Turn your voice into musical notes, instantly

BACKBONE SINGER TONE

Version 1.0

By Xavinet

Content

What is Backbone Singer Tone 3

Requirements and platform permissions 3

Quick Start 4

Prefab: MicToneDetector 5

- Public properties 5

Usage examples 6

- Display Hz / Note / MIDI in UI 7
- React to notes with UnityEvents 7
- Select microphone in code 8

Troubleshooting 8

- No pitch detected 8
- Permissions and WebGL specifics 8
- Latency vs stability and performance 9

What is Backbone Singer Tone

Backbone Singer Tone is a Unity asset that converts live microphone audio into real-time musical pitch data—frequency (Hz), MIDI note, cents offset, and note name—exposed as properties and UnityEvents.

Add the MicToneDetector prefab, select a mic, and it handles capture, level gating, and optional smoothing via a configurable analysis window, ready to drive UI and gameplay on desktop, mobile, and WebGL.

Key features

- Robust monophonic pitch tracking tuned for voice.
- Low latency windowed analysis (pitchWindowMs).
- Level gate with optional "clear on silence."
- Mode-based smoothing over recent detections.
- Clean runtime API: last values as properties, events for UI/logic.
- Works on Desktop, Mobile, and WebGL (with mic permissions).

Typical uses

- Vocal pitch visualizers and tuners.
- Singing gameplay, note-hit logic, vocal UI feedback.
- Educational apps for ear training and intonation.

Requirements and platform permissions

This section explains what Unity versions and audio setup you need, and how microphone permissions work on each platform. Read it before shipping to desktop, mobile, or WebGL so the mic starts reliably and users see the correct permission prompts.

Minimun requirements

- **Unity:** 2020.3+ (Mono or IL2CPP).
- **Audio:** microphone available.
- **Platforms:** Windows, macOS, Linux, Android, iOS, WebGL.

Permissions by platform

- **Windows/macOS/Linux:** grant mic access when prompted by the OS.
- **Android:** runtime permission RECORD_AUDIO. Unity handles the request; ensure your app targets an SDK that supports runtime permissions.
- **iOS:** add NSMicrophoneUsageDescription to **Info.plist** with a non-empty message.

Notes and Tips

- Headsets reduce bleed and false detections.
- For low latency, start with pitchWindowMs = 80–120.
- If your device reports unknown mic caps, the asset falls back to the project sample rate.

Quick Start

Backbone Singer Tone is quite simple to use: drop the MicToneDetector prefab, tweak a few fields, optionally wire UnityEvents, and you're detecting sung notes in real time—no boilerplate, no custom audio plumbing, just plug the mic and go.

1) **Add prefab:** Drag **MicToneDetector** to your scene. It includes an AudioSource (loop, muted).

2) **Inspector setup:**

- autoStart = true
- pitchWindowMs = 80–120
- minVolumeDB ≈ -20
- consecutiveDetections = 5
- useClipRead = true

3) **Hook events (optional UI):**

- OnPitchHz(float) → update Hz label
- OnMidiNote(int) → update MIDI label or logic
- OnMidiCents(int) → show cents
- OnNoteName(string) → show note text

4) **Run:** Play the scene and allow microphone permission. Sing. Values update in real time.

5) **Start/Stop manually (optional):**

Use this in your code

```
var mic = FindObjectOfType<MicToneDetector>();
mic.StartMic();          // or mic.StartMic("YourDeviceName");
mic.StopMic();
```

6) **Read latest values (polling)**

If you wish to use your own detection instead of being called by events, use:

```
float hz = mic.LastFrequencyHz;
int midi = mic.LastMidiNote;
string name = mic.LastNoteName;
```

