



**HAL**  
open science

## Web Audio Modules 2.0: Audio Plugins for the Web Platform!

Michel Buffa

► **To cite this version:**

Michel Buffa. Web Audio Modules 2.0: Audio Plugins for the Web Platform!. ADC 2022 - Audio Developer Conference, Nov 2022, Londres, United Kingdom. hal-03871660

**HAL Id: hal-03871660**

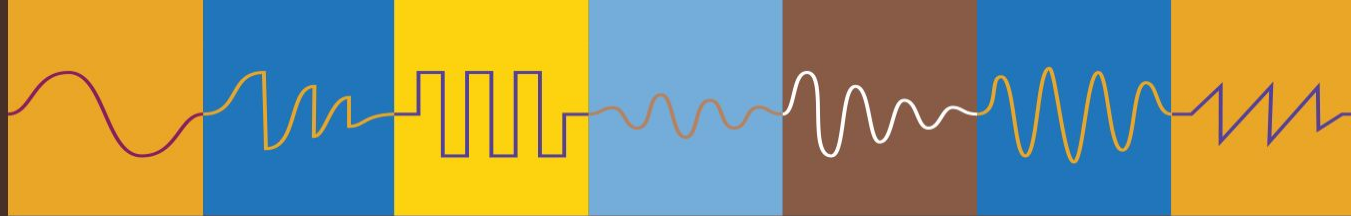
**<https://inria.hal.science/hal-03871660v1>**

Submitted on 25 Nov 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# ADC<sup>22</sup>



## WEBAUDIO MODULES 2.0: *AUDIO PLUGINS FOR THE WEB PLATFORM!*

MICHEL BUFFA



# Who am I? What is our group?

- Professor / researcher at Université Côte d'Azur (UCA), France
  - Member of the WIMMICS research group common to INRIA and I3S lab from CNRS
  - W3C Advisory Committee Representative for UCA
  - Member of the W3C WebAudio Working Group since 2014
  - [michel.buffa@univ-cotedazur.fr](mailto:michel.buffa@univ-cotedazur.fr), @micbuffa
- Other members of the WAM group:
  - WAM original creators: Jari Kleimola  
And Oliver Larkin,
  - Developers, academic researchers, PhD  
Students : Shihong Ren, Owen Campbell,  
Tom Burns, Steven Yi, Stéphane Letz,  
Hugo Mallet...
  - Thanks to: Jordan Sintès, Guillaume Etevenard, GRAME friends...



Web Audio Modules: VSTs for the Web!

**SpectrumModalModule**

Time: 0.30, FbK: 0.60, Wet: 0.30  
Base: 0.00, Width: 0.00, Sharp: 0.00  
Ping/Pong:

Contour: 0.00, Bow: 0.00, Blow: 0.00, Strike: 1.00  
Mallet: 0.00, Geometry: 0.69, Brightness: 0.30  
Bow Timbre: 0.00, Blow Timbre: 0.00, Strike Timber: 0.50, Damping: 0.30, Position: 0.30, Space: 1.28

**webCZ-101**

P-01  
BRASS ENS. 1

OSC 1, OSC 2, OSC 3  
WAVEFORM: SINE, SQUARE, TRIANGLE, PULSE

**webOBXD // ABS-OBXD-Custom Shop v1.6b**

OSCILLATORS: OSC 1, OSC 2, OSC 3  
MIX: MIX, MIX, MIX  
FILTER: CUTOFF, RESO, ENV, ENV  
AMPLIFIER ENVELOPE: A, D, S  
VOICE VARIATION: FILTER, ENV, ENV  
CONTROL: LEGATO, VOICES, VAL, LEARN, CLEAR

**TEMPERK**

High, HighMid, mDepth, mid, lowMid, nFreq, low, earlyOff, damp, size, wet, 160

**DualPitchShifter**

Mix, Shift, Shift, Windows Size

**VoxAmp 30**

Clean / Crunch

Fuzz Face, Fuzz, Level, Buss Cut, Effect, Humbucker Sim, Distortion Wah, Pitch, Formant, Csound PitchShifter

**JPVerb**

damping, diffusion, modFreq, delayTime, feedback, size, modDepth, GREYHOLE

**BIG MUFF**  
**WAH BABY**  
**UTILITY**  
**NOISE GATE**

**Phaser Rix**  
**StereoFreqShifter**  
**TimeZeroFlanger**  
**WetPhase**

**DISTO MACHINE**

VOLUME MASTER, DRIVE, BASS, MIDDLE, TREBLE, REVERB, PRESENCE

**DECAY**  
**MIX**  
**SHIMMER**  
**FAUSTEQ**

**Blues Machine**  
**MODERN METAL MACHINE**

**DEAD ZONE**  
**NOISE GATE**  
**KBVerb**  
**DEXED**  
**Smooth Delay**

**AUGUR (TRACK 1)**  
**OSC**  
**FILTER**  
**AMP**  
**GLOBAL**  
**MIXENV LFO**  
**ENV MATRIX WAVES**

**STONE PHASER**

**Portamento**  
**LFO**  
**Oscillator**  
**Mixer**  
**Filter**

**DrumSampler**

OH, Crash, Ride, Low Tom, Mid Tom, High Tom, CH

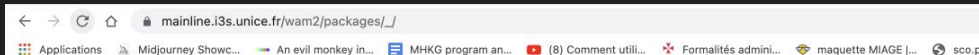
Kick, Output: Tone, Pan, Gain

**PianoRollModule**

Settings, C3, G3, Output: Tone, Pan, Gain

**Fuzz**  
**RESONANCE**  
**OscTube**

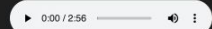
# Wams in hosts... here with a host from the WAM distrib <https://mainline.i3s.unice.fr/wam2/packages/> / /



## Example WAM Host

NEW: See [WAM 2.0 API docs](#)

NEW: See [WAM 2.0 Parameter Manager docs](#)



Select MIDI input device **Select...**

Select live input device **Par défaut - MacBook Pro Microphone (Built-in)**

Live input: NOT ACTIVATED, click to toggle on/off

Enter any WAM Plugin URL

LOAD PLUGIN

Midi

- Simple MIDI Keyboard
- Midi Virtual keyboard No Sound (just emits events)
- Random note generator
- Simple Transport
- MIDI Sequencer

Pure JavaScript

- Quadrafuzz
- Quadrafuzz without builder
- Disto Machine without builder
- PingPongDelay
- Graphic Equaliser (pureJS)
- WAM Example (pureJS)
- TinySynth, a GM synth without GUI. Accepts program changes, volume change, notes...
- Wam Event Viewer
- WebMIDI Output

Faust

- faustPingPongDelay
- faustPingPongDelayDefaultUI
- Guitar AmpSim 60s (generated by faust IDE)
- Stone Phaser (generated by faust IDE)
- TSS Overdrive (generated by faust IDE)
- BigMuff Fuzz (generated by faust IDE)
- faust Flute MIDI

Typescript

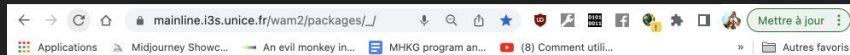
- LiveGain
- Oscilloscope
- Spectroscope
- Spectrogram

C#sound

- Cound PitchShifter



```
SAVE STATE RESTORE STATE
+ instance.descriptor : { "name": "OBXD", "vendor": "Jari Kleimola 2017-2020 (jari@webaudiomodels.org)", "description": "", "version": "1.0.0", "apiVersion": "1", "instrument": true, "website": "", "hasAudioInput": true, "hasAudioOutput": true }
+ gui.properties.dataWidth.value : undefined, (you should define get properties in Gui.js)
+ gui.properties.dataHeight.value : undefined, (you should define get properties in Gui.js)
+ instance.audioNode.getParameterInfo() :
```



Select MIDI input device **Select...**

Select live input device **Par défaut - MacBook Pro Microphone (Built-in)**

Live input: NOT ACTIVATED, click to toggle on/off

Enter any WAM Plugin URL

LOAD PLUGIN

Midi

- Simple MIDI Keyboard
- Midi Virtual keyboard No Sound (just emits events)
- Random note generator
- Simple Transport
- MIDI Sequencer

Pure JavaScript

- Quadrafuzz
- Quadrafuzz without builder
- Disto Machine without builder
- PingPongDelay
- Graphic Equaliser (pureJS)
- WAM Example (pureJS)
- TinySynth, a GM synth without GUI. Accepts program changes, volume change, notes...
- Wam Event Viewer
- WebMIDI Output

