# Introducing AudioMass ([**https://audiomass.co**](https://audiomass.co)) an open-source web based audio and waveform editing tool.

AudioMass lets you record, or use your existing audio tracks, and modify them by trimming, cutting, pasting or applying a plethora of effects, from compression and paragraphic equalizers to reverb, delay and distortion fx. AudioMass also supports more than 20 hotkeys combinations and a dynamic responsive interface to ensure ease of use and that your producivity remains high. it is written solely in plain old-school javascript, weights approximately 65kb and has no backend or framework dependencies.

It also has very good browser and device support.

##### :: Feature List ::

* Loading Audio, navigating the waveform, zoom and pan
* Visualization of frequency levels
* Peak and distortion signaling
* Cutting/Pasting/Trimming parts of the audio
* Inverting and Reversing Audio
* Exporting to mp3
* Modifying volume levels
* Fade In/Out
* Compressor
* Normalization
* Reverb
* Delay
* Distortion
* Pitch Shift
* Keeps track of states so you can undo mistakes
* Offline support!

And all this, in only 83kb of JS!

Getting Started

To get started, drag and drop an audio file, or try the included sample.

Once the file is loaded and you can view the waveform, zoom in, pan around, or select a region.

### Recording Audio

To record audio, simply press the Recording button, or the *[R]* key.

### Exporting to mp3

In order to export back to mp3, click on 'File', then 'Export to mp3', and follow the modal's instructions.

### The story behind AudioMass. And a short rant on web interfaces.

I wrote AudioMass back in June 2018 and it stayed dormant on my hard disk until I decided to share it with the world today (July 13th, 19 -- but you might be seeing this in 2020. Hi!!). It started as a personal small tool for quick visualization of waveforms. Later I added the ability to cut/copy/paste as well as fade in and out. And soon after good 'ol feature creep and perfectionism took over! Soon after, it turned to a challenge to see how close to a full-featured waveform editor it can get, whilst maintaining acceptable performance and small filesize.

In general I am a big fan of the interfaces of DAWs *(Digital Audio Workstations)*, they are extremely, complex, intricate, versatile, and they manage to remain visually pleasant even through their infinite options and knobs. Many times I feel the web has taken a very wrong turn, as amazing interfaces such as...

Sonar



Fruity loops



Existed for more than 10-15 years now, while we are struggling with animating some rectangles for 60fps... So for AudioMass I wanted to try and make a fast and performant interface. Drawing inspiration from the examples I mentioned earlier rather than the tradiional web development practices. This is my unconvincing but truthfull excuse as to why the code is ugly; it is focused on being fast and getting the job done, with little regard on structure.

### Building the interface

Let's say we have a *"PLAY"* button and when we press it the track begins to play. I suppose we would want the button's color and state to reflect that the track is now playing. So naively we would do something like;

btn.onclick = function () {  
 this.classList.add ('active');  
};

But what happens if we have a hotkey that triggers the same action? Let's say we press *[SPACEBAR]* and the track begins to play. Do we modify that button's class in the spacebar's handler?

document.onkeypress = function ( e ) {   
 if ( e.keyCode === 32) {  
 e.preventDefault ();  
  
 document.querySelector ('.playbtn').classList.add ('active');  
 }  
};

And what happens, if there are 2 buttons, or one gets dynamically removed? Do we do selectAll and iterate? Hmmm...

And if the track is playing and we hit *[SPACEBAR]* or the play button again, we need to stop playing. What do we do then? You can see how this becomes messy very quickly as everything gets very tighlty coupled together in a big dependency ball.

Introducing the observer pattern. Actions are represented by events. So the above logic and be expressed as;

btn.onclick = function () {  
 FireEvent ('RequestTogglePlay');  
};  
  
On ('WillPlay', function () {  
 btn.classList.add ('active');  
});  
  
On ('WillStop', function () {  
 btn.classList.remove ('active');  
});  
  
  
  
document.onkeypress = function ( e ) {  
 if ( e.keyCode === 32) {  
 e.preventDefault ();  
  
 FireEvent ('RequestTogglePlay');  
 }  
};  
  
  
  
On ('RequestTogglePlay', function () {  
 if (track.is\_playing) {  
 FireEvent ('WillStop');  
 track.stop ();  
 }  
 else {  
 FireEvent ('WillPlay');  
 track.play ();  
 }  
});  
  
track.onPlayStart = function () {  
 FireEvent ('DidPlay');  
};  
track.onPlayStop = function () {  
 FireEvent ('DidStop');  
};

Now this is completely decoupled and dependency free! The button will set its state according to the events it receives, and both the button and the spacebar key rely on the same mechanisms. You may notice the vocabulary we are using. *"Request"*, *"Will"* and *"Did"*. These are arbitrarilly chosen to impose some extra structure.

*"Request"* denotes intent to perform an action, it is not guaranteed that the action will execute as there might be conditions preventing it (eg unitialized or still loading objects). *"Will"* means that the conditions passed and we are attempting to perform the action. And *"Did"* means that the action just got performed.

It might be a bit too verbose, but it worked very well for structuring AudioMass's interface.

### Dockable UI

One thing I love about DAW interfaces, is that every window can be pulled out of the main host. I fondly remember having 3 screens full of VST plugins. So can we do the same in the browser?

Yes! And it is using some of the oldest tricks in the book. Essentially we create a pop-up window with *window.open* and just pass buffers to its *documentWindow* object. Surprisingly it is quite performant on all browsers except IE Edge. I believe they are serializing in ascii or base64 every packet or something. Also Chrome has an interesting bug where you can't pass more than 512 byte buffers.

So for the undocked window, we call *window.open*. But how do we make it work when it is docked? It would be quite cumbersome to write the same functionality twice, once as a standalone page, and once as an in-page script. Luckily we can avoid that completely, and re-use the same page by using iframes.

The only difference is that the undocked version uses *window.opener* to refference its parent, whereas the iframe uses *window.parent*.

### Future work and performance considerations

The next big feature that is planned, is multitrack support. The ability to mix tracks and different sounds is currently lacking and its usefulness as it stands is limited.

There is also a lot of room for improvement in almost all aspects.

First of all we can further reduce the filesize by around 20kb by removing the library we are using for rendering the waveform. We use only a fraction of its functionality so there is no reason to include it all.

We can also optimize a lot the rendering of the waveform. I heavily modified the library used to compute and draw only the visible range. However it is still clearing and re-drawing all of the canvas at each frame. We can take advantage of 2d Context's translate calls, and shift the canvas around instead of redrawing all of the pixels.

We can also move some operations to a background thread, such as the filters processing so that the UI does not freeze when applying a long chain of effects.

However, the biggest issue I encountered, is the Web Audio API itself. Every operation results in iterating over multiple long arrays per frame. Eventually the garbage collector fires and crackling is introduced. Only way to go around this is to use a small fftSize, but then the frequency range we have to work with is very narrow. Perhaps a pure WASM implementation would outperform trying to modify audio signals with JS. Only one way to find out I guess :)

Additionally, *decodeAudioData* provides no progress callback, and no way of cancelling it. So if you attempt to load a huge audio file, you will waste resources until it gets processed. There is no way around this currently and it can get annoying if you push a big file by mistake.