Instructions for
Web Audio Evaluation Tool

Nicholas Jillings, Brecht De Man and David Moffat

May 15, 2016

These instructions are about use of the Web Audio Evaluation Tool on Windows and Mac OS X platforms.

We request that you acknowledge the authors and cite our work when using it [1], see also CITING.txt.

The tool is available for academic use in its entirety including source code on `https://github.com/BrechtDeMan/WebAudioEvaluationTool`, under the GNU General Public License v3.0 (`http://choosealicense.com/licenses/gpl-3.0/`), see also LICENSE.txt.

The SoundSoftware project page, including a Mercurial repository, is `https://code.soundsoftware.ac.uk/projects/webaudioevaluationtool/`.

Contents

1 Installation

Download the folder (`https://code.soundsoftware.ac.uk/hg/webaudioevaluationtool/archive/tip.zip`) and unzip in a location of your choice, or pull the source code from `https://github.com/BrechtDeMan/WebAudioEvaluationTool.git` (git) or `https://code.soundsoftware.ac.uk/hg/webaudioevaluationtool` (Mercurial).

1.1 Contents

The folder should contain the following elements:

Main folder:

- `CITING.txt, LICENSE.txt, README.md`: text files with, respectively, the citation which we ask to include in any work where this tool or any portion thereof is used, modified or otherwise; the license under which the software is shared; and a general readme file.

- `demo.html`: Several demonstrations of listening tests, using examples from the example_eval folder

- `index.html`: webpage where interface should appear (append link to configuration XML, e.g. index.html?url=config.xml)

- `pythonServer.py`: webserver for running tests locally

- `pythonServer-legacy.py`: webserver with limited functionality (no automatic storing of output XML files)

Analysis of results (`./analysis/`)

- `analyse.html`: analysis and diagnostics of a set of result XML files (legacy)

- `analysis.css`: analysis page style file

- `analysis.js`: analysis functions

- `index.html`: web page where analysis of stored results can be performed

CSS files (`./css/`)

- `core.css`: core style file (edit to change appearance)

Documentation (`./docs/`)

4

- AESPosterComp: PDF and LaTeXsource of Audio Engineering Society UK Sustaining Members event at Solid State Logic, Begbroke

- DMRN+10: PDF and LaTeXsource of poster for 10th Digital Music Research Network One-Day workshop ("soft launch")

- Instructions: PDF and LaTeXsource of these instructions

- Project Specification Document (LaTeX/PDF)

- Results Specification Document (LaTeX/PDF)

- SMC15: PDF and LaTeXsource of 12th Sound and Music Computing Conference paper [1]

- WAC2016: PDF and LaTeXsource of 2nd Web Audio Conference paper [2]

- WAC2016Poster: PDF and LaTeXsource of 2nd Web Audio Conference poster

Interface files (`./interfaces/`)

- Each interface class has a JavaScript file and an optional CSS style file. These are loaded as needed.

JavaScript code (`./js/`)

- `core.js`: JavaScript file with core functionality

- `jquery-2.1.4.js`: jQuery JavaScript Library

- `loudness.js`: Allows for automatic calculation of loudness of Web Audio API Buffer objects, return gain values to correct for a target loudness or match loudness between multiple objects

- `specification.js`: decodes configuration XML to JavaScript object

- `WAVE.js`: decodes and performs WAVE file byte level manipulation

- `xmllint.js`: XML validation

Media files (`./media/`)

- `example`: contains example audio files 0.wav-10.wav which are short recordings at 44.1kHz, 16bit of a woman saying the corresponding number (useful for testing randomisation and general familiarisation with the interface).

PHP scripts (`./php/`)

- `keygen.php`: generates a unique file name for saved results

- **pseudo.php**: allows for pseudo-random selection from a range of configuration XML files

- **save.php**: PHP script to store result XML files to web server

- PHP analysis scripts

Python scripts (./python/)

- Helpful Python and PHP scripts for extraction and visualisation of data.

Output files (./saves/)

- The output XML files of tests will be stored here by default by the `pythonServer.py` script.

Test creation tool (./test_create/)

- Webpage for easily setting up your own test without having to delve into the XML.

Tests (./tests/)

- This is where you can store your configuration XML files.

- Contains a folder with examples.

XML specifications (./xml/)

- **scaledefinitions.xml**: marker text and positions for various scales

- **test-schema.xsd**: definition of configuration and result XML file structure

1.2   Compatibility

As Microsoft Internet Explorer doesn't support the Web Audio API[1], you will need another browser like Google Chrome, Safari or Firefox (all three are tested and confirmed to work).

The tool is platform-independent and works in any browser that supports the Web Audio API. It does not require any specific, proprietary software. However, in case the tool is

---

[1]`http://caniuse.com/#feat=audio-api`

hosted locally (i.e. you are not hosting it on an actual webserver) you will need Python (2.7 or 3.x), which is a free programming language - see the next paragraph.

2 Quick start

This document aims to provide an overview of all features and how to use them. However, if you are just trying out this tool, or you need to put together a test very quickly, or you simply don't want to read through all the details first, this section gives you the bare necessities to put together a simple listening test very quickly.

- Download the tool (see Section 1)

- Copy the tool to a PHP-enabled web server if you have access to one.

- Go to `test_create.html` and configure your test.

- Save your test file in the folder `.\tests\`.

- Your test will be live at `[webserveraddress]/index.html?url=tests/[testname]` `.xml`. If you are not