Faust

- faustPingPongDelay
- faustPingPongDelayDefaultUI
- Guitar AmpSim 60s (generated by faust IDE)
- Stone Phaser (generated by faust IDE)
- TSS Overdrive (generated by faust IDE)
- BigMuff Fuzz (generated by faust IDE)
- faust Flute MIDI

Typescript



SAVE STATE RESTORE STATE

```
+ instance.descriptor : { "name": "Faust Flute MIDI", "vendor": "Grame", "description": "Faust MIDI Instrument", "version": "1.0.0", "apiVersion": "2.0.0", "thumbnail": "screenshot/flute", "faust": true, "instrument": true, "website": "", "hasAudioInput": true, "hasAudioOutput": true }
+ gui.properties.dataWidth.value : undefined, (you should define get properties in Gui.js)
+ gui.properties.dataHeight.value : undefined, (you should define get properties in Gui.js)
+ instance.audioNode.getParameterInfo() :
+ /flute/otherParams/vibratoFreq : { "id": "/flute/otherParams/vibratoFreq", "label": "/flute/otherParams/vibratoFreq", "units": "" }
+ /flute/midi/freq : { "id": "/flute/midi/freq", "label": "/flute/midi/freq", "type": "float", "units": "" }
+ /flute/midi/lastNote : { "id": "/flute/midi/lastNote", "label": "/flute/midi/lastNote", "type": "float", "units": "" }
```



# WAMs in hosts... WIP open source DAW (100% WAM-based)

The screenshot displays the WAM-OpenStudio DAW interface. At the top, the browser address bar shows the URL `wam-openstudio.vidalmazuy.fr`. Below the browser bar is a transport control bar with a play button, a stop button, a time display of `00:00:32.424`, and volume controls. The main workspace is divided into several sections:

- TRACKS:** A vertical list on the left contains five tracks: `bass.ogg`, `song.ogg`, `cymbals.ogg`, `guitar.ogg`, and `kick.ogg`. Each track has a volume knob and a solo button.
- WAVEFORMS:** The main area shows waveforms for each track. The `guitar.ogg` track is highlighted in purple and features a white envelope curve with key points labeled: `0.59, 18.02`, `15.12, 88.87`, `26.74, 9.92`, and `41.64, 88.87`.
- PLUGINS:** A bottom section contains a grid of virtual analog modules. The `GuitarAmpSim60s` plugin is the largest and most prominent, featuring a `VoxAmp 30` interface with knobs for `Volume`, `Bass`, `Middle`, `Treble`, and `Master`, and a `Clean / Crunch` selector. Other visible plugins include `StonePhaser`, `Faust BigMuff`, `Lowgain Multiwa`, `Highgain Multiwa`, `QuadraFuzz`, and `Faust Delay`.
- MIXER:** On the right side, there are three panels: `Banks` (with a `New Bank` button and a `Rock Pop` preset), `Presets`, and `Information`. A blue waveform is visible across these panels, likely representing the master mix.

# WAMs in hosts... here the WASABI pedalboard

The screenshot shows a digital guitar pedalboard software interface titled "PedalBoard" running in a web browser. The interface features a central workspace where various virtual guitar pedals are connected in a signal chain. The pedals include:

- Dead Gate** (teal)
- Blue Dream** (blue)
- QuadraFuzz** (black and white)
- DISTO MACHINE** (grey with red and white stripes)
- FAUST STEREO FREQUENCY SHIFTER** (teal)
- FAUST ZitaRev** (grey with a pixelated face)
- CHANNEL 1** and **CHANNEL 2** (orange)

Below the workspace is a category menu with tabs for: UNDEFINED (1), NORBERT2019 (1), WAC2019 (6), DYNAMIC (1), FAUSTIDE (19), TUNER (1), GATE (1), PERCUSSION (1), AMPSIM (3), EXPRESSION (1), MIXING (3), DELAY (1), MODULATION (6), DISTORTION (2), REVERB (1). Under the "FAUSTIDE (19)" tab, a row of pedals is displayed, including:

- Humbucker Sim
- BIG MUFF
- DUSTO RIDER
- JFIB
- DwShimmer
- GreyHoleRaw
- OverdriveRix
- OwDIrty
- Stereo Chorus
- MANBABY
- isTube



# WAMs in hosts... <https://sequencer.party/>

The screenshot displays the sequencer.party web interface. The browser address bar shows the URL: `editor.sequencer.party/sessions/39406024a97445e49199f5b780b64597/track/ufosxp/`. The page title is "sequencer.party: Wimmics test 1".

The main interface is divided into two main sections:

- Left Section (Piano Roll):** A piano roll titled "PianoRollModule" with a red "Settings" bar at the top. The piano roll shows a grid with three notes on the C3 staff (middle C) and one note on the C2 staff. The notes are represented by red bars.
- Right Section (Signal Chain):** A series of interconnected modules:
  - SpectrumModelModule:** A blue module with various parameters: Contour (0.00), Bow (0.00), Blow (0.00), Strike (0.70), Model (Chords), Flow (0.00), Mallet (0.50), Geometry (0.20), Brightness (0.67), Bow Timbre (0.00), Blow Timbre (0.00), Strike Timbre (0.28), Damping (0.56), Position (0.69), and Space (0.58).
  - greyholePlugin:** A purple module with parameters: damping, diffusion, modFreq, delayTime, feedback, size, and modDepth. The name "GREYHOLE" is prominently displayed at the bottom.
  - untitledPlugin:** A colorful module with parameters: DECAY, MIX, SHIMMER, and TONE.

Red and blue lines connect the piano roll to the SpectrumModelModule, and the SpectrumModelModule to the greyholePlugin, and finally to the untitledPlugin, indicating the signal flow.

# Wams in hosts... <https://app.ampedstudio.com/>

The screenshot displays the Amped Studio web application interface. At the top, the browser address bar shows the URL [app.ampedstudio.com](https://app.ampedstudio.com/). The interface includes a top navigation bar with various icons and a digital display showing a time of 0:02:12.50, a tempo of 120 bpm, and a 4/4 time signature. The main workspace features a timeline with tracks for Backing Track, Guitar main, Track 3, Track 4, and Master Track. The Guitar main track is active, showing a green waveform. Two virtual guitar pedals are overlaid on the track: the 'AMP SIM UTILITY (GUITAR MAIN)' and the 'DISTORTION MACHINE (GUITAR MAIN)'. The Amp Sim Utility pedal has controls for Input Gain and Noise Gate. The Distortion Machine pedal has controls for Volume, Master, Drive, Bass, Middle, Treble, Reverb, and Presence. A device chain at the bottom shows the sequence of these two pedals. On the right side, there is a Sound Library panel with a search bar and a list of sound presets, including 'Joan Jett - I love Rock'n'Roll' and 'Recorded Audio Nov 2, 20...'. At the bottom right, there are buttons for 'BUY SOUNDS' and 'CONTACT US'.

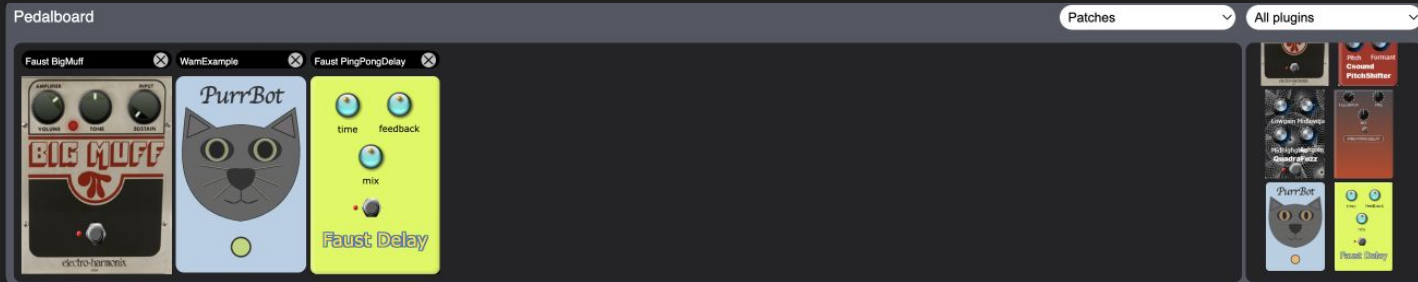
# WAMs in hosts... JSPatcher, aka Max MSP in the browser (<https://github.com/Fr0stbyteR/jspatcher>)