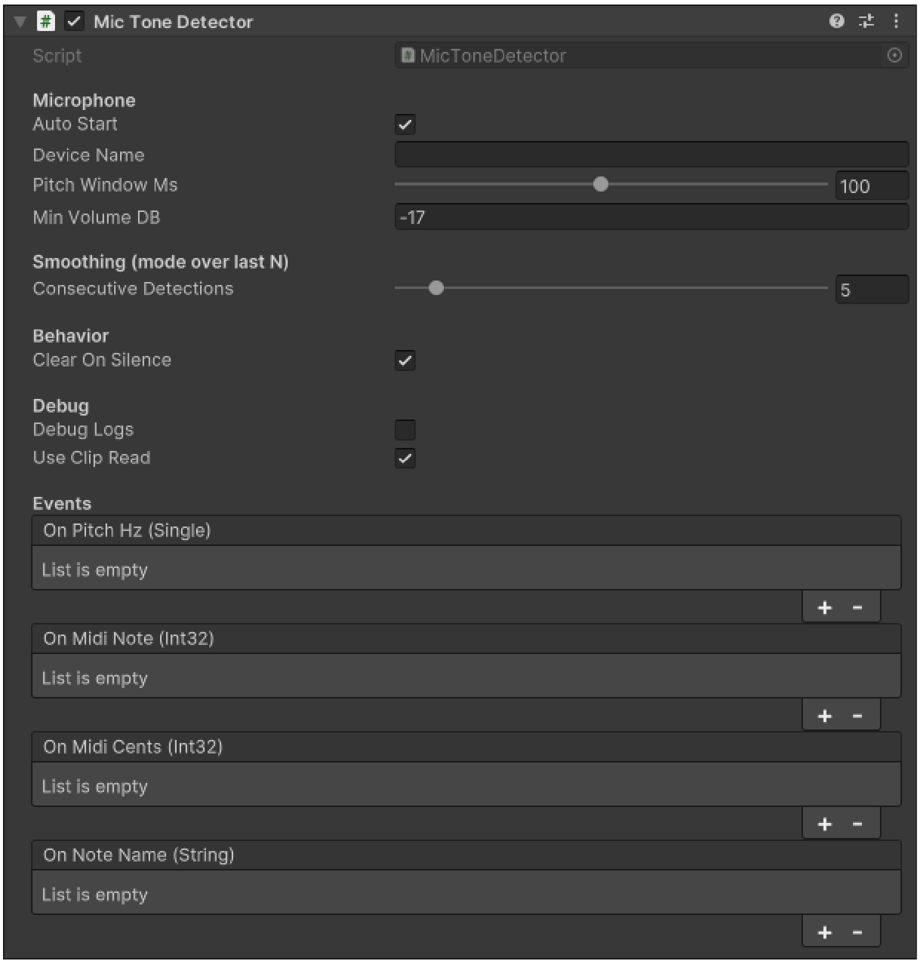
Prefab: MicToneDetector

MicToneDetector is your plug-and-play vocal pitch box for Unity. Drop it in a scene and turn voice into music data in real time:

- **Listen:** captures the microphone via a muted AudioSource to avoid feedback.
- **Filter + analyze:** voice-tuned DSP pipeline finds stable monophonic pitch.
- **Gate:** a level threshold decides when a tone is valid.
- **Deliver:** exposes frequency (Hz), MIDI note, cents, and a human-readable note name.
- **Emit:** fires **UnityEvents** so UI and gameplay can react without extra glue code.

Think of it as a "voice → game" adapter. Pick a mic, set the analysis window and volume gate (or simply leave as it comes), wire the events, and ship. The prefab handles the rest.

Public properties



- **autoStart (bool):** Flip it on and the mic spins up as soon as the object is enabled. Fire-and-forget.
- **deviceName (string):** Choose a mic by name. Empty picks the first device—good defaults, zero friction.
- **pitchWindowMs (10–200):** Analysis window. Bigger windows breathe stability; smaller windows snap faster.
- **minVolumeDB (float):** Level gate. Below this, detection rests. Keeps noise from masquerading as pitch.
- **consecutiveDetections (1–50):** Mode smoothing. A small memory that favors what you keep singing.
- **clearOnSilence (bool):** When the room goes quiet, emit clean zeros so your UI doesn't ghost old notes.
- **debugLogs (bool):** Switch on the cockpit lights. See RMS, dB, decisions.
- **useClipRead (bool):** Read mic clip directly. The straightest path from mouth to math.
- **LastFrequencyHz (ro):** Latest frequency in Hz. Zero means "no tone."
- **LastMidiNote (ro):** Latest MIDI note. Also zero when silent.
- **LastMidiNotePrecise (ro):** Note as float (note + cents/100) for tight intonation work.
- **LastCents (ro):** Offset in cents, −50..+50. The nudge you need to center a note.
- **LastNoteName (ro):** Friendly label, e.g., "A# 4".
- **Active (ro):** Mic running status.
- **ListDevices():** Snapshot of available input devices.
- **StartMic(string device = null):** Manual ignition.
- **StopMic():** Full stop.

