WebAudio tutorial

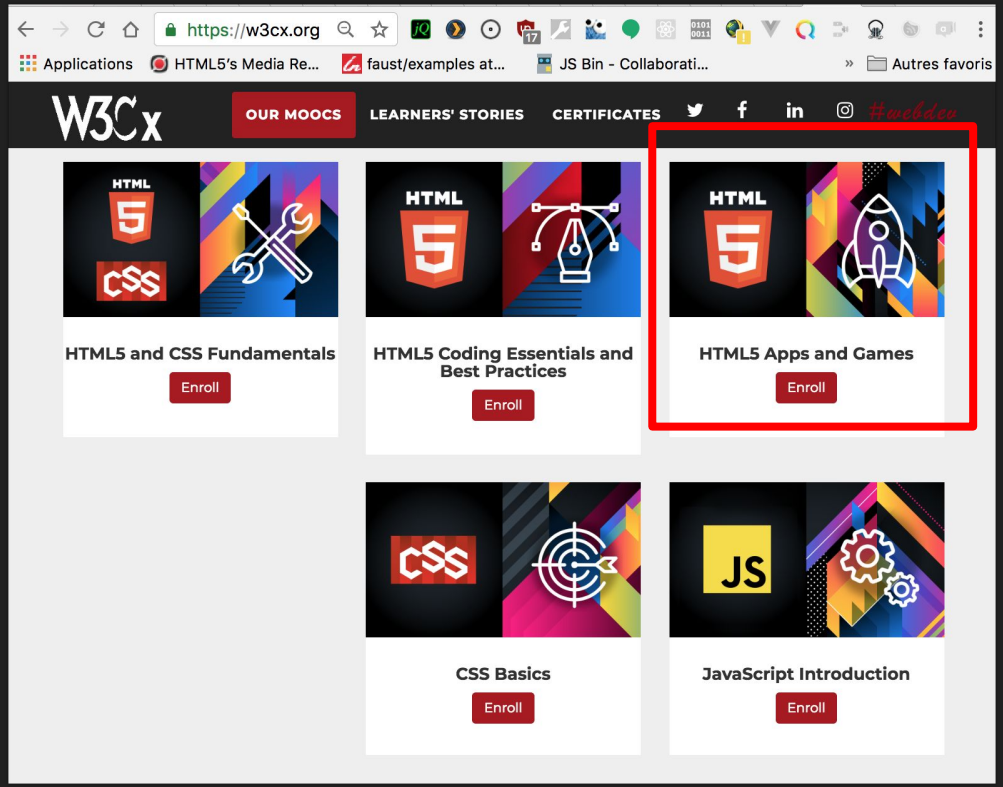
Audio Effects and Musical Instruments in the Browser

TheWebConference 2018, Lyon France
W3C Track,

Michel Buffa,
Université Côte d'Azur, France,
I3S/CNRS/INRIA labs
buffa@i3s.unice.fr, @micbuffa

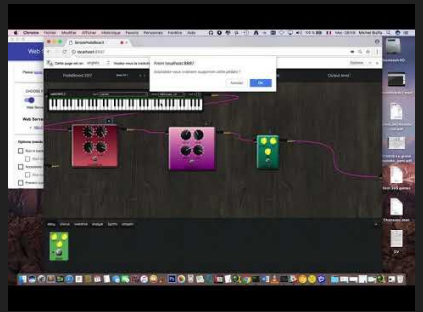
Stéphane Letz letz@grame.fr,
Yann Orlarey orlarey@grame.fr
GRAME/Lyon

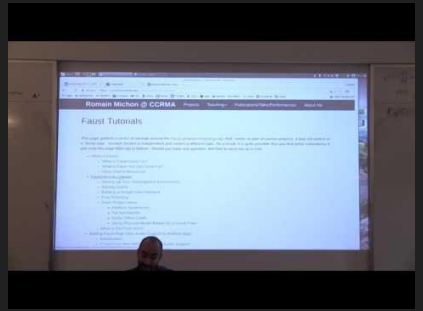
Some WebAudio examples are detailed in this MOOC at [W3Cx.org](W3Cx.org)



Outline

- History / Background / Concepts
- PART 1 : Developing high level DSP code using WebAudio standard nodes
- PART 2 : Developing low level DSP code using the AudioWorklet node
  - Can be written using C/C++/DSL languages such as FAUST
  - Including support for WebAssembly / Asm.js





History

2010/2011: Audio Data API (Mozilla)

2010: proposal for a WebAudio API and discussion about its design concepts

15/12/2011: first draft specification of the WebAudio API

2012: implementations in Firefox/Webkit browsers

End of 2012 :

- Support for real time input: Media Capture and Streams API
  - getUserMedia() / MediaDevices
- First draft of the WebMidi API, implementation in Google Chrome

2018: AudioWorklet node implemented in Google Chrome

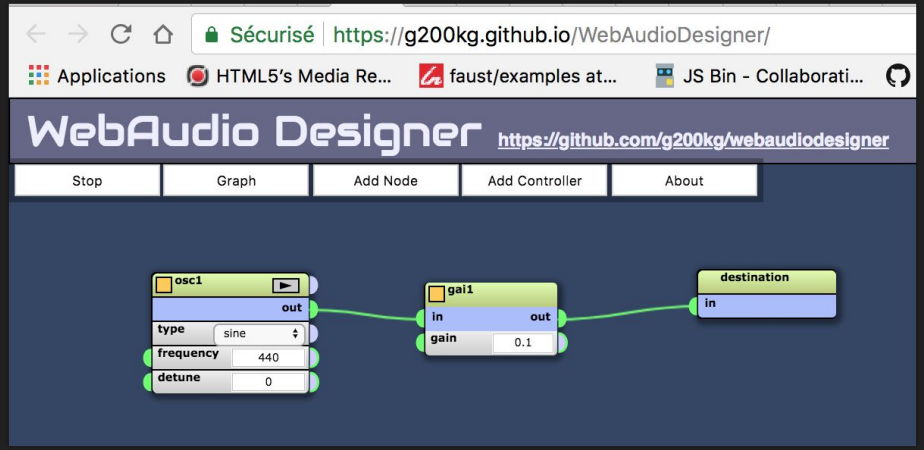
WebAudio Concepts

Audio operations occur inside an Audio Context

```
let ctx = new AudioContext();
```

Modular design:

- Audio operations performed with **audio nodes**
- Audio nodes form an **audio graph**




```
let osc1 = ctx.createOscillator();
osc1.frequency.value = 440;
let gain1 = this.audioctx.createGain();
gain1.gain.value = 0.1;

osc1.connect(gain1).connect(ctx.destination);
```

WebAudio comes with a set of standard nodes

WebAudio provides a set of standard nodes:

- Nodes for defining audio sources (file, stream, sound card input…)
- A special node for the audio destination (speakers)
- Gain (change volume of incoming signal)
- Filters (i.e make an equalizer)
- WaveShaper (i.e add distortion…)
- Convolver (i.e reverberation, phonish sound etc.)
- Compressor (i.e prevent clipping)
- Merger / Splitter (i.e mono / stereo / 5.1 etc.)
- Panner (i.e balance left/right)
- Delay (deliver output signal after a delay… i.e make echo effects…)
- ScriptProcessor (obsolete) replaced by AudioWorklet (low level DSP)

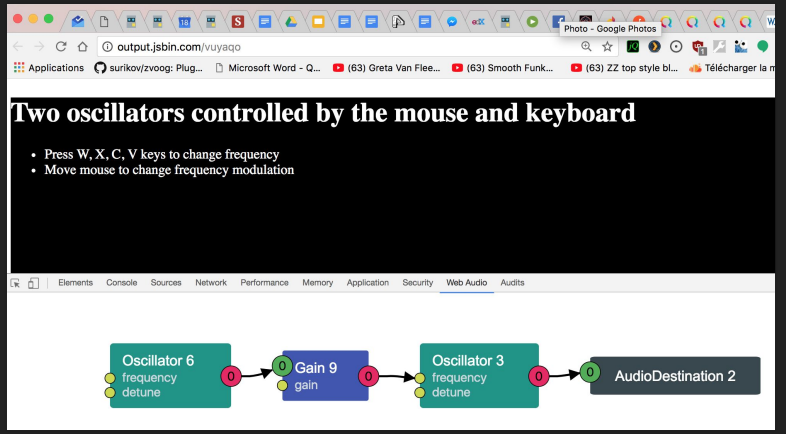
Why high-level nodes?

Advantages:

- People with nearly no DSP background can start experimenting
  - College students for example
- Easy to debug Appealing to JavaScript developers with no C/C++ experience
- DEMO TIME!

