```
//*********************************************************
//        VIOLIN2.3. Jeremy Leach 2nd September 2023
//*********************************************************

//IMPORTANT:
//1. You must have V9 or higher of the ToneProcessor Firmware to use DepthCC:Modulation for Vibrato and Tremolo as shown in the LFO section below (because V9 fixes a bug). If you haven't got this firmware loaded then set these to DepthCC:None
//2.    Enable the 'Harmonic pruning' option in Tools>Settings to benefit from faster timbre refresh rate.

//INFORMATION ABOUT A VIOLIN TONE :
//==========================
//The range of a Violin is from G3 to A7, so the harmonic levels of the lowest note sectors (C2 and below, C#2 to E3) can be left at zero.
//The basic waveform of a violin needs to be a Sawtooth. The action of the bow upon the string (it's a repeated slipping and grabbing action) and the action of the string upon the bridge produce a sawtooth wave in the bridge itself.
//This sound energy is modified by the many bridge and body resonances.

//GENERAL APPROACH:
//==============
//In this patch, we're going to :
//1. Use all 5 waveforms in the Waveform Block, have them all set to identical saw-like waveforms, apart from some harmonic randomness that we add,    and slowly modulate across these waveforms.
//2. Use a dramatically set filter, with lots of resonant peaks, because this is a hallmark of stringed instrument acoustics.
//3. Use delayed Vibrato and Tremolo to simulate how a player expresses notes. We're going to control the depth of this expression with the modulation wheel.
//4. Low-pass filter the harmonics of the lower intensities, and also the higher note ranges.
//5. Give lower notes a slightly longer release time, because in reality the lower strings will 'ring' a little longer after playing.
//6. Use the Railsback inharmonicity algorithm, because violin strings have stiffness that result in inharmonicity.

//Optional first step to set the patch to a default state.
//Run("\ClearAll.txt");

Define.WaveSet
{
        //First clear all harmonic levels in the current WaveSet.
        WaveSet.SetCurrentNoteSector(NoteSector:0);
        WaveSet.SetCurrentIntensityLayer(IntensityLayer:0);
        CurrentWaveformBlock.SetCurrentWaveform(Waveform:0);
        CurrentWaveform.SetHarmonicLevelsFromCSV(LevelCSV:0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
        CurrentWaveform.CopyAcrossAllDimensions();

        //Select Waveform0 in the waveform block specified by note sector 2 (F3 to G#4, which covers G3 the lowest violin note) and intensity layer 0.
        WaveSet.SetCurrentNoteSector(NoteSector:2);
        WaveSet.SetCurrentIntensityLayer(IntensityLayer:0);
        CurrentWaveformBlock.SetCurrentWaveform(Waveform:0);

        //Set Waveform0 to a sawtooth.
        CurrentWaveform.SetHarmonicLevelsFromWaveformShape(WaveformShape:Saw);
        //Raise some of the upper harmonics to make it brighter. This seems to help.
        CurrentWaveform.SetHarmonicLevel(HarmonicID:4,Level:25000);
        CurrentWaveform.SetHarmonicLevel(HarmonicID:6,Level:20000);
        CurrentWaveform.SetHarmonicLevel(HarmonicID:8,Level:10000);

        //Copy the sawtooth to all the other waveforms in the Waveform Block.
        CurrentWaveform.CopyToRange(WaveformFrom:0,WaveformTo:4);

        //Vary tone
        CurrentWaveform.ScaleHarmonicLevelsFromCSV(ScaleCSV:1.000,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5);
        CurrentWaveformBlock.SetCurrentWaveform(Waveform:1);
        CurrentWaveform.ScaleHarmonicLevelsFromCSV(ScaleCSV:1.000,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8);
        CurrentWaveformBlock.SetCurrentWaveform(Waveform:2);
        CurrentWaveform.ScaleHarmonicLevelsFromCSV(ScaleCSV:1.000,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8,0.8);

        //Now copy this Waveform block across NoteSectors 2 to 4 and across all Intensity Layers.
        CurrentWaveformBlock.CopyAcrossNoteSectors(NoteSectorFrom:2,NoteSectorTo:4);
        CurrentIntensityLayer.CopyToRange(IntensityLayerFrom:0,IntensityLayerTo:2);

        //For intensity layer 0 , we want to mute the high frequencies a little, because when the Vioilin is played softly there is less energy being delivered to the instrument , and since high frequency evergy gets damped faster then this is effectively low-pass filtering.
        CurrentIntensityLayer.ShapeTheHarmonics(Direction:Up,HarmonicIDFrom:2,Slope:-3dB);

        //We now want to morph the harmonics between Intensity layer 0 and 2, so that layer1 is a midway timbre.
