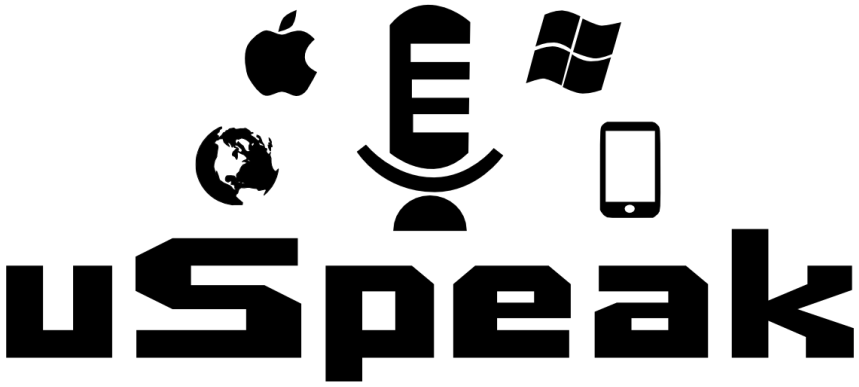

USpeak™ Developer Documentation

USpeak v3.0

Hello. My name is Alan Stagner, and I'm the creator of USpeak. I started this project because I needed an integrated voice chat solution for my own games - and I just couldn't find any good solutions. USpeak is intended to provide fast, efficient, and high-quality voice chat for any multiplayer game, whether it be a browser MMO, a mobile multiplayer game, or a standalone game aimed at serious players - USpeak covers it all. I hope you enjoy this package, and I can't wait to see the games that use it.
That being said, let's dive right in!

General Setup
A standard USpeak setup is made of the following components:

- A USpeaker component (Components → USpeak → USpeaker)
- A data handler component (any monobehavior that implements the ISpeechDataHandler interface)
- (Optional) A talk controller component (any monobehavior that implements the IUSpeakTalkController interface). A default talk controller is included (Components → USpeak → Default Talk Controller)

What do these components do? Let's take a look.
USpeaker does most of the work. It handles recording audio data, processing and encoding it, and wrapping it up in a network-friendly format before handing it off to the *Data Handler*, which is in charge of sending data generated from USpeaker over the network, as well as passing received data back to the USpeaker.
The USpeaker also uses an optional *Talk Controller*, which controls whether or not data should be recorded and sent at any given time. If no controller is present, data is always sent. The

included example controller uses keypresses to determine whether data should or shouldn't be sent, and lets you configure which key is used and whether the behavior is toggling talk on and off, or holding down the key to talk (walkie talkie style). You can also create your own talk controller.

USpeaker Properties

The USpeaker component itself has quite a few properties available, here's the complete list:

Property	**Script Property**	**Description**
Codec	USpeaker.Codec	Which codec this USpeaker uses to encode/decode. This is an index into the codec list as defined in the Codec Manager
Speaker Mode	USpeaker.SpeakerMode	Whether the USpeaker is a Local USpeaker (should send data) or a Remote USpeaker (should only receive data)
Bandwidth Mode	USpeaker.BandwidthMode	The recording frequency to use. Options are Narrow (8kHz), and Wide (16kHz).
Sending Mode	USpeaker.SendingMode	The behavior of sending, whether to send at the same time as recording, or to first record, and then send, data.
Volume Activated	USpeaker.UseVAD	When enabled, audio is compared against a threshold. If audio is too quiet, it will not be sent.
Debug Playback	USpeaker.DebugPlayback	If the USpeaker is a Local USpeaker, whether audio data should be played back. Useful for debugging purposes.
Send Rate	USpeaker.SendRate	How many times per second should audio data be serialized.
3D Mode	USpeaker._3DMode	What level of 3D is applied to the played audio data. None = no 3D, Speaker Pan = audio pans across speakers, Full 3D = audio is played with all 3D effects (doppler, distance, reverb, etc)

Ask For Mic Permission	USpeaker.AskPermission	[Only affects web builds] Whether the USpeaker should automatically display a Microphone Permissions dialogue. Usually it's best to disable this and call Application.RequestUserAuthorization from the main menu, for example.
Speaker Volume	USpeaker.SpeakerVolume	The playback volume of this speaker.
Volume Threshold	USpeaker.VolumeThreshold	When Volume Activated is enabled, audio is thresholded against this value. Only if it exceeds the threshold will it be sent.
N/A	USpeaker.Mute	Whether to mute incoming voice data for the given USpeaker.

There are also a few static properties that are useful for game settings:
USpeaker.RemoteGain : [Default 1.0] How much to boost incoming audio data. A fractional value means quieter, a value more than 1.0 means louder.
USpeaker.LocalGain : [Default 1.0] How much to boost recorded data before encoding and sending it. Similar to RemoteVolume.
USpeaker.MuteAll : [Default 'false'] Whether to mute all incoming voice data. Note that this does not save bandwidth.

And, finally, most games include some kind of in-game overlay, which shows a list of the players who are actively speaking. You can do that in USpeak by iterating the static *USpeaker.USpeakerList,* and for each USpeaker in the list check the *IsTalking* property.