Disadvantages:

- Serious applications may need more control over the signal (AudioWorklets)



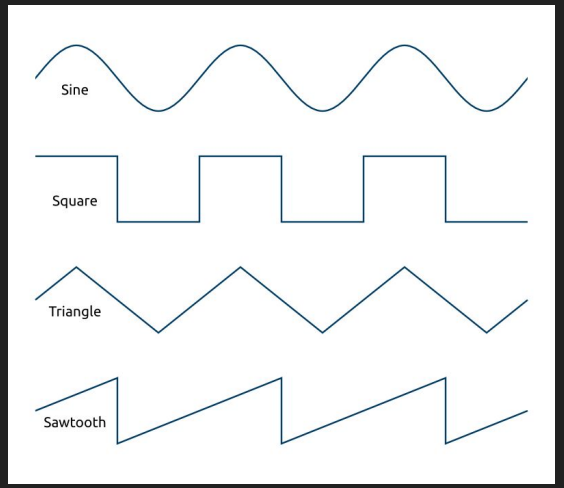
Audio source nodes: AudioBufferSourceNode

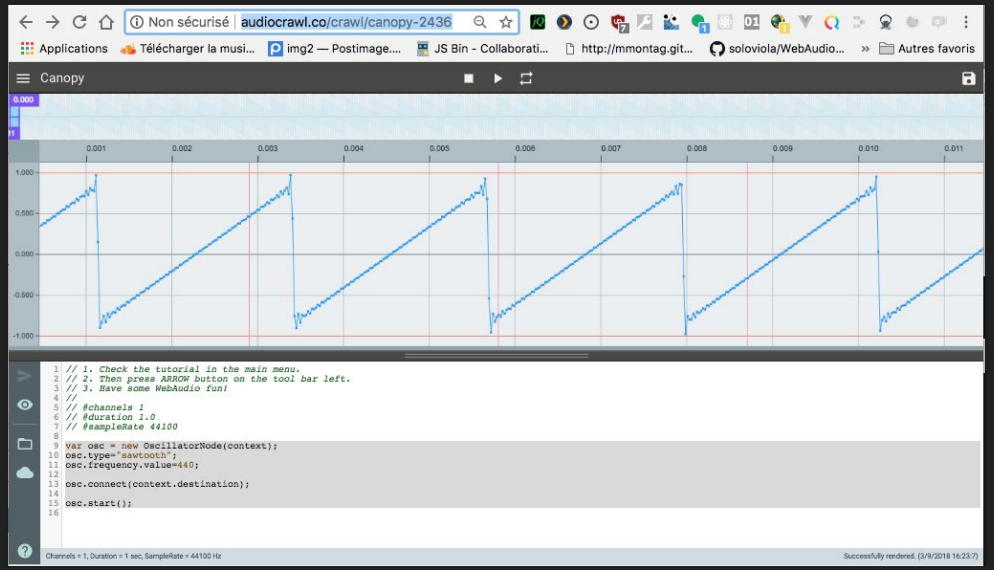
- **OscillatorNode**: sound synthesis, support single waveforms
- **AudioBufferSourceNode**: sound from files (audio, video) stored and decoded in memory
- **MediaElementAudioSourceNode**: a stream from an HTML element (audio, video)
- **MediaStreamAudioSourceNode**: a stream from WebRTC or a device (microphone, webcam, sound card input…)

- Let’s see examples of each of them!

Audio source nodes: OscillatorNode

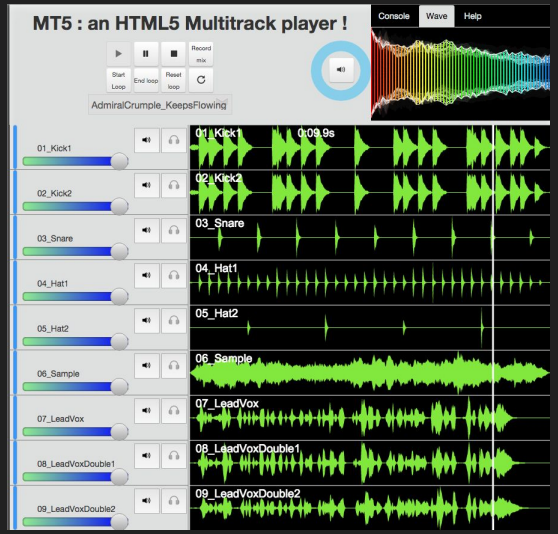
- **OscillatorNode**: sound synthesis, support single waveforms
  - DEMO with Canopy
  - Link this to mouse events
  - WebAudiox lib for retro games, see also this link
  - Oscillators are at the heart of synthesizers! More on that later...

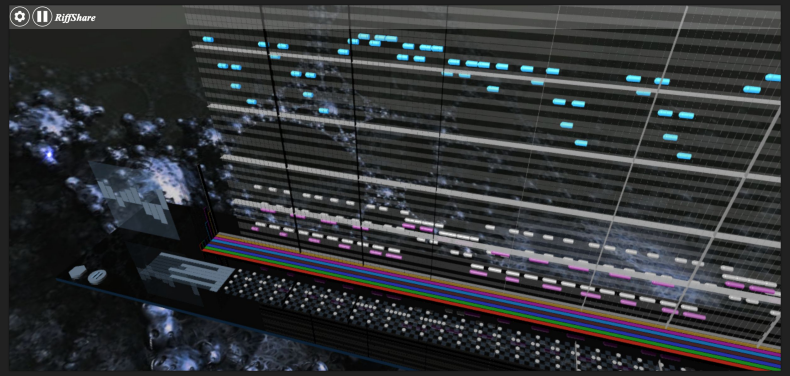




Audio source nodes: AudioBufferSourceNode

- AudioBufferSourceNode: sound from files (audio, video) stored and decoded in memory
- Useful for games, sample based instruments, has low cpu usage
- Multitrack player, soundfont based midi player (see WebAudioFont)
- See this simple example on jsBin





- Click on the above images for demos

Good utility: an asset loader (images, sounds)

Sécurisé | https://codepen.io/w3devcampus/pen/QpRGrz?editors=0010

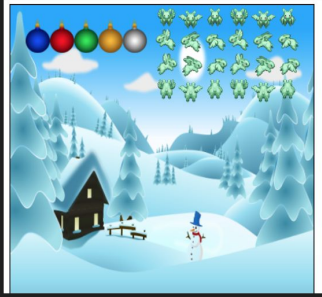
Applications | HTML5's Media Re... | faust/examples at... | JS Bin - Collaborati... | surikov/zvoog: Plug... | Microsoft Word - Q... | (63) Greta Van Flee... | Autres favoris

Untitled
A PEN BY W3Cx Pro

Save | Fork | Settings | Change View

HTML | CSS | JS

```
var assetsToLoadURLs = {
    backgroundImage: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/images/background.png' }, // http://www.clipartlord.com/category/weather-clip-art/winter-clip-art/
    logo1: { url: "https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/images/SkywardWithoutBalls.png" },
    logo2: { url: "https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/images/BoundsWithoutBalls.png" },
    bell: { url: "https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/images/bells.png" },
    spriteSheetBunny: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/images/bunnySpriteSheet.png' },
    plop: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/sounds/plop.mp3', buffer: false, loop: false, volume: 1.0 },
    humbug: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/sounds/humbug.mp3', buffer: true, loop: true, volume: 1.0 },
    concertino: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/sounds/christmas_concertino.mp3', buffer: true, loop: true, volume: 1.0 },
    xmas: { url: 'https://mainline.i3s.unice.fr/mooc/SkywardBound/assets/sounds/xmas.mp3', buffer: true, loop: true, volume: 0.6 }
};
```



Assets to load can be images, sound samples streamed music

[Used by this game!]

Scheduling / playing sounds at a given time

- The start method of the previous source nodes takes optional parameters:

```
sourceNode.start( time );
sourceNode.stop( time + delay );
```

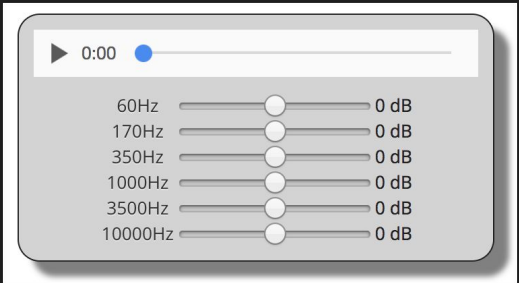
- If you run this, the sound will start after a given time, and will stop after a given delay.
  - You don’t have to worry, notes or samples will be played on time and last the correct duration…
  - You can schedule all notes of a music sheet in advance if you like..
- If you want to change the tempo, or other parameters in real time, you need some strategies: only schedule a few nodes in advance and add them on the fly. Look at this online example.
- See A Tale of Two Clocks - Scheduling Web Audio with Precision
- Libraries like ToneJS will simplify all this for you :-)

Audio source nodes: MediaElementSourceNode

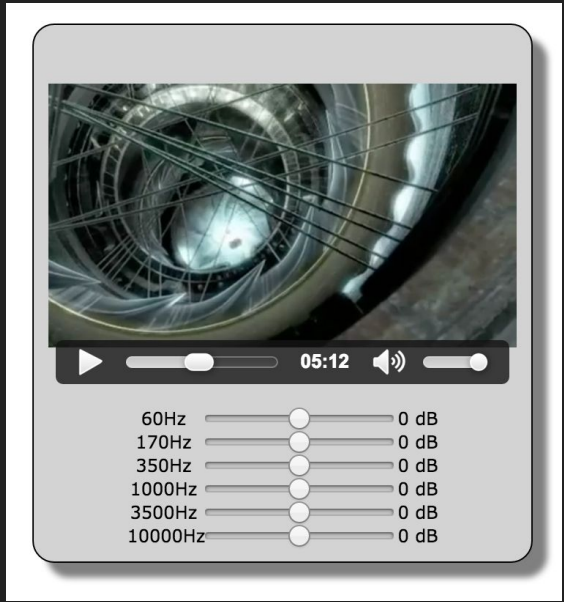
- **MediaElementAudioSourceNode**: a stream from an HTML element (audio, video). See this simple example:

```
<audio id="player" src="http://.../guitarRiff1.mp3" controls ></audio>
```

```
ctx = new AudioContext();
var player = document.querySelector("#player");
var source = ctx.createMediaElementSource(player);
source.connect(ctx.destination);
```



