Hi-precision audio in listening tests - also in the browser?

Benjamin Pedersen
DELTA, SenseLab
Venlighedsvej 4, 2970 Hørsholm, Denmark
bep@delta.dk

ABSTRACT

In listening tests, detailed sound control is sometimes mandatory down to each individual digital sample value and guarantee is needed that they are not unintentionally altered. At other times, a lesser degree of control is acceptable, if on the other hand test execution becomes less restricted. Detailed control of sound is often possible only under "laboratory" conditions where hardware and software are under complete control and sound pressure levels can be accurately calibrated. On the other hand, if test persons can do listening tests at home, via an internet browser for example, collecting large amounts of data becomes faster and cheaper (no laboratory facilities required, and more persons can do tests in parallel). Online listening tests made possible by the Web Audio API offers great flexibility in test execution, but compromises in precise stimulus control must be accepted. This paper analyzes such compromises by discussing technological limitations of Web Audio API followed by validation measurements of sound playback in popular internet browsers. The measurements show that at the detailed level there are significant differences in actual performance of different browsers and behavior is not always as expected. Finally, a solution is presented where audio presentation is delegated to an external audio presenter for situations where the limitations of Web Audio API are not acceptable.

1. INTRODUCTION

When evaluating sound against human perception, listening test is often the preferred tool to gain insight into aspects such as sound quality, discriminability and preference. This is for example relevant when designing products for sound reproduction, when comparing competing technologies, to support marketing claims or when investigating aspects of perception in general. Both hardware and software (signal processing algorithms) can be evaluated in listening tests. In such tests, it is important to have detailed control of audio presentation as not to introduce any bias and for the test system to be "transparent" and not influence results. It is often required to comply with specific standards (for listening tests) to support specific claims. Such standards may have detailed requirements for the audio presentation.

When running many listening tests with different requirements, flexibility in test execution is important, including the possibility for assessors (test persons) to be located at their homes or in a controlled laboratory [4]. This makes detailed hardware control possible (in the laboratory), but when large amounts of data are required and detailed control of sound is less important assessors can sit at home using their own personal computers increasing the rate and flexibility of data collection. Further, having people do tests at home can significantly lower cost and sometimes assessors may not be near the laboratory (in for example culture dependent studies).

When doing listening tests at a large scale there are also several important tasks, not directly related to the execution of a test, such as: (1) Maintaining panels of assessors and their history, (2) keeping collected data in a uniform format in a centralized data store with data history and traceability, (3) execution of statistical analyses for immediate feedback and automatic result processing, and (4) simple and fast configuration of new tests via user interface avoiding error prone manual editing of scripts and configuration files.

It is important with the right tools to facilitate the entire test process to make it economically feasible and to meet strict deadlines and avoid costly errors and at the same time be able to collect the amount of data required for hi-quality data analysis in a constrained amount of time (running tests in parallel).

At SenseLab we have developed the system SenseLabOnline to solve the mentioned problems. Since 2009 SenseLabOnline has matured over several iterations using different technologies available at different times. It has been used for collecting and analyzing data for more than 500 listening tests. Several other software tools for listening tests are available, some of which are free, but usually their focus is only on few of the tasks mentioned. A presentation of one such system is for example described by [3] who also provide a review of some other tools.

This paper will in the following focus on detailed audio presentation and not listening tests in general. Specifically, the technological (in software terms) options for audio presentation are reviewed and audio presentation via Web Audio API is evaluated in greater depth.

2. TECHNOLOGICAL OVERVIEW

The options for online audio presentation have changed over time and SenseLabOnline has used Java, Silverlight and

Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution**: owner/author(s).

Web Audio Conference WAC-2017, August 21–23, 2017, London, UK.

© 2017 Copyright held by the owner/author(s).

currently Web Audio API. Java and Silverlight are not based on web standards and only available in browsers via third party plug-ins. For security reasons, most modern browsers no longer allow installation of plug-ins and therefore Java and Silverlight are not options for future (and current) versions of online listening test systems. Avoiding browser plug-ins has the benefit that only a standard compliant browser is required without further installation of plug-ins (and associated version problems). Further, standard compliant browsers are available across devices and platforms to a larger extent than plug-ins, making it possible to execute tests on for example tablets or smartphones.

