!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Sunn O)))-ish Drone — MIDI + Looper + WAV (Single File)</title>

<style>

:root { --bg:#000; --fg:#e5e5e5; --mut:#9ca3af; --card:#0b0b0b; --accent:#ef4444; --accent2:#10b981; }

html,body{height:100%;background:var(--bg);color:var(--fg);font-family:ui-sans-serif,system-ui,-apple-system,Segoe UI,Roboto,Inter,Helvetica,Arial,sans-serif;}

.wrap{max-width:1100px;margin:0 auto;padding:20px}

h1{font-size:22px;margin:0 0 12px 0;font-weight:900;letter-spacing:0.3px}

h2{font-size:16px;margin:0 0 10px 0;font-weight:700}

.grid{display:grid;gap:14px}

.g2{grid-template-columns:repeat(2,minmax(0,1fr))}

.g3{grid-template-columns:repeat(3,minmax(0,1fr))}

.card{background:var(--card);border:1px solid #1f2937;border-radius:16px;padding:14px}

.row{display:flex;gap:10px;align-items:center}

.btn{background:#111827;color:#e5e7eb;border:1px solid #374151;border-radius:12px;padding:8px 12px;cursor:pointer}

.btn:hover{background:#1f2937}

.btn.red{background:#7f1d1d;border-color:#b91c1c}

.btn.green{background:#064e3b;border-color:#10b981}

.btn.small{padding:6px 10px;font-size:12px}

.badge{font-size:11px;color:#e5e7eb;background:#111827;border:1px solid #374151;border-radius:999px;padding:3px 8px}

label{font-size:12px;color:var(--mut)}

input[type=range]{width:100%}

.mono{font-family:ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono", "Courier New", monospace}

.mut{color:var(--mut)}

.warn{color:#fca5a5}

.note{font-size:12px;opacity:.8}

.slot{border:1px dashed #334155;border-radius:12px;padding:10px}

</style>

</head>

<body>

<div class="wrap">

<header class="row" style="justify-content:space-between;align-items:center">

<h1>Sunn O)))-ish Drone — MIDI + Looper + WAV</h1>

<button id="power" class="btn green">Power On</button>

</header>

<p class="note">Start with your system volume <strong>low</strong>. Works offline. For MIDI in Firefox: open <span class="mono">about:config</span> and set <span class="mono">dom.webmidi.enabled</span> → <strong>true</strong>, then restart the tab.</p>