# WAMs in plugins that acts as hosts (i.e pedalboard)

These ones  
are open  
source

(from [wam-examples](#)  
and [wam-community](#)  
github repositories)



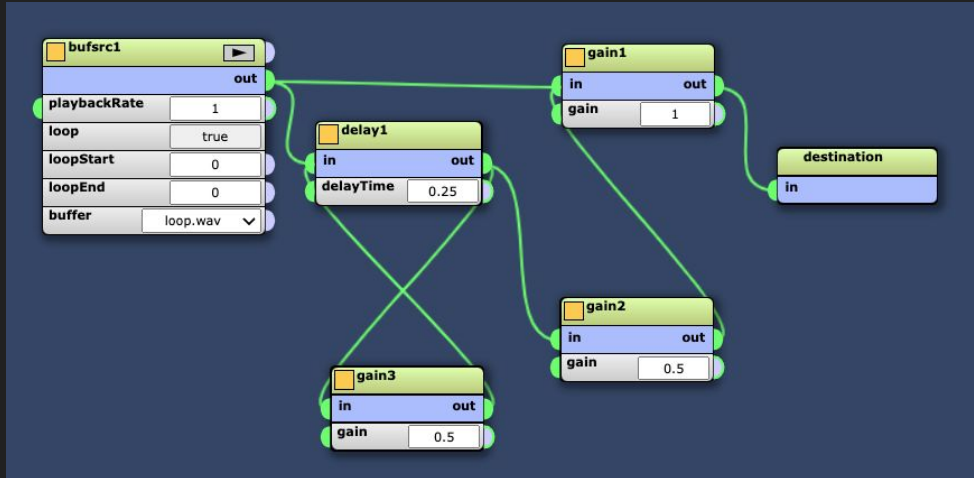
... and loaded in a DAW!

The image shows a screenshot of a Digital Audio Workstation (DAW) interface. The top part displays the browser address bar with the URL `wam-openstudio.vidalmazuy.fr` and various browser tabs. Below the browser, the DAW interface includes a transport bar with a play button and a time display of `00:00:26.266`. The main area contains several audio tracks with waveforms. The tracks are labeled on the left as `Cysoasa.ogg`, `guitar.ogg`, `song.ogg`, `kick.ogg`, and `snaretoms.ogg`. Each track has volume and pan controls. A red arrow points from the text **This is the pedalboard plugin** to the `song.ogg` track. At the bottom, a red-bordered panel displays a collection of guitar pedalboard plugins. The plugins shown include `DeathGate`, `WeirdPhaser`, `ThruZeroFlan...`, `KPP distortion`, and `DistoMachine (No builder)`. The `DistoMachine` plugin is highlighted, showing its interface with knobs for `Volume`, `Master`, and `Drive`. To the right of the plugins, there are sections for `Banks` and `Presets`, with a `New Bank` and `New Preset` button respectively. The `Banks` section shows a list of banks including `Rock`, `Pop`, and `Jordan`. The `Presets` section shows a list of presets including `Stonesour`. The `Information` section shows a waveform visualization.

A few reminders before proceeding...

# W3C WebAudio API: connecting nodes and build an audio graph, it's a low level API...

Example: an audio delay effect, made with standard audio nodes. You assemble the graph in JavaScript (main thread), then control the parameters, the GUI. Audio rendering is done in the browser, in the audio thread.



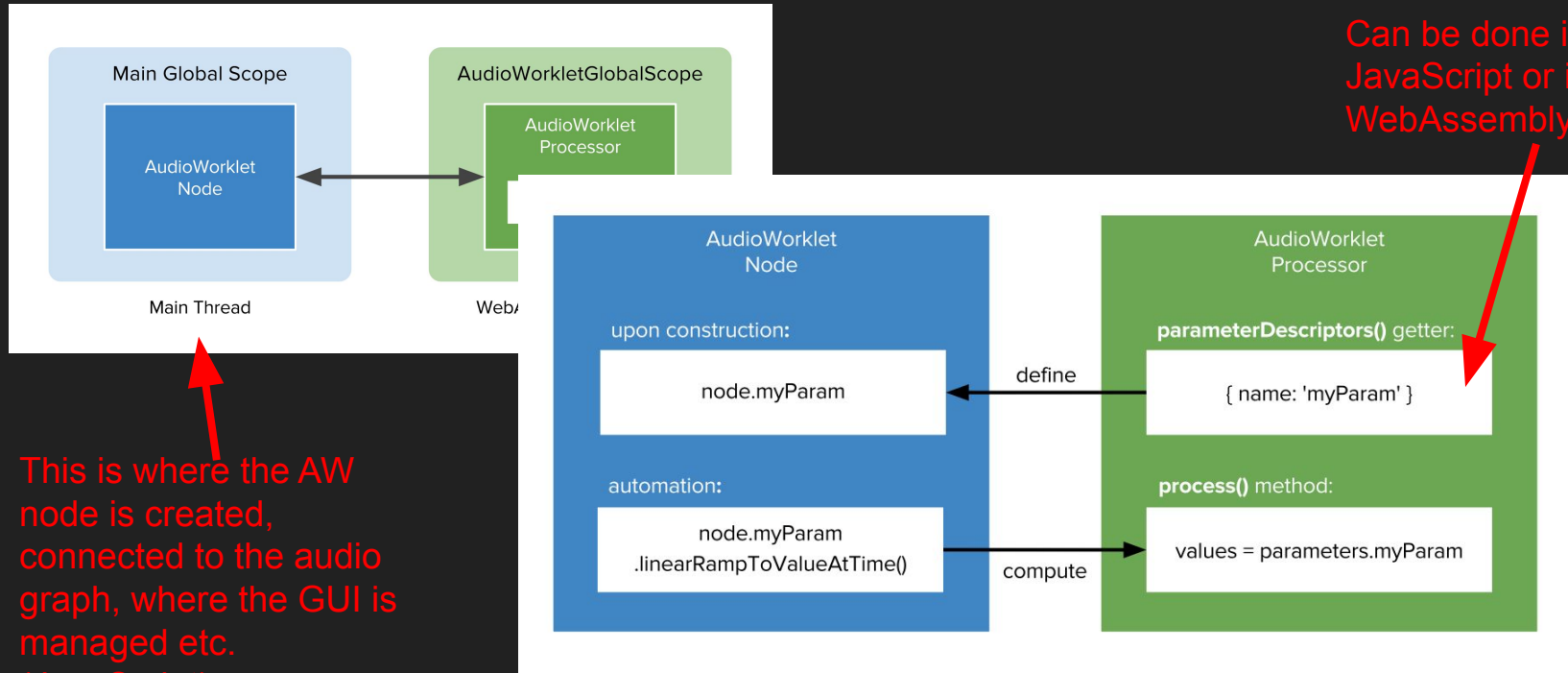
```
this.nodes.gain1 = this.audioctx.createGain();
this.nodes.bufsrc1 = this.audioctx.createBufferSource();
this.nodes.bufsrc1.loop = true;
this.nodes.bufsrc1.buffer = this.buffers['loop.wav'].data;
this.nodes.delay1 = this.audioctx.createDelay();
this.nodes.delay1.delayTime.value = 0.25;
this.nodes.gain2 = this.audioctx.createGain();
this.nodes.gain2.gain.value = 0.5;
this.nodes.gain3 = this.audioctx.createGain();
this.nodes.gain3.gain.value = 0.5;
```

```
this.nodes.gain1.connect(this.nodes.destination);
this.nodes.bufsrc1.connect(this.nodes.delay1);
this.nodes.bufsrc1.connect(this.nodes.gain1);
this.nodes.delay1.connect(this.nodes.gain2);
this.nodes.delay1.connect(this.nodes.gain3);
this.nodes.gain2.connect(this.nodes.gain1);
this.nodes.gain3.connect(this.nodes.gain1);
```

# The Web Audio API also supports custom DSP programming with the AudioWorklet

This is where custom DSP processing is done!

Can be done in JavaScript or in a WebAssembly module!