Before going into details of audio presentation a brief overview of other requirements for SenseLabOnline and chosen solutions are given to understand why a web browser/web server solution is adequate. While SenseLabOnline is the primary tool for our own listening tests we also want to: (1) Grant customers access to an online service where they can carry out own experiments and (2) make it possible for customers to install and use SenseLabOnline on their own hardware allowing them to keep all collected data in-house, which might be required in commercial development projects. The first option (online access) requires a centralized server and preferably clients do not have to install anything on their own hardware (other than a standard compliant browser). For the second option (entire SenseLabOnline system on customer's hardware) it should preferably be very easy to install, host and maintain. For a traditional web system, this often means installing and maintaining a separate web server and database server which often is rather complicated and requires server licenses. To get around this, the relatively new open source and cross platform .NET Core framework (developed and maintained by Microsoft) is used as a base for the web server implementation and SQLite (open source) is used as database. These choices have the following benefits: A .NET Core web application can be hosted on traditional web servers (like Microsoft's web server Internet Information Services or the open source cross platform NGINX), but it can also run "standalone" with inbuild web server. SQLite is a popular, well tested, single-file, serverless (in-process) database requiring no server installation, but clients access the database file directly. For simple installations, running SenseLabOnline is as simple as running a single executable file. Actually, installation is not even required, the system can be started directly from for example a USB stick (though this is not the recommended way of running SenseLabOnline). The database is a single file, which can easily be copied and backed up.

Configuring a test often involves many details and can be error prone. The philosophy in SenseLabOnline is to avoid file based configuration and provide a user interface with sensible options helping to avoid mistakes, which potentially invalidate collected data for its intended use.

Security is an important aspect of online systems, and SenseLabOnline has several layers of data access, so for example, a test person can only access data for tests he/she participates in. Also, there is data separation between different clients configuring and running tests on the same system.

While being able to execute standard listening tests, the core of SenseLabOnline is, via an add-inn architecture, designed so it is relatively easy to extend with new types of tests (not limited to listening tests). This also provides the needed flexibility to quickly implement tests with requirements not immediately met by the existing system.

A good user interface with dynamic user interaction, as required in a listening test, can be challenging to implement in a browser (HTML/JavaScript). And further, there may be special requirements for for example rating scales and labelling. As an example, Figure 1 illustrates how scales can be in a "not rated" state and how a guideline helps positioning a rating in relation to other ratings. Angular (referring to Angular 2 and newer, not AngularJS) has been used to implement this (with TypeScript code) and the ability of Angular to "data bind" rating data (and other data) is very helpful. Bootstrap (HTML/CSS style framework) is also used, enabling the user interface to dynamically adapt to different screen/windows sizes.

3. AUDIO PRESENTATION IN SOFTWARE

At the lowest level (sound card driver) audio samples are written directly into a fixed size output buffer for the sound card for immediate playback. The audio driver requests data to be written into the buffer by interrupts or callbacks. Depending on audio driver and configuration it is possible to specify desired buffer size, sample rate and channel count for the hardware. Also, it is possible to request information of bit-depth (sample format) at the hardware level as to obtain "bit perfect" playback. This is for example possible by using ASIO drivers or WASAPI (Windows Audio Session API) drivers (in "exclusive" mode). Controlling buffer size also makes it possible to obtain very small latencies (to obtain fast response times to user actions). Note that audio software typically must share audio device with other software thus not having the privilege of setting sample rate and buffer size and often there is no guarantee that other software is not generating interfering sounds. The concept of "exclusive mode" remedies this by letting a specific program take ownership of audio hardware and thus preventing other programs from using the same audio device. "Exclusive mode" must be implemented at the driver (lowest) level of the audio chain.

3.1 JavaScript limitations

In this paper "JavaScript" is used in a broad sense and refers to the language (ECMAScript specification) and available APIs in a web browser (as standardized by W3C). JavaScript is the programming language for the browser, so audio programming for a web browser is subject to limitations of JavaScript.

Audio files can be rather large, and there is no standard way of storing large files locally in JavaScript. There is a concept of "LocalStorage", but with limitations for data size, making it inappropriate for audio files. This means audio files must reside in memory on the client device. There is no restriction on memory use in JavaScript, but the amount of physical memory on the client may be a problem. In a listening test, switching between sounds is required on a single trial, so not all audio for an entire test needs to reside in memory at the same time, but only audio files for individual trials. There is also the possibility of "streaming" audio directly from the server, but switching and crossfading between streamed audio is hard to obtain without glitches.