<div class="grid g2">

<section class="card">

<h2>Play Control</h2>

<div class="row" style="flex-wrap:wrap;gap:12px">

<button id="midiBtn" class="btn">MIDI: Off</button>

<span id="midiStatus" class="badge">No device</span>

<button id="manualBtn" class="btn">Manual Drone: Off</button>

<button id="panic" class="btn red">Panic (All Notes Off)</button>

</div>

<div class="grid g3" style="margin-top:12px">

<div>

<label>Fundamental (Hz)</label>

<input id="baseFreq" type="range" min="20" max="120" step="0.5" value="49">

<div class="mono" id="baseRead">49.0 Hz</div>

</div>

<div>

<label>Voices</label>

<input id="voices" type="range" min="1" max="12" step="1" value="4">

<div class="mono" id="voicesRead">4</div>

</div>

<div>

<label>Detune (±¢)</label>

<input id="detune" type="range" min="0" max="50" step="1" value="12">

<div class="mono" id="detuneRead">±12¢</div>

</div>

<div class="row" style="gap:18px">

<label><input id="subOct" type="checkbox" checked> Sub Oct</label>

<label><input id="fifth" type="checkbox" checked> + Fifth</label>

<label><input id="octUp" type="checkbox" checked> + Octave</label>

</div>

<div>

<label>Preamp Drive (dB)</label>

<input id="drive" type="range" min="0" max="60" step="1" value="36">

<div class="mono" id="driveRead">36 dB</div>

</div>

<div>

<label>Post Level (dB)</label>

<input id="postDb" type="range" min="-48" max="0" step="1" value="-18">

<div class="mono" id="postRead">-18 dB</div>

</div>

</div>

<div class="grid g3" style="margin-top:12px">

<div>

<label>Bass (dB)</label>

<input id="bass" type="range" min="-12" max="12" step="1" value="6">

<div class="mono" id="bassRead">6</div>

</div>

<div>

<label>Mids (dB)</label>

<input id="mids" type="range" min="-12" max="12" step="1" value="2">

<div class="mono" id="midsRead">2</div>

</div>

<div>

<label>Treble (dB)</label>

<input id="treble" type="range" min="-12" max="12" step="1" value="-4">

<div class="mono" id="trebleRead">-4</div>

</div>

<div>

<label>Low Cut (Hz)</label>

<input id="lowCut" type="range" min="10" max="60" step="1" value="25">

<div class="mono" id="lowRead">25</div>

</div>

<div>

<label>High Cut (Hz)</label>

<input id="highCut" type="range" min="1500" max="10000" step="100" value="8000">

<div class="mono" id="highRead">8000</div>

</div>

<div>

<label>Reverb Mix</label>

<input id="rvMix" type="range" min="0" max="1" step="0.01" value="0.4">

<div class="mono" id="rvMixRead">40%</div>

</div>

<div>

<label>Reverb Time (s)</label>

<input id="rvTime" type="range" min="2" max="20" step="0.5" value="10">

<div class="mono" id="rvTimeRead">10.0</div>

</div>

<div>

<label>Reverb Decay</label>

<input id="rvDecay" type="range" min="0.5" max="6" step="0.1" value="3.5">

<div class="mono" id="rvDecayRead">3.5</div>

</div>

<div>

<label>Amp Noise (dB)</label>

<input id="noiseDb" type="range" min="-60" max="-12" step="1" value="-36">

<div class="mono" id="noiseRead">-36 dB</div>

</div>

</div>

<p class="note">Velocity → drive; Pitch bend → slow detuned glides. Safety first: keep <span class="mono">Post Level</span> low and raise gradually.</p>

</section>

<section class="card">

<h2>Recorder (Master Out → WAV)</h2>

<div class="row" style="gap:10px;flex-wrap:wrap">

<button id="recBtn" class="btn">Record</button>

<button id="stopRecBtn" class="btn">Stop & Save WAV</button>

<a id="downloadLink" class="btn small" style="display:none" download="doom-drone.wav">Download</a>

<span id="recStatus" class="badge">Idle</span>

</div>

<p class="note">Records exactly what you hear. The file saves locally (no server).</p>

<hr style="border:none;border-top:1px solid #1f2937;margin:10px 0" />

<h2>Loopers (x4)</h2>

<div class="row" style="gap:14px;margin-bottom:8px">

<label><input id="includeLoops" type="checkbox"> Include other loops in new recordings (overdub)</label>

</div>

<div class="grid g2">

<div class="slot" id="slot0"></div>

<div class="slot" id="slot1"></div>

<div class="slot" id="slot2"></div>

<div class="slot" id="slot3"></div>

</div>

</section>

</div>

<section class="card" style="margin-top:14px">

<h2>Tips</h2>

<ul class="note">

<li>For monolithic weight: enable <span class="mono">Sub Oct</span>, set <span class="mono">Low Cut</span> 18–24 Hz, keep <span class="mono">High Cut</span> ~4–6 kHz.</li>

<li>For smeared walls: widen <span class="mono">Detune</span> (±18–22¢) and lengthen reverb.</li>

<li>Use the loopers to build layers: bass bed → mid growl → airy overtone → noise layer.</li>

</ul>

</section>

<footer class="note" style="margin-top:10px">Homage to the aesthetic of heavy sustained drones. No endorsement implied.</footer>

</div>

<script>

(function(){

const el = id=>document.getElementById(id);

const ctxObj = { ctx:null };

let master, preIn, postGain, waveShaper, lowShelf, midPeak, highShelf, hp, lp, verb, verbWet, verbDry, comp, noiseSrc, noiseGain;

let loopBus, loopMixOut; // bus for summing loops

let includeLoopsInCapture = false;

const state = {

isOn:false,

midiEnabled:false,

manualOn:false,

activeNotes:new Map(), // midiNote -> {osc[], gain[]}

pitchBend:0,

};

function dbToGain(db){ return Math.pow(10, db/20); }

function createWaveshaperCurve(amount=400, samples=44100){

const curve = new Float32Array(samples);

const k = typeof amount === 'number' ? amount : 50;

const deg = Math.PI/180;

for(let i=0;i<samples;i++){

const x = (i\*2)/samples - 1;

curve[i] = ((3+k)\*x\*20\*deg) / (Math.PI + k\*Math.abs(x));

}

return curve;

