Creative Audio Programming for the Web with Tone.js

Anna Xambó

anna.xambo@dmu.ac.uk

Music, Technology and Innovation - Institute for Sonic Creativity (MTI^2)

De Montfort University

Women Who Code, School of Media Arts, Columbia College Chicago

25.09.2020

Links

- Tone.js
- Codepen.io
- Dropbox

Abstract

Inspired by common tools used in gaming and music production, the Web Audio API is a cross-browser solution that allows for the development of sophisticated audio applications in JavaScript that can be freely accessible from anywhere in the world. Several libraries and frameworks are built upon this API, which facilitates prototyping music and audio ideas quickly. In this hands-on workshop, we will explore Tone.js, a web audio framework that facilitates the creation of interactive music for the web. The workshop is designed to accommodate both beginners and experts in programming using a team-based learning approach with collaborative online tools.

> *I hear and I forget.*
>
> *I see and I remember.*

> *I do and I understand.*
>
> Chinese Proverb.

Schedule

- 10' Introductions
  - People
  - Learning outcomes
  - How are we going to operate?

- 30' Sound synthesis
  - HTML - CSS - JavaScript
  - Web Audio overview
  - Tone.js overview
  - Understanding what is possible:
    - Example 1: Tone.js Demos
    - Example 2: Partipatory Mobile Pieces
    - Example 3: Tone Toss by Cameron Lee (Audio Arts and Acoustics, Columbia College Chicago)

  - Understanding the framework: Online examples from Tone.js website explored on CodePen / JSFiddle
  - Same examples on VS Code (local environment).
    - Running the local server
    - Workflow VS Code / Browser Dev Tools / API

- 30' Interactivity
  - Examples
    - Envelope
    - Arpeggiator
    - Transport
    - BPM

  - Team task

- 30' Musical prototype building
  - Team task

- 10' Publishing, Resources, FAQ
- 10' Questionnaire & Closing
  - Questionnaire

- Final words (JAES SI WA, WAC 2021 Conference)

Introductions

- Who are you?
- What do you do? (e.g. student, faculty, independent...).
- Three topics of your interest.

Learning Outcomes

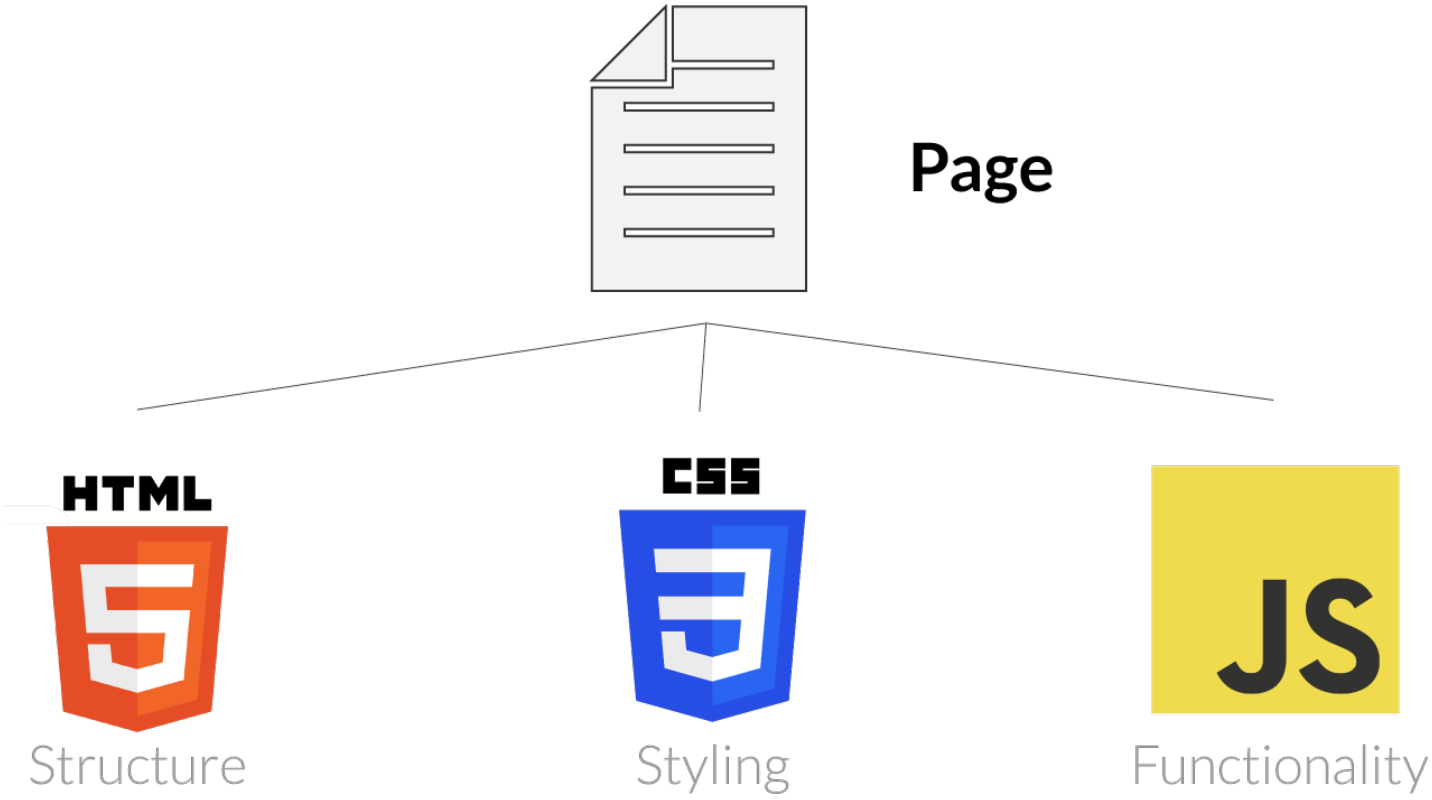
- Get familiar with the framework Tone.js.
- Develop programming skills using collaborative techniques.
- Be able to take ownership of code related to interactive music.
- Get an understanding about how to develop online musical prototypes.
- Get interest in the web audio and computer music communities.

