Table of Contents

Introduction

Estrada is a drop-in replacement for Unity Microphone with WebGL support.

Features:

- Full compatibility with Unity Microphone API
- One line of code to switch from Unity Microphone to Estrada
- WebGL support (with the limitations listed below)
- Uniform microphone permission handling for all platforms
- Automatic browser feature detection
- Uses AudioWorkletNode in HTTPS mode and falls back to ScriptProcessorNode in HTTP mode

Supported Unity editors:

- 2021.2 and above

Limitations

WebGL:

- Unity doesn't support *AudioClip.GetData()*. Use *Estrada.Microphone.GetCurrentData()* instead
- Unity doesn't support *AudioClip* streaming, simultaneous audio recording and playback is not supported
- Microphone permission is required to view available devices

Known issues

AudioClip cannot be read in WebGL

Symptom:

> *(Error in browser console) Cannot get data on compressed samples for audio clip "clipName". Changing the load type to DecompressOnLoad on the audio clip will fix this.*

Fix: Update your version of Unity to one of the following versions

Getting started

```
using System.Collections;
using UnityEngine;

// Add this line to your code to replace Unity Microphone with Estrada Microphone
using Microphone = Estrada.Microphone;

public class EstradaExample : MonoBehaviour
{
    private IEnumerator Start()
    {
        // Get permission
        if (Microphone.RequiresPermission())
        {
            yield return Microphone.RequestPermission();

            if (!Microphone.HasPermission())
            {
                Debug.LogError("Permission to record a microphone is not granted");
                yield break;
            }
        }

        // Start microphone and connect it to AudioSource
        const string deviceName = null;
        const int length = 5;
        const int frequency = 16000;
        const bool loop = false;

        var audioSource = gameObject.AddComponent<AudioSource>();

        audioSource.clip = Microphone.Start(deviceName, loop, length, frequency);
        audioSource.loop = loop;

        // Wait for microphone to start
        while (Microphone.GetPosition(deviceName) == 0)
        {
            yield return null;
        }

        // Wait until recorded
        while (Microphone.IsRecording(deviceName))
        {
            yield return null;
        }

        // Play recorded audio
        audioSource.Play();
    }
}
```

API

Member	Description
property public static string[] devices	Returns names of available devices.
public static bool RequiresPermission	Whether a microphone permission is required
public static IEnumerator RequestPermission	Request permission to record microphone.
public static bool HasPermission	Whether a permission to record microphone has been granted.
public static AudioClip Start	Start recording.
public static void End	Stops recording
public static bool IsRecording	Whether recording is active.
public static bool GetCurrentData	Fills an array with sample data from the clip. If not recording, samples from previous clip will be used.
public static int GetPosition	Get current position of the recording.
public static void GetDeviceCaps	Get the frequency capabilities of a device.

property public static string[] `devices`

Returns names of available devices.

public static bool `RequiresPermission()`

Whether a microphone permission is required

Kind	Member	Description
return		True if required.

public static IEnumerator `RequestPermission()`

Request permission to record microphone.

Kind	Member	Description
return		Enumerator to run coroutine on.

public static bool `HasPermission()`

Whether a permission to record microphone has been granted.

Kind	Member	Description
return		[object Object]

public static AudioClip `Start(string deviceName, bool loop, int lengthSec, int frequency)`

Start recording.

Kind	Member	Description

return		Returns AudioClip or null if the recording fails to start.
param	deviceName	Name of the device to be recorded. Null or empty string means default device.
param	loop	Indicates whether the recording should continue recording if lengthSec is reached, and wrap around and record from the beginning of the AudioClip.
param	lengthSec	Length of the recording before overlap.
param	frequency	The sample rate of the AudioClip produced by the recording.
exception	InvalidOperationException	If permission to record was not granted or no devices available.
exception	ArgumentException	[object Object]

public static void `End(string deviceName)`

Stops recording

Kind	Member	Description
param	deviceName	The name of the device.

public static bool `IsRecording(string deviceName)`

Whether recording is active.

Kind	Member	Description
return		Recording state.

public static bool `GetCurrentData(float[] data, int offsetSamples)`

Fills an array with sample data from the clip. If not recording, samples from previous clip will be used.

Kind	Member	Description
return		Whether the microphone data was accessed.
param	data	Array to fill.
param	offsetSamples	Internal buffer offset.

public static int `GetPosition(string deviceName)`

Get current position of the recording.

Kind	Member	Description
return		Current recording position in samples.

public static void `GetDeviceCaps(string deviceName, out int minFreq, out int maxFreq)`

Get the frequency capabilities of a device.

Kind	Member	Description
param	deviceName	Device name. Null or empty string means default device.
param	minFreq	Minimal supported sample rate.

param	maxFreq	Maximal supported sample rate.