        WaveSet.MorphAcrossIntensityLayers(IntensityLayerFrom:0,IntensityLayerTo:2);

        //We also want to reduce the overall gain of the highest note sector (C#6 and above), to make it just a little less shrill
        WaveSet.HighestNoteSectorGain(80%);

        //We sprinkle in some randomness into the Harmonic levels across the timbral landscape, to add authenticity.
        //This also means that all waveforms are noticeably different when we apply slow timbral modulation between them.
        WaveSet.AddHarmonicRandomness(25%);
}

Define.Filter
{
        //Violin bridges and bodies have multiple, very narrow resonant frequencies (also called formants). When a player adds vibrato this may push a tone in and out of a peak formant region, making for a characteristic,    very dynamic sound.
        //Multiple formants are generally below 2KHz, with an overall peak frequency response at 2KHz to 4KHz, which happens to be the range Humans are most sensitive to.    Above 4KHz the frequency response drops quite sharply at around 12dB/Octave.
        //There are limitations on how narrow we can make the resonance peaks on the module. Also it's unlikely that the Timbre refresh rate will be high enough to pick up on pitch Vibrato taking the pitch in and out of a formant peak.
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:0,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:1,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:2,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:3,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:4,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:5,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:6,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:7,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:8,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:9,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:10,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:11,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:12,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:13,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:14,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:15,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:16,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:17,Level:2);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:18,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:19,Level:1);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:20,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:21,Level:3);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:22,Level:4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:23,Level:4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:24,Level:3);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:25,Level:2);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:26,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:27,Level:0);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:28,Level:-1);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:29,Level:-4);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:30,Level:-3);
        CurrentFilter.SetFilterBandLevel(FilterBandNumber:31,Level:-3);
}
Define.Patch
{
        //===== GENERAL : Misc ==========
        //Add 1 cent of random detuning, because even the best player will have some inaccuracy.
        General.SetDetuning(DetuningType:Random,Detuning:1.0%);
        General.SetDetuning(DetuningType:Regular,Detuning:0%);
        //We turn on Railsback Inharmonicity, because a violin has wire strings with stiffness that will result in a certain amount of inharmonicity.
        General.SetHarmonicAlgorithm(HarmonicAlgorithm:RailsbackInharmonicity);
        General.SetPortamento(Enabled:False,PortamentoAmount:100%);
        General.SetScalingSplit(Note:g3);

        //The timbre can overload a little so reduce the patch gain
        General.SetPatchGain(Gain:45%);

        //===== GENERAL : Oscillators ==========
        General.SetActiveOscillators(OscillatorCount:2);
        General.SetDetuningMode(DetuningMode:Cents);
        General.SetOscillatorDetuning(Oscillator:0,Detuning:0.00);
        General.SetOscillatorDetuning(Oscillator:1,Detuning:0.50);
```

```
    //===== GENERAL : Envelope control ==========
    General.SetEnvelopeGainController(Envelope:Amplitude,InitialLevel:0,GainCC:Velocity);
    General.SetEnvelopeGainController(Envelope:TimbreLFODepth,InitialLevel:0,GainCC:Velocity);
    General.SetEnvelopeGainController(Envelope:TimbreMorph,InitialLevel:100,GainCC:Velocity);

    //===== GENERAL : LFOs ==========
    //Violin Vibrato is generally at a rate of 3-8Hz with 25-35 cent pitch deviation.    Control depth via modulation wheel.
    General.SetLFO(LFOType:Vibrato,Enabled:True,WaveType:Sine,Frequency:3.00,FrequencyCC:None,DepthCC:Modulation);
    //Surprisingly, due to the physics of the violin, the player's vibrato also creates amplitude modulation (tremolo). Control depth via modulation wheel.
    General.SetLFO(LFOType:Tremolo,Enabled:True,WaveType:Sine,Frequency:3.00,FrequencyCC:None,DepthCC:Modulation);
    //We're using TimbreLFO to simulate the bowing action, where it's modulating between two slightly different timbres.
    //Bowing rate will depend on the music tempo etc, but here we have to pick one value. Let's assume in 2 seconds the bow goes back and forth. A triangle LFO wave makes sense, to linearly move from one timbre to the other.
    General.SetLFO(LFOType:TimbreLFO,Enabled:True,WaveType:Triangle,Frequency:3.0,FrequencyCC:None,DepthCC:None);

    //Amplitude Envelope: Relatively slow swell compared to a piano. Constant through sustain and release fading quickly rather than an abrupt cutoff.
    //Vibrato Envelope: Violin Players tend to introduce it after the initial bowing action, and modify it to suit the requirements of the music.