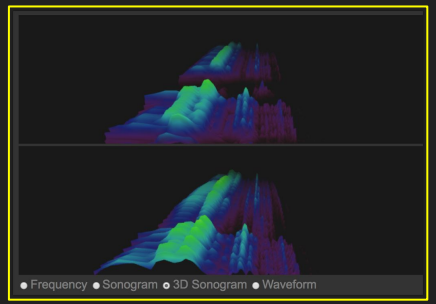
Click on images to see the code

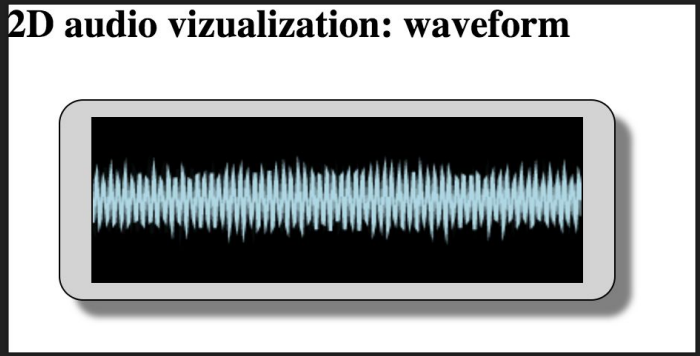


Audio source nodes: MediaStreamAudioSourceNode

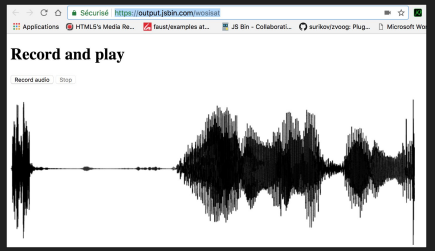
- **MediaStreamAudioSourceNode**: a stream from WebRTC or a device (microphone, webcam, soundcard input…)

```
navigator.mediaDevices.getUserMedia({audio: true})
.then((stream) => {
    sourceNode = ctx.createMediaStreamSource(stream);
    // build the audio graph
    sourceNode.connect(...);
});
```





Audio destinations



- `ctx.destination` is the “default” destination (speakers)
  - Using the MediaStream API it’s possible to choose the “device”, but no support for sound cards with multiple outputs
- You can use a MediaStreamDestinationNode to redirect the output to a stream
  - Record in a blob, then save on disk, a remote server, send through WebRTC
  - Assign to the `src` attribute of an HTML audio player
  - Or use the MediaStream constructor to add it as a video track, for example.
  - See this online demo

```
var ctx = new AudioContext();
var osc = ctx.createOscillator();
var dest = ctx.createMediaStreamDestination();
var mediaRecorder = new MediaRecorder(dest.stream);
osc.connect(dest);
mediaRecorder.start(); // starts recording...
```

Let’s look at a first, simple example (GainNode)

An HTML audio player

With a GainNode connected to it

The GainNode is connected to `ctx.destination` (speakers)

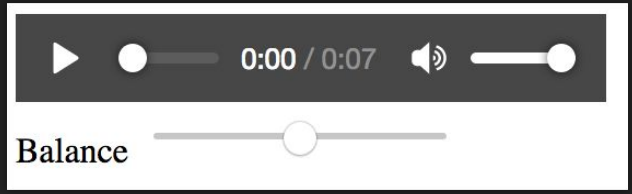
[Online example at JsBin]

```
var g = ctx.createGain();

source.connect(g);
g.connect(ctx.destination);

g.gain.value = 3; // louder x 3!
```

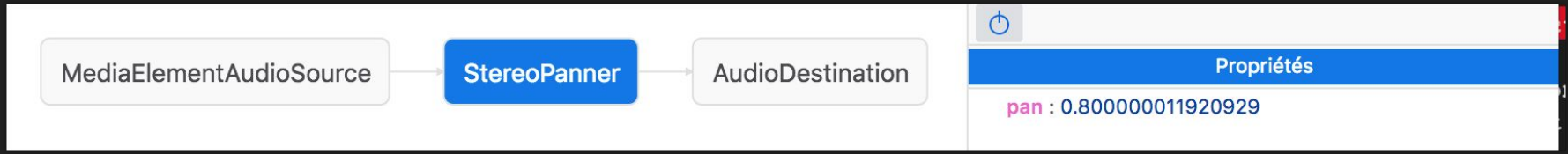
After volume, add stereo / balance control

An HTML audio player

With a StereoPanner node connected to it



The StereoPanner is connected to `ctx.destination` (speakers)



[Online example at JsBin]

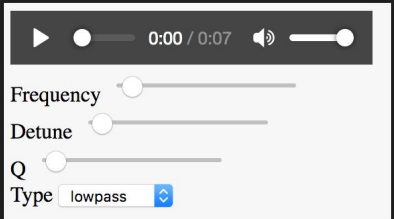
```
var p = ctx.createStereoPanner();

source.connect(p);
p.connect(ctx.destination);

p.pan.value = 0.8; // right > left
```

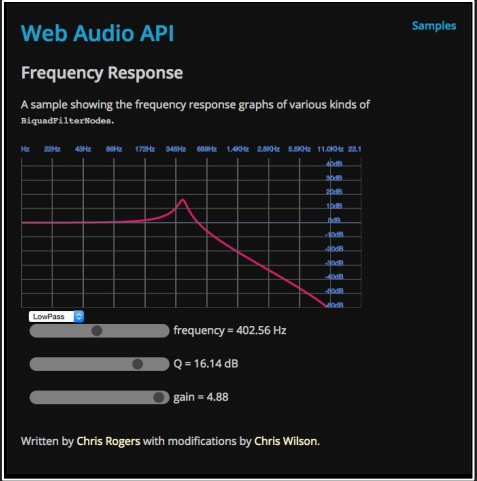
Control frequencies: filters

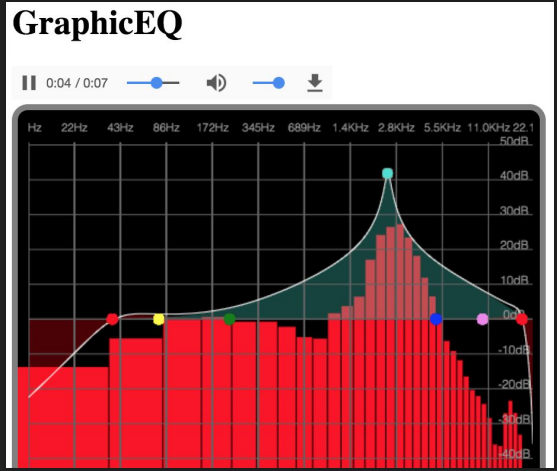
WebAudio comes with two types of filters:

- Biquad filters (most common ones: highpass, lowpass, peaking etc.)
- IIRfilters: custom impulse response function (IIR filter workshop, filter playground)

```
var f =
ctx.createBiquadFilter();

source.connect(f);
f.connect(ctx.destination);

f.type = "lowpass";
f.frequency.value = 200; // Hz
f.gain=-6; // dB
```





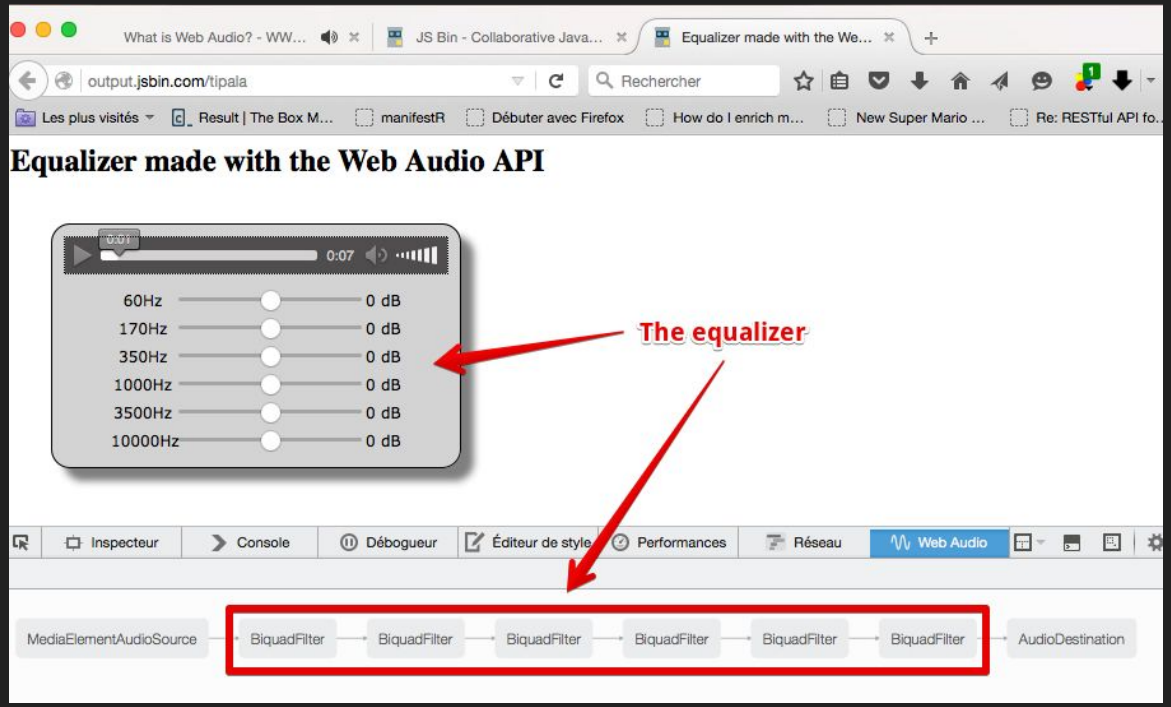


Control frequencies: filters, a multiband equalizer

Step by step tutorial available on the free MOOC by W3Cx: HTML5 Apps and Games, Week 1.

