# ECS512 Sound Design - Lab 2: Working with the Web Audio API

Required resources:

• Code and sound examples contained in the **Lab2-resources.zip** file – available on QM+.

• **Google Chrome** browser.

• **Web Audio Inspector** extension for Chrome – to visualize the audio processing chain in the browser.

• **Atom** – a free on-line text and source code editor.

• **atom-live-server package** – a free add-on for Atom, which will launch a development http server.

• **Sonic Visualiser** – a free analysis and visualization software package developed at QMUL.

We will also use NexusUI, which is a collection of HTML5 interfaces and Javascript helper functions to assist with building web audio instruments in the browser. However, you will get this as part of the assignment.

This lab builds on the first lab, Introduction to the Web Audio API, by introducing more advanced concepts, and building more functional applications. Please note that this lab is not being graded .

The lab is divided into 2 parts;

1. Creating a Beep, with adjustable timing.
2. Changing the interface with the UI package NexusUI.

The last two parts have some questions as parts of the steps that you should follow. Doing the exercises and typing in the code yourself is a lot better way for you to learn than just reading the assignment and running the code examples.

Topics covered in this and other labs are explained in more detail at <https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API> , and links therein. I encourage you to check it out since it is a thorough guide, will reinforce everything being demonstrated here today and will give you a more grounded understanding for future assignments.

## Part 1. Beep

1. Download lab2-resources.zip from QMPlus, unzip it and save it where you want to store your code.

2. Make sure you have the Atom text and source code editor, that you’ve installed the atom-live-server add-on, and that you have installed Chrome and set it to be your default browser.

3. In Atom, from the menu go to File -> Open Folder and select the ‘Beep’ folder from within your unzipped ‘lab2-resources’

4. Create a simple html file, with 4 user interface elements; a ‘Beep’ button, a Frequency range (slider) control, an Attack range control, and a Decay range control. The Attack and Decay are in milliseconds, so could range from 1 to 1000.

Your code will look something like this;

<html>

<div align="center">

<input type="button" value="Beep" id="triggerBeep">

<p>Frequency</p>

<input type="range" min="0" max="8000" value="440" id="Frequency">

<span id="Frequency label"></span>

<p>Attack</p>

<input type="range" min="1" max="1000" value="50" step="any" id="Attack">

<span id="Attack label"></span>

<p>Decay</p>

<input type="range" min="1" max="1000" value="50" step="any" id="Decay">

<span id="Decay label"></span>

</div>

<script src="Beep.js"></script>

</html>

**beep.html**

5. Now create a separate javascript file with callbacks showing the values of the interface elements.

triggerBeep.onclick = function() { }

Frequency.oninput = function() {

document.getElementById("Frequency label").innerHTML = this.value

}

Attack.oninput = function() {

document.getElementById("Attack label").innerHTML = this.value

}

Decay.oninput = function() {

document.getElementById("Decay label").innerHTML = this.value

}

**beep.js**

and include this javascript file near the end of the html file using <script src="Beep.js"></script>. We’ve kept the Javascript file as small as possible. It uses the element values directly, rather than defining new variables based on the elements.

6. Add a square wave using an Oscillator Node, but connect it to a Gain Node with gain set to 0. We do this rather than not starting it or disconnecting it, since in the next step we will *slowly ramp up* the gain, rather than just connect or start it.

So this part of the Javascript file will look like;

var context = new AudioContext();//Define audio context

var gain = new GainNode(this.context,{gain:0});

var osc = new OscillatorNode(this.context,{type:'square'});

osc.connect(gain);

gain.connect(context.destination);

osc.start();

Note that we could have defined the gain and osc nodes on one line and set parameters on the next, but this is more compact.

7. Now we introduce parameter scheduling. When the Beep button is clicked, we want to ramp up the gain from 0 to 1, and back down again.

This is the callback function for when the Beep button.

triggerBeep.onclick = function() {

if (context.state === 'suspended') context.resume()

let now = context.currentTime;

osc.frequency.value = Frequency.value;

Volume.gain.setValueAtTime(0.0, now);

Volume.gain.linearRampToValueAtTime(1, now + Attack.value/1000);

Volume.gain.linearRampToValueAtTime(0, now + Attack.value/1000 + Decay.value/1000);

}

A few things to point out here.

* Chrome often defaults the audio context to ‘suspended’, so that it won’t produce sound. There are a variety of reasons for this, but mainly its because users tend not to want a blast of sound when they go to a web page, unless they request it. So the context needs to be resumed after a user gesture on the page, like clicking the mouse on a button. That’s what if (context.state === 'suspended') context.resume does. The *if* part of the statement is not essential, but it prevents unnecessarily resuming the context for later button clicks.
* linearRampToValueAtTime takes the time in seconds as one of the inputs. The Attack and Decay were given in milliseconds on the interface, so we need to divide them by 1,000.
* We don’t need to specify a start time for linearRampToValueAtTime just the end time. This is because the change starts at the time specified for the previous event.