How Are We Going To Operate?

- **A hands-on approach**: everyone should have access to the materials and interact with them.
- **A collaborative approach**: mixed groups for two blocks of the workshop using collaborative technologies (sharing screen in Zoom, creation of teams in Zoom, use of VS Code Live Share.
- **Folder** with the files and documentation: https://www.dropbox.com/sh/0x3uk81eba41kex/AACpe-Mg4PIC6T0s-tRoIFI5a?dl=0

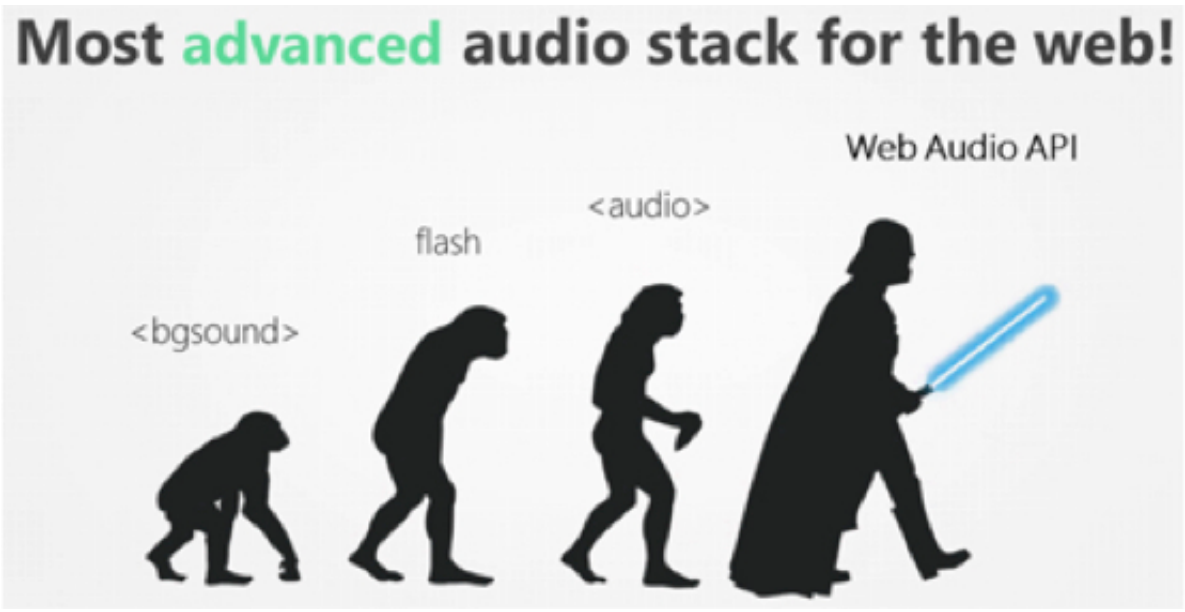
Part 1: Sound synthesis

HTML - CSS - JavaScript

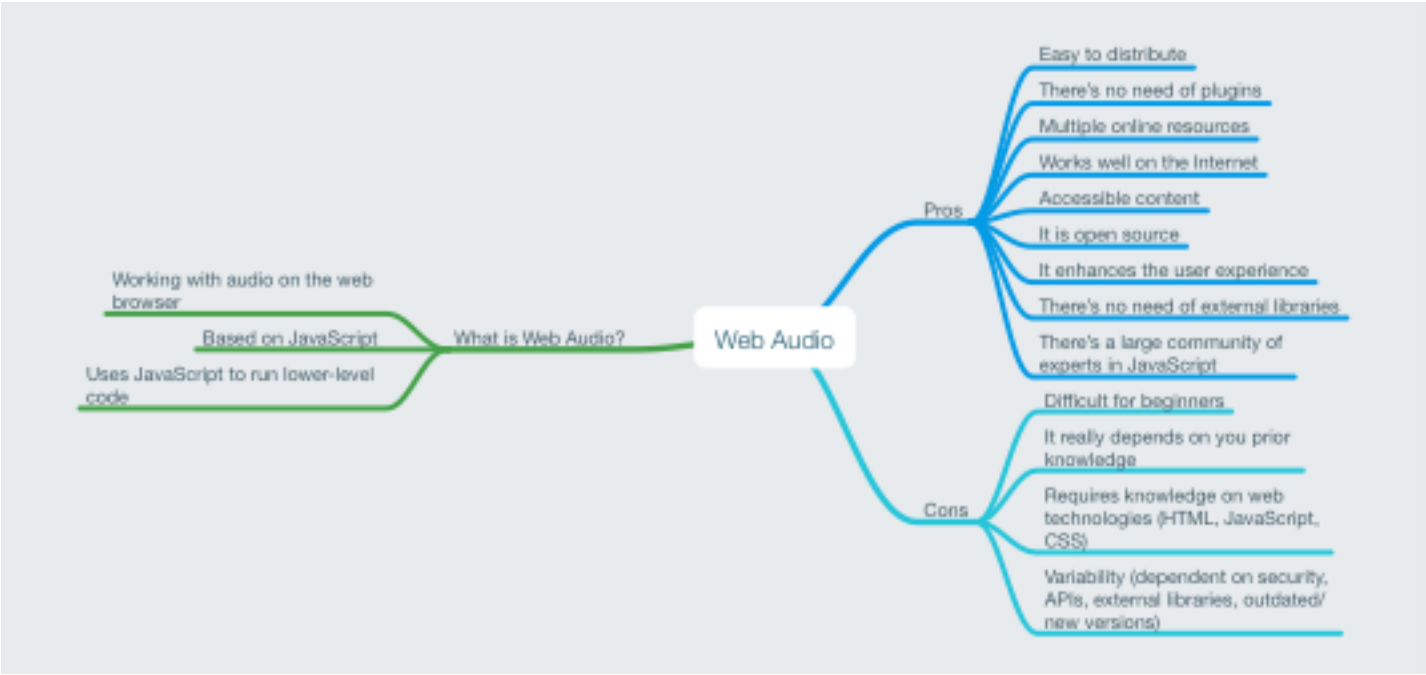


Web Audio overview

- The Web Audio API provides a powerful and versatile system for controlling audio on the Web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects (such as panning) and much more.



Web Audio - Pros and Cons



Source: Xambó et al. (2019)

Tone.js overview

- A web audio framework for creating interactive music in the browser.
- A framework is more constrained than a library because it gives functionality that needs to follow a particular structure.
- The framework is characterised by:
  - **Musicality**: musical scores are possible with JSON files.
  - **Modularity**: elements can be connected e.g. DSP and synthesis building blocks.
  - **Synchronization**: elements or building blocks can share the same timeline / clock.

Source: Mann (2015)

Tone.js overview: Musical time

With `Tone.Time`, AudioContext time can be expressed in tempo-relative terms that are translated into seconds with the `toSeconds` method. Delay times, for example, can be expressed in terms of beats: "4n" would translate to 0.5 seconds at 120 bpm (4/4 time signature).

The notation style is inspired by Max/MSP's metrical timing and Ableton Live's transport time representation.