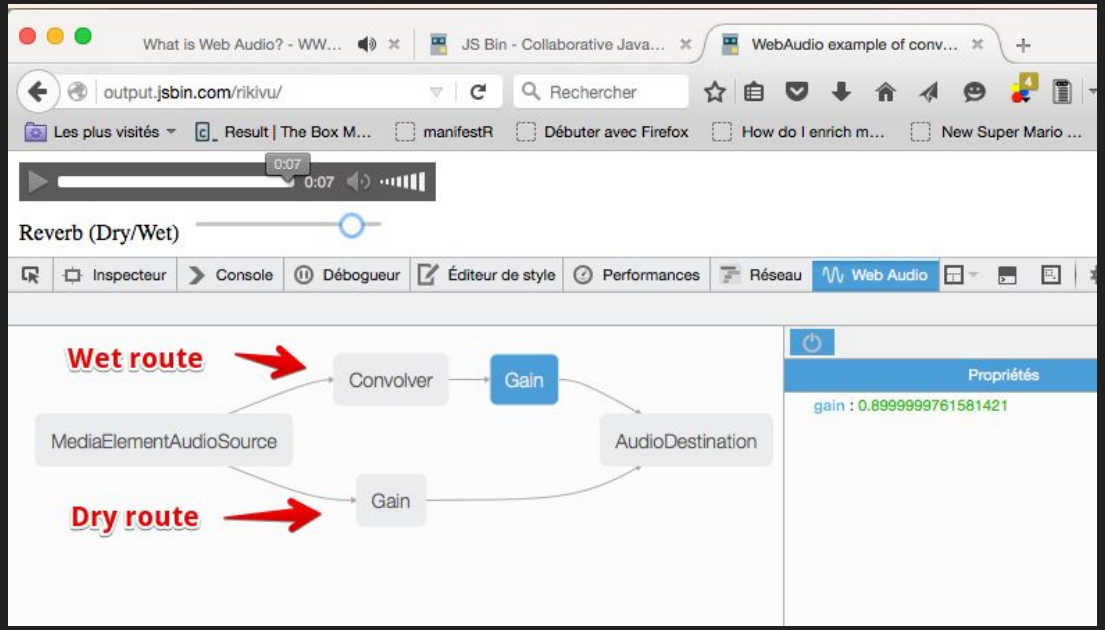
Let's give it a look!

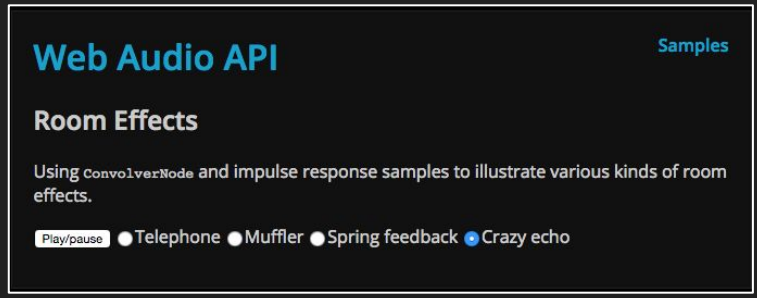
- JsBin code
- Course page (need to be online on the edX platform)



Add reverberation and other convolution effects

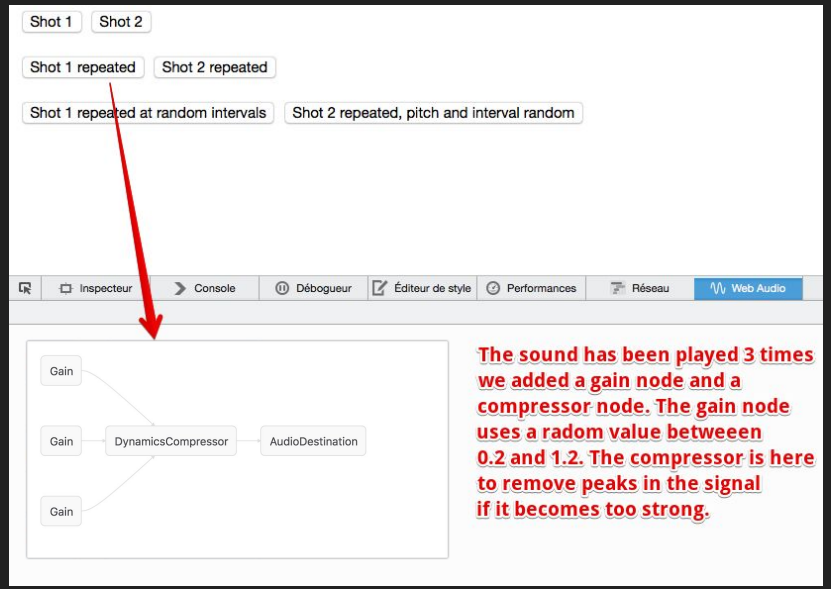
An HTML audio player with a ConvolverNode connected to it + 2 GainNode to make a "wet" and a "dry" route to `ctx.destination` (speakers). The slider adjusts the two gains, setting the amount of reverb we add to the signal.





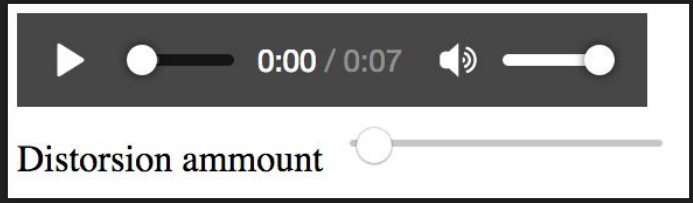
Avoid clipping using a compressor node

An HTML audio player to a gain connected to a DynamicCompressorNode for avoiding clipping / saturation. Other example = multiple sound sample played at the same time needs to be limited.

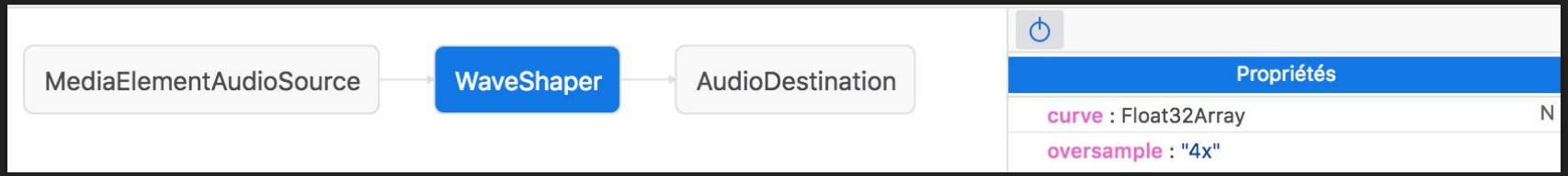


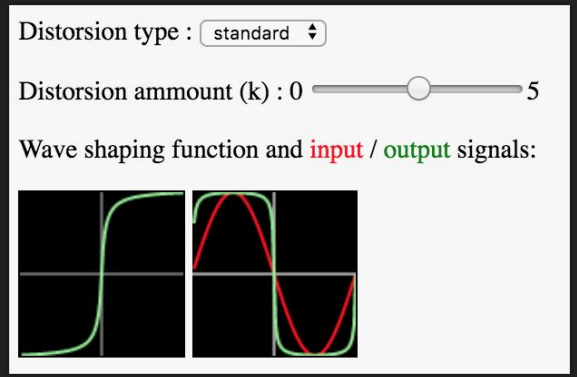
Distortion: using WaveShaper nodes

An HTML audio player

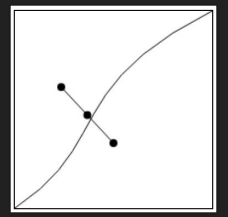


With a [WaveShaperNode] connected to it
The WaveShaperNode is connected to `ctx.destination` (speakers)
Wave shapers use a “transfer function curve”.





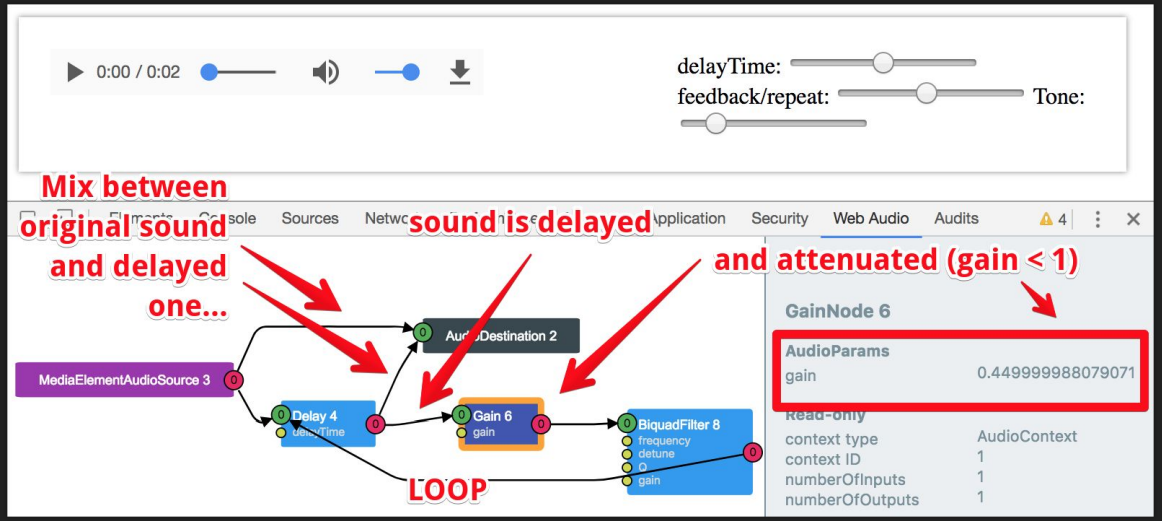

```
var ws = ctx.createWaveShaper();

source.connect(ws);
ws.connect(ctx.destination);

// k = “squarifying” param
ws.curve = makeDistortionCurve(k);
```



A example of transfer function editor

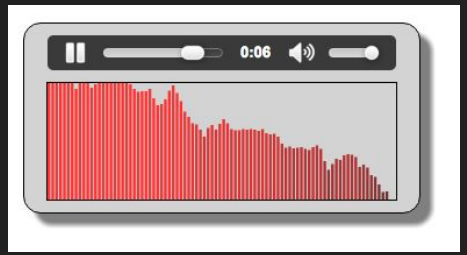
Make “echo” effects: use a delay node!