}

function generateIR(ctx, duration=6, decay=3, reverse=false){

const rate = ctx.sampleRate;

const length = rate\*duration;

const impulse = ctx.createBuffer(2, length, rate);

for(let ch=0; ch<2; ch++){

const data = impulse.getChannelData(ch);

for(let i=0;i<length;i++){

const n = reverse ? length - i : i;

data[i] = (Math.random()\*2-1) \* Math.pow(1 - n/length, decay);

}

}

return impulse;

}

function midiToFreq(n){ return 440 \* Math.pow(2, (n-69)/12); }

function ensureCtx(){

if(ctxObj.ctx) return;

const ctx = new (window.AudioContext||window.webkitAudioContext)({latencyHint:'interactive'});

ctxObj.ctx = ctx;

master = ctx.createGain(); master.gain.value = 0.9; master.connect(ctx.destination);

// Preamp chain entry for synth voices

preIn = ctx.createGain();

waveShaper = ctx.createWaveShaper();

waveShaper.curve = createWaveshaperCurve(400);

lowShelf = ctx.createBiquadFilter(); lowShelf.type='lowshelf'; lowShelf.frequency.value=80;

midPeak = ctx.createBiquadFilter(); midPeak.type='peaking'; midPeak.frequency.value=700; midPeak.Q.value=0.7;

highShelf = ctx.createBiquadFilter(); highShelf.type='highshelf'; highShelf.frequency.value=3000;

hp = ctx.createBiquadFilter(); hp.type='highpass';

lp = ctx.createBiquadFilter(); lp.type='lowpass';

verb = ctx.createConvolver(); verb.buffer = generateIR(ctx, 10, 3.5, false);

verbWet = ctx.createGain(); verbDry = ctx.createGain(); verbWet.gain.value=0.4; verbDry.gain.value=0.6;

comp = ctx.createDynamicsCompressor();

comp.threshold.value = -28; comp.knee.value = 24; comp.ratio.value = 8; comp.attack.value = 0.02; comp.release.value = 0.3;

postGain = ctx.createGain(); postGain.gain.value = dbToGain(-18);

// Loop bus

loopBus = ctx.createGain(); // where all loop players sum

loopMixOut = ctx.createGain(); // loop overall volume

loopBus.connect(loopMixOut);

// Amp noise

const noiseBuf = ctx.createBuffer(1, ctx.sampleRate\*5, ctx.sampleRate);

const nd = noiseBuf.getChannelData(0); for(let i=0;i<nd.length;i++){ nd[i] = (Math.random()\*2-1)\*0.3; }

noiseSrc = ctx.createBufferSource(); noiseSrc.buffer = noiseBuf; noiseSrc.loop = true;

noiseGain = ctx.createGain(); noiseGain.gain.value = dbToGain(-36);

// Wire synth preamp path

preIn.connect(waveShaper);

waveShaper.connect(lowShelf);

lowShelf.connect(midPeak);

midPeak.connect(highShelf);

highShelf.connect(hp);

hp.connect(lp);

// Split to reverb

const sum = ctx.createGain();

lp.connect(verbDry);

lp.connect(verb);

verb.connect(verbWet);

verbWet.connect(sum);

verbDry.connect(sum);

// Add loop bus into main sum

loopMixOut.connect(sum);

// Final stages

sum.connect(comp);

comp.connect(postGain);

postGain.connect(master);

// Start noise

noiseSrc.connect(noiseGain); noiseGain.connect(preIn); noiseSrc.start();

// Expose taps for recorders

ctx.\_sumNode = sum; // pre-comp sum

ctx.\_postNode = postGain; // post-comp/postgain (what you hear)

ctx.\_preIn = preIn;

ctx.\_loopBus = loopBus;

// AudioWorklet for WAV recording

if (ctx.audioWorklet) {

const workletCode = `

class RecorderProcessor extends AudioWorkletProcessor {

constructor(){

super();

this.\_bufferL = [];

this.\_bufferR = [];

this.\_isRecording = false;

this.port.onmessage = (e)=>{

const {cmd} = e.data;

if(cmd==='start'){ this.\_isRecording=true; this.\_bufferL=[]; this.\_bufferR=[]; }

if(cmd==='stop'){ this.\_isRecording=false; this.port.postMessage({cmd:'data', left:this.\_bufferL, right:this.\_bufferR}); }

};