JavaScript executes code synchronously in a single thread

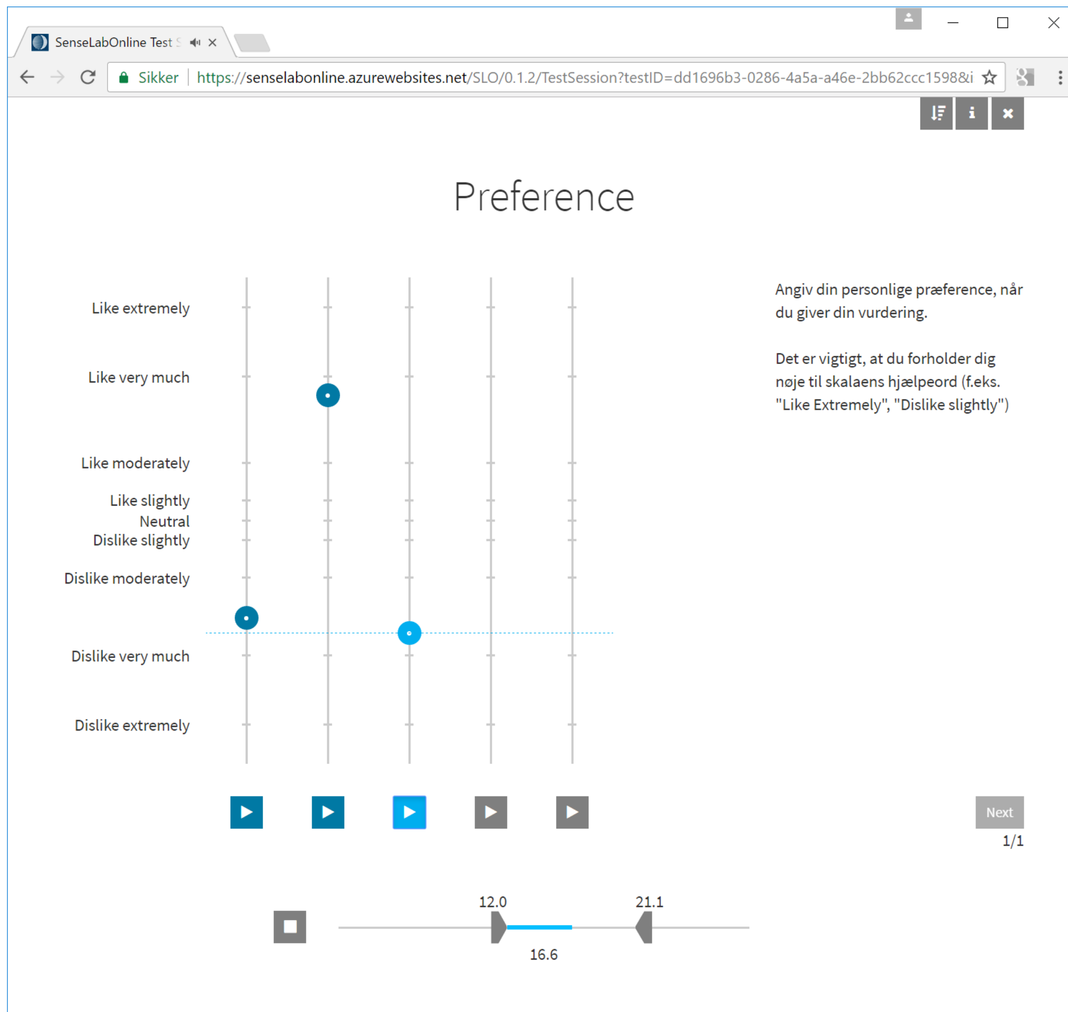

Figure 1: Example of test session in SenseLabOnline. Labels for rating scales can be configured with irregular intervals (a modified version of Labelled Hedonic Scale in this case [5]). Helping guideline is provided for the actively (dragged) rated scale. Playback status and control is in the bottom of the screen. The active section of the sound ("zoom") can be set by the assessor. When configuring the test, automatic looping of the selected range can be enabled or disabled.