This is where the AW node is created, connected to the audio graph, where the GUI is managed etc. (JavaScript)



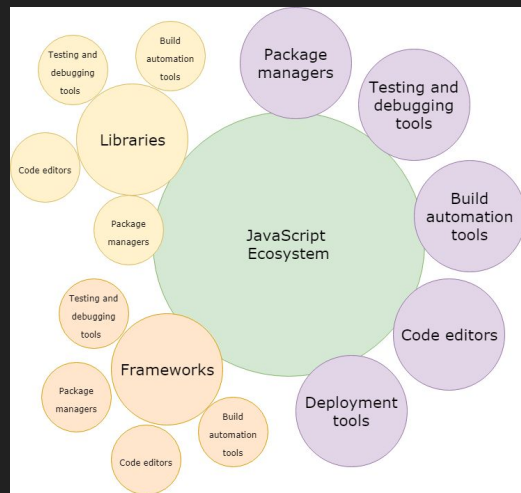
# Web Audio development

## Web developers

- Use plain HTML/CSS/JavaScript but very often also bundlers/minifiers (webpack, parcel, rollup), npm modules, frameworks (react, vueJS), and also code with TypeScript, etc.

## Audio developers

- Use C++/Rust, DSLs like FAUST, Csound, CMajor, patchers like Max, etc.
- Use plugin standards: VSTs, AU, AAX, RTAS, JUCE, CLAP, iPlug2, etc.



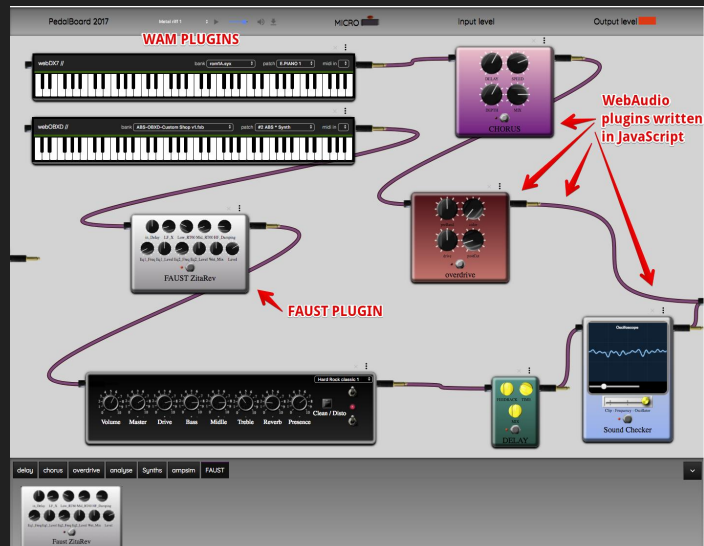
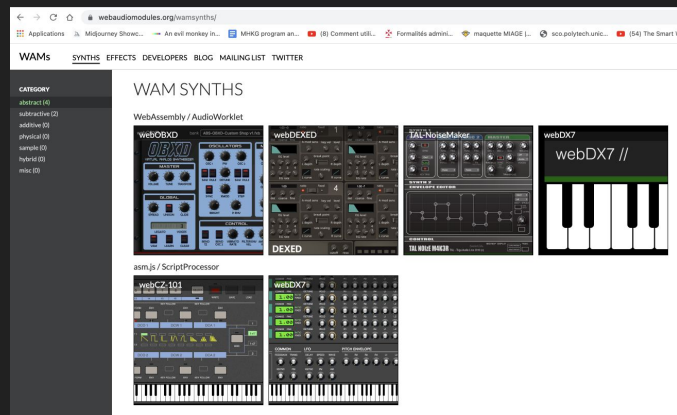
# How to combine all this?

## 2015: First WAM proposal by J.Kleimola and O.Larkin

- Attract native developers, help going from C++ plugins to AudioWorklet/ASM.js and later WebAssembly,
- <http://webaudiomodules.org> has impressive synths ported from VST/JUCE/iPlug2

## 2018: Enlarge the proposal (Buffa and al.)

- Please web developers,
- Support DSLs like FAUST, other improvements...



# WebAudioModules version 2 (aka WAM or WAM2)

- **2021-2022: WebAudio Modules 2.0**

- A WAM plugin can be loaded using a simple URI!
- A WAM plugin is a JavaScript module,
- A WAM can be made of a single AudioWorklet Node, or made of multiple nodes, it will behave like a single AudioNode.
- Plugin parameters are handled by the WamParamMgr,
- Focus on performance (ring buffer, audio thread isolation)
- Plain modern JS or build systems for JS / TS / frameworks
- Support for C/C++
- Support for DSL (Faust, CSound)
- Parameter Automation, MIDI support,
- host/plugin interaction as an API (+ rich SDK). The API can be entirely re-implemented for low-level plugins

**Example WAM Host**

NEW: See WAM 2.0 API docs  
NEW: See WAM 2.0 Parameter Manager docs

0:00 / 2:57

Select MIDI input device [ ]  
Select live input device [microphone 1]  
Live input: NOT ACTIVATED, click to toggle on/off!

Enter any WAM Plugin URI  
[pedalboard/dist/index.js]

LOAD PLUGIN

Midi

- Simple MIDI Keyboard
- Midi Virtual keyboard No Sound (just emits events)
- Random note generator

Pure JavaScript

- Quadrafuzz
- Quadrafuzz without builder
- Disto Machine without builder
- PingPongDelay
- Grmire's Reverbizer (pureJS)
- WAM Example (pureJS)
- TinySynth, a GM synth without GUI. Accepts program changes, volume change, notes...

Faust

- FaustPingPongDelay
- FaustPingPongDelayDefaultUI
- Guitar AmpSim 60s (generated by faust IDE)
- Stone Dancer (generated by faust IDE)
- TS9 Overdrive (generated by faust IDE)
- BigMuff Fuzz (generated by faust IDE)

```
initOnce, descriptor :
{
  "name": "Pedalboard",
  "vendor": "WebAudioModule",
  "description": "A rack to load multiple WAMs",
  "version": "1.0.0",
  "sdkVersion": "1.0.0",
  "thumbnail": "screenshot.png",
  [1],
  "isInstrument": false,
  "website": "",
  "hasAudioInput": true,
  "hasAudioOutput": true,
  "gui.properties.dataWidth.value": undefined,
  (you should define get properties in Gui.js)
  "gui.properties.dataHeight.value": undefined,
  (you should define get properties in Gui.js)
  "instance.audioNode.getParameterInfo()":
  [
    {
      "id": "0",
      "label": "Faust BigMuff//BigMuff /Drive",
      "type": "float",
      "defaultValue": 3,
      "maxValue": 100,
      "discreteStep": 0,
      "exponent": 0,
      "choices": [1],
      "units": ""
    },
    {
      "id": "1",
      "label": "Faust BigMuff//BigMuff"
    }
  ]
}
```

How to start with WAMs!

# Github repo, Home page of the project...

WebAudioModules (WAM) is an old standard (2015), and WAM2 is the updated version:

- [webaudiomodules.org](https://webaudiomodules.org) will remain the home of the project (not yet up to date! Soon with a section about WAM2!)
- The official github repo is the regular webaudiomodules one: <https://github.com/webaudiomodules>.
- Everything is under MIT/MPL/Apache 2.0 open source licence...
- Also available as [npm modules](#)

The image displays three overlapping screenshots. The top one is the website 'WAMs' (Web Audio Modules) with a blue header and 'Web Audio' text. The middle one is the NPM profile for 'webaudiomodules', showing a list of packages: '@webaudiomodules/api' (WebAudioModules Plugin API, published 2.0.0-alpha.0), '@webaudiomodules/sdk' (WebAudioModules Plugin SDK, published 0.0.2), and '@webaudiomodules/sdk-parammgr' (Parameter Manager SDK for WebAudioModules Plugin, published 0.0.1). The bottom one is the GitHub repository page for 'webaudiomodules/sdk-parammgr', showing it's a JavaScript package under the MIT license, with 3 forks and 21 stars, updated on May 24.

