BeaqleJS: HTML5 and JavaScript based Framework for the Subjective Evaluation of Audio Quality

Sebastian Kraft and **Udo Zölzer**
Department of Signal Processing and Communications
Helmut-Schmidt-University
Holstenhofweg 85, 22043 Hamburg, Germany
sebastian.kraft@hsu-hh.de

Abstract

Subjective listening tests are an essential tool for the evaluation and comparison of audio processing algorithms. In this paper we introduce BeaqleJS, a framework based on HTML5 and JavaScript to run listening tests in any modern web browser. This allows an easy distribution of the test environment to a significant amount of participants in combination with simple configuration and good expandability.

Keywords

listening test, subjective audio evaluation, HTML5, JavaScript

1 Introduction

Frequently used physical measures to judge the quality of audio signals, like the signal to noise ratio or signal distortion, do not correlate well with the perception of quality by the human hearing system. Therefore, listening tests, also named subjective audio evaluations, play an important role in the comparison of signal processing algorithms like audio effects and codecs.

The setup of a test environment and the selection of items under test is crucial to yield significant and non-biased results. Some guidance and standards can be found for example in the International Telecommunication Union (ITU) recommendations and in particular in [1]. Still, one of the biggest problems is to address an adequate number of qualified participants. Closely connected is the problem of distributing the test environment to the various platforms of the participants and how the results could be merged and evaluated afterwards.

In this paper BeaqleJS (browser based evaluation of audio quality and comparative listening environment) is presented, which is a framework to easily setup and run listening tests in any modern web browser. To achieve this, BeaqleJS purely relies on open web standards like HTML5 and JavaScript, without the need of further browser plugins or extensions. It is published under the GPLv3 open source license and its source code is available on GitHub[1].

The following section 2 will first introduce some background information about listening tests and common standards in general. Afterwards, the BeaqleJS framework is described in section 3 and section 4 outlines advanced usage scenarios like modifying or implementing new test schemes as well as server side evaluation and data collection. Finally section 5 will give a conclusion and outlook.

2 Listening test standards and basics

The difficulty in setting up a listening test comes from the fact that humans are rarely objective in their judgements. Therefore, the challenge is to design a test environment that minimizes external influences and yields non-biased results. To avoid mistakes it is advised to stick close to standardised instructions and test procedures as they are for example defined by the ITU in [1][2][3].

Test items should be presented in random order together with neutral names avoiding any association to the underlying algorithms. If several different algorithms are compared in one test, the corresponding items should always appear at a random position to prevent that the listeners recognize or learn the connection between a rating and its item position.

It is also necessary to find a way to judge the ability of the participants to understand the test procedure or to even recognize if they are able to perceive any differences between the items at all. For this purpose, a hidden reference and an anchor signal can be mixed among the test items. In valid test results, the participants should always rate the hidden reference with the same quality as the visible reference. In contrary, an anchor signal is an obviously bad test

[1] GitHub is a source code hosting platform using the git version control system http://www.github.com

item, for example heavily lowpass filtered, that is expected to always catch the worst rating and will set the bottom end of the scale.

The experience of the participants can have a strong influence on the results. People that are trained to hear analytically and know what artefacts they have to listen for can usually give more detailed feedback. On the other hand, they might be quite biased in their understanding of audio quality and typical consumers may highlight completely different aspects. Therefore, it makes sense to consider and document the background of the participants.

In a decentralized and distributed test setup it is important to assure comparable playback conditions. This is best achieved when all participants make use of high quality studio headphones as these completely reduce the influence of room acoustics and can be expected to be quite linear over a broad frequency range.

The selected audio test items should be well selected to reflect and underline a variety of characteristics of the tested algorithms. This can include for example transient, noisy and harmonic signals. A good starting point for choosing audio is the SQAM (Sound Quality Assessment Material) CD from the European Broadcasting Union (EBU) [4], which is well established in the audio coding field. The individual test item should be quite short and not exceed a length of 10 seconds. On the one hand, this helps to keep the attention of all listeners focused to the same part of the item, but also to avoid that exhaustion of the participants will influence the results. For the same reason, the amount of time that is necessary to perform the whole test should be kept below 15 minutes.

3 The BeaqleJS framework

BeaqleJS provides a framework to create browser based listening tests and is purely based on open web standards like HTML5 and JavaScript. For the user interface and to simplifiy Document Object Model (DOM) manipulations, the well known jQuery and jQueryUI libraries [5] are used.

