
audiolib.js

Overview

audiolib.js is a powerful toolkit for audio written in JS.

It ships with most of the common tools such as:

- Reverbs
- Comb/IIR/Biquad/All-Pass/Low-Pass/Band-Pass/High-Pass filters
- Delays
- Oscillators
- FFT and other analyzing tools
- Step sequencers
- Envelope controls
- Noise generators
- Samplers

In addition, it hosts these tools in a comprehensive framework, that makes it simple to write a single effect and provides the higher level abstraction on top of that, making the whole system comfortable for both users and plugin authors.

As for the higher level abstraction, audiolib.js features a sophisticated automation API, alongside with pre-processing hooks, sample level access and buffer level management.

audiolib.js is bundled with some tools to make an audio developer's life in a browser much easier, such as sink.js for a consistent API between the experimental browser audio APIs. To complement that, audiolib.js also bundles pcmdata.js that is a WAV encoder/decoder, so that you can turn the recordings you make using Sink.js into WAV files that the user can save. Other tools include the AudioWorker API that allows you to create web workers from strings or functions, bundling audiolib.js and its plugins, all ready to use from the worker.

How to get it

audiolib.js is available both in browser and CLI environments.

Node

To install via npm:

```
1 $ npm install audiolib
```

Please note that you'll need grunt for this to work.

Browser

For browser environments, download the latest version here, or get the source code from GitHub and build it yourself. Don't worry, instructions are included.

Documentation

Documentation is available at <http://docs.audiolibjs.org/> . Tutorials can be found in the wiki

Demos

(if you have your own, please fork & add | msg me)

- JSMad
- Orbisyn
- Audiocogs codecs

Libraries bundled with audiolib.js

- sink.js, for output and buffer handling.
- PCMDData.js, for WAV codecs. (project deprecated and adopted)
- binary.js, for PCMDData.js and general binary data processing. (project deprecated and adopted)
- fft.js, for super fast FT. (project deprecated and adopted)

Related libraries

- XAudioJS is an alternative audio sink with built-in resampling and a Flash fallback. More developer-controlled output environment, that might be more sane for example games.
- dynamicaudio.js is a Flash fallback for Mozilla Audio Data API.
- Audiolet is a graph-based audio routing framework with a lot of nice stuff.
- DSP.js is an extensive DSP toolkit originally designed for the Mozilla Audio Data API.

Plugins

Specifications for plugin developers can be found in <https://github.com/jussi-kalliokoski/audiolib.js/tree/master/spec>

Credits

This project is maintained by Jussi Kalliokoski, with significant contributions from David Govea.

License

Licensed under MIT license.

Example usage

```
1  /* Create an output. */
2
3  var dev = audioLib.Sink(function(sampleBuffer){
4      // Fill the buffer here.
5  }, channelCount, preBufferSize, sampleRate);
6
7  /*
8   Note that all the arguments are optional,
9   so if you want to create a write-only
10  device, you can leave the arguments blank.
11  Also, it is highly discouraged to set any
12  of the arguments if you aren't sure that you
13  need them. Use null if you need to skip
14  arguments.
15  */
16
17  /* Writing buffers: */
18  dev.writeBuffer(buffer);
19
20  /*
21   You can also attach multiple listeners
22   to the same Sink instance.
23  */
24  dev.on('audioprocess', function(...){});
25
26
27  /* Effects */
28
29  var del = audioLib.Delay(sampleRate, delay, feedback);
30
31  var flt = audioLib.IIRFilter(sampleRate, cutoffFreq, resonance);
32
33  var flt = audioLib.LP12Filter(sampleRate, cutoffFreq, resonance);
34
35  var flt = audioLib.Reverb(sampleRate, channelCount, wet, dry, roomSize,
36      damping);
```

```
37 var dist = audioLib.BiquadFilter(sampleRate, b0, b1, b2, a1, a2);
38
39 /* to feed a new input sample */
40 effect.pushSample(sample);
41 /* to get the output */
42 sample = effect.getMix();
43
44 /* Synthesis */
45
46 var osc = audioLib.Oscillator(sampleRate, frequency);
47
48 /* to generate a new sample */
49 osc.generate();
50 /* to get the output */
51 osc.getMix();
52
53 /* Sampler */
54
55 var sampler = audioLib.Sampler(sampleRate, sampleBuffer, defaultPitch);
56
57 /* Envelopes */
58
59 var adsr = audioLib.ADSREnvelope(sampleRate, attack, decay, sustain,
    release);
60
61 /* to trigger the gate */
62 adsr.triggerGate(isOpen);
63 /* to update the value ** Do this on every sample fetch for this to
    work properly. */
64 adsr.generate();
65 /* Get the value */
66 adsr.value; // 0.0 - 1.0, unless you put something more as sustain
67
68 var stepSeq = new audioLib.StepSequencer(sampleRate, stepLength,
    stepArray, attack);
69
70 /* To start the sequence over */
71 stepSeq.triggerGate();
72 /* to update the value ** Do this on every sample fetch for this to
    work properly. */
73 stepSeq.generate();
74 /* Get the value */
75 stepSeq.value; // 0.0 - 1.0
76
77 /* Recording */
78
79 var rec = dev.record();
80
81 /* To stop */
82 rec.stop();
83 // To export wav
```

```
84 var audioElement = new Audio(  
85     'data:audio/wav;base64,' +  
86     btoa( rec.toWav() ) // presuming btoa is supported  
87 );  
88  
89 /* Resampling buffers */  
90 audioLib.Sampler.resample(buffer, fromSampleRate,  
91     fromFrequency, toSampleRate, toFrequency);  
92  
93 /*  
94  If you are used to buffer based approach (for example DSP.js)  
95  and don't need to do any raw manipulation, all the effects  
96  can be used as buffer based too.  
97  */  
98  
99 var bufFx = audioLib.Delay/* or any effect */.createBufferBased(  
100     channelCount, /* the parameters needed by the specific effect */);  
101  
102 bufFx.append(buffer);
```

Audio Workers

You can also use audiolib.js inside Audio Workers (Firefox 6.0+ only), but this is a whole another story. There are many approaches to that, you can include audiolib.js via an external javascript worker file, but audiolib.js offers an alternative approach to this: inline workers. Inline audio workers include the source code already downloaded, and thus creates a new worker that already contains audiolib.js. Inline Audio Workers also allow you to inject code into workers. Here is some code to get started, also see tests/audioworker.html.

```
1  
2 var worker = audioLib.AudioWorker(function(){  
3     device = audioLib.Sink(function(buffer, channelCount){  
4         /* Do some audio processing, like you weren't in a worker. */  
5     });  
6 }, true /* enables injections */);  
7  
8 /* Injection */  
9  
10 worker.inject(function(){  
11     /* Execute some code inside the worker. */  
12 });  
13  
14 /* Close the worker */  
15  
16 worker.terminate();
```

It's important to remember that even though that code looks like it's running in the same environment

as the code it's written in, it's actually not and runs in the context of the worker, meaning you can't cross-reference variables. Also, the injections are sandboxed, so if you need to create a global variable, drop var.