# API vs SDK

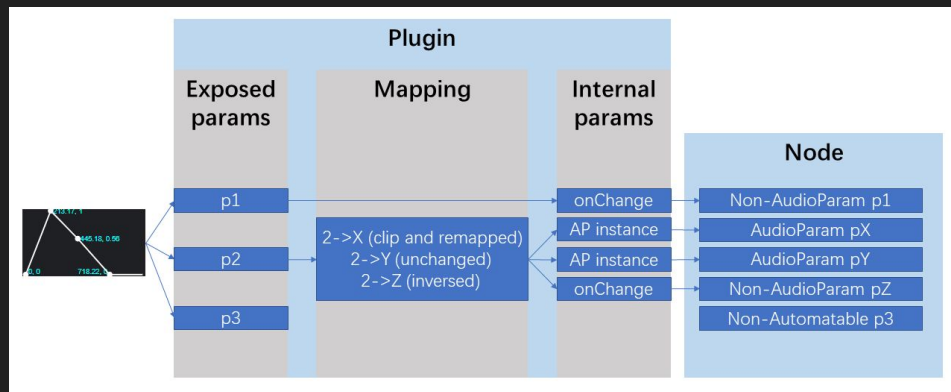
API (Standard)	SDK (Implementation/Tools)
<ul style="list-style-type: none"><li>- Defines required methods</li><li>- Abstract classes</li></ul>	<ul style="list-style-type: none"><li>- Reference API implementations</li><li>- Utility classes and example plugins</li></ul>

- Developers can choose to adapt their existing code to the API
- Others can use the SDK that implements the API (much easier), inherit classes etc.

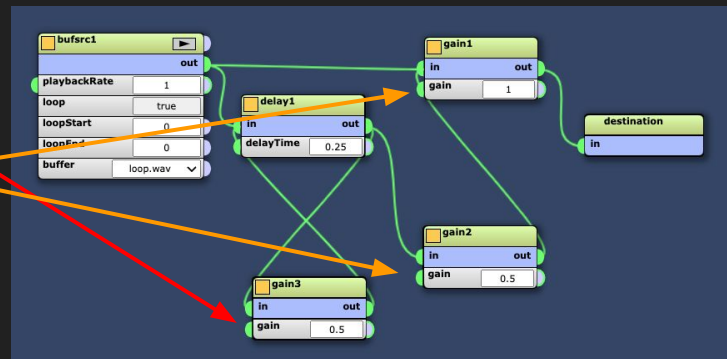
# The sdk-parammgr repository

Dedicated to plugins made of an audio graph:

- Exports a **CompositeNode** class, the plugin will be seen as a single node!
- Deals with parameter mapping and automation (figure).

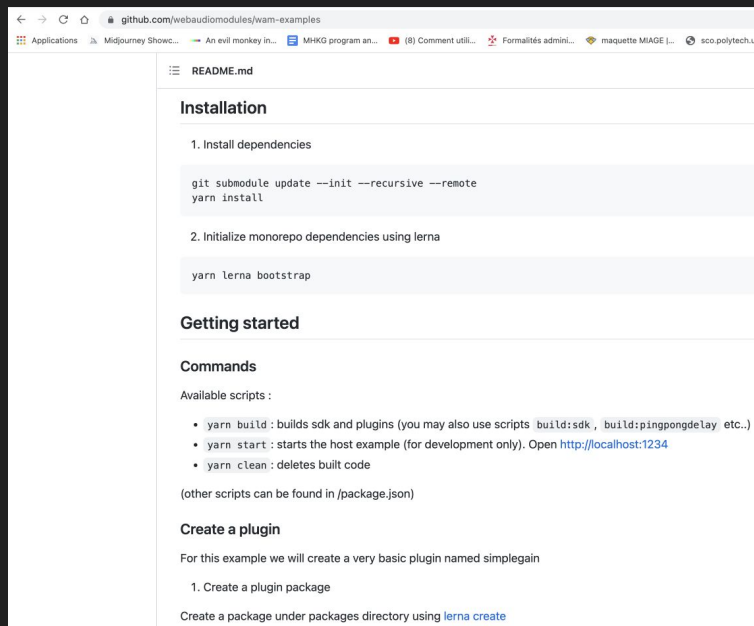


3 exposed  
parameters



Dozens of internal parameters

# The wam-examples repository



github.com/webaudiomodels/wam-examples

## Installation

1. Install dependencies

```
git submodule update --init --recursive --remote
yarn install
```
2. Initialize monorepo dependencies using lerna

```
yarn lerna bootstrap
```

## Getting started

## Commands

Available scripts :

- `yarn build` : builds sdk and plugins (you may also use scripts `build: sdk`, `build: pingpongdelay` etc...)
- `yarn start` : starts the host example (for development only). Open <http://localhost:1234>
- `yarn clean` : deletes built code

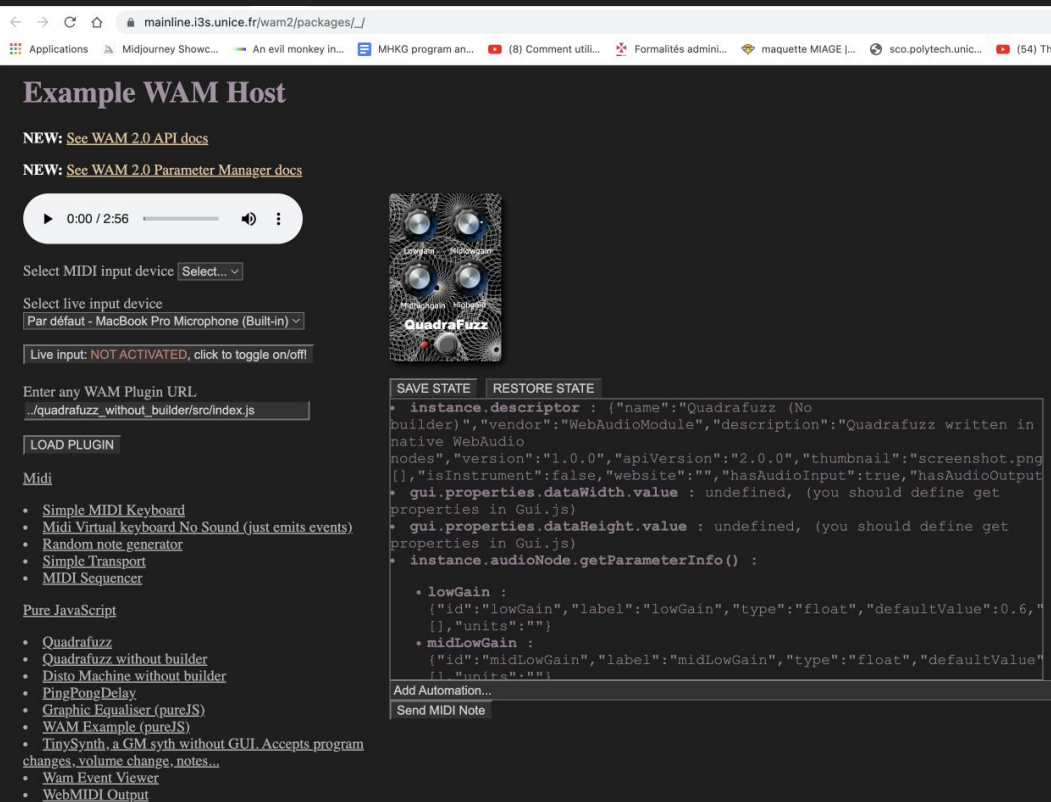
(other scripts can be found in /package.json)

## Create a plugin

For this example we will create a very basic plugin named simplegain

1. Create a plugin package

Create a package under packages directory using [lerna create](#)



mainline.i3s.unice.fr/wam2/packages/\_/

## Example WAM Host

**NEW:** See [WAM 2.0 API docs](#)

**NEW:** See [WAM 2.0 Parameter Manager docs](#)

0:00 / 2:56

Select MIDI input device:

Select live input device:

Live input: NOT ACTIVATED, click to toggle on/off!


