USpeak^{TM} Developer Documentation

USpeak v2.0a

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**
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.
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	USpeaker.Is3D	Whether to apply directional pan effects to the audio data based on the USpeaker position and the Main Camera position.
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.

Play Buffer Length	USpeaker.PlayBufferSize	How long to buffer audio data before playing. Smaller can result in more glitches and gaps in the audio, but larger means a longer delay between sending and playback.
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.RemoteVolume : [Default 1.0] How much to boost incoming audio data. A fractional value means quieter, a value more than 1.0 means louder.
USpeaker.LocalVolume : [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.

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: Mu-Law Alternate Codec
Included with USpeak is an alternate codec known as Mu-Law. While the default codec results in a 4:1 compression ratio and Mu-Law only results in a 2:1, the quality of Mu-Law is superior to that of ADPCM.
To use the codec, anywhere in your code you can call the following:
USpeakAudioClipCompressor.Codec = new MuLawCodec();
Make sure you include the namespaces *MoPhoGames.USpeak.Codec* and *MoPhoGames.USpeak.Core*

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.
4. Anywhere in your code, assign the new codec via:
    a. [make sure to include the MoPhoGames.USpeak.Core namespace]
    b. USpeakAudioClipCompressor.Codec = new MyCodec();
    c. Be sure to replace 'MyCodec' with the name of your codec.

Voila! You now have a custom codec in place. Note that two clients using different codecs are NOT compatible.

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!