}

process(inputs){

if(!this.\_isRecording) return true;

const input = inputs[0];

if(input && input[0]){

const ch0 = input[0];

const ch1 = input[1] || input[0];

this.\_bufferL.push(new Float32Array(ch0));

this.\_bufferR.push(new Float32Array(ch1));

}

return true;

}

}

registerProcessor('recorder-processor', RecorderProcessor);

`;

const blob = new Blob([workletCode], {type:'application/javascript'});

const url = URL.createObjectURL(blob);

ctx.audioWorklet.addModule(url).then(()=>{

// ready

});

}

}

function setUI(){

// Ranges → labels

const bind = (id, out, fmt=(v)=>v)=>{

const inp = el(id), lab = el(out);

const update = ()=>{ lab.textContent = fmt(inp.value); applyParam(id, parseFloat(inp.value)); };

inp.addEventListener('input', update); update();

};

bind('baseFreq','baseRead',v=>parseFloat(v).toFixed(1)+' Hz');

bind('voices','voicesRead',v=>v);

bind('detune','detuneRead',v=>'±'+v+'¢');

bind('drive','driveRead',v=>v+' dB');

bind('postDb','postRead',v=>v+' dB');

bind('bass','bassRead',v=>v);

bind('mids','midsRead',v=>v);

bind('treble','trebleRead',v=>v);

bind('lowCut','lowRead',v=>v);

bind('highCut','highRead',v=>v);

bind('rvMix','rvMixRead',v=>(Math.round(v\*100))+'%');

bind('rvTime','rvTimeRead',v=>parseFloat(v).toFixed(1));

bind('rvDecay','rvDecayRead',v=>parseFloat(v).toFixed(1));

el('subOct').addEventListener('change', ()=>applyParam('partials'));

el('fifth').addEventListener('change', ()=>applyParam('partials'));

el('octUp').addEventListener('change', ()=>applyParam('partials'));

el('power').addEventListener('click', async()=>{

ensureCtx();

if(!state.isOn){ await ctxObj.ctx.resume(); state.isOn=true; el('power').textContent='Power Off'; el('power').classList.remove('green'); el('power').classList.add('red'); }

else { killAllNotes(); state.isOn=false; el('power').textContent='Power On'; el('power').classList.add('green'); el('power').classList.remove('red'); }

});

el('manualBtn').addEventListener('click',()=>{

if(!state.manualOn){ startManualDrone(); state.manualOn=true; el('manualBtn').textContent='Manual Drone: On'; }

else { stopManualDrone(); state.manualOn=false; el('manualBtn').textContent='Manual Drone: Off'; }

});

el('panic').addEventListener('click', ()=>{ killAllNotes(); });

el('midiBtn').addEventListener('click', ()=>{

if(!state.midiEnabled) enableMIDI(); else disableMIDI();

});

el('includeLoops').addEventListener('change', (e)=>{ includeLoopsInCapture = e.target.checked; });

setupRecorderUI();

setupLoopers();

}

function applyParam(id, val){

if(!ctxObj.ctx) return;

const ctx = ctxObj.ctx;

switch(id){

case 'bass': lowShelf.gain.setTargetAtTime(val, ctx.currentTime, 0.01); break;

case 'mids': midPeak.gain.setTargetAtTime(val, ctx.currentTime, 0.01); break;

case 'treble': highShelf.gain.setTargetAtTime(val, ctx.currentTime, 0.01); break;

case 'lowCut': hp.frequency.setTargetAtTime(parseFloat(el('lowCut').value), ctx.currentTime, 0.01); break;

case 'highCut': lp.frequency.setTargetAtTime(parseFloat(el('highCut').value), ctx.currentTime, 0.01); break;

case 'rvMix': verbWet.gain.setTargetAtTime(parseFloat(el('rvMix').value), ctx.currentTime, 0.05); verbDry.gain.setTargetAtTime(1-parseFloat(el('rvMix').value), ctx.currentTime, 0.05); break;

case 'rvTime': verb.buffer = generateIR(ctx, Math.max(1, parseFloat(el('rvTime').value)), Math.max(0.1, parseFloat(el('rvDecay').value))); break;

case 'rvDecay': verb.buffer = generateIR(ctx, Math.max(1, parseFloat(el('rvTime').value)), Math.max(0.1, parseFloat(el('rvDecay').value))); break;

case 'postDb': postGain.gain.setTargetAtTime(dbToGain(parseFloat(el('postDb').value)), ctx.currentTime, 0.03); break;

case 'drive': /\* handled per-voice when spawning \*/ break;

case 'partials': /\* next note will use \*/ break;

default: break;

