve as an example of how to incorporate a **User Interface(UI**) to get the player values needed to send to Vivox. You can do all this thru code (*if using other Unity/3rd Party services for user identity*) without user input if you want.

Most of the **public void** Methods in Example scripts stated above are linked to **button events** or other **UI events** in the **Canvas**.

All the **public/protected override** methods are **inherited from EasyManager** and overridden. They are all called when a **Vivox event fires** I left the base methods
[base.MethodNameHere();] in as an example but you can delete them. There is no major functionality in the base methods, just a simple Unity **Debug.Log()** statement for each event that fires. Some of the overrides update the **Text UI Chat Panel** in the Demo Scene and is not necessary just an example of what information may be relevant to common use cases.

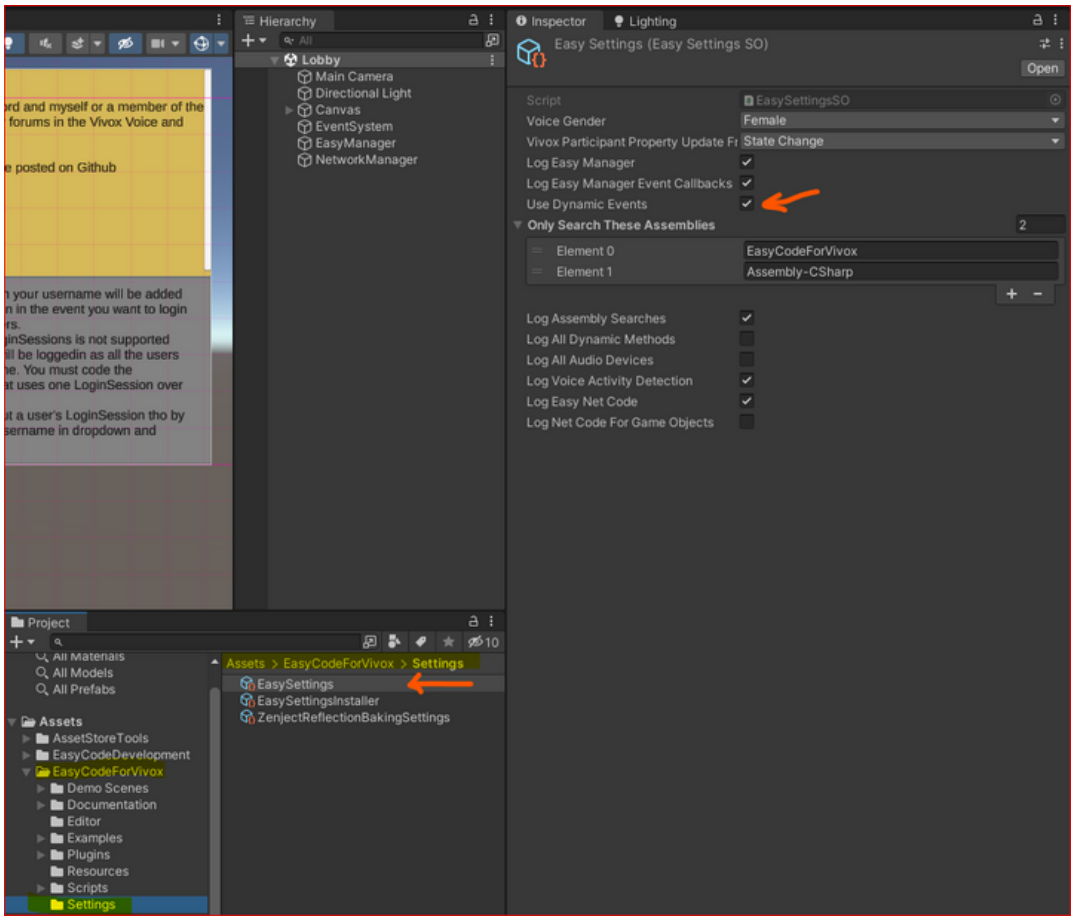
You can Attach any Example script to a gameObject and hook up your own UI to it. You can create a duplicate of the Example scripts and/or rename them to something that fits your naming convention and delete the methods / events you don't need.

Certain events update numerous times a second and should be used wisely to avoid spamming. Such as

protected override void **OnUserValuesUpdated**(IParticipant participant)

Test out which events you would like to subscribe to and delete the ones you don't need (In Example Scripts only...**Not EasyManager.cs**)

Setup EasySettings



EasySettings are located at **Assets/EasyCodeForVivox/Settings/EasySettings**

You can configure **Logging, Dynamic Events, Text-To-Speech Voice Gender**, and how often **User updates/events are invoked/fired.**

If using Dynamic Events you should specify which assemblies you want searched. I have chosen to only search **EasyCodeForVivox** and **Assembly-CSharp** (**default Unity project Assembly** unless you create your Assembly Definitions). If **OnlySearchTheseAssemblies** is left blank **EasyCode** will search all assemblies asynchronously at startup to avoid UI/thread blocking. If there are a lot of Assemblies to search and players start playing the game immediately some Dynamic events may not be registered in time and will not be invoked/fired

I had to implement like this because searching all assemblies inside a project with a lot of assets/packages took a long time (**7+ seconds at startup** even after ignoring most of Unity's packages)

How to use EasyManager

Initialize EasyManager and Vivox- Using async void makes **Initialization** faster especially if using **Dynamic Events**

```
private async void Awake()
{
   await InitializeClient();
}
```

Unitialize EasyManager and Vivox

```
private void OnApplicationQuit()
{
   UnitializeClient();
}
```

Login

- Logs into Vivox if the username is valid
- Sets the transmission mode to All so any Voice or Text chat automatically switches to a new channel whenever a channel is joined or switched.
- There is an additional parameter that allows the player to joined as muted. This is useful for conference type settings like zoom chat where Admins/Moderators controls who can speak.