It is often useful to schedules values relative to the current AudioContext time from Web Audio. Prefixing

any of the above representations with a plus sign as a string ("+") will add the AudioContext's currentTime to the following value.

Source: Mann (2015)

Timing Web Audio Events

- The browser's provides two built-in methods to deal with time: `setTimeout()` and `setInterval()`. They use the same thread as the rest of the DOM.
- **Challenge!** JavaScript is asynchronous. Events cued to be executed as soon as possible, thus time precision is difficult.
- Solution: Using the internal clock of Web Audio. The web audio clock operates on a separate thread than the rest of the DOM: `audioContext.currentTime`
- The currentTime read-only property of the BaseAudioContext interface returns a double representing an ever-increasing hardware timestamp in seconds that can be used for scheduling audio playback, visualizing timelines, etc. It starts at 0. More info here.
- The Web Audio API includes a collection of methods for scheduling changes in audio parameter values at present or in the future:
  - `setValueAtTime(arg1,arg2)`
  - `exponentialRampToValueAtTime(arg1,arg2)`
  - `linearRampToValueAtTime(arg1,arg2)`
  - `setTargetAtTime(arg1,arg2,arg3)`
  - `setValueCurveAtTime(arg1,arg2,arg3)`

Tone.js overview: Transport

A single transport is central to many music production environments since it allows for tightly synchronized and coordinated events.

`Tone.Transport` schedules callback functions: it has a `setInterval` method that accepts a callback and an interval (in `Tone.Time`).

```
    Transport.setInterval(function(time){
        //check application state
        //trigger an event using 'time'
    }, "8n"); // invoked every 8th note
```

`Tone.Transport`'s API also includes a `setTimeout` method for scheduling single events in the future relative to the current clock time, and a `setTimeline` method which schedules methods along a loop-able and seek-able global timeline.

Source: Mann (2015)

Tone.js overview: Buses, routing and master output

Buses

In addition to making it easy to share a single reverb effect across many audio nodes, buses also promote loose-coupling between audio modules.

```
    synth.send("reverb", 0.5);
    //...in a separate audio module
    reverbEffect.receive("reverb");
```

Routing

It can be useful to conditionally route audio to or from another AudioNode; Tone.Route and Tone.Select do just that. `Tone.Route` has any number of inputs and only one output. By setting its gate to an input number, Tone.Route selectively routes that input to the output, stopping all others. `Tone.Select` routes a single input to one of the outputs depending on the value of the Select's gate.

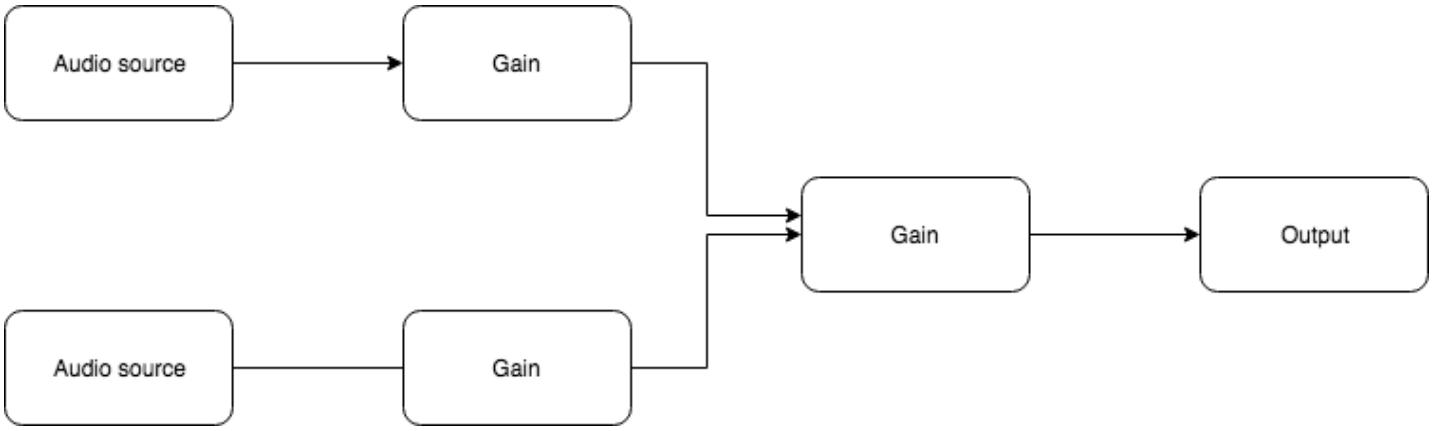
Master output

The master output is an abstraction on the native `AudioDestinationNode`. With this abstraction, it makes adjusting the global volume or muting the entire application easier. Also, by placing a node before the final output, global e↵ects, compressors, and limiters can be applied to the entire mix.