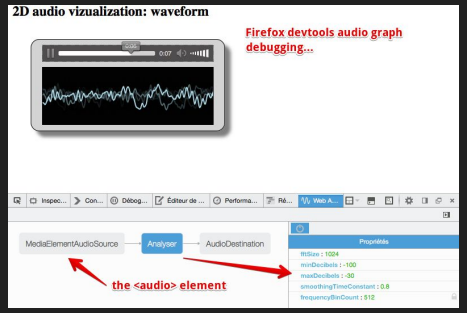
- Audio player connected to the `ctx.destination`
- But also to a DelayNode that will “delay” the output of the sound,
- Delayed sound is attenuated through a gain node (gain < 1) and filtered
- Then re-injected into the delayNode. This loops makes the “echo” effect

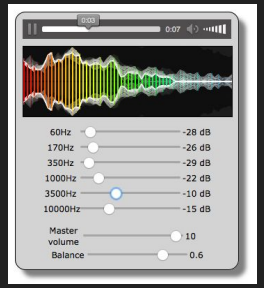


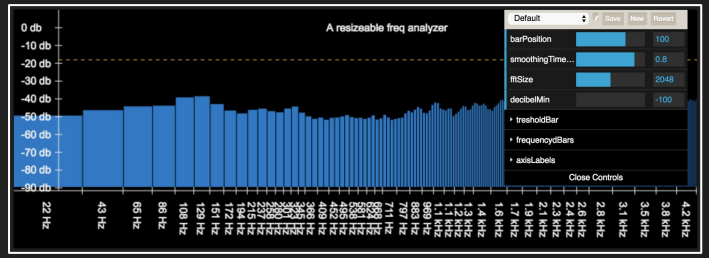
Analyser node: visualize waveforms, frequencies, volume

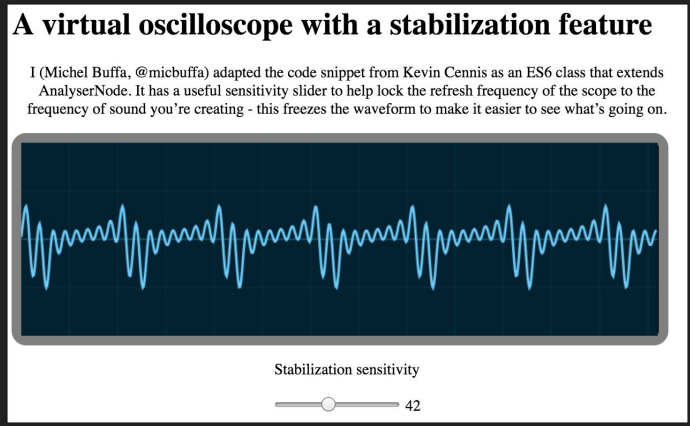
- AnalyserNode step by step tutorials available on the free MOOC by W3Cx: HTML5 Apps and Games, Week 1.











Demos: effects and instruments, only using high level nodes

- These standard nodes can be assembled into an “audio graph”, which developers can use to write more complex audio effects or instruments.
- It’s possible to have hundreds of such nodes (ex: vocoder)

Conclusion of part 1: high-level WebAudio in JS...

- **Lots of demos / examples**, but not competing with pro native audio apps (a bit the Office / Google doc comparison)
- **Lots of JS libraries**, some impressive ones (toneJS, tunaJS, howlerJS etc.)
- **Few commercial products** SoundTrap / BandLab: **only two serious DAWs available in the browser**, Hearing Tests, online music schools
- **High-level nodes gave opportunity to all JS coders to make some noise :-)**
- **Not only for computer music** (enhance existing multimedia applications -players-, sound effects in games, computed music, midi music, P2P etc.)
  - Tight integration with the Media Capture and Streams API
- **Not enough for professional computer music application needs**

Custom programmed nodes : why ?

- **Developers may need special algorithms or synthesis models not available in the native API (physical models, modal synthesis…)**
- **Describing complex audio computation as Audio Graphs suffer from the “one buffer in loops” limitation**
- **So doing sample level descriptions is mandatory in this case**
- **Porting well established native music languages or systems on the Web (Csound, PureData/Heavy…)**
- **Developing new pure Web environments**
    - **Flocking :** https://github.com/colinbdclark/Flocking
    - **genish.js**: http://www.charlie-roberts.com/genish/
    - ...

Custom programmed nodes : how ?

Developed in pure JavaScript, or produced by another mean.

WebAssembly(wasm) is now the preferable way to deploy fast code in the Web:

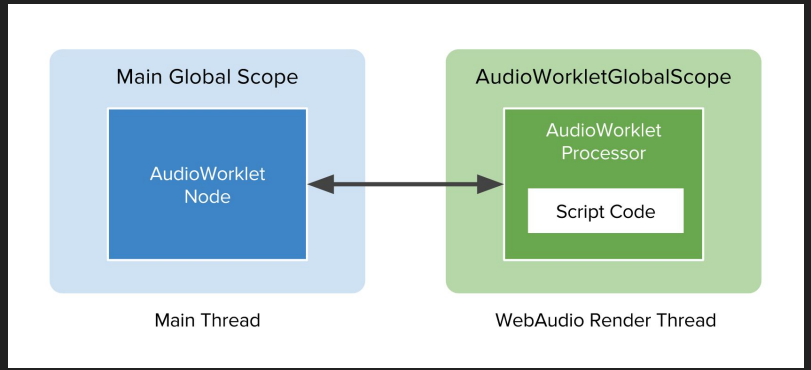
- **wasm can be generated using a C++ ⇒ wasm path (Emscripten compiler from C/C++ or Rust compiler)**
- **wasm can be directly generated by Domain Specific Language (DSL) like Faust**

ScriptProcessorNode/AudioWorkletNode

- **Old ScriptProcessorNode model:**
  - **runs in the main thread**
  - **suffers from high latency and glitches (thread priority)**
- **New AudioWorkletNode model:**
  - **runs in a dedicated audio processors thread**
  - **lower latency (running using 128 frames buffers like the native part)**
  - **but still in specification/implementation (only in Chrome for now)**
- **General page:**
  - **https://googlechromelabs.github.io/web-audio-samples/audio-worklet/**
- **Good introduction by Hongchan Choi (Google/Chrome) here:**
  - **https://developers.google.com/web/updates/2017/12/audio-worklet**

The AudioWorkletNode/AudioWorkletProcessor model (1)

- **Combining AudioWorkletNode and AudioWorkletProcessor**
- **AudioWorkletProcessor exists in a separated AudioWorkletGlobalScope**
- **The actual processor audio code runs in a dedicated thread**



The AudioWorkletNode/AudioWorkletProcessor model (2)

- **Example of AudioWorkletProcessor**

```
// This is "processor.js" file, evaluated in AudioWorkletGlobalScope upon
// audioWorklet.addModule() call in the main global scope.
class MyWorkletProcessor extends AudioWorkletProcessor {
  constructor() {
    super();
  }

  process(inputs, outputs, parameters) {
    // audio processing code here.
  }
}

registerProcessor('my-worklet-processor', MyWorkletProcessor);
```