EasyCode automatically validates if username is accepted by Vivox. You are still responsible to make sure another user with the same name is not already in the same channel

```
public void Login()
{
   LoginToVivox("username");
}

public void LoginAsMuted()
{
   LoginToVivox("username", true);
}
```

Logout

- Logs player out of Vivox.

```
public void Logout()
{
   LogoutOfVivox("username");
}
```

// **Login Event Callbacks**

You can use protected override to see what events you can override. Keeping the base methods are not necessary. They are simply Debug.Logs(). Feel free to delete them

```
protected override void OnLoggingIn(ILoginSession loginSession)
{
    base.OnLoggingIn(loginSession);
}

protected override void OnLoggedIn(ILoginSession loginSession)
{
    base.OnLoggedIn(loginSession);
}

protected override void OnLoggingOut(ILoginSession loginSession)
{
    base.OnLoggingOut(loginSession);
}

protected override void OnLoggedOut(ILoginSession loginSession)
{
    base.OnLoggedOut(loginSession);
}

protected override void OnLoginAdded(AccountId accountId)
{
    base.OnLoginAdded(accountId);
}

protected override void OnLoginRemoved(AccountId accountId)
{
    base.OnLoginRemoved(accountId);
}

protected override void OnLoginUpdated(ILoginSession loginSession)
{
    base.OnLoginUpdated(loginSession);
}
```

Joining an Echo Channel
This type of channel is used for Mic testing.

- Joins the player "username" to an Echo channel called "echo".
- Audio is activated in channel so the player can test their mic.
- Since "username" is the only player in the channel, Text capabilities are turned off.
- Transmission is switched to the current channel which overrides any previous settings set in the current LoginSession (This means only Audio/Voice/Text will transmit through this Echo channel and no other channel).
- Channel type chosen is Echo channel.
- Join muted is set to false because we want the player to hear themselves speaking so they can test their mic
- No 3d Settings are created/added because it is not a 3d channel

```
public void JoinEchoChannel()
{
    JoinChannel("username", "echo", true, false, true, ChannelType.Echo, joinMuted: false);
}
```

Joining a Non-Positional (Conference) Channel
This type of channel is normal chat

- Joins the player "username" to a NonPositional (Conference) channel called "chat".
- Audio is activated in channel so the player can talk to other players.
- Text capabilities are turned on so player can chat with other players.
- Transmission is not switched to current channel. Any previous settings set in the current LoginSession will be persisted.
- Channel type chosen is NonPositional channel.
- Join muted is set to false because we want the player to hear other players in channel as soon as they join
- No 3d Settings are created/added because it is not a 3d channel

```
public void JoinChannel()
{
    JoinChannel("username", "chat", true, true, false, ChannelType.NonPositional, joinMuted: false);
}
```

Joining a 3D Channel
This type of channel is for 3d video games where Audio/Voice and Text capabilities are based on how close you are to other players.

- Joins the player "username" to a 3DPositional channel called "3D".
- Audio is activated in channel so the player can talk to other players.
- Text capabilities are turned off because I don't want players text/chat to be lost/discarded if they aren't close to each other on the map. I also don't want everyone to be able to text each other, only squads can text within their squad (use NonPositional Channel).
- Transmission is not switched to current channel. Any previous settings set in the current LoginSession will be persisted.
- Channel type chosen is 3DPositional channel.
- Join muted is set to false because we want the player to hear other players in channel as soon as they join
- I have created 3d Settings with the default values that Vivox recommends, and I am passing in the settings so when the 3d channel is created it will apply my chosen settings.

```
public void Join3DChannel()
{
    var channelProperties = new Channel3DProperties(32, 1, 1.0f, AudioFadeModel.InverseByDistance);

    JoinChannel("username", "3D", true, false, false, ChannelType.Positional, joinMuted: false,
 channel3DProperties: channelProperties);
}
```

Want more EasyCode documentation check out Assets/EasyCodeForVivox/Documentation/Offline GitBook Docs

Vivox Features You Can Access From EasyCodeForVivox

- **Login Multiple Users/Player Sessions (requires custom logic)**
- **Join 1-10 Non-Positional Channels (Conference Channels)**
- **Join 1 3D Positional Channel**
- **Maximum Joined Channels is 10 (9 Non-Positional, 1 Positional)**
- **Send Channel Messages**
- **Send Direct Messages (as long you know the User's Name)**
- **Toggle Voice/Audio in channel**
- **Toggle Text in channel**
- **Adjust Local Players Volume**
- **Adjust Remote Player's Volume if you know their name**
- **Mute Self**
- **Mute other players in channel if you know their name**
- **Text-To-Speech(TTS) - All of Vivox TTS options available**
- **Push-To-Talk functionality**
- **Raise Hand feature where Admin/Teacher/Host can mute/unmute anyone in the current channel. Anyone who is not Admin/Host/Teacher can raise their hand and then the Admin/Host/Teacher gets to decide if they want to unmute them and when to mute them. In the LoginToVivox method in EasyChatExample.cs or Easy3DExample.cs make sure the joinMuted parameter is set to true for all users and whoever you decide gets to be the Admin leave false. It is false by default.**

Features Supported By Vivox but not available in EasyCodeForVivox

- **Presence is not added because there is currently a bug in Vivox SDK for Unity. Also, this feature will most likely get deprecated in favor of Unity Friends**
- **IOS/Mac/Linux support is not added or confirmed (Vivox does support IOS/Mac/Linux but EasyCodeForVivox doesn't)**
- **No client/server architecture**
- **No Server-to- Server API support**