}

}

// ---------- Manual Drone (non-MIDI) ----------

let manualNodes = [];

function makeOscStack(freq, velocity=100){

const ctx = ctxObj.ctx;

const voices = parseInt(el('voices').value);

const detune = parseInt(el('detune').value);

const baseDrive = parseFloat(el('drive').value);

const partials = [freq];

if(el('subOct').checked) partials.push(freq/2);

if(el('fifth').checked) partials.push(freq\*1.5);

if(el('octUp').checked) partials.push(freq\*2);

const oscNodes = [];

partials.forEach(pf=>{

for(let v=0; v<voices; v++){

const osc = ctx.createOscillator();

osc.type = (v%2===0)?'sawtooth':'triangle';

const cents = (Math.random()\*2-1)\*detune + state.pitchBend; // include bend

osc.detune.value = cents;

osc.frequency.value = pf;

const preGain = ctx.createGain();

const velDb = (velocity-64)/64 \* 6; // +/- ~6 dB swing from velocity

const perVoiceDb = baseDrive - 20\*Math.log10(voices\*partials.length) + velDb;

preGain.gain.value = dbToGain(perVoiceDb);

osc.connect(preGain).connect(ctx.\_preIn);

osc.start();

oscNodes.push({osc, preGain});

}

});

return oscNodes;

}

function startManualDrone(){ ensureCtx(); stopManualDrone(); manualNodes = makeOscStack(parseFloat(el('baseFreq').value)); }

function stopManualDrone(){ if(!manualNodes) return; manualNodes.forEach(n=>{ try{n.osc.stop();}catch(e){}; try{n.osc.disconnect();}catch(e){} }); manualNodes=[]; }

// Reflect UI value live change

el('baseFreq')?.addEventListener('input', ()=>{ if(state.manualOn){ stopManualDrone(); startManualDrone(); } });

// ---------- MIDI ----------

let midiAccess = null;

async function enableMIDI(){

try{

ensureCtx();

midiAccess = await navigator.requestMIDIAccess({sysex:false});

midiAccess.inputs.forEach(input=> input.onmidimessage = onMIDIMessage);

midiAccess.onstatechange = (e)=>{ el('midiStatus').textContent = midiDeviceSummary(); };

state.midiEnabled = true; el('midiBtn').textContent='MIDI: On';

el('midiStatus').textContent = midiDeviceSummary();

}catch(err){ el('midiStatus').textContent = 'MIDI unavailable'; }

}

function disableMIDI(){

if(midiAccess){ midiAccess.inputs.forEach(input=> input.onmidimessage = null); }

state.midiEnabled=false; el('midiBtn').textContent='MIDI: Off'; el('midiStatus').textContent='Disabled';

}

function midiDeviceSummary(){

const ins = []; midiAccess && midiAccess.inputs.forEach(i=>ins.push(i.name));

return ins.length? ('Input: '+ins.join(', ')) : 'No device';

}

function onMIDIMessage(e){

const [status, d1, d2] = e.data; const cmd = status>>4; const ch = status & 0xf;

if(cmd===9 && d2>0){ noteOn(d1, d2); }

else if((cmd===8) || (cmd===9 && d2===0)){ noteOff(d1); }

else if(cmd===14){ // pitch bend

const value = ((d2<<7) + d1) - 8192; // -8192..+8191

const bendCents = (value/8192) \* 100; // +/- 100 cents

state.pitchBend = bendCents;

// update detune of all active note oscs

state.activeNotes.forEach(stack=>{

stack.forEach(n=>{ if(n.osc) n.osc.detune.value = n.osc.detune.value + 0; /\* already drifting; ignore \*/ });

});

}

}

function noteOn(note, vel){ ensureCtx(); if(state.manualOn) { /\* can layer with manual \*/ }

if(state.activeNotes.has(note)) return; // already sounding

const freq = midiToFreq(note);

const nodes = makeOscStack(freq, vel);

state.activeNotes.set(note, nodes);