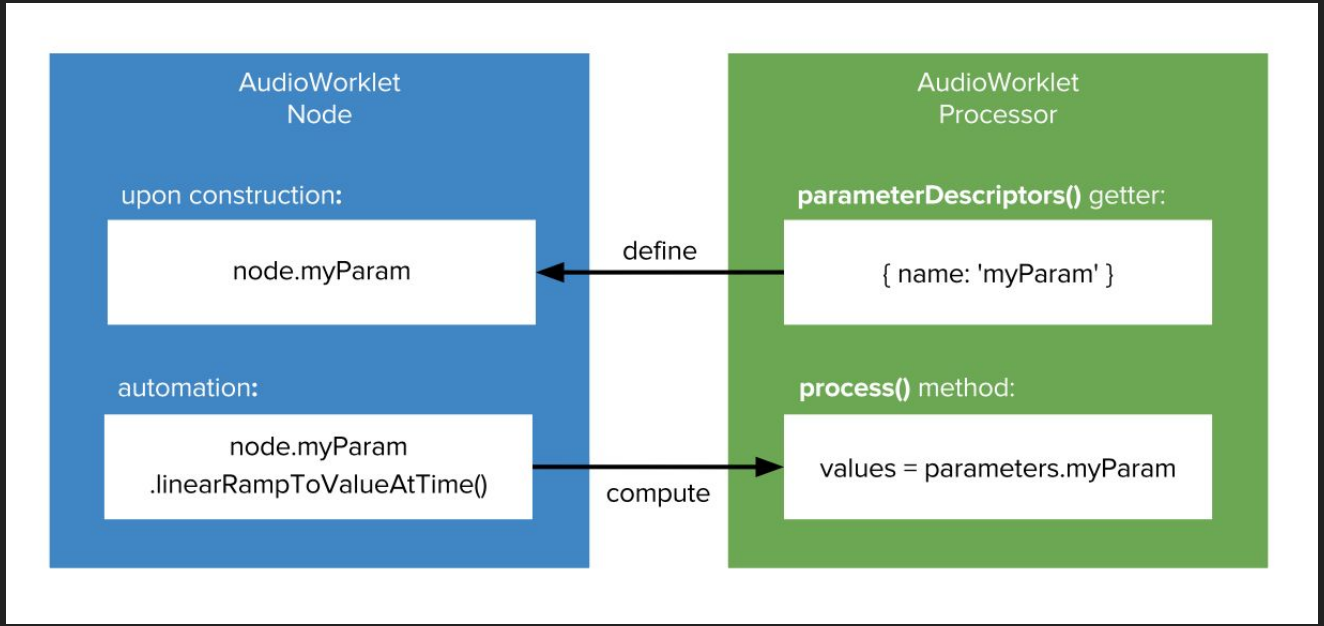
The AudioWorkletNode/AudioWorkletProcessor model (3)

- **Example of AudioWorkletNode**

```
// The code in the main global scope.
class MyWorkletNode extends AudioWorkletNode {
  constructor(context) {
    super(context, 'my-worklet-processor');
  }
}

let context = new AudioContext();

context.audioWorklet.addModule('processors.js').then(() => {
  let node = new MyWorkletNode(context);
});
```

The AudioWorkletNode/AudioWorkletProcessor model (4)

- **Custom AudioParams:**



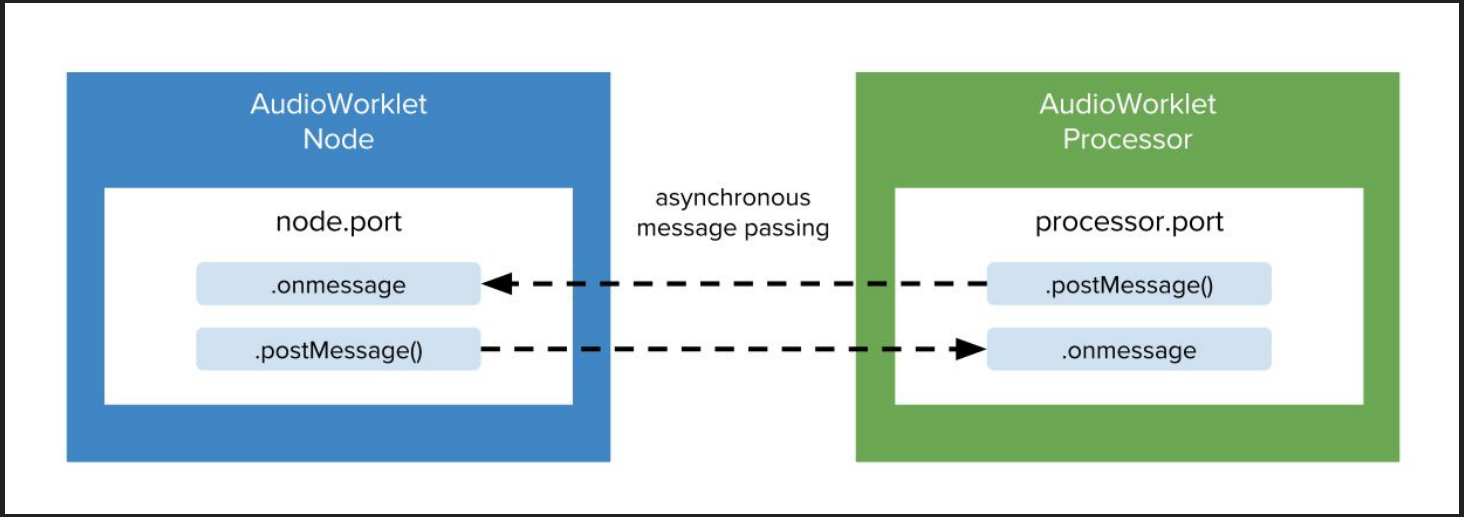
The AudioWorkletNode/AudioWorkletProcessor model (5)

- **Example of “process” code:**

```
/* AudioWorkletProcessor.process() method */
process(inputs, outputs, parameters) {
  // The processor may have multiple inputs and outputs. Get the first input and
  // output.
  let input = inputs[0];
  let output = outputs[0];

  // Each input or output may have multiple channels. Get the first channel.
  let inputChannel0 = input[0];
  let outputChannel0 = output[0];

  // Get the parameter value array.
  let myParamValues = parameters.myParam;

  // Simple gain (multiplication) processing over a render quantum (128 samples).
  // This processor only supports the mono channel.
  for (let i = 0; i < inputChannel0.length; ++i) {
    outputChannel0[i] = inputChannel0[i] * myParamValues[i];
  }

  // To keep this processor alive.
  return true;
}
```

MessagePort to communicate between Node/Processor

- **“onmessage” callback and “postMessage” function**
- **can be used to communicate MIDI messages (using the WebMIDI API), MIDI banks/patches...**



Demo of custom Noise node

Using Faust Domain Specific Language

Faust is a DSP for audio programming : effect, synthesis, instrument design…

Faust tutorial used for the demo: https://faust.grame.fr/tutorial/

Deploying Faust coded WebAudio nodes

- **The Faust compiler produces a wasm module description (as a wasm file or buffer)**
- **This module has to be “compiled” (using WebAssembly.Module JS API) then “instantiated” (using WebAssembly.Instance JS API)**
- **Generating JS glue code to define the AudioWorkletNode and AudioWorkletProcessor pair**
- **DEMO**
  - **Testing the Faust source on FaustEditor**
  - **exporting is using the compilation service**
  - **or faust2wasm -worklet Gain.dsp**
  - **Creates the Gain.js and Gain-processor.js files**
  - **Use them in Gain.html test page**
  - **Doing the same for FilterBank.dsp**

Emscripten generated code

- **Emscripten is a general purpose C/C++ ⇒ wasm compiler (Mozilla)**
- **Audio C/C++ DSP can be easily deployed as WebAudio nodes**
  - **using the same “produce the wasm module”, add “glue JS code” model**
  - **Jari Kleimola Web Audio Modules:** https://www.webaudiomodules.org/blog/wam-story/
  - **DEXED :** https://webaudiomodules.org/demos/wasm/dexed.html
  - **OBDX :** https://webaudiomodules.org/demos/wasm/obxd.html

Now let’s discuss what is still missing...

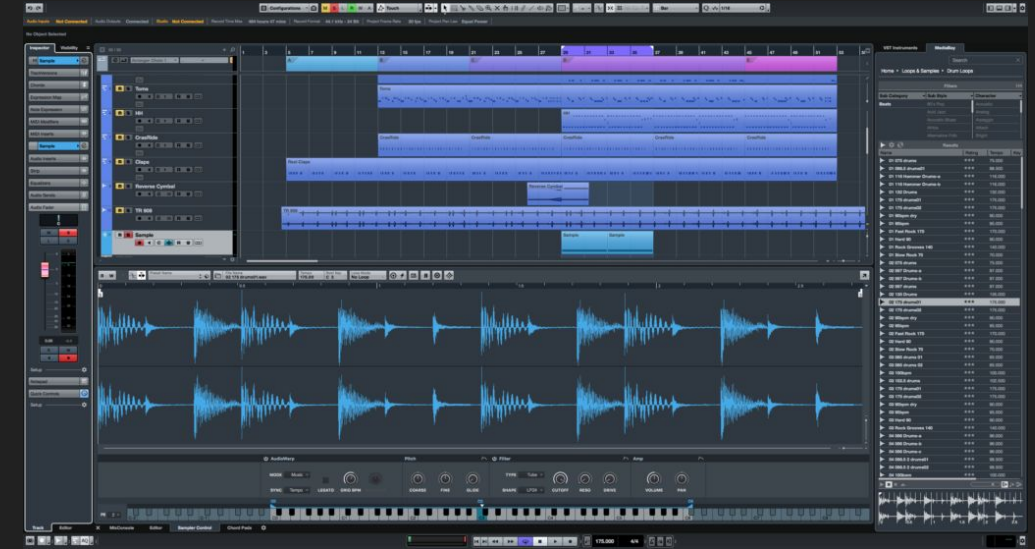
First, let’s look at some computer music history….

At the industry…

At the “programming model”...

The Electronic Music landscape

- Market ruled by commercial actors and non Web-based software
  - 1996 (Atari ST, Steinberg's Cubase and Virtual Studio Technology aka VST)
- Articulated around DAWs (Digital Audio Workstations) : Cubase (Steinberg), Logic Pro, Ableton Live, etc.
- And plugins: DAWs are "hosts" for plugins (effects, instruments)
  Many closed, commercial standards VST, Apple Audio Units, AVID AAX, etc.
  - **Literally thousands of plugins have been developed in C/C++, meta standard exists (JUCE, iPlug2...)**




Vibe Phaser
Tri-Knob Fuzz
Spring Reverb

The open source Computer Music landscape

Note: the only “professional-quality” DAW on Linux is commercial and not Open Source (Reaper)

- The GMPI group ("Generalized Music Plugin Interface", started in 2003)
  - Took the best parts of each existing commercial standard
  - Put them together into a cross-platform specification

- The LV2 (LADSPA version 2) open standard implemented partially the GMPI specification
  - Widely adopted on Linux
  - LV2 vs GMPI comparison table

WebAudio and Audio on the Web

Did you really listen to what we said earlier?

