Introduction to soundgen

This introduction to using the **soundgen** server discusses examples that are used to create psychoacoustic experiments. These experiments can be debugged and carried out in the 6.182 Laboratory (36-744) or anywhere you have access to earphones and a computer that is capable of producing sound. The experiment control programs that are illustrated here are written in JavaScript/HTML. Although many books describe these two languages, two that are useful are: *Learning JavaScript* (S. Powers) and *HTML & XHTML: The Definitive Guide* (C. Musciano and B. Kennedy). Also useful, less as texts than as references to be accessed while you work online, are the worldwide web sites: http://www.w3schools.com/js/ and http://www.w3schools.com/html/ You will run your experiments using the Firefox web browser. You may also benefit from using *mootools* (http://mootools.net/).

Your programs will be written in two parts. Part I (the **head**) is the control code, written in JavaScript. It includes the definition of global variables and functions used in the program (that contain local variables). Part II (the **body**) is the user interface, written in HTML, that calls the functions, causes sounds to be played, and responses to be gathered.

Example I: Creating and playing a tone burst plus a noise burst.

The first example causes a tone burst in a noise burst to be created and played.

The tone signal and noise signal are specified as a JavaScript array (**soundArray**) created by the internal functions (**Noise** and **Tone**) and passed as an argument to the **soundgen** procedure, which executes a Remote Procedure Call to the server that generates the samples of the sound. After the Remote Procedure Call executes, **soundgen_callback** is automatically called. The RPC defines two variables: **obj.sound,** which specifies the URL containing the sound, and **obj.image**, which specifies the URL containing the image of the sound. The image of the sound id is **jpg** format and can be displayed by simply going to the URL. They can also be used in JavaScript code, as discussed below.

In this example, the noise burst starts at t=0 (**noise_ip**), with rise time (**noise_rt**) of 10 ms, on time (**noise_ot**) of 470 ms, and fall time (**noise_ft**) of 5 ms, for a total duration of 485 ms. The tone burst starts at t = 20 (**tone_ip**), with rise time of 25 ms (**tone_rt**), on time of 415 ms (**tone_ot**), and fall time of 25 ms (**tone_ft**), for a total duration of 490 ms. The noise is a sample of Gaussian noise, filtered by a high-pass Butterworth filter of order 4 and a low-pass Butterworth filter of order 3 to have a power spectrum that is flat over the range 100-4000 Hz (**noise_lp – noise_hp**) . Because **noise_seed = -1**, the noise is generated fresh each time the program is run. The tone is a burst of 845 Hz (**tone_fq**) sinusoid that begins with zero phase (**tone_ph**). Because **noise_ch = 'b'**, the nose is played from both the left and right channels of the sound card, while the tone is played only from the left (to**ne_ch = 'l').** Note that the tone occurs entirely within the noise.

The calibration level (**calibration_level**) is the difference between 0 dB SPL and the maximum sound output. It will vary by machine/headphones etc. and should be determined either from specification sheets or by measurement. A value of -100 is only nominal. The tone level (**tone_db)** and noise level (**noise_db**) can then be then specified in dB SPL. The sum of **calibration_level** and **tone_db** (or **noise_db**) must never exceed 0.

In this example, the body consists of the **span** tag and the specification of the **onclick** event that specifies the JavaScript function (**begin**) that is executed when the event occurs. In HTML, the **** tag is used to group or specify elements in a document. In this case, an empty **** tag is used as a placeholder to store and play the sound file that will be generated by **soundgen**. The **soundgen_callback(obj)** is called upon completion of the **soundgen** RPC. In this example, JavaScript is used to place the URL of the sound (contained in **obj.sound**) inside of the element with id **"soundspan",** which is inside the **body** of the program.