The general structure of BeaqleJS can be divided in three blocks (Fig. 1). There is a common HTML5 `index.html` file to hold the main HTML structure with some basic place holder blocks whose content will be dynamically created by the JavaScript backend. The styling is completely independent and done with the help of cascading style sheets (CSS). Style sheets,

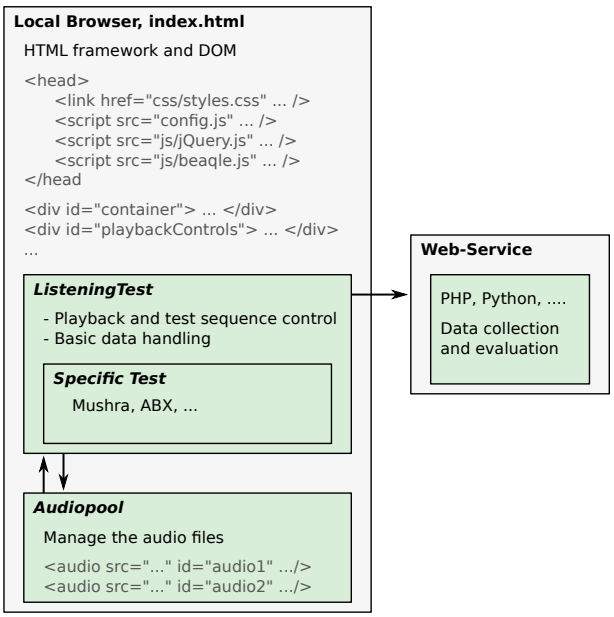

Figure 1: Schematic overview of the BeaqleJS framework.

config files and all necessary JavaScript libraries are loaded in the header of the `index.html`. Most of the descriptive text, like introduction and instructions, are placed in hidden blocks inside this file and their visibility is controlled by the scripts.

The JavaScript backend consists of two main classes. The first one is the `AudioPool` which takes care of audio playback and buffering. It pools a set of HTML5 <`audio`>-tags in a certain AudioPool <`div`>-tag. There are simple functions to add and load a new file, connect and address it with an ID, manage playback and looping as well as synchronized pause and stop operations.

The `ListeningTest` class provides the main functions of an abstract listening test. This includes the setup and management of basic playback controls (play, pause, looping, time line display, ...), reading of the test configuration as well as storage of the results and also main control over the test sequence.

To create a certain test type the abstract `ListeningTest` class is inherited and specific functions for the actual arrangement of test items or storage and evaluation of the results need to be implemented (cf. section 4.1). Based on this modular approach it is very easy to extend the framework with additional test types or to create variants of existing ones without the need to reimplement all the necessary basics.

If the test is performed distributed over the

Table 1: Overview of supported codecs and audio formats in current web browsers.

Browser	Internet Explorer	Firefox	Chrome	Opera	Safari
WAV PCM	no	> 3.5	yes	> 11.00	> 3.1
Ogg Vorbis	no	> 3.5	yes	> 10.50	with XiphQT
MP3	> 9.0	> 26, not OS X	yes	> 14	> 3.1
AAC	> 9.0	> 26, not OS X	yes	> 14	> 3.1

internet or on several local computers, the `ListeningTest` main class is also able to send the final ratings to a web service for centralised collection and evaluation.

3.1 Codec support

Although the HTML5 markup language is already widely used in the internet there is no final adopted standard but only various drafts from the world wide web consortium (W3C). This may be one of the reasons why its degree of implementation can differ a lot between the different web browsers.

Our main interest regarding browser compatibility is the HTML5 <`audio`> element and its support for various file types, whereas in particular lossless formats like WAV or FLAC would be best suited for the desired application. An overview of the supported formats is given in Table 1 and unfortunately no browser supports FLAC[2] or other lossless codecs so far. The only lossless, but also uncompressed, format widely accepted is WAV PCM with 16 bit sample precision. Solely the Internet Explorer is not capable to play back this file type.

The described overall situation regarding the support of a common codec is quite unsatisfying. At the time of writing the only recommendation for an audio listening test environment would be to use the WAV PCM format. The circumstance that it is completely uncompressed (data rate approx. 94 kB/s per channel at 44.1 kHz sample rate) is relativised by the fact that the individual audio test items are recommended to be quite short, usually not more than 10 seconds, and therefore, the overall amount of audio data to be loaded is limited to around 1-2 MB per test item.

[2]It should be noted that there is a JavaScript based audio decoder framework named Aurora.js that is available together with a FLAC decoder at GitHub (https://github.com/audiocogs/aurora.js). However, its adaptability still has to be investigated.

3.2 Predefined tests

As described in the beginning of section 3 the main `ListeningTest` class only provides an abstract implementation with the core functionality of a generic listening test. Two implementations of specific listening tests are currently available in BeaqleJS. The most simple one is the so called ABX test and it is best suited to understand the functioning and internals of the whole framework. The other one is the so called MUSHRA (multi stimulus test with hidden reference and anchor) which is widely used in many evaluation scenarios and therefore, one of the most common test types. It is defined by the ITU in the BS.1534-1 recommendation [3].