}

function noteOff(note){ const nodes = state.activeNotes.get(note); if(!nodes) return; nodes.forEach(n=>{ try{n.osc.stop();}catch(e){}; try{n.osc.disconnect();}catch(e){} }); state.activeNotes.delete(note); }

function killAllNotes(){ state.activeNotes.forEach((nodes)=>{ nodes.forEach(n=>{ try{n.osc.stop();}catch(e){}; try{n.osc.disconnect();}catch(e){} }); }); state.activeNotes.clear(); stopManualDrone(); }

// ---------- WAV Recorder (Master Out) via AudioWorklet ----------

let recNode=null, recIn=null, recCollecting=false, recBuffers=null;

function setupRecorderUI(){

el('recBtn').addEventListener('click', startMasterRecord);

el('stopRecBtn').addEventListener('click', stopMasterRecord);

}

async function ensureRecorder(){ ensureCtx(); const ctx = ctxObj.ctx; if(recNode) return; if(!ctx.audioWorklet){ el('recStatus').textContent='AudioWorklet not supported'; return; }

await new Promise(r=> setTimeout(r,0));

recIn = ctx.createGain();

ctx.\_postNode.connect(recIn); // tap what you hear

recNode = new AudioWorkletNode(ctx, 'recorder-processor');

recIn.connect(recNode);

recBuffers = {L:[], R:[]};

recNode.port.onmessage = (e)=>{

if(e.data.cmd==='data'){

recBuffers.L = e.data.left; recBuffers.R = e.data.right;

const wav = encodeWAV(recBuffers, ctx.sampleRate);

const url = URL.createObjectURL(wav);

const a = el('downloadLink'); a.href=url; a.style.display='inline-block';

}

};

}

async function startMasterRecord(){ await ensureRecorder(); if(!recNode) return; recNode.port.postMessage({cmd:'start'}); recCollecting=true; el('recStatus').textContent='Recording…'; }

function stopMasterRecord(){ if(!recNode||!recCollecting) return; recNode.port.postMessage({cmd:'stop'}); recCollecting=false; el('recStatus').textContent='Stopped'; }

function mergeFloat32(arrays){ let total=0; arrays.forEach(a=> total+=a.length); const out = new Float32Array(total); let off=0; arrays.forEach(a=>{ out.set(a, off); off+=a.length; }); return out; }

function floatTo16BitPCM(view, offset, input){ for (let i = 0; i < input.length; i++, offset += 2) { let s = Math.max(-1, Math.min(1, input[i])); view.setInt16(offset, s < 0 ? s \* 0x8000 : s \* 0x7FFF, true); } return offset; }

function writeString(view, offset, str){ for (let i = 0; i < str.length; i++) { view.setUint8(offset + i, str.charCodeAt(i)); } }

function encodeWAV(buffers, sampleRate){

const left = mergeFloat32(buffers.L); const right = mergeFloat32(buffers.R);

const interleaved = new Float32Array(left.length + right.length);

for(let i=0,j=0;i<left.length;i++,j+=2){ interleaved[j]=left[i]; interleaved[j+1]=right[i]; }

const buffer = new ArrayBuffer(44 + interleaved.length \* 2);

const view = new DataView(buffer);

writeString(view, 0, 'RIFF');

view.setUint32(4, 36 + interleaved.length \* 2, true);

writeString(view, 8, 'WAVE');

writeString(view, 12, 'fmt ');

view.setUint32(16, 16, true);

view.setUint16(20, 1, true);

view.setUint16(22, 2, true);

view.setUint32(24, sampleRate, true);

view.setUint32(28, sampleRate \* 4, true);

view.setUint16(32, 4, true);

view.setUint16(34, 16, true);

writeString(view, 36, 'data');

view.setUint32(40, interleaved.length \* 2, true);

floatTo16BitPCM(view, 44, interleaved);

return new Blob([view], {type:'audio/wav'});

}

// ---------- Loopers (x4) ----------

const loopSlots = [0,1,2,3].map(i=>({index:i, recNode:null, recIn:null, buffers:null, bufSrc:null, gain:null, isRecording:false, isPlaying:false}));

function setupLoopers(){ loopSlots.forEach(initLoopSlot); }

async function initLoopSlot(slot){ ensureCtx(); const div = document.getElementById('slot'+slot.index); const ctx = ctxObj.ctx;

div.innerHTML = `

<div class="row" style="justify-content:space-between">

<strong>Loop ${slot.index+1}</strong>

<span class="badge" id="lstat${slot.index}">Empty</span>

</div>

<div class="row" style="flex-wrap:wrap;gap:8px;margin-top:8px">

<button class="btn small" id="lrec${slot.index}">Rec</button>

<button class="btn small" id="lstop${slot.index}">Stop</button>

<button class="btn small" id="lplay${slot.index}">Play</button>

<button class="btn small" id="lclear${slot.index}">Clear</button>

<label>Vol <input type="range" id="lgain${slot.index}" min="0" max="1.5" step="0.01" value="1" /></label>