This example illustrates a simple type of interaction between the **head** and the **body** of the program. When the page loads, the text "Click here for sound!" is displayed. When the user clicks on **here** in the display the JavaScript function **begin()** is called with no arguments. Calling **begin** causes the variables to be defined and **soundgen** to be called.

```
<Example-1.html>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<head>
<script type="text/javascript"
   src="http://tuliptree.mit.edu/soundgen/soundgen2.js">
</script>

<script type="text/javascript">

  function begin() {

    var noise_db = 60;
    var noise_ip = 0;
    var noise_rt = 10;
    var noise_ot = 470;
    var noise_ft = 20;
    var noise_hp = 100;
    var noise_lp = 5000;
    var noise_hp_ord = 4;
    var noise_lp_ord = 3;
    var noise_seed = -1;
    var noise_ch = 'b';

    var tone_db = 85;
    var tone_ip = 20;
    var tone_rt = 20;
    var tone_ot = 415;
    var tone_ft = 25;
```

```
      var tone_fq = 845;
      var tone_pha = 0;
      var tone_ch = 'l';


      var soundArray = new Array();
      var calibration_level = -105;
      var sample_rate = 11025;
   var xtone_db = tone_db + calibration_level;
   var xnoise_db = noise_db + calibration_level;


   soundArray[0] =
      new Noise noise_ip, noise_rt, noise_ot, noise_ft,
        xnoise_db, noise_lp, noise_hp,
        noise_lp_ord, noise_hp_ord, noise_seed, noise_ch);
   soundArray[1] =
      new Tone ( tone_ip, tone_rt, tone_ot, tone_ft,
        xtone_db, tone_fq, tone_ph, tone_ch);
   soundgen(soundArray, sample_rate);
   }

   function soundgen_callback(obj) {
      document.getElementById("soundspan").innerHTML=
        "<embed src='"+obj.sound+
        "'hidden=true autostart=true loop=false>";
     }

</script>
</head>

<body>
<span id="soundspan"></span>
   Click <a href="#" onclick="javascript:begin()">here</a> for sound!
</body>
```

Legal Argument Values

The duration arguments (**`tone_ip, tone_rt, tone_ot, tone_ft, noise_ip, noise_rt, noise_ot, noise_ft`**) must floating point numbers in the range 0-60000 (ms) inclusive (0-60 sec). The level arguments (**`tone_db, noise db`**) must be a floating point numbers in the range –300-0 (dB). The channel arguments (**`tone_ch and noise_ch`**) must be one of the strings **`'l'`** for left, **`'r'`** for right, or **`'b'`** for both.

The frequency of the tone (tone_fq) must be a floating point number in the range 10-20000 (Hz). The phase of the tone must be a floating point number in the range –pi - +pi (–3.14159… - 3.14159…), inclusive.

The noise filer cutoff values (**noise_lp, noise_hp**) must be floating point numbers in the range 1-20000 (Hz) or –1 for no filtering. The filter orders (**noise_lp_ord, noise_hp_ord**) must be in the range 1-100 inclusive. The random noise seed must be either an integer between 0 and 2^32 - 1 inclusive for "frozen" noise or the integer -1 for

"random" noise, i.e. a randomly selected seed.;

Example II: Multiple Sounds

This examples and subsequent examples may be conveniently found on the web at http://web.mit.edu/6.162/www.

It is possible for the **soundgen** procedure to produce more than one tone and/or noise. If the time parameters are identical, the sounds will be added and played simultaneously. If the offsets are adjusted properly, the sounds will be played in disjoint time intervals.

In this example, two sounds are heard. During the first 500 ms interval, beats are heard because two tones of slightly different frequency (300 Hz and a tone of increasing frequency) are played simultaneously. During the second interval (500 ms, beginning 400 after the first interval) a single higher frequency tone is played at a lower level. It is possible to play multiple sounds (either simultaneously or in disjoint time intervals) because **soundArray** is an array.

The code for this example reports the frequency of both tones played during the first interval by means of an **Alert Box.** The code which summons the Alert Box is:

```
alert("tone_fq0 = " + tone_fq0 +
 " Hz\ntone_fq1 = " + tone_fq1 + " Hz")
```

In this **alert** the **+** causes concatenation of strings rather than addition and the **\n** causes the second tone frequency to appear on a separate line.

Example III: Adding buttons that affect the tone level.