Enter any WAM Plugin URL:

### Midi

- Simple MIDI Keyboard
- Midi Virtual keyboard No Sound (just emits events)
- Random note generator
- Simple Transport
- MIDI Sequencer

### Pure JavaScript

- Quadracruz
- Quadracruz without builder
- Disto Machine without builder
- PingPongDelay
- Graphic Equaliser (pureJS)
- WAM Example (pureJS)
- TinySynth, a GM syth without GUI. Accepts program changes, volume change, notes...
- Wam Event Viewer
- WebMIDI Output



SAVE STATE RESTORE STATE

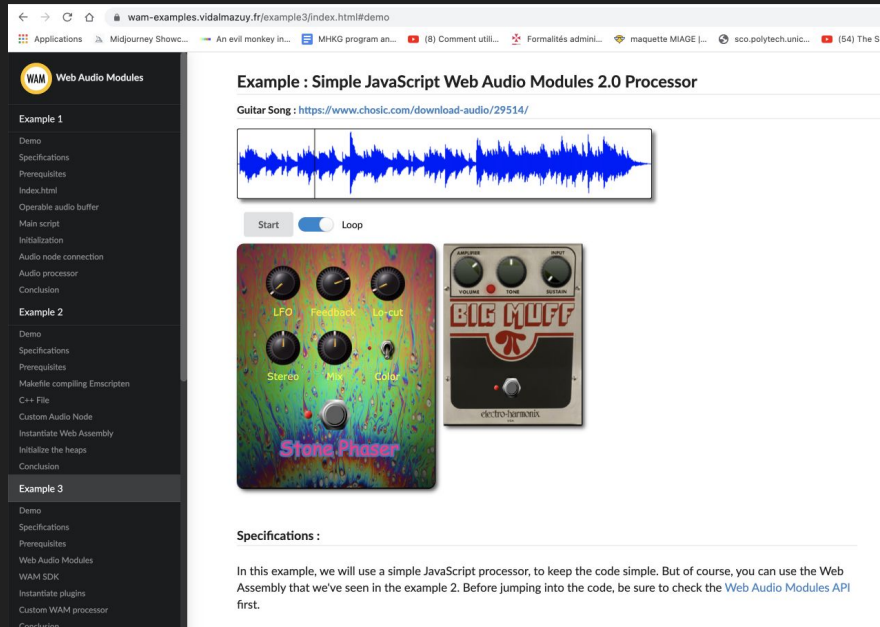
```
* instance.descriptor : {"name":"Quadracruz (No builder)", "vendor":"WebAudioModule", "description":"Quadracruz written in native WebAudio nodes", "version":"1.0.0", "apiVersion":"2.0.0", "thumbnail":"screenshot.png", "isInstrument":false, "website":"","hasAudioInput":true, "hasAudioOutput":true, "gui.properties.dataWidth.value": undefined, (you should define get properties in Gui.js), "gui.properties.dataHeight.value": undefined, (you should define get properties in Gui.js), "instance.audioNode.getParameterInfo() : [{"id":"lowGain", "label":"lowGain", "type":"float", "defaultValue":0.6, "units":""}, {"id":"midLowGain", "label":"midLowGain", "type":"float", "defaultValue":1, "units":""}]}

Add Automation...
Send MIDI Note
```



# WAM2 step by step tutorials

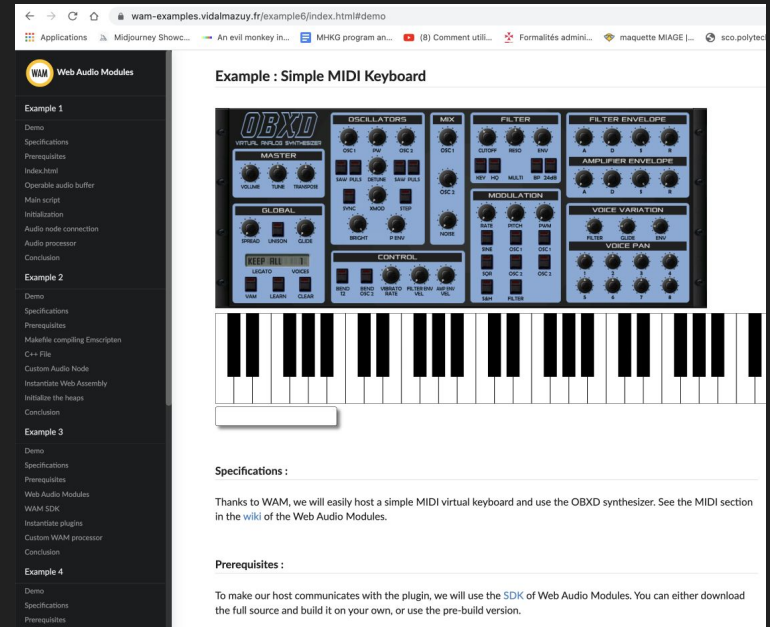
Several tutorials are available at <https://wam-examples.vidalmazuy.fr/>



The screenshot shows a web browser window with the URL `wam-examples.vidalmazuy.fr/example3/index.html#demo`. The page title is "Example 3 : Simple JavaScript Web Audio Modules 2.0 Processor". Below the title, it says "Guitar Song : <https://www.chosic.com/download-audio/29514/>". There is a blue audio waveform visualization. Below the waveform, there is a "Start" button and a "Loop" toggle switch which is currently turned on. At the bottom, there are two images of guitar pedals: a "Stone Phaser" and a "Big Muff".

**Specifications :**

In this example, we will use a simple JavaScript processor, to keep the code simple. But of course, you can use the Web Assembly that we've seen in the example 2. Before jumping into the code, be sure to check the [Web Audio Modules API](#) first.



The screenshot shows a web browser window with the URL `wam-examples.vidalmazuy.fr/example6/index.html#demo`. The page title is "Example 6 : Simple MIDI Keyboard". The main content is a virtual synthesizer interface for the OBXD synthesizer. It features various sections: MASTER (VOLUME, TUNE, RESONANCE), GLOBAL (SPREAD, UNISON, GLUE), CONTROL (KEEP, HOLD, TILT, LEGATO, VOICE), OSCILLATORS (OSC1, PAW, OSC2, SAW PULS, DETUNE, SAW PULS, TRIM), MIX (OSC1, OSC2), FILTER (CUTOFF, RESO, ENV), FILTER ENVELOPE (A, D, S, R), AMPLIFIER ENVELOPE (A, D, S, R), MODULATION (ENV, MOD, MULTI, BP, ZAMP), VOICE VARIATION (FILTER, CUTOFF, ENV), and VOICE PAN (L, C, R). Below the synthesizer is a MIDI keyboard with a white key highlighted. The left sidebar contains a navigation menu with "Example 6" selected.

**Specifications :**

Thanks to WAM, we will easily host a simple MIDI virtual keyboard and use the OBXD synthesizer. See the MIDI section in the wiki of the Web Audio Modules.

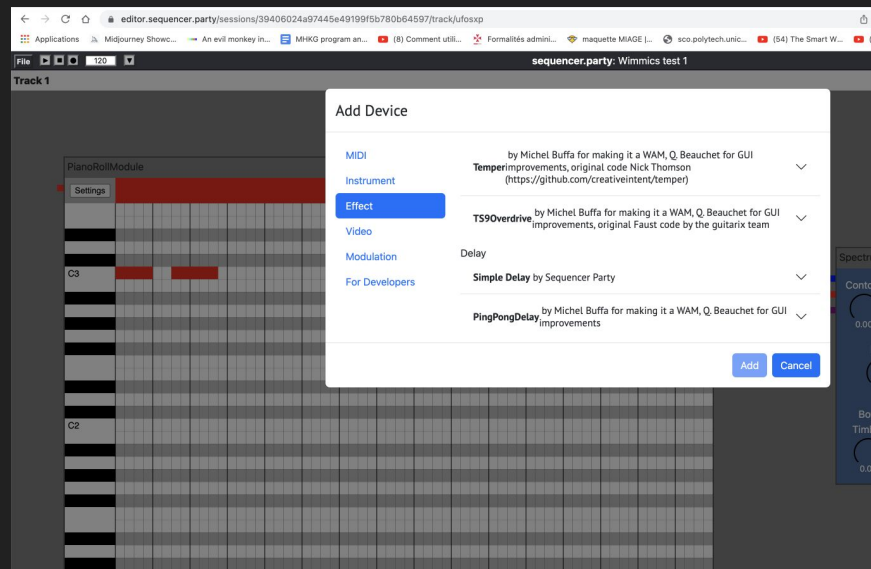
**Prerequisites :**

To make our host communicates with the plugin, we will use the SDK of Web Audio Modules. You can either download the full source and build it on your own, or use the pre-build version.

# Also, check the wam-community repository <https://github.com/boourns/wam-community>

Used by the community to publish and share “ready to use” plugins!

- Remember that a plugin is just a URI!
- **Several dozens of plugins available**, 99% also available with source code to study ([github.com/boourns/bourns-audio-wam](https://github.com/boourns/bourns-audio-wam))
- Cover all classic effects, proposes some instruments and utility plugins.
- All plugins available in the <https://sequencer.party> host.



# Build a WebAssembly WAM in seconds with FAUST DSL



Functional  
Audio  
Stream

**FAUST: a DSL for DSP programming, born in 2002 at GRAME-CNCM, France**

Used in artistic productions, education and research, open source projects and commercial applications.