Usage examples

Below are small, drop-in snippets that show common patterns:

- **Display** live data in UI (Hz, note name, MIDI, cents).
- **React** to specific notes with UnityEvents for gameplay or FX.
- **Select** a microphone programmatically at runtime.

Use them as templates: wire the prefab, paste a script, hit Play. Adjust thresholds and smoothing to fit your needs.

Prerequisite: add a MicToneDetector to your scene (the prefab or component).
Use these as templates: wire the prefab, paste a script, hit Play. Adjust thresholds and smoothing to fit your scene.

Display Hz / Note / MIDI in UI

Shows live pitch data on screen. Wire **Backbone Singer Tone** events (OnPitchHz, OnNoteName, OnMidiNote, OnMidiCents) to UI text fields to display frequency, note name, MIDI note, and cents in real time while singing.

```
using UnityEngine;
using TMPro; // or UnityEngine.UI

public class PitchHUD : MonoBehaviour
{
    public MicToneDetector mic;
    public TMP_Text hzLabel, noteLabel, midiLabel, centsLabel;

    void OnEnable()
    {
        mic.OnPitchHz.AddListener(OnHz);
        mic.OnNoteName.AddListener(OnNote);
        mic.OnMidiNote.AddListener(OnMidi);
        mic.OnMidiCents.AddListener(OnCents);
    }
    void OnDisable()
    {
        mic.OnPitchHz.RemoveListener(OnHz);
        mic.OnNoteName.RemoveListener(OnNote);
        mic.OnMidiNote.RemoveListener(OnMidi);
        mic.OnMidiCents.RemoveListener(OnCents);
    }

    void OnHz(float hz) { if (hzLabel) hzLabel.text = hz > 0 ? $"{hz:F2} Hz" : "—"; }
    void OnNote(string name) { if (noteLabel) noteLabel.text = string.IsNullOrEmpty(name) ? "—" : name; }
    void OnMidi(int note) { if (midiLabel) midiLabel.text = note > 0 ? note.ToString() : "—"; }
    void OnCents(int cents) { if (centsLabel) centsLabel.text = $"{cents:+#;-#;0} c"; }
}
```

React to notes with UnityEvents

Trigger gameplay or FX when specific notes are detected. Subscribe to **Backbone Singer Tone** events (e.g., OnMidiNote) and run your logic on matching notes—lights, animations, scoring, or state changes.

```
using UnityEngine;

public class LightOnNote : MonoBehaviour
{
    public MicToneDetector mic;
    public Light target;
    public int triggerNote = 69; // A4

    void OnEnable() => mic.OnMidiNote.AddListener(OnMidi);
    void OnDisable() => mic.OnMidiNote.RemoveListener(OnMidi);

    void OnMidi(int note)
    {
        if (!target) return;
        target.enabled = (note == triggerNote);
    }
}
```

Select microphone in code

Programmatically choose the input device at runtime. List Microphone.devices, pick a name (e.g., by keyword or index), and call StartMic("DeviceName") on **Backbone Singer Tone** to capture from that specific mic.

```
using UnityEngine;

public class MicSelector : MonoBehaviour
{
    public MicToneDetector mic;

    void Start()
    {
        var devices = mic.ListDevices();
        if (devices.Length == 0) { Debug.LogError("No microphones."); return; }

        // Example: pick by partial name or fallback to first
        string chosen = null;
        foreach (var d in devices) if (d.Contains("USB")) { chosen = d; break; }
        if (string.IsNullOrEmpty(chosen)) chosen = devices[0];

        mic.StartMic(chosen);
        Debug.Log($"Using mic: {chosen}");
    }
}
```

Troubleshooting

No pitch detected

- **Mic permission**: verify OS prompt was accepted. In Editor, check Console for permission errors.
- **Input level**: raise voice or lower minVolumeDB (e.g., to -25). Watch the dB logs by enabling debugLogs.
- **Window size**: increase pitchWindowMs to 120–160. Small windows can miss weak fundamentals.
- **Noise/bleed**: use a headset or move away from speakers. Disable reverb/FX.
- **Device**: confirm deviceName is valid: Debug.Log(string.Join(", ", Microphone.devices)).
- **Silence clearing**: if clearOnSilence is on, UI will show zeros below gate. That is expected.
- **Mono content**: the detector expects a single voiced source. Polyphonic audio will fail.