Buttons can be created to increase and decrease the amplitude of the tone. In this example, three buttons are used. One calls the **begin** function, one the function **increase_level**, and one **decrease_level**. The latter two functions increment or decrement **tone_db** by 5 (within limits) causing the level of the tone to increase or decrease by 5 dB. The value of **tone_db** is displayed at the end of the **Tone dB SPL:** text. When **increase_level** or **decrease_level** are clicked, the function **update_display** is called (with no arguments). Function **update_display** changes the value displayed after the **Tone dB SPL:** text by executing the statement

```
document.getElementById('show_tone_db').innerHTML = tone_db;
```

thereby effectively modifying the original content of the statement to the new value of **tone_db**. If the value of **tone_db** is 60, and the function **increase_level** is called (so that the value of **tone_db** is now 65) executing the statement

```
document.getElementById('show_tone_db').innerHTML = tone_db;
```

effectively causes the statement

```
Tone dB SPL: <span id="show_tone_db">60</span>
```

To execute as if it read

```
Tone dB SPL: <span id="show_tone_db">65</span><
```

Example IV:

A more elaborate example allows the user to specify both tone and noise levels (using text boxes) and keeps track of how many times the user responds that he hears the tone.

The HTML code in the **body** section starting with the **id** whose value is **setup** defines text boxes for the variables **sel-calibration, sel_tone_db, sel_tone_fq,** and **sel_noise_db** with initial values of –100 (dB), 53 (dB SPL), 800 (Hz), and 60 (dB SPL) respectively. On clicking the **begin** button, the function **begin** is called. This function, among other things, reads the values of the variables **calibration, tone_db, tone_fq,** and **noise_db**. For example, the level of the tone is read by as the value of

```
parseFloat(document.getElementById('sel_tone_db').value))
```

where the **parseFloat** function converts the text read by

```
document.getElementById('sel_tone_db').value)
```

to a number.

The statements which set certain HTML text strings to **none**, e.g.

```
document.getElementById('instructions').style.display='none'
```

have the effect of turning off the display of the instructions and the text boxes (and the **begin** button) because a string with a **display** value equal to **none** is not displayed.

By contrast, the statement

```
document.getElementById('respond').style.display="";
```

in function **getresponse** causes the string whose **id** is **respond** and which is initialized to **display:none** to be displayed.

This example also illustrates another way of communicating the results of an experiment. The variable **res** is initialized by the statement

```
res = document.getElementById('results');
```

Now, **`res.innerHTML`** refers to the HTML content of element **`res`**. Thus setting it equal to the HTML string

```
"<h2>Results</h2>" + . . . + ' dB SPL.<br>'
```

causes the text to be displayed as part of **`res`**. Finally the display of **`res`** is "turned on" by the statement

```
res.style.display="";
```

Debugging

Most likely when programs are written, bugs are created. These may be difficult to understand and eliminate in JavaScript/HTML programs because of the interaction of the two languages. Several simple tools exist for debugging JavaScript code.

Comments

An effective technique is to use commenting to prevent the execution of sections of code. For example, if you suspect the line

```
tonedur = tone_rt + tone_ot + tone_ft;
```

is causing a bug, you can prevent its execution by preceding the line with the comment designator, **//:**

```
// tonedur = tone_rt + tone_ot + tone_ft;
```

Similarly, if you suspect the code block

```
if(noisedur >= tonedur) {
  tone0_ip = (noisedur - tonedur)/2;
  noise0_ip = 0;
  noise1_ip = noisedur + gap_dur
}
```

you can prevent the entire block from executing by enclosing it in **/* … */**

```
/*
            if(noisedur >= tonedur) {
  tone0_ip = (noisedur - tonedur)/2;
  noise0_ip = 0;
  noise1_ip = noisedur + gap_dur
   }
*/
```

Segments of the **body** may likewise be commented by enclosing the segment in the following strings "**<!--**" and "**-->**" as in

```
<!–
   <div style="width:850px; clear:both; padding-top:15px;
   margin:auto;">
      <div id="seldb">
         <input type="text" class="text" id="sel_calibration" size="6"
         value="-105">
         db SPL - Calibration
      </div>
   </div>
-->
```

Alerts

Alerts allow you to examine the value of variables while the JavaScript code is executing. For example, the URL associated with the **`obj.image`** variable can be displayed in an alert box by executing the statement

```
alert("Image URL = " + obj.image);
```

Detecting exceptions: **`try/catch/finally`**

The **`try`** statement delimits a block of code that's enclosed in an exception-handling mechanism. If an exception is detected, the code in the subsequent **`catch`** block is executed. The optional subsequent **`finally`** block contains code that is executed whether or not a exception is detected.