- Very young compared to the “native” computer music world
- Nearly no commercial products, lots of “enhanced multimedia demos”
- Very, very few ambitious realizations in the Computer Music field

But… no plugin standard, no “hosts”, no programming model...

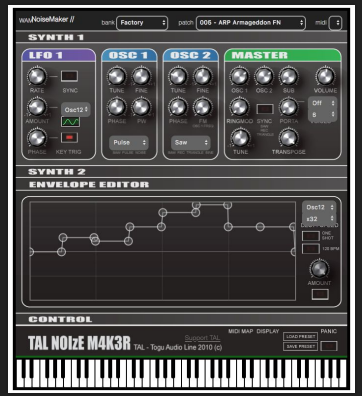
We find some very good JavaScript libraries (i.e. toneJS)

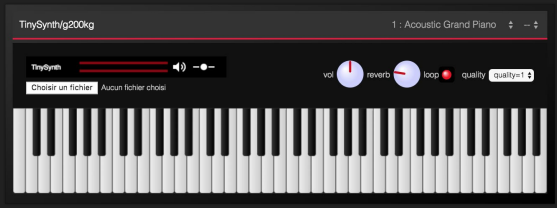
Some open source github repositories (i.e. https://webaudiodemos.appspot.com/)

Some online tools for music synthesis (genish.js etc.)

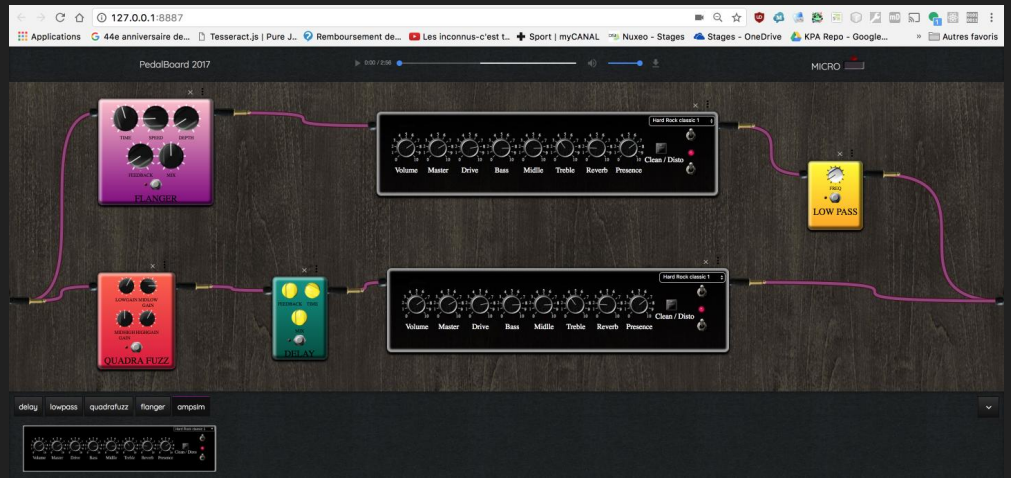
Some DSL for DSP programming (FAUST)

Some effects and instruments





With some researchers and developers we decided to start working on an open plugin standard for WebAudio



August 2017, WebAudio conference, Jari Kleimola from WAMs saw my presentation and contacted me...

We made a team with different researchers / developers, that share same concerns with different approaches

- **1 - Bringing native developers to the Web**
  a. Jari Kleimola (Aalto University Espoo, Southern Finland, now at Webaudiomodules.org),
  b. Oli Larkin (PhD University of Huddersfield Middlesex University)

- **2 - Bringing low level DSP developers to the Web**
  a. Yann Orlarey/Stéphane Letz (researchers at GRAME, Lyon, authors of the FAUST DSL/compiler)

- **3 - Attract Web Developers / JavaScript developers**
  a. Tatsuya Shinyagaito, aka g200kg, (Audio and WebAudio developer, huge WebAudio contributor, Yokohama, Kanagawa, Japan)
  b. Jerôme Lebrun and Michel Buffa (I3S/INRIA)

Audio on the Web vs Native audio

- **Advantages of WebApps**
  - **distribution** (no installation,updates),
  - **collaboration** (collaborative aspects can be implemented using WebSockets for example),
  - **platform independence** (including GUI),
  - **sandboxing** (security).
- **Disadvantages of WebApps**
  - **efficiency** (JavaScript is usually slower than native code, with issues that affect real-time audio performance such as a garbage collector),
  - **latency** (audio drivers on Linux/Windows),
  - **sandboxing** (access to native resources, local hard disk access is forbidden or limited).
  - **do not run in popular, native apps**
- **This is changing**
  - WebAssembly + AudioWorklet brings the performance of native audio to the browser
  - **May attract native developers**: compile C++ code to WebAssembly / WebAudiomodules toolchain, FAUST dsp langage now compiles to WASM

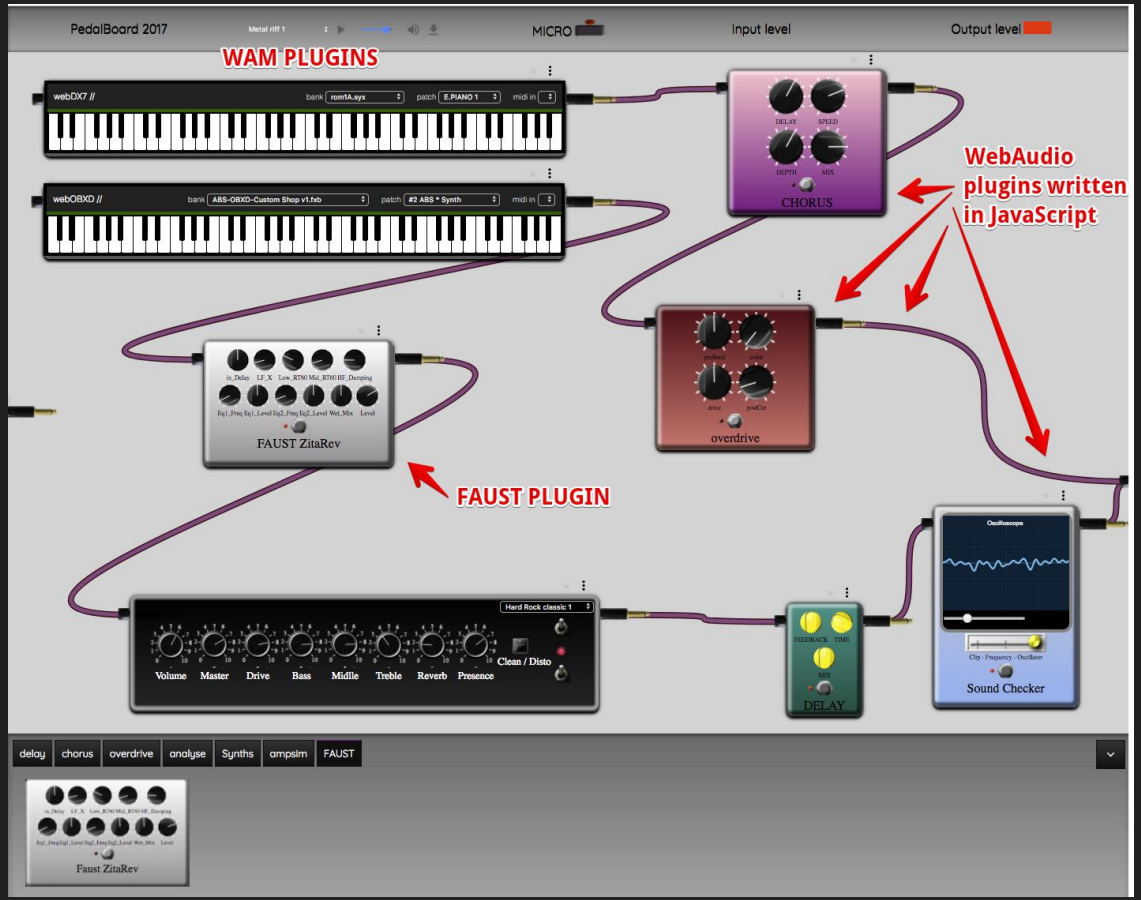
An open standard = API/specification ? Or more… ?