</div>

`;

await ensureLoopRecorder(slot);

el('lrec'+slot.index).addEventListener('click', ()=> startLoopRecord(slot));

el('lstop'+slot.index).addEventListener('click', ()=> stopLoopRecord(slot));

el('lplay'+slot.index).addEventListener('click', ()=> toggleLoop(slot));

el('lclear'+slot.index).addEventListener('click', ()=> clearLoop(slot));

el('lgain'+slot.index).addEventListener('input', (e)=>{ if(slot.gain) slot.gain.gain.value = parseFloat(e.target.value); });

}

async function ensureLoopRecorder(slot){ const ctx = ctxObj.ctx; if(slot.recNode) return; if(!ctx.audioWorklet) return;

slot.recIn = ctx.createGain();

// Capture from synth sum + optionally existing loops

const captureSource = ctx.createGain();

ctx.\_sumNode.connect(captureSource);

captureSource.gain.value = 1.0;

captureSource.connect(slot.recIn);

// Include loops option handled by also connecting loop bus

if(includeLoopsInCapture){ ctx.\_loopBus.connect(slot.recIn); }

slot.recNode = new AudioWorkletNode(ctx, 'recorder-processor');

slot.recIn.connect(slot.recNode);

slot.buffers = {L:[], R:[]};

slot.recNode.port.onmessage = (e)=>{

if(e.data.cmd==='data'){

slot.buffers.L = e.data.left; slot.buffers.R = e.data.right;

// Create AudioBuffer from floats

const L = mergeFloat32(slot.buffers.L); const R = mergeFloat32(slot.buffers.R);

const len = Math.min(L.length, R.length);

const ab = ctx.createBuffer(2, len, ctx.sampleRate);

ab.copyToChannel(L.subarray(0,len),0); ab.copyToChannel(R.subarray(0,len),1);

// Build/replace player

if(slot.bufSrc){ try{slot.bufSrc.stop();}catch(e){}; try{slot.bufSrc.disconnect();}catch(e){} }

const src = ctx.createBufferSource(); src.buffer = ab; src.loop = true;

const g = ctx.createGain(); g.gain.value = parseFloat(document.getElementById('lgain'+slot.index).value);

src.connect(g).connect(ctx.\_loopBus);

slot.bufSrc = src; slot.gain = g;

el('lstat'+slot.index).textContent = 'Captured ('+(ab.duration.toFixed(2))+'s)';

}

};

}

function startLoopRecord(slot){ if(slot.isRecording) return; slot.recNode.port.postMessage({cmd:'start'}); slot.isRecording=true; el('lstat'+slot.index).textContent='Recording…'; }

function stopLoopRecord(slot){ if(!slot.isRecording) return; slot.recNode.port.postMessage({cmd:'stop'}); slot.isRecording=false; }

function toggleLoop(slot){ if(!slot.bufSrc){ el('lstat'+slot.index).textContent='Empty'; return; } if(slot.isPlaying){ try{slot.bufSrc.stop();}catch(e){} slot.isPlaying=false; el('lstat'+slot.index).textContent='Stopped'; } else { const src = ctxObj.ctx.createBufferSource(); src.buffer = slot.bufSrc.buffer; src.loop = true; const g = ctxObj.ctx.createGain(); g.gain.value = slot.gain?slot.gain.gain.value:1; src.connect(g).connect(ctxObj.ctx.\_loopBus); src.start(); slot.bufSrc = src; slot.gain = g; slot.isPlaying=true; el('lstat'+slot.index).textContent='Playing'; } }

function clearLoop(slot){ if(slot.bufSrc){ try{slot.bufSrc.stop();}catch(e){}; try{slot.bufSrc.disconnect();}catch(e){} } slot.bufSrc=null; slot.buffers={L:[],R:[]}; slot.isPlaying=false; el('lstat'+slot.index).textContent='Empty'; }

// ---------- Start ----------

setUI();

})();

</script>

</body>

</html>