Source: Mann (2015)

Gain node in web audio

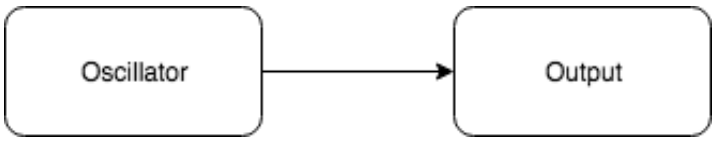
- The gain node can be used to split and combine input sources.
- Gain nodes facilitate independent volume control of the audio sources.
- Gain nodes are used as virtual mixing channels.
- A wrapper around the Native Web Audio GainNode exists: Tone.Gain.



Tone.js overview: Sound sources

- Tone.js includes sound sources such as oscillators of different types (basic oscillator, FMOscillator, AMOscillator, GrainPlayer, LFO, noise, and so on) as well as an audio or multiple file players (Player, Players).

How can a sound source be incorporated?



In Web Audio:

1. Invoke the method to create the node e.g. AudioContext.createGain().
2. Connect the object (sound source, effect) in the signal chain.
3. Modify the properties and methods of the effects node.

```
! Web Audio, ex: oscillator
context = new AudioContext();
oscillator = context.createOscillator();
oscillator.connect(context.destination);
oscillator.frequency.value = 440;
oscillator.type = "sine"; // "triangle", "square", "sawtooth", "triangle"
oscillator.start();
```

In Tone.js:

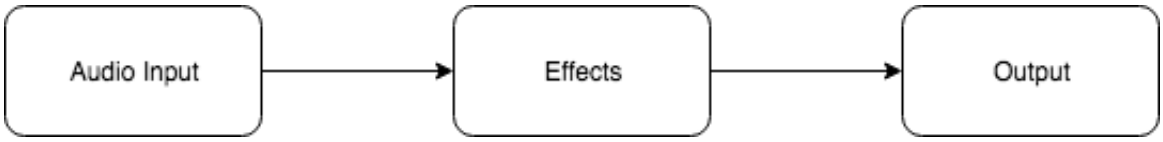
1. Connect the object (sound source) in the signal chain.
2. Modify the properties and methods of the sound source node.

```
! Tone.js, ex: oscillator
// make and start a 440hz sine tone
const osc = new Tone.Oscillator(440, "sine").toDestination().start();
```

Tone.js overview: Instruments and effects

- Tone.js includes instruments such as an FM (Frequency Modulation) synthesizer and a Karplus-Strong plucked string modeling synthesizer.
- Tone.js has a number of audio effects including Tone.PingPongDelay, Tone.Freeverb, and Tone.BitCrusher.

How can the effects be incorporated?



In Web Audio:

1. Invoke the method to create the node e.g. AudioContext.createGain().
2. Connect the object (sound source, effect) in the signal chain.
3. Modify the properties and methods of the effects node.

```
! Web Audio, ex: StereoPannerNode
var source = audioCtx.createMediaElementSource(myAudio);
var panNode = audioCtx.createStereoPanner();
source.connect(panNode);
panNode.connect(audioCtx.destination);
```

In Tone.js:

1. Connect the object (sound source, effect) in the signal chain.
2. Modify the properties and methods of the effects node.

```
! Tone.js, ex: Freeverb
var freeverb = new Tone.Freeverb().toDestination();
freeverb.dampening = 1000;
// routing synth through the reverb
var synth = new Tone.NoiseSynth().connect(freeverb);
synth.triggerAttackRelease(0.05);
```

Presets, States and Scores in JSON

The states of instruments and effects in Tone.js can be set through JSON (JavaScript Object Notation) descriptions in the constructor and set method.

```
    var fastPanner = new Tone.AutoPanner({
        "frequency" : "16n",
        "type" : "square"
    });
```

This format makes it simple to create and share presets.

The JSON descriptions in Tone.js are inspired by the unit generator/score file distinction introduced in MUSIC by Max Mathews which has been a feature in many subsequent computer music languages.