3.2.1 ABX

In an ABX test (see Fig. 2) three items named A, B and X are presented to the listener, whereas X is randomly selected to be either the same as A or B. The listener has to identify which item is hidden behind X, or which one (A or B) is closest to X. If the listener is able to find the correct item, it reveals that there are perceptual differences between A and B.

A typical application of ABX tests would be the evaluation of the transparency of audio codecs. For example item A could be an unencoded audio snippet and B is the same snippet but encoded with a lossy codec. When the listener is not able to identify if A or B was hidden in X (results are randomly distributed), one can assume that the audio coding was transparent.

3.2.2 MUSHRA

In a MUSHRA test (see Fig. 3) the listener gets presented an item marked as reference together with several anonymous test items. By using a slider for each test item he has to rate how close the items are to the reference on top. Among the test items there is usually also one hidden reference and one, or several, anchor signals to prove the validity of the ratings and the quali-

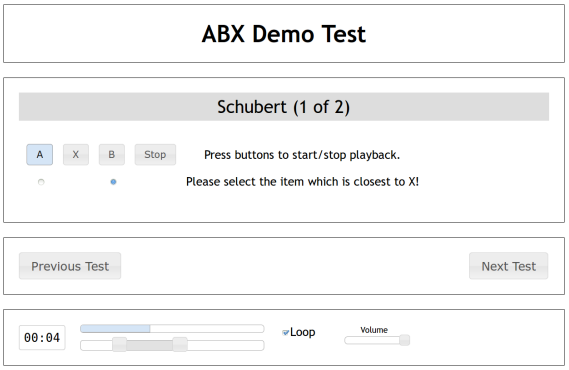

Figure 2: Screenshot of the ABX demo test.

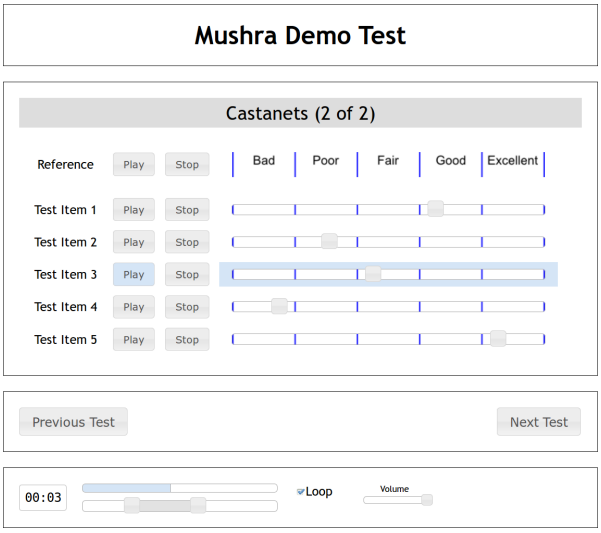

Figure 3: Screenshot of a MUSHRA test.

fication of the participants.

Contrary to ABX tests the MUSHRA procedure allows more detailed evaluations as it is possible to compare more than one algorithm to a reference. Furthermore, the results are on a continuous scale allowing a direct numerical comparison of all algorithms under test.

4 Advanced usage

If one of the predefined test classes already covers the desired test requirements, it is only necessary to create a set of test items and to define the corresponding paths in the config script. But it is also quite simple to slightly modify existing test layouts and structures or to even implement completely new test schemes.

4.1 Implementation of new tests

All new test classes have to inherit the base functionality from the main `ListeningTest` class. Inheritance in JavaScript is achieved by prototypes and this means to define a new class

```
// inherit from ListeningTest
function MyTest(TestData) {
  ListeningTest.apply(this, arguments);
}
MyTest.prototype = new ListeningTest();
MyTest.prototype.constructor = MyTest;

// implement the necessary functions
MyTest.prototype.createTestDOM = ...
MyTest.prototype.saveRatings   = ...
MyTest.prototype.readRatings   = ...
MyTest.prototype.formatResults = ...
```

Listing 1: Creation of a new test class `MyTest` inheriting from `ListeningTest`.

```
ListeningTest.TestState = {
// main public members
'CurrentTest': -1,
'TestIsRunning': false,
'FileMappings': {},
'Ratings': {},
'EvalResults': {},
// ...
// optionally add own fields
// ...
}
```

Listing 2: The `TestState` structure.

`MyTest` and then set its prototype to the base class. As this overwrites the constructor it has to be reset to the child constructor afterwards (listing 1). The child class can access the `TestState` (listing 2) and the `TestConfig` (listing 3) from the parent. The first one can be used to store random file mappings, ratings and other status variables, but can also be dynamically expanded with specific fields required by the child class. The `TestConfig` structure is just a mapping of the BeaqleJS config file into the class namespace. It has to contain at least the fields and structure as in listing 3 but it is possible to add additional sections which are then only read by the child class.