Faust offers end-users a high-level alternative to C/C++ to develop audio applications for a large variety of platforms.

The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, WebAssembly, etc.).

[Online doc / tutorial so that you can experiment yourself](#), create, build GUI, export WAM2 plugins directly from the [FAUST online IDE](#).

# Conclusion / Perspectives

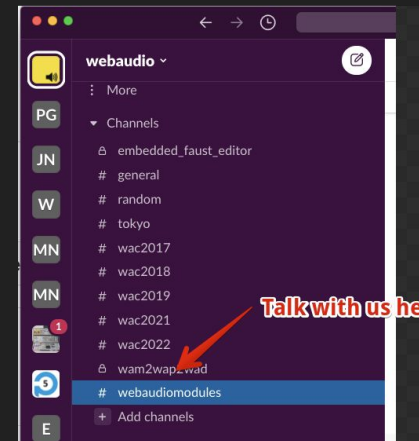
The WAM standard is stable now and comes with many examples.

**OUR MAIN CONCERN NOW: get inputs from developers regarding barriers to entry / adoption!**

Things that will come soon:

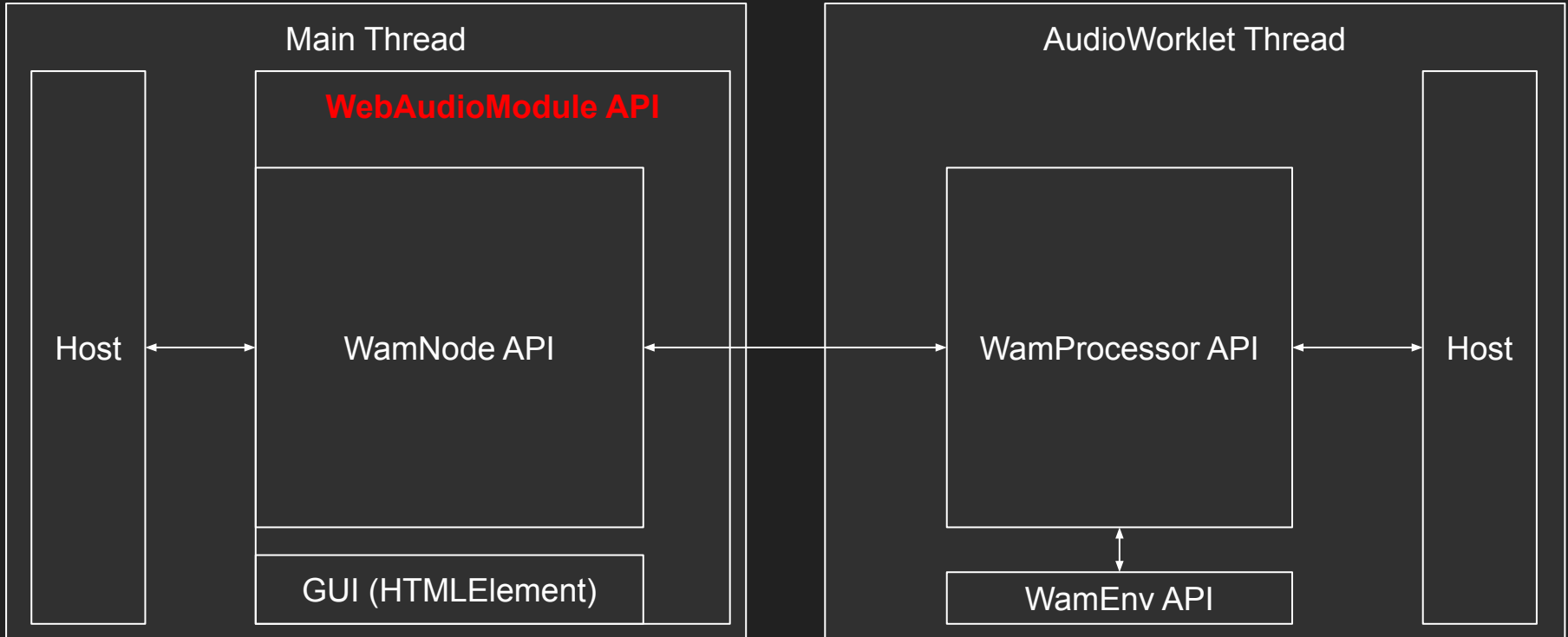
- The wam-community repo is growing :-)
- The WAM SDK [has been extended to support 3D WebGL/GLSL/Video extensions](#)
- A WAM based DAW is under development and will be open source
- More examples using WASM, in particular C++/WASM
- Remote plugin server with API