These six exceptions can be caught:

1) **`EvalError`** – **`eval`** is used incorrectly
2) **`RangeError`** – numeric value of of range
3) **`ReferenceError`** – Invalid reference
4) **`SyntaxError`** – Invalid syntax
5) **`TypeError`** – variable not typed as expected
6) **`URIError`** – **`encodeURI()`** or **`decodeURI`** are used incorrectly

An example of the use of this mechanism is

```
try{
          var ansArray = null;
          alert(ansArray[18]);
   }
catch(err){
  if (err.instanceof TypeError) {
  alert(TypeError: + err.message);
   }
finally{
           var ansArray = null;
   }
```

The existence of code in the **`finally`** block serves to remind us that the code in the **`try`** block is not actually executed, it is merely tested.

Firebug

Although it is possible to gain some information about the cause of bugs from the Firefox **`Error Console`** (Tools Menu), more useful information can be gathered by running Firebug (Tools Menu).

`Soundgen` Errors

`Soundgen` detects and reports many erroneous conditions, such as illegal argument values. The following code fragment occurs in the **`soundgen_callback(obj)`** function:

```
if (obj.result != 'success') {
   document.getElementById('result_obj').innerHTML = obj.result;
}
```

Together with the following statement in the body of the .html file:

```
<div id="result_obj"></div>
```

the following message is displayed if an attempt is made to play a 111 dB tone when the calibration is set to –105 dB.

> **There was an error processing your request. The following problems were found:**
>
> **The dB of tone at index 1 is invalid. It must be between -300 and 0 inclusive.**

Calibration

The Sennheiser HD-580 headplones produce 98 dB SPL at a flat-plate coupler when driven by a 1V rms tone at 1kHz.

Assuming that the gain of the Tucker-Davis PA8 headphone drivers is set to +6 dB, the maximum rms level of the tone produced by the 24 bit Lynx-One Digital to Analog converters (achieved with a xtone_db=0) is 2.2 volts at the HD-580 headphones. The peak-level of this tone is 1.414 times the ms level, or 3.1 volts. Thus the maximum rms level of a tone produced by the headphones is

$$98 + 20 \log (2.2/1.0) = 105 \text{ dB SPL.}$$

You should set the calibration to -105. Then the tone levels you specify (as tone_db) will then be at the specified level (in dB SPL). In particular, if you specify a tone of 105 dB SPL, xtone = 105-105 = 0, causing a 2.2 volt (rms) tone to be generated by the Digital to Analog converters, producing a 1000 Hz tone of 105 dB SPL.

A noise with a bandwidth of 100-5000 Hz produces a peak voltage 3.1 volts when is rms value is approximately 1.0 volts rms or 98 dB SPL at the HD-580 headphones when noise_db = 90 or xnoise_db = 0.

The delivered noise level will then be 7 dB less than that you specify. To achieve a 75 db SPL noise, you should specify a noise level of 82 dB. When 82 is added to –105, the result is –23, or 98-23 = 75 dB SPL. To achieve a 45 db SPL noise, you should specify a noise level of 52 dB. When this is added to –105, the result is –53, or 98-53 = 45 dB SPL.

The spectrum level of the noise produced by the digital to analog converters (achieved with a xtone_db=0) is roughly constant at –37 dB volts/Hz. Thus when the noise cutoffs are 100-15000 Hz, the rms level of the noise produced is roughly 1.5 volts, or 102 dB SPL; 100-10000 Hz produces 1.33 volts or 100 dB SPL; 100-5000 produces 1.0 volts or 98 dB SPL; 100-2000 Hz produces 0.64 volts or 94 dB SPL, and 100-1000 produces 0.45 volts or 91 dB SPL; and 100-500 Hz produces 0.29 volts or 87 dB SPL.

SAFETY

If you were a subject in one of our research experiments we would not ordinarily expose you to tones in excess of 90 dB SPL nor broadband noises in excess of 95 db SPL. The maximum exposure time would also be limited to 2 hours per day. These are believed to be very conservative limits on sound exposure.

While the exposures planned for Laboratory 1 are clearly within these limits, the capabilities of the equipment are not: 105 dB SPL tones and 102 dB broadband noises can be produced by the equipment. The best protection from exposure to excessively intense sounds is to remove your headphones if you ever experience uncomfortable sounds.