Source: Mann (2015)

Example: Defining the instruments

An example “orchestra” (to use Csound terminology) with Tone.js might be a Tone.PluckSynth connected through a Tone.Chorus to the master output. The score part would look like:

```
    pluckSynth.set({
        "attackNoise" : 0.8,
        "resonance" : 0.6
    });

    chorus.set({
        "rate" : 0.75,
        "delayTime" : 3.5,
        "depth" : 0.7,
    });
```

Source: Mann (2015)

Example: A score

Scores are another JSON-based description used in Tone.js. A score can be parsed by `Tone.Note.parse` which schedules note events along the transport's timeline.

Scores are represented in JSON with the name of the instrument or channel as the object's key and an array of events as the value. Instruments and other components can then listen for these events using `Tone.Note.route`. `Tone.Note.route` is invoked with a score's keys and an event callback function.

```
var score = {
    "drums" : [["0:0","kick"],["0:1","snare"],...
};
Tone.Note.parse(score);
Tone.Note.route("drums",function(time,sample){
    //play drum sample at time
});
```

Source: Mann (2015)

Understanding what is possible

- Example 1: Tone.js Demos
- Example 2: Partipatory Mobile Pieces
- Example 3: Tone Toss by Cameron Lee (Audio Arts and Acoustics, Columbia College Chicago)

Hands-on I: Procedure to Follow the Examples

- Examples from Tone.js website: https://tonejs.github.io
- Online HTML-CSS-JavaScript editor:
  - Codepen.io or JSFiddle.net
- Code in the HTML window:

```
!html
<script src="https://unpkg.com/tone@latest"></script>
<button type="button" id="button1">Start</button>
<button type="button" id="button2">Stop</button>
```

- Code in the JavaScript window:

```
!javascript
// COPY INSTANTIATION CODE HERE
document.querySelector("#button1").addEventListener('click', function() {

      // COPY THE CODE RELATED TO THE PLAY BUTTON BEHAVIOR HERE
      // console.log("play");
  });
document.querySelector("#button2").addEventListener('click', function() {

      // COPY THE CODE RELATED TO THE STOP BUTTON BEHAVIOR HERE
      console.log("stop");
  });
```

Hands-on II: Interactivity

Hands-on III: Musical prototype building

Publishing, Resources, FAQ

Publishing

Responsive Design

- Responsive Design Principles
- Responsive Design Frameworks:
  - W3.CSS responsive stylesheet
  - Bootstrap uses HTML, CSS and jQuery to make responsive web pages.

Touch Events

- Touch events are supported by Chrome and Firefox on desktop, and by Safari on iOS and Chrome and the Android browser on Android, as well as other mobile browsers like the Blackberry browser.
- There are three basic touch events:
  - *touchstart* (a finger is placed on a DOM element),
  - *touchmove* (a finger is dragged along a DOM element) and
  - *touchend* (a finger is removed from a DOM element). — Your applications should support both touch and mouse.

- More info here.

Resources

Note: Also check the references below.

Books

- Smus, Boris. (2013). Web Audio API: Advanced Sound for Games and Interactive Apps. O'Reilly Media, Inc.
  - Online access
  - Buy on Amazon

API

- Web Audio API:
  - https://www.w3.org/2011/audio
  - https://developer.mozilla.org/en-US/docs/Web/API/Web*Audio*API

Groups

- W3C Audio Working Group

Newsletters

- Web Audio Weekly

Videos

- The Future of Web Audio: with Chris Wilson and Chris Lowis
- Introducing the Web Audio API
- Web Audio Conference Channel

Awesome Web Audio

- amilajack/awesome-web-audio: A list of resources and projects to help learn about audio.
- jonahvsweb/awesome-web-audio-resources: A repository of helpful Web Audio related links and other online resources/inspiration.

FAQ

- Frequently Asked Questions

References

- Mann, Y. (2015) Interactive Music with Tone.js. In Proceedings of the Web Audio Conference 2015. Paris, France.
- Smus, B. (2013) Web Audio API: Advanced Sound for Games and Interactive Apps. 1st edition. O'Reilly Media, Inc., Sebastopol, CA.
- Xambó, A., Martín, S. R., Roma, G. (eds.) (2020). JAES Special Issue in Web Audio. Journal of Audio Engineering Society, volume 68, issue 10.
- Xambó, A., Støckert, R., Jensenius, A.R. and Saue, S. (2019) "Facilitating Team-Based Programming Learning with Web Audio". In Proceedings of the Web Audio Conference 2019. Trondheim, Norway. pp. 2–7.