(conceptually at least, and with the exception of Web Workers) and this has consequences for audio presentation as will be discussed later, but basically this means audio samples cannot be processed individually in code.

3.1.1 Web Audio API limitations

Web Audio API is a set of JavaScript functions and objects which can be used for audio programming. The behavior of these are described by a standardized specification [2].

The specification states that "implementations must use block processing, with each AudioNode processing 128 sample-frames in each block". The 128 sample block is somewhat comparable to the buffer size mentioned earlier, but it is important to realize that they are different things, where the 128 block size for processing merely gives an absolute lower limit with respect to buffer latency and the actual audio buffer size at the hardware level is most likely larger depending on the actual Web Audio API implementation and its use of the operating system's audio drivers. Somewhat related to the block size is "AudioContext.currentTime", which can be used for precise timing. It should be noted that "AudioContext.currentTime" is not elapsed time, but rather it "is the time in seconds of the sample frame immediately following the last sample-frame in the block of audio most recently processed" [2]. This definition may be somewhat problematic as it is not totally clear what happens if "most recently processed"' frame progresses during a single JavaScript event loop. Potentially "AudioContext.currentTime" is not constant in a single JavaScript event loop which is somewhat contradictory to "normal" JavaScript behavior.

There are no functions available in Web Audio API to set sample rate or request "exclusive" mode and audio channel and sample rate resampling is performed in Web Audio API if required. In most use cases this is a good thing, because it makes misuse of audio hardware harder for malicious web pages, but it also means that "bit perfect" audio repro-

duction cannot be guaranteed. Also, note that an internet browser potentially must mix audio of several opened web pages. As stated, bit perfect reproduction cannot be guaranteed in Web Audio API, but on the other hand, it might also be possible under specific circumstances when the right browser and operating system audio settings are (and can be) made. For example, the Chrome internet browser until recently had a (probably unknow by most) switch enabling "exclusive" mode use of audio drivers in Windows. Unfortunately, this feature of Chrome now seems to be abandoned.

In many listening tests "bit perfect" reproduction is not a requirement, but in some it is. SenseLabOnline online supports both cases, being able to present audio using Web Audio API, but also being able to redirect audio presentation to an "external presenter" with detailed hardware control as will be described later.

3.2 Requirement and implementation

This section describes requirements for sound control and how this can be obtained using JavaScript and Web Audio API and related problems. Several functions from the API are discussed and the reader is assumed to be familiar with these or consult their documentation elsewhere (for example [6] and [7]).

3.2.1 Looping and zooming

When comparing sound samples in a listening test it is often desirable to have a given sound repeat itself, and further, it is also desirable to be able to focus on only a part of the sound (to "zoom"). This behavior is implemented in SenseLabOnline and the user interface for this is illustrated in Figure 1 ("play bar" in the bottom). Given the availability of relatively high-level functions in Web Audio API such as "linearRampToValueAtTime" and "start" and "stop" at given times, this might initially seem quite trivial to implement, but the requirement of Web Audio API to schedule ahead in time and at the same time also be able to immediately react to user interactions make this surprisingly difficult, though possible. Also, there is no "seek" operation in Web Audio API, so this has to be implemented by for example "stop" followed by "start" at a given position. An "AudioBufferSourceNode" in Web Audio API has a "loop" property, but when fading has to be applied at beginning and end of the loop, implementation of loop is more easily achieved by a scheduled fade out and then "stop" followed by a scheduled "start" and fade in. When at any given time the listener is allowed to change the "zoom" range, already scheduled events have to be cancelled and new scheduled events established. The scheduling has to be implemented using for example JavaScript's "setTimeOut" which, because of JavaScript's single thread, is not particularly accurate. Therefore, implementation requires elaborate "book keeping" of Web Audio API event scheduling and general JavaScript scheduling which must also take care of rescheduling potentially involving "cancelScheduledValues" for Web Audio API events and "clearTimeout" for JavaScript events. Also, it can be noted that even though Web Audio API provides a few events which can be listened for (sound source "ended" event for example) it is the authors experience that these are not working reliable across different internet browsers and all event timing must be via JavaScript timers ("setTimeOut" for example) and tight time alignment carried out in timer event handlers with Web Audio API's "AudioContext.currentTime" as reference.