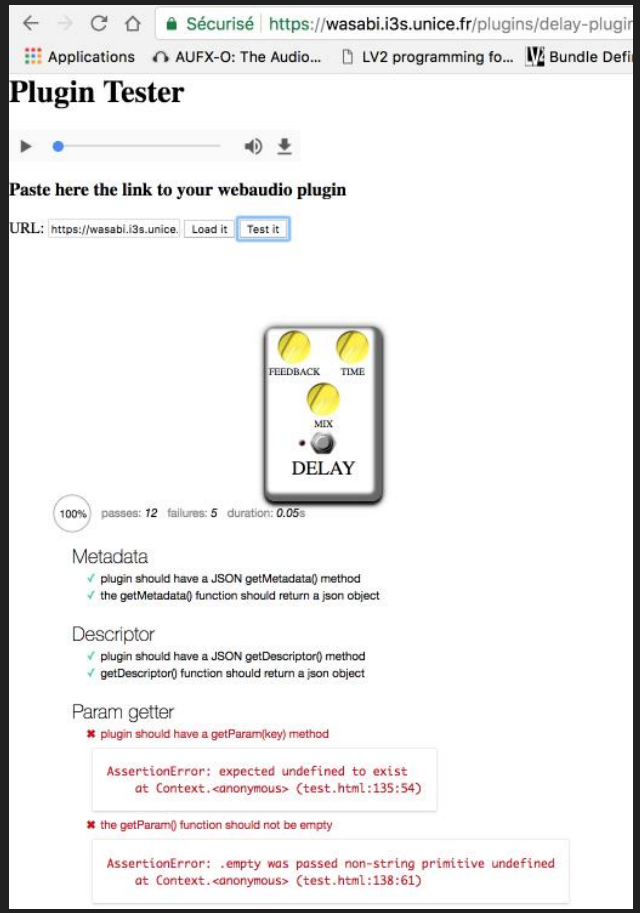
- **DOGMA: being Web-Aware!**
  - Use URIs,
  - Use rich metadata
  - Support local or remote plugins (REST repositories)
  - **Avoid naming conflicts** (HTML ids, JS names, CSS rules)
  - **Plugins can be headless or with a GUI**
  - **…but also an API or at least guidelines on how to package and publish plugins**

So… where how do we start working on a standard???

1. **look also at GMPI spec / LV2 and compare to them**
2. **Make lots of pocs with different plugins by different developers**
3. **Try to attract both JS and native devs**
4. **Think “universal”, the WebPlatform is now the “portable standard”**
5. **Report to the W3C WebAudio Working Group**

DEMOS


PedalBoard 2017
MICRO
Input level
Output level
WAM PLUGINS
webDX7 //
webOBXD //
CHORUS
WebAudio plugins written in JavaScript
overdrive
FAUST ZitaRev
FAUST PLUGIN
Volume
Master
Drive
Bass
Midlle
Treble
Reverb
Presence
Clean / Disto
DELAY
Sound Checker
delay
chorus
overdrive
analyse
Synths
ampsim
FAUST
Faust ZitaRev


Sécurisé https://wasabi.i3s.unice.fr/plugins/delay-plugin
Applications
AUFX-O: The Audio...
LV2 programming fo...
Bundle Defi
Plugin Tester
Paste here the link to your webaudio plugin
URL: https://wasabi.i3s.unice.
Load it
Test it
FEEDBACK
TIME
MIX
DELAY
100%
passes: 12
failures: 5
duration: 0.05s
Metadata
plugin should have a JSON getMetadata() method
the getMetadata() function should return a json object
Descriptor
plugin should have a JSON getDescriptor() method
getDescriptor() function should return a json object
Param getter
plugin should have a getParam(key) method
AssertionError: expected undefined to exist
at Context.<anonymous> (test.html:135:54)
the getParam() function should not be empty
AssertionError: .empty was passed non-string primitive undefined
at Context.<anonymous> (test.html:138:61)

Simple example

Sécurisé | https://jsbin.com/xarotez/edit?js,output

Applications | HTML5's Media Re... | faust/examples at... | JS Bin - Collaborati... | surikov/zvoog: Plug... | Microsoft Word - Q...

File | Add library | Share | HTML | CSS | JavaScript | Console | Output

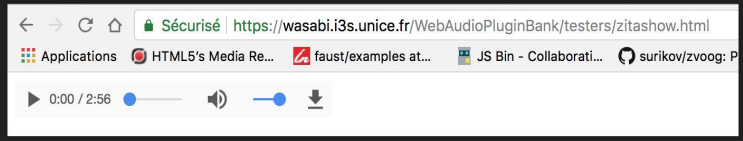
JavaScript

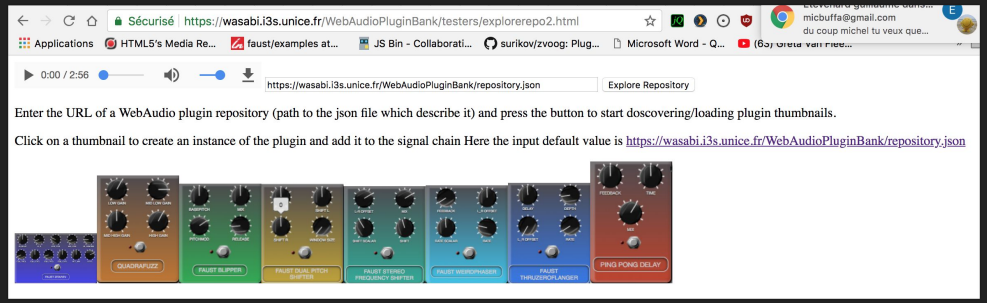
```
var ctx = new AudioContext();
var player = document.getElementById("soundSample");

 var mediaSource = ctx.createMediaElementSource(player);
 var pluginURL = "https://wasabi.i3s.unice.fr/WebAudioPluginBank/WASABI/PingPongDelay2";
 var plugin = new WAPlugin.WasabiPingPongDelay(ctx, pluginURL);

 plugin.load().then((node)=>{
   plugin.loadGui().then((elem)=>{
     document.body.appendChild(elem);
   });
   mediaSource.connect(node.getInput(0));
   node.connect(ctx.destination);
 });
```

Output

0:00 / 2:56

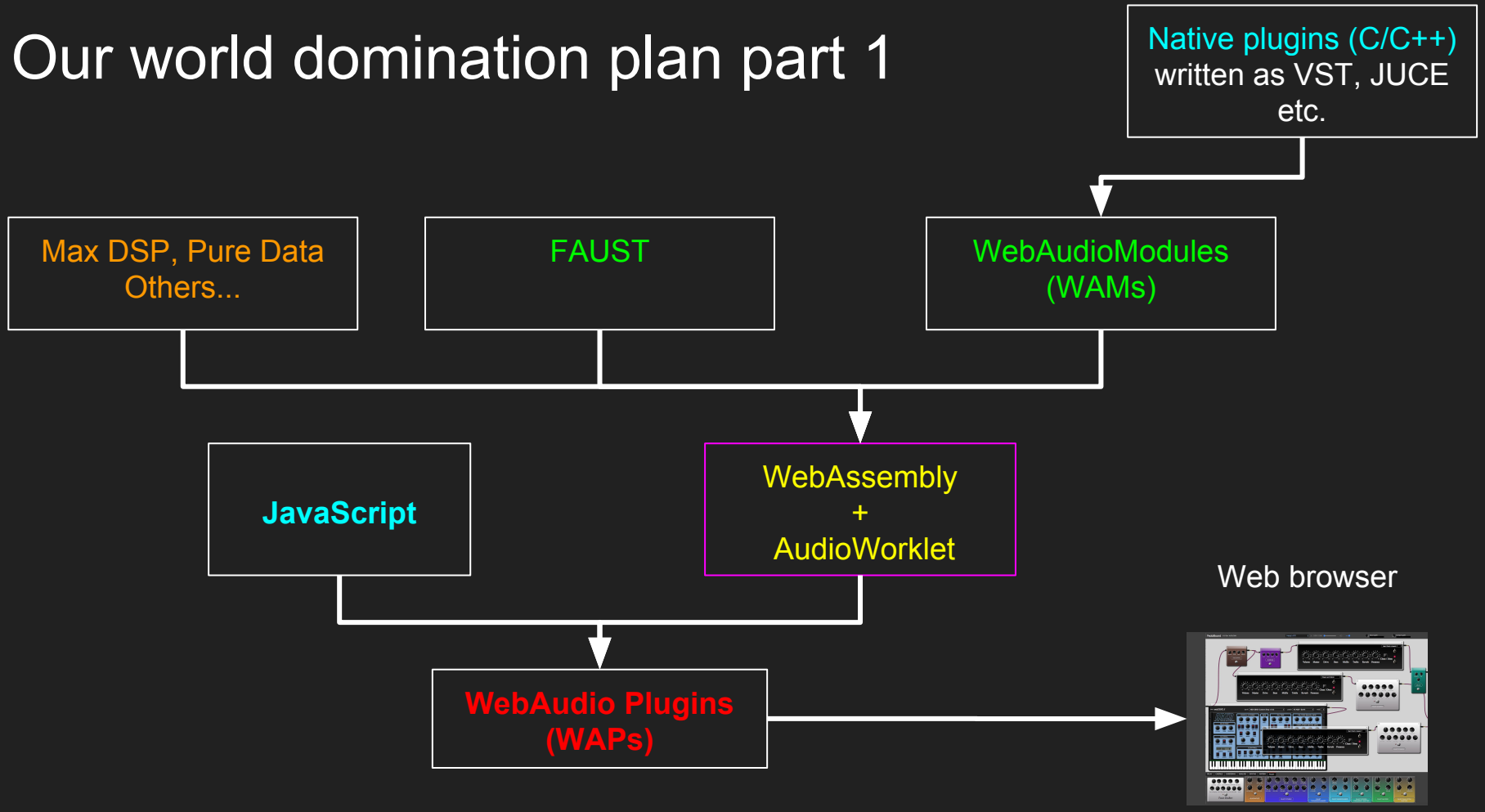
DEMOS





[Dynamic pedalboard]

Our world domination plan part 1



Our world domination plan part 2 (...)

WebAudio Plugins (WAPs)

WebAudio host runs in a Web browser

Wrap embedded browser in a native plugin shell

Now the browser runs as a native plugin

Load any WebAudio plugin in native hosts, no need to rewrite GUI or DSP code