Questions:

Volume.gain.setValueAtTime(0.0, now); first sets the gain parameter value to 0. So every time the button is clicked, it ramps up from 0 to 1, and then back down to 0.

1. What happens when the button is clicked again before the ramping is finished?
2. Now remove this line where the value is set to 0. How does this sound different from before, when the button is clicked again before ramping is finished?
3. Replace both calls to linearRampToValueAtTime with exponentialRampToValueAtTime . How does this change the sound? Which version sounds more natural?

## Part 2. Beep with Nexus UI

The labs are focused on using the Web Audio API. But its useful to know how to work with other software packages to add extra functionality on top of that. NexusUI is a collection of user interface elements and other tools for use with Web audio. Its not explicitly for WAA, and can be used without it, but presenting good interfaces to go with an audio context is clearly where it is most useful.

1. Read through the documentation at <https://nexus-js.github.io/ui/> . Then go to <https://github.com/nexus-js/ui/> and download the sourcecode using the ‘clone or download’ button.

2. From the downloaded source code, go to the ‘dist’ folder, and put the file NexusUI.js in a new folder called ‘Beep with Nexus’ where you want your code for this part of the lab.

3. Now. Open ‘Beep with Nexus’ in Atom, and create a new file there called Beep.html .

4. In Beep.html, load the NexusUI and add html sections for triggering the Beep and setting the parameters Frequency, Attack and Decay. Your html file should look something like this;

<html>

<script src="NexusUI.js"></script>

<div align="center">

<div id="triggerBeep"></div>

<p>Frequency</p>

<div id="frequency"></div>

<p>Attack</p>

<div id="attack"></div>

<p>Decay</p>

<div id="decay"></div>

</div>

</html>

5. Now add Nexus slider controls by adding the following code after the last div section ;

<script>

let triggerBeep = new Nexus.TextButton('triggerBeep', {'mode': 'button','size': [120,55],'state': false,'text': 'Beep'});

let frequency = new Nexus.Slider('frequency',{'size':[200,20],'mode':'relative','min':80,'max':8000,'value':800});

let attack = new Nexus.Slider('attack',{'size':[200,20],'mode':'relative','min':1,'max':1000,'value':50});

let decay = new Nexus.Slider('decay',{'size':[200,20],'mode':'relative','min':1,'max':1000,'value':50});

</script>

5. So now we have sliders. They look slightly different from the ones without using Nexus, but otherwise this is the same. Lets add number boxes to display the slider values, and link them so that the value can be changed either by typing in the box or moving the slider. So this section of code becomes;

<script>

let triggerBeep = new Nexus.TextButton('triggerBeep', {'mode': 'button','size': [120,55],'state': false,'text': 'Beep'});

let frequency = new Nexus.Slider('frequency',{'size':[200,20],'mode':'relative','min':80,'max':8000,'value':800});

let frequencyDisplay = new Nexus.Number('frequencyDisp');

frequencyDisplay.link(frequency);

let attack = new Nexus.Slider('attack',{'size':[200,20],'mode':'relative','min':1,'max':1000,'value':50});

let attackDisplay = new Nexus.Number('attackDisp');

attackDisplay.link(attack);

let decay = new Nexus.Slider('decay',{'size':[200,20],'mode':'relative','min':1,'max':1000,'value':50});

let decayDisplay = new Nexus.Number('decayDisp');

decayDisplay.link(decay);

</script>

7. And finally, have the Frequency slider callback update the oscillator’s frequency parameter, in a javascript file called Beep.js .

context = new AudioContext();

triggerBeep.on('change', function(v) {

if (v) {

context.resume()

let now = context.currentTime;

masterGain.gain.setValueAtTime(0.0, now);

masterGain.gain.linearRampToValueAtTime(1, now + attack.value/1000);

masterGain.gain.linearRampToValueAtTime(0, now + attack.value/1000 + decay.value/1000);

}

});

frequency.on('change', function(v) {osc.frequency.value=frequency.value});

var masterGain = new GainNode(context,{gain:0});

var osc = new OscillatorNode(context,{type:'square',frequency:800});

osc.connect(masterGain);

masterGain.connect(context.destination);

osc.start();

This is almost exactly the same as the JavaScript file for Beep without the NexusUI in the previous part.

8. Add <script src="Beep.js"></script> to the end of the html file so that this JavaScript is embedded in the html.

Questions/problems:

1. Check that this produces exactly the same output as the Beep without Nexus, and that the slider values can be changed using the number boxes.
2. Replace the sliders from NexusUI with dial controls, and check that it still works.
3. (optional) Replace the Attack and Decay sliders with a multislider from NexusUI.
4. (optional) Create shortcut keys to trigger the Beep and move the slider up and down. See <https://medium.com/@melwinalm/crcreating-keyboard-shortcuts-in-javascripteating-keyboard-shortcuts-in-javascript-763ca19beb9e> for examples of how to do this.