3.2.2 Fading curve

When switching between two sound samples ITU-R BS.1534-3 requires raised 5-ms raised cosine fading curves. This can be obtained with the "setValueCurveAtTime" of at gain node in Web Audio API. Alternatively, sample-by-sample control is also possible (and has been tried out) using the "ScriptProcessorNode", but this method has now been deprecated and has problems with glitches caused by the "single threaded-ness" of JavaScript. The concept of an "AudioWorker" (or "AudioWorklet" in the latest "Editor's Draft" of the specification [1]) has newly been introduced as a replacement, but is hardly implemented yet across browsers. As described by the Web Audio API specification the use of AudioWorker-code potentially lowers priority of the audio rendering thread increasing the possibility of audio glitches. Therefore "setValueCurveAtTime" currently seems to be the best solution for detailed control of wave shape. The "linearRampToValueAtTime" and "exponentialRampToValueAtTime" are also available for fading, but are not able to fulfill the "raised-cosine" requirement of the standard. The "setValueCurveAtTime" method takes two parameters: (1) An array of numbers describing the shape of the curve and (2) a number describing the total length (in seconds) of the curve. The values of the value curve are equidistantly distributed over the specified duration and linearly interpolated by Web Audio API to fit the relevant sample rate. If the parameters for "setValueCurveAtTime" are chosen such that the resolution of the numbers for scaling is the same as that of the sampling rate, sample scaling can be controlled at single sample level.

3.3 Validation measurements

Even though a procedure for fading and looping can be theoretically described it is also of interest to measure the performance of actual implementation executed in various internet browsers. This gives some insight into how well Web Audio API is implemented across browsers and how similar their performance is. In this section, such control measurements of the following scenario are presented: A listener listens to sound A and presses a button in the web interface to switch (fade) to sound B. Two "AudioBufferSourceNode"s (one for sound A and one for sound B) were used. Each of these were connected to a "GainNode" used for fading. The output of these were connected to "AudioContext.destination" (the overall audio output). When the listener clicked the button, fading out of sound A was scheduled immediately ("setValueCurveAtTime" called on gain node with start time parameter equal to "AudioContext.currentTime"). Simultaneously fade in was scheduled for sound B with start time equal to "AudioContext.currentTime" plus the length of the fading curve for sound A (that is, sound B starts fading in immediately after sound A has faded out). "stop" (sound A) and "start" (sound B) were scheduled on buffer nodes to align with end of fade out (sound A) and beginning of fade in (sound B).

The exact same code implementing this scenario was executed in three different browsers: Chrome (version 56.0.2924.87), Edge (version 38.14393.0.0) and Firefox (version 51.0.1). Running these browsers under Windows 10, the following test was made: Two wav sound files were generated with a sample frequency of 44.1 kHz, two channels

and a resolution of 16 bit per sample (signed integer). The sample values contained in each file were of constant value where, in one file (sound A), the constant value was half of full scale and in the other (sound B) the value was half of the negative full scale. Raised cosine fading curves to be used with "setValueCurveAtTime" were generated by the JavaScript function "buildFaderCoefficients" with code listed in Code 1. In the testing a 5 ms fader duration was specified and a sample rate of 44.1 kHz was given as parameter to the "buildFaderCoefficients" function.

Code 1: JavaScript function for generating fading curves

```
function buildFaderCoefficients
 (duration, fs) { // fs is samplerate fx 44100
  var faderLength = Math.ceil(duration * fs);
  if (faderLength < 2) faderLength = 2;
  var inFader = new Float32Array(faderLength);
  var outFader = new Float32Array(faderLength);
  for (var i = 0; i < faderLength; i++) {
    var raisedCosine = 0.5 *
     (1+Math.cos(i/(faderLength-1)*Math.PI));
    inFader[i] = 1 - raisedCosine;
    outFader[i] = raisedCosine;
  }
  return {
    duration: faderLength / fs,
    coefficients: {
      in: inFader,
      out: outFader
    }
  };
};
```