Implementing the Interfaces
In most cases, you can just use the default talk controller and create a component that implements the *ISpeechDataHandler* interface. Here's how you do that:

1. Create a new Monobehavior script.
2. At the top of this script, add 'using MoPhoGames.USpeak.Interface'
3. Implement the two functions:
    a. USpeakOnSerializeAudio(byte[] data)
        i. This might, for example, call an RPC and pass the byte array, and in that RPC you'd call USpeaker.ReceiveAudioData(data)

    b. USpeakInitializeSettings(int data)
        i. This is passed an integer value to be passed over the network. **NOTE: This call MUST be buffered and sent to any newly connected players**. This might call an RPC, and in the RPC it calls USpeaker.InitializeSettings(data)
4. Depending on the context, this script should also set the USpeaker.SpeakerMode property depending on whether it belongs to the local player or the remote player.
5. Add the new script to the same object as the USpeaker.

If you need a custom way to trigger audio sending (for example, an onscreen button in Mobile games), you can also implement the *IUSpeakTalkController* interface:

1. Create a new Monobehavior script.
2. At the top of this script, add 'using MoPhoGames.USpeak.Interface'
3. Implement the two functions:
    a. OnInspectorGUI
        i. This is used internally by DefaultTalkController. Generally you can just leave it empty
    b. ShouldSend
        i. Returns a boolean. Evaluated every time audio data is about to be encoded. If it returns true the audio data is encoded and buffered before being sent. If it returns false the audio data is discarded.
4. Add the new script to the same object as the USpeaker.

Using USpeak With Other Networking Engines

You'll notice that the given network example uses Unity Networking. This can be easily modified for Photon Unity Networking and similar middleware, but what about technologies such as Photon Server and Player.IO?

In this case, you'll want to take a look at USpeakUtilities.

Specifically, the workflow looks like this:

1. Upon connecting to the server, download a list of players. Pass string array of player IDs to USpeakUtilities.ListPlayers
2. While connected, if a new player connects, pass player ID to USpeakUtilities.PlayerJoined
3. While connected, if a player disconnects, pass player ID to USpeakUtilities.PlayerLeft
4. When sending voice data, make sure when you receive it you have some way to access player ID (most networking engines feature a way to send player ID with the message)
5. When receiving voice data, use the sending player's ID to find the USpeaker by calling USpeakOwnerInfo.FindPlayerByID(playerID) and accessing the 'Speaker' property of the returned USpeakOwnerInfo

Advanced Topic: Changing Input Devices

USpeak by default uses the 'default' microphone device (the one found at Microphone.devices[0]). You can, however, force USpeak to use a different device by calling the static function USpeaker.SetInputDevice(int deviceID). USpeaker automatically restarts the recording process with the given microphone, and also automatically caps the given device ID when it exceeds the device range.

Note that 'deviceID' corresponds to the indices of the string entries in Microphone.devices.

Advanced Topic: Alternate Codecs

Included with USpeak are two alternate codecs: ADPCM and MuLaw. Both of these codecs offer far less CPU usage as compared to Speex (and may be a better fit for mobile platforms), with different tradeoffs. The quality of ADPCM is sacrificed for bandwidth (and it can often sound scratchy and staticky), whereas MuLaw takes up twice the bandwidth but has improved audio fidelity.

To change to either of these codecs, simply choose them from the Codec dropdown in the inspector. Don't worry about clients using different codecs; USpeak automatically handles this for you.

Advanced Topic: Custom Codecs

Let's say you have a custom codec you've written. It's the best codec EVAR! But how do you plug it into USpeak? The answer is simple:

1. Create a new script file (NOT a Monobehavior)
2. At the top of the script, add 'using MophoGames.USpeak.Codec'
3. Make a class to represent your custom codec, and make it implement the ICodec interface with the following functions:
    a. byte[] Encode(short[] data)
        i. This encodes an array of 16-bit PCM samples into a byte array. The format of the byte array doesn't matter
    b. short[] Decode(byte[] data)
        i. This decodes an encoded byte array into an array of 16-bit PCM samples.
    c. int GetSampleSize(recordingFrequency)
        i. Returns the input length your codec expects when encoding. Return 0 to indicate that your codec can handle any size
4. Add your codec to the Codec Manager
    a. Open Window | USpeak | Codec Manager
    b. Hit the "Add" button
    c. Select your codec from the dropdown
    d. Hit "Save"
5. Select your codec in the USpeaker inspector (it will automatically be available in the dropdown)

Voila! You now have a custom codec in place.

A Note On Advertising USpeak

Included in the Scenes folder you'll notice a SplashScreen folder. This folder contains two example splash screens with the USpeak logo. While not strictly required, it is recommended to include one of these splashscreens in your game. If you need a more customized splashscreen, you can find the source PSD files in the PromoAssets folder.

And you're done.

Congratulations, at this point you should now have fully-functional, realtime voice chat for your game. USpeak will add a whole new level of player-to-player communication and strategization, and overall will improve the game experience dramatically.

I hope you enjoy USpeak as much as I do.

Happy coding!