Interoperability/Reusability of high level WebAudio components
Michel Buffa

To cite this version:
Michel Buffa. Interoperability/Reusability of high level WebAudio components. W3C Workshop on Web Games, Jun 2019, Redmond, United States. hal-02366763

HAL Id: hal-02366763
https://inria.hal.science/hal-02366763
Submitted on 16 Nov 2019
Interoperability/Reusability of high level WebAudio components

Games on Web W3C Workshop, Seattle 2019

Michel Buffa
Université Côte d’Azur
France
I3S/CNRS/INRIA labs
buffa@i3s.unice.fr
@micbuffa
Who am I?

● Professor / researcher at Université Côte d’Azur, in the South of France
  ○ member of the WIMMICS research group common to INRIA and I3S lab from CNRS
● National coordinator of the WASABI ANR research project, with WebAudio at its heart,
● W3C Advisory Committee Representative for UCA
● I participate to the WebAudio working group
Some ambitious WebAudio examples...

- AudioGraphs of these apps use high-level WebAudio nodes and/or AudioWorklets + WebAssembly.
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, etc.)

Some effects and instruments
In early 2018, with some researchers and developers we decided to start working on an open plugin standard for WebAudio.
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 (Developer of VirtualCZ, Endless Series, WDL-OL/iPlug, iPlug2)

- 2 - Bringing low level DSP developers to the Web
  a. Stéphane Letz (senior researcher at GRAME, Lyon, co-author of the FAUST DSL/compiler)

- 3 - Attract Web Developers / JavaScript audio app developers
  a. Tatsuya Shinyagaito, aka g200kg, (Audio and WebAudio developer, huge WebAudio contributor, Yokohama, Kanagawa, Japan)
  b. Jerôme Lebrun and Michel Buffa (I3S/INRIA)
An open standard = API/specification ? Or more… ?

Be Web-Aware!

- **Use URIs**: support local or remote plugins, audio and midi routing
- Asynchronous events in the lifecycle
- Plugins can be headless or with a GUI
- **API for the “audio processor part”**, as close as possible to AudioNode
  - i.e use `plugin.connect(gain)` or `gain.disconnect(plugin)`, etc.
- Propose an **API or at least guidelines** on how to package and publish plugins on REST repositories
- Avoid **naming conflicts** (HTML ids, JS names, CSS rules), metadata...
Let’s play with some WAPs (WebAudio Plugins)
Native plugins (C/C++)
written as VST, JUCE etc.

Max DSP, Pure Data
Others...

FAUST

WebAudioModules
(WAMs)

JavaScript

WebAssembly + AudioWorklet

WebAudio Plugins
(WAPs)

Web browser
Once the script has been loaded instance the plugin:

```javascript
buildPlugin(className, baseURL);

// will be executed before the onload above...
document.head.appendChild(script);

// instance the plugin
function buildPlugin(className, baseURL) {
  var plugin = new window[className](ctx, baseURL);
  console.log(plugin);

  plugin.load().then((node) => {
    // loads and initialize the audio processor part
    // Then use the factory to create the HTML custom element that holds the GUI
    // The loadGui method takes care of inserting the link rel-import part,
    // not doing it twice, and associate the node with its GUI.
    if (checkbox.checked) {
      plugin.loadGui().then((elem) => {
        console.log("ADDING PLUGIN");
        // show the GUI of the plugin, the audio part is ready to be used
        document.querySelector("#WAP").appendChild(elem);
        // mediaSource connect(node);
        node.connect(ctx.destination);
        // Add node to the chain of plugins
      });
    }
  });
  document.body.querySelector("#WAP").insertAdjacentHTML('afterbegin', '<h2>' + className + '</h2>');
  try {
    mediaSource.connect(node);
  } catch (error) {
    console.log("this plugin does not use audioworkletnode or compositenode");
  }
  node.connect(ctx.destination);
  bt_buildIt.addEventListener('click', () => {
    mediaSource.disconnect();
    node.disconnect();
    document.querySelector("#WAP").innerHTML = "";
    document.querySelector("#mocha").innerHTML = "";
  });
}
```

Paste here the link to your webaudio plugin

Plugin URL (repository where your main.json file is)
https://wasabi.js.unice.fr/webAudioPluginBank#tardan-6
GUI
```javascript
bt_buildIt;
```

Try different plugins click on links to fill the form field, then press the "build" button:

- Metal Machine amp sim
- Acoustic guitar amp sim
- PingPongDelay3
- ZitaRev
- Blipper
- Flanger

WasabiDistoMachine

![DISTO MACHINE](https://wasabi.js.unice.fr/webAudioPluginBank#tardan-6)
Simple example of plugin loaded by host...
Low level DSP automatized -> WASM
Online testers (individual plugin, repository)
Conclusion: Where are we today?

SDK for JS developers

FAUST scripts and IDE that compile .dsp files to WAPs, embedded GUIs builder, publish to remote servers

WebAudioModules C/C++ toolchain for native audio developers (and WAMs are WAPs)

Multiple examples of hosts that load plugins dynamically using URIs

Tools: plugin validator, repository validator, GUI editor

Check the GitHub!

Check the pedalboard demo!
Guitar Hero / Rocksmith for the Web?

Yes, this is possible :-)

- Good low cpu amp simulation + effects
- Pitch detection (i.e. tuner)

Still the latency problem need to be addressed on Windows/android (Guitar to Speaker)

\[ \text{Mac OSX : 18-23ms with an audio buffer of 2x128 samples (Chrome, Edge, Opera), FF 32ms, Safari is 70ms.} \]

\[ \text{Windows : Chrome/Edge 81 ms, FF 109 ms.} \]

AudioContext OutputLatency\ is not implemented by any browser yet, baseLatency only by Chrome.

Figure 7: Example of measure in Audacity, here the WebAudio amp sim with Google Chrome and a Presonus 44VSL sound card.