Every new test class has to implement at least four new functions:

- `createTestDOM(TestIdx)` creates the visible layout and HTML structure of the test with the index `TestIdx` based on the test configuration. All the necessary information from the config is available inside the object in `this.TestConfig.*`. Audio files should be appended to the `AudioPool` with `this.addAudio()` and can then be connected to play buttons by unique file IDs. Random mapping of file-

```
ListeningTest.TestConfig = {
"TestName": "Test",
"LoopByDefault": true,
"EnableABLoop": true,
"EnableOnlineSubmission": false,
"BeaqleServiceURL": "http://...",
"SupervisorContact": "super@visor.com",
"Testsets": [
    {
        "Name": "Testset 1",
        "Files": {
            // ...
        }
    },
    {
        "Name": "Testset 2",
        "Files": {
            // ...
        }
    }
    ],
// ...
// further test specific settings
// ...
}
```

Listing 3: The `TestConfig` structure.

names to IDs can be stored in the prepared `this.TestState.FileMappings[TestIdx]` structure (listing 2).

- `saveRatings(TestIdx)` is used to obtain the ratings from the sliders or buttons in the DOM and to store them in an arbitrary format in the predefined `this.TestState.Ratings[TestIdx].*` object.

- `readRatings(TestIdx)` is intended to read the ratings for `TestIdx` from `this.TestState.Ratings[TestIdx]` and to reapply them to the sliders or buttons in the DOM. This is primarily used during switching back and forth in the test sequence.

- `formatResults(TestIdx)` is automatically called after the final test in the sequence. It is supposed to evaluate and summarize the ratings and to store the final results in `this.TestState.EvalResults`. It should return a string containing the results formatted in a human readable manner (HTML). This will be presented to the listener after the last test and the `EvalResults` structure may be send to a web service (section 4.2).

4.2 Server side data collection and evaluation

Unfortunately it is not possible to directly send emails with JavaScript locally from a web browser or to store files. Therefore, to automatically collect the results it is necessary to have some kind of web service reachable from your network. This can be implemented for example with Python, Node.js or simply PHP.

The `ListeningTest` class includes the basic functionality to pack the results object `this.TestState.EvalResults` into a JSON (JavaScript Object Notation) structure and to transfer it to a web service for collection and further evaluation. A simple PHP example is included in the `beaqleJS_Service.php` file. It receives a JSON encoded data structure and writes its content into a text file with the time stamp as filename.

In the future, a more enhanced server side evaluation could include the automatic visualisation and statistical analysis of all the collected results. This can be combined with the capability to export the data in various formats for further analysis in scientific tools like SciPy, R or Matlab.

5 Conclusion

One big difficulty in setting up a proper listening test for the subjective evaluation of audio is its distribution to a significant number of participants. This is addressed by BeaqleJS which supplies all necessary components to run listening tests in any modern web browser in a flexible manner. It enables various usage scenarios ranging from complete online tests, over semi-public distribution in the intranet, down to local installation on a single computer with direct attendance of a supervisor. The presented framework, and its predecessor MushraJS, has already been used in various evaluations and proved its practical capabilities [6][7].

However, to assure significant and unbiased results, it is always advisable to closely stick to predefined and established test methods as they were introduced in section 2.

Further development could include a more extensive server side data evaluation and visualisation, but of course also the addition of more test schemes. The code is available at `https://github.com/HSU-ANT/beaqlejs` and the reader's contribution and feedback are highly appreciated.

References

[1] ITU-R. RECOMMENDATION ITU-R BS.1284-1: General methods for the subjective assessment of sound quality. Technical report, 2003.

[2] ITU-R. RECOMMENDATION ITU-R BS.1116-1: Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems. Technical report, 1997.

[3] ITU-R. RECOMMENDATION ITU-R BS.1534-1: Method for the subjective assessment of intermediate quality level of coding systems (01/03). Technical report, 2003.

[4] European Broadcasting Union. Sound Quality Assessment Material recordings for subjective tests (SQAM), Audio CD. `https://tech.ebu.ch/publications/sqamcd`.

[5] The jQuery Foundation. Homepage of the jQuery and jQueryUI javascript libraries. `http://jquery.com/`.

[6] Sebastian Kraft, Martin Holters, Adrian von dem Knesebeck, and Udo Zölzer. Improved PVSOLA Time-Stretching and Pitch-Shifting for Polyphonic Audio. In *Proc. 15th Int. Conf. on Digital Audio Effects*, 2012.

[7] Marco Fink, Martin Holters, and Udo Zölzer. Comparison of Various Predictors for Audio Extrapolation. In *Proc. 16th Int. Conf. on Digital Audio Effects*, 2013.