The audio device (RME Fireface UC) was configured for two channel playback at 44.1 kHz with 24 bits per sample. 44.1 kHz was also reported as the sample rate in the three tested browsers ("AudioContext.sampleRate" in Web Audio API). A digital loopback was configured on the sound card, which means the exact sound output from each browser could be recorded (recording was also made at 44.1 kHz). Fade in and out according to the described scenario was recorded for the three browsers and the results are presented in Figure 2. The figure also presents fading when directly using ASIO drivers via external presenter (described later). The graphs in the figures represent consistent behavior of the browsers and not atypical cases. The dashed line indicates the ideal target curve. As can be seen, none of the browsers behaves perfectly, but have individual deviations from the desired target curve. It is not the purpose of this paper to identify errors in internet browsers, but examine if they behave similarly (with respect to audio reproduction) provided the same code. So, whether the "error" is in the browser or in the executed code is not important in this context. However, a few comments can be made about the individual browsers: From close inspection, it is seen that Chrome does not scale the first sample by the fading curve when fading in. Edge does not scale what appears to be the first 128 samples (the Web Audio API block size) when fading out. Firefox does not appear to scale samples by the fading curve at all, but correctly switches between the two sounds. If better understanding of this behavior is wanted, it may be relevant to note that fade in by "setValueCurveAtTime" is scheduled at "AudioContext.currentTime" (i.e. immediately) and other events are scheduled 5 and 10 ms ahead of time respectively. It can also be noted that, when everything is scheduled sufficiently ahead of time, Edge and Firefox can be made to hit the target curve and Chrome's spike can be removed by explicitly setting the scale for the first sample.

3.3.1 Browser idiosyncrasies

As illustrated, browsers are not behaving equal. It is also the experience of the author that this problem is not restricted to the specific case presented here. Under non-critical circumstances, hardly anybody would notice the differences in fading curves as presented earlier. But in critical situations, the differences would be clearly heard which might be problematic in some listening tests, and furthermore, one can hardly claim to do a test according to a standard if for example it in its specification have requirements for fading which are clearly violated. To get around this, one may try to make "work arounds" for specific browsers. This is, however, not very desirable as one is no longer relying on web standards, but implementation details of individual browsers and versions. On the positive side, it is the author's experience that "steady state" audio rendering works well across browsers and even devices (computers, phones, tablets).

3.4 External presenter - detailed control and video

Web Audio API can be used for audio presentation with a degree of control and quality that is sufficient in many cases, but for the cases where better control of stimulus presentation is required SenseLabOnline has the possibility of sending control commands to an external presenter. The same browser based user interface (as for example seen in Figure 1) is used whether using Web Audio API audio presentation or the external presenter. An abstract command based code interface has been developed and the commands given via the user interface are then sent to a selected implementation of that interface in this case being either Web Audio API based implementation or the external presenter implementation. The external presenter is implemented as a regular Windows program thus able to directly utilize ASIO (or WASAPI) drivers for detailed hardware control. Efficient communication between browser (user interface) and external presenter is made possible by the Web Sockets web standard. The actual implementation uses Microsoft's SignalR library for this, and the external presenter is in itself also a websocket server. In less technical words this means that using web standards the browser based user interface can be used to control a presenter program with full hardware rights. This also opens new possibilities as the external presenter can control everything regarding hardware so presentation is not necessarily restricted to audio only presentation. Specifically, an external presenter which is able to present both audio and video has been developed where great care was taken for audio and video to be synchronized while being able to switch between different audio processings for the same video. And also, the external presenter implementation is able to handle uncompressed (raw) video, which would not be possible using a purely web standard based implementation.

4. CONCLUSIONS

It has been argued that web standards provide a good platform for implementation of user interface for doing listening tests (both configuration management, and test

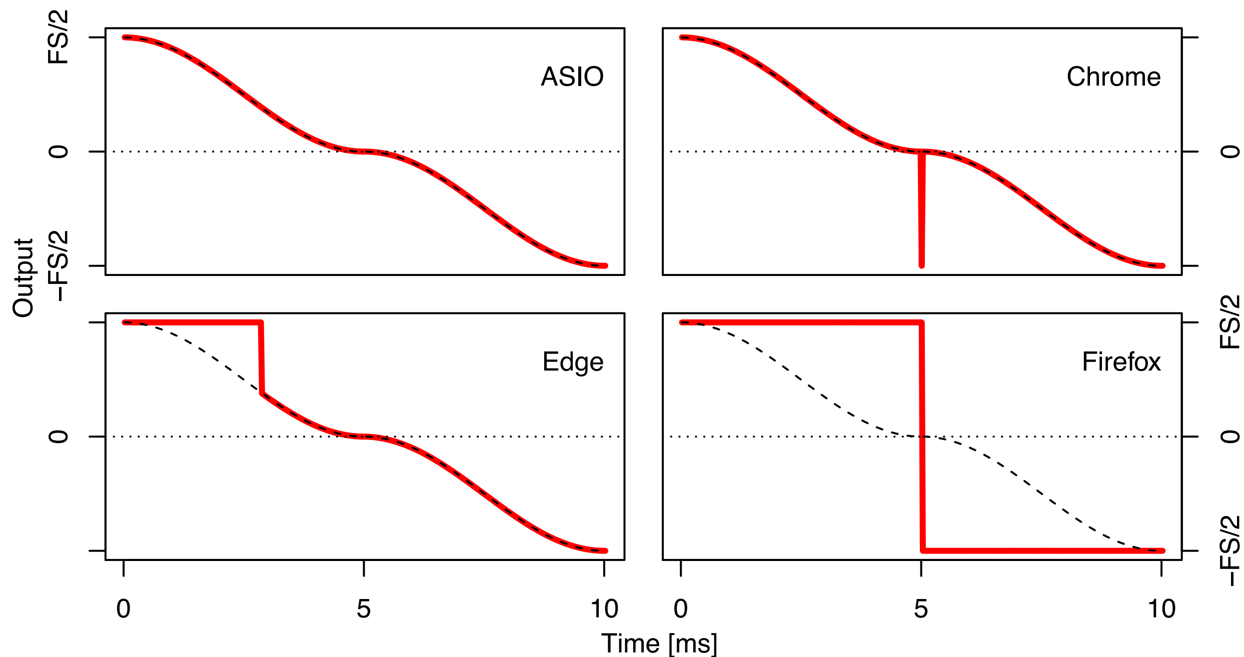

Figure 2: Fading using external presenter (ASIO) and three different browsers. A positive constant (FS/2, half of full scale) is faded out and immediately after, a negative constant (-FS/2) is faded in. The fading curve has raised cosine shape and a duration of 5 ms. Dashed curve indicates theoretical target curve and red curve represents digitally measured values.

execution proper). Using Web Audio API, the only requirement to run the test is a standard compliant internet browser, but if the requirement for audio precision is high, other means of audio presentation can be necessary. Some of the problems seem to stem from the definition of "AudioContext.currentTime" (bound to frame most recently processed). "currentTime" may be in the past when related JavaScript code is actually executed. It is up to the JavaScript programmer to guess (or probe) how much ahead of time to schedule events to avoid glitches. Because this depends on hardware and browser implementation, it seems reasonable that the Web Audio API should somehow provide this information (or at least some hint of a "safety" offset for "currentTime").

In the future, browser implementation of Web Audio API may improve, and the Web Audio API specification may develop (AudioWorklets [1] for example) so alternative means for precision audio presentation becomes irrelevant. Whether this happens, will probably depend on how important precise audio presentation is deemed by developers of web browsers. In most use cases (non-listening tests), the differences between browsers illustrated earlier, hardly would be noticed by anyone and consequently resolution of this type of issues may have low priority in browser development. So, even though Web Audio API provides a good platform for many listening tests, critical listening tests may have to rely on other means in the foreseeable future.

5. REFERENCES

[1] P. Adenot and R. Toy. Web Audio API, W3C Editor's Draft 21 June 2017. https://webaudio.github.io/web-audio-api, 2017.

[2] P. Adenot and C. Wilson. Web Audio API, W3C Working Draft 08 December 2015. https://www.w3.org/TR/2015/WD-webaudio-20151208/, 2015.

[3] N. Jillings, B. De Man, D. Moffat, J. D. Reiss, and R. Stables. Web Audio Evaluation Tool: A framework for subjective assessment of audio. In *2nd Web Audio Conference*, apr 2016.

[4] S. V. Legarth, J. Ramsgaard, G. Le Ray, and N. Zacharov. A performance comparison of Home Usage Testing and Central Location Testing in small impairment listening tests. In *3rd International Workshop on Perceptual Quality of Systems (PQS)*, Dresden, Germany, 2010.

[5] J. Lim, A. Wood, and B. G. Green. Derivation and evaluation of a labeled hedonic scale. *Chemical Senses*, 34(9):739–751, 2009.

[6] Mozilla. JavaScript. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[7] Mozilla. Web Audio API. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API.