    //Timbre Morph envelope: Starts at Waveform0 then in the attack phase morphs to Waveform2 (50%).
    //Timbre Morph FLO Depth: Depth zero for attack and decay, then in sustain phase increases in depth to 50% so we get a modulation between Waveform2 and Waveform1. See Help>Desiging and editing patches>LFO Masterclass to understand how this works.

    //===== ADSR SECTIONS ==========
    ADSR.ConfigureSection(Section:Attack,Enabled:True,Duration:50,EndKSU:0%,EndKSL:0%,Sample:None,SampleMode:OneShot);
    ADSR.ConfigureSection(Section:Decay,Enabled:True,Duration:100,EndKSU:0%,EndKSL:0%,Sample:None,SampleMode:OneShot);
    ADSR.ConfigureSection(Section:Sustain,Enabled:True,Duration:60000,EndKSU:0%,EndKSL:0%,Sample:None,SampleMode:OneShot);
    //Add some key-scaling of the release duration, so that the lower notes have longer duration.
    ADSR.ConfigureSection(Section:Release,Enabled:True,Duration:700,EndKSU:-1%,EndKSL:0%,Sample:None,SampleMode:Looped);

    //----- ENVELOPES : for ADSR section 'Attack' ----------
    ADSR.ConfigureEnvelope(Section:Attack,Envelope:Amplitude,EnvelopeType:Exponential,Target:100.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:1%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Attack,Envelope:VibratoDepth,EnvelopeType:Exponential,Target:00.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:2%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Attack,Envelope:TremoloDepth,EnvelopeType:Exponential,Target:0.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:2%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Attack,Envelope:TimbreMorph,EnvelopeType:Linear,Target:100.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:1,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:3%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Attack,Envelope:TimbreLFODepth,EnvelopeType:Exponential,Target:0.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:4%,ExpMultKSU:0%,ExpMultKSL:0%);

    //----- ENVELOPES : for ADSR section 'Decay' ----------
    ADSR.ConfigureEnvelope(Section:Decay,Envelope:Amplitude,EnvelopeType:Exponential,Target:95.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:1%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Decay,Envelope:VibratoDepth,EnvelopeType:Exponential,Target:00.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:2%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Decay,Envelope:TremoloDepth,EnvelopeType:Exponential,Target:00.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:2%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Decay,Envelope:TimbreMorph,EnvelopeType:Linear,Target:99.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:1,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:3%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Decay,Envelope:TimbreLFODepth,EnvelopeType:Exponential,Target:0.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:4%,ExpMultKSU:0%,ExpMultKSL:0%);

    //----- ENVELOPES : for ADSR section 'Sustain' ----------
    ADSR.ConfigureEnvelope(Section:Sustain,Envelope:Amplitude,EnvelopeType:Exponential,Target:95.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:1%,ExpMultKSU:0%,ExpMultKSL:0%);
    //Make the tremolo depth a bit less than the vibrato depth.
    ADSR.ConfigureEnvelope(Section:Sustain,Envelope:VibratoDepth,EnvelopeType:Exponential,Target:35.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:5%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Sustain,Envelope:TremoloDepth,EnvelopeType:Exponential,Target:25.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:5%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Sustain,Envelope:TimbreMorph,EnvelopeType:Linear,Target:98.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:1,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:3%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Sustain,Envelope:TimbreLFODepth,EnvelopeType:Exponential,Target:98.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:4%,ExpMultKSU:0%,ExpMultKSL:0%);

    //----- ENVELOPES : for ADSR section 'Release' ----------
    //Add some key-scaling of the release amplitude envelope, so that the lower notes decay more slowly.
    ADSR.ConfigureEnvelope(Section:Release,Envelope:Amplitude,EnvelopeType:Exponential,Target:0.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:0.5%,ExpMultKSU:3%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Release,Envelope:VibratoDepth,EnvelopeType:Exponential,Target:0.0%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:40%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Release,Envelope:TremoloDepth,EnvelopeType:Exponential,Target:0.0%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:40%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Release,Envelope:TimbreMorph,EnvelopeType:Linear,Target:97.00%,TargetKSU:0%,TargetKSL:0%,LinearDelta:1,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:3%,ExpMultKSU:0%,ExpMultKSL:0%);
    ADSR.ConfigureEnvelope(Section:Release,Envelope:TimbreLFODepth,EnvelopeType:Exponential,Target:0.0%,TargetKSU:0%,TargetKSL:0%,LinearDelta:0,LinearDeltaKSU:0%,LinearDeltaKSL:0%,ExpMult:20%,ExpMultKSU:0%,ExpMultKSL:0%);
}
```