**Join us on slack WebAudio channel / #webaudiomodules!**








# Interact with a host, be it in main or audio thread

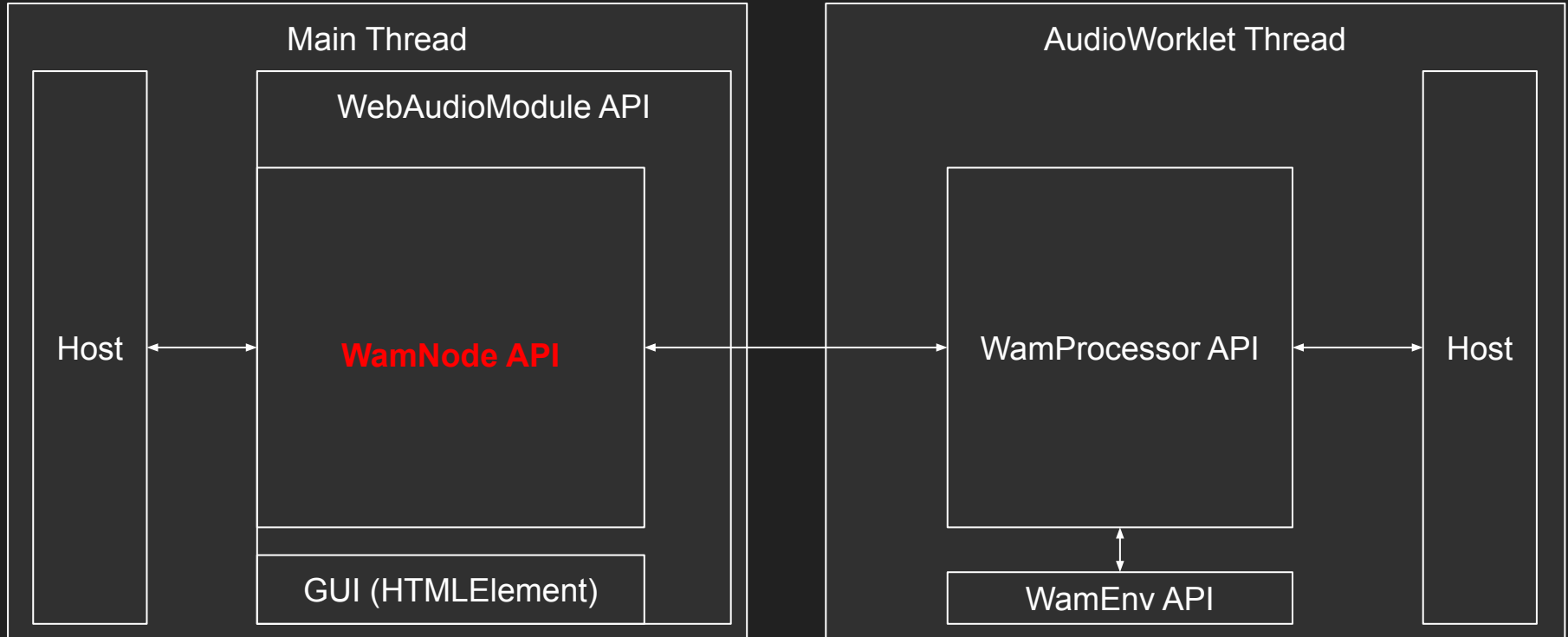
## 1 - The WebAudioModule API



# A plugin = instance of a WAM = “a WAM” API - WebAudioModule

```
export interface WebAudioModule<Node extends WamNode = WamNode> {  
  /** should return `true` */  
  readonly isWebAudioModule: boolean;  
  /** The `AudioContext` where the plugin's node lives in */  
  audioContext: BaseAudioContext;  
  /**  
   * The `AudioNode` that handles audio in the plugin  
   * where the host can connect to/from  
   */  
  audioNode: Node;   
  /** This will return true after calling `initialize()`. */  
  initialized: boolean;  
  /** The identifier of the current WAM, composed of vendor + name */  
  readonly moduleId: string;   
  /** The unique identifier of the current WAM instance. */  
  readonly instanceId: string;   
  /** The values from `descriptor.json` */  
  readonly descriptor: WamDescriptor;  
  /** The WAM's name */  
  readonly name: string;  
  /** The WAM Vendor's name */  
  readonly vendor: string;  
}   
  
/**  
 * This async method must be redefined to get `AudioNode` that  
 * will be connected to the host  
 * It can be any object that extends `AudioNode` and implements `WamNode`  
 */  
createAudioNode(initialState?: any): Promise<WamNode>;  
  
/**  
 * The host will call this method to initialize the WAM with an initial state.  
 *  
 * In this method, WAM devs should call `createAudioNode()`  
 * and store its return `AudioNode` to `this.audioNode`,  
 * then set `initialized` to `true` to ensure that  
 * the `audioNode` property is available after initialized.  
 *  
 * These two behaviors are implemented by default in the SDK.  
 *  
 * The WAM devs can also fetch and preload the GUI Element in while initializing.  
 */  
initialize(state?: any): Promise<WebAudioModule>;  
/** Redefine this method to get the WAM's GUI as an HTML `Element`. */  
createGui(): Promise<Element>;   
/** Clean up an element previously returned by `createGui` */  
destroyGui(gui: Element): void
```

## 2 - The WamNode API

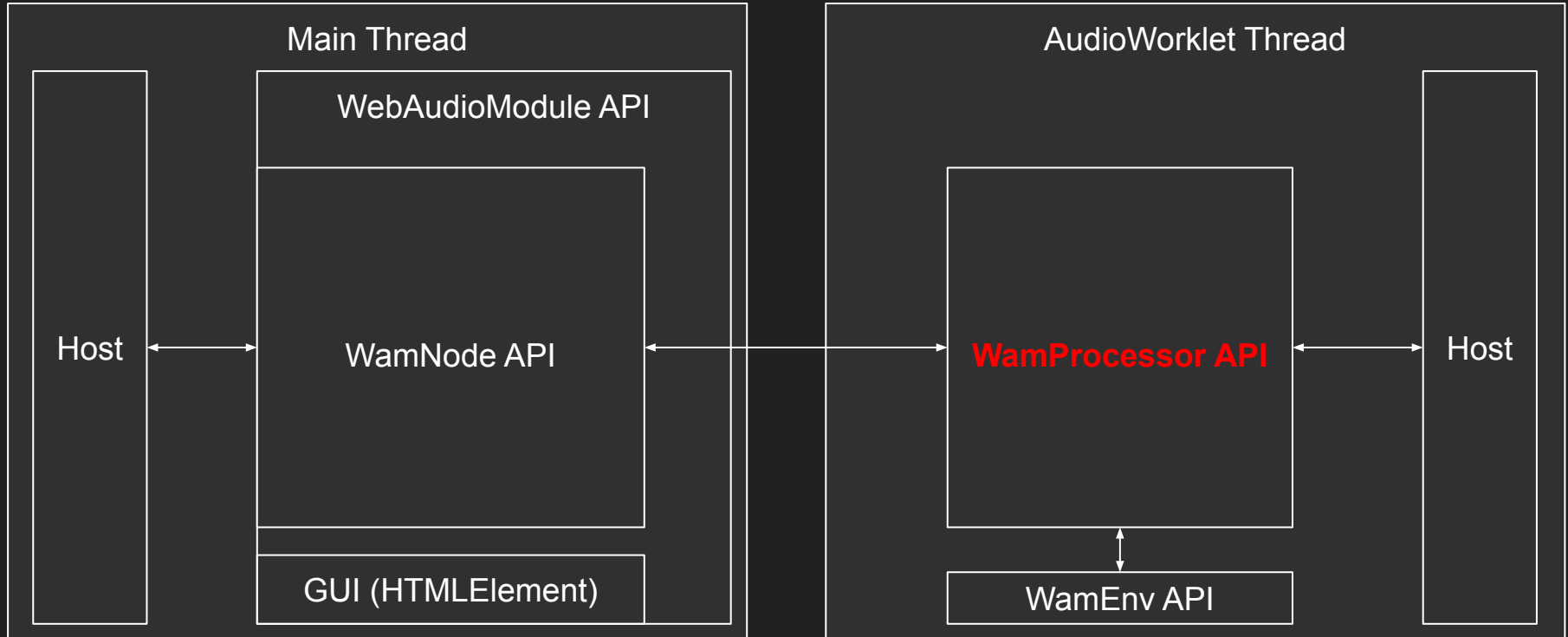


# A WAM contains a WamNode, here is the API

```
export interface WamNode extends AudioNode, Readonly<WamNodeOptions> {  
  readonly module: WebAudioModule;  
  /** Get parameter info for the specified parameter ids, or omit argument to get info for all parameters. */  
  getParameterInfo(...parameterIdQuery: string[]): Promise<WamParameterInfoMap>;  
  /** Get parameter values for the specified parameter ids, or omit argument to get values for all parameters. */  
  getParameterValues(normalized?: boolean, ...parameterIdQuery: string[]): Promise<WamParameterDataMap>;  
  /** Set parameter values for the specified parameter ids. */  
  setParameterValues(parameterValues: WamParameterDataMap): Promise<void>;  
  /** Returns an object (such as JSON or a serialized blob) that can be used to restore the WAM's state. */  
  getState(): Promise<any>;  
  setState(state: any): Promise<void>;  
  /** Compensation delay hint in samples */  
  getCompensationDelay(): Promise<number>;  
  /** Register a callback function so it will be called when matching events are processed. */  
  addEventListener<K extends keyof WamEventMap>(type: K, listener: (this: this, ev: CustomEvent<WamEventMap[K]>) => any, options?: boolean | AddEventListenerOptions): void;  
  addEventListener(type: string, listener: (this: this, ev: CustomEvent) => any, options?: boolean | AddEventListenerOptions): void;  
  addEventListener(type: string, listener: EventListenerOrEventListenerObject, options?: boolean | AddEventListenerOptions): void;  
  /** Deregister a callback function so it will no longer be called when matching events are processed. */  
  removeEventListener<K extends keyof WamEventMap>(type: K, listener: (this: this, ev: CustomEvent<WamEventMap[K]>) => any, options?: boolean | EventListenerOptions): void;  
  removeEventListener(type: string, listener: (this: this, ev: CustomEvent) => any, options?: boolean | AddEventListenerOptions): void;  
  removeEventListener(type: string, listener: EventListenerOrEventListenerObject, options?: boolean | EventListenerOptions): void;  
  /** Schedule a WamEvent. Listeners will be triggered when the event is processed. */  
  scheduleEvents(...event: WamEvent[]): void;  
  /** Clear all pending WamEvents. */  
  clearEvents(): void;  
  /** Connect an event output stream to another WAM. If no output index is given, assume output 0. */  
  connectEvents(to: WamNode, output?: number): void;  
  /** Disconnect an event output stream from another WAM. If no arguments are given, all event streams will be disconnected. */  
  disconnectEvents(to?: WamNode, output?: number): void;  
  /** Stop processing and remove the node from the graph. */  
  destroy(): void;  
}
```



# 3 - The WamProcessor API



# WAMs also have an explicit/implicit WamProcessor API - WamProcessor

```
export interface WamProcessor extends AudioWorkletProcessor {  
  readonly moduleId: string;  
  readonly instanceId: string;  
  /** Compensation delay hint in seconds. */  
  getCompensationDelay(): number;  
  /** Schedule a WamEvent. Listeners will be triggered when the event is processed. */  
  scheduleEvents(...event: WamEvent[]): void;  
  /** Schedule events for all the downstream WAMs */  
  emitEvents(...events: WamEvent[]): void;  
  /** Clear all pending WamEvents. */  
  clearEvents(): void;  
  /** Process a block of samples. Note that `parameters` argument is ignored. */  
  process(inputs: Float32Array[][], outputs: Float32Array[][], parameters: Record<string, Float32Array>): boolean;  
  /** Stop processing and remove the node from the WAM event graph. */  
  destroy(): void;  
}
```

# WamEnv and WamGroup: manage plugin chains

Send events downstream to a list of chained plugins..., manage group (states) etc.