Permissions and WebGL specifics

- **Windows/macOS/Linux**: allow mic in OS privacy settings. Restart the app after changing it.
- **Android**: ensure runtime permission RECORD_AUDIO is granted. Test on device, not just in Editor.
- **iOS**: set NSMicrophoneUsageDescription in Info.plist with a non-empty message.

- **WebGL**:
    - Serve over **HTTPS**, not HTTP.
    - Browser prompts need a **user gesture** (click) before capture starts.
    - Some browsers block autoplay; call StartMic() from a button.
    - Check the browser console for permission/SSL errors.

Latency vs stability and performance

- **Lower latency**: decrease pitchWindowMs (e.g., 60–80). Expect more jitter; compensate with consecutiveDetections = 3–4.
- **Higher stability**: increase pitchWindowMs (e.g., 120–160) and consecutiveDetections = 5–8. Accept extra latency.
- **CPU**:
    - Keep useClipRead = true.
    - Avoid heavy per-frame listeners; batch UI updates.
    - Reduce log spam by disabling debugLogs.
- **Gate tuning**: set minVolumeDB just below typical voice level so silence clears quickly without cutting notes.
- **UI smoothing**: display OnNoteName and OnMidiNote from the **mode** output (already smoothed) to avoid flicker.

FAQ

Does it detect multiple voices or chords?
No. Monophonic voice only.

Will it work on instruments?
Often yes for clean monophonic sources, but it is tuned for singing.

Why do I get zeros on events?
Level gate is below threshold or silence. Lower minVolumeDB or disable clearOnSilence.

Latency feels high. How do I reduce it?
Lower pitchWindowMs (e.g., 60–80) and keep consecutiveDetections small. Expect more jitter.

Notes flicker. How do I stabilize?
Increase pitchWindowMs and consecutiveDetections, or smooth in UI.

How do I pick a specific mic?
Call mic.ListDevices(), choose a name, then mic.StartMic("DeviceName").

WebGL shows no input. Why?
Serve over HTTPS and start after a user gesture. Browser must grant mic permission.

Android/iOS permission issues?
Android: runtime RECORD_AUDIO. iOS: set NSMicrophoneUsageDescription in Info.plist.

Which sample rate is used?
Uses project output sample rate by default. Device caps may override max capture rate.

Can I run multiple detectors?
Yes, but each active mic/analysis costs CPU. Profile on target hardware.

Sharps vs flats in note names?
Output uses sharps by default (e.g., "A# 4"). Change using DSP helpers if needed.

What is the cents range?
-50..+50 around the detected MIDI note.

Does the prefab play audio back?
No. AudioSource is muted. It only routes mic data internally.

How do I start/stop at runtime?
StartMic() / StopMic() or set autoStart = true.

Does it allocate every frame?
No significant GC in steady state. Logs and custom listeners may allocate—keep them light.

Technical reference — read this before diving in

Backbone Singer Tone's runtime is a full DSP stack: windowed analysis, band-shaping IIR filters, coarse-to-fine pitch search, interpolation, overlap handling, and history smoothing. The code exposes many levers (filter prototypes, window sizes, offsets, thresholds) that interact in non-obvious ways. This reference is aimed at experienced developers who need to extend or audit the internals—e.g., swapping filter responses, experimenting with search curves, or integrating custom latency/smoothing strategies.

If you only need to **detect notes and drive gameplay**, stick to the prefab and high-level API. If you intend to modify the algorithms, read the class docs in order: SingerToneDSP → SingerToneTracker → IIRFilter → CircularBuffer<T> → SingerToneDetector. The Doxygen pages include parameter constraints, data flow, and edge-case notes.

Link: http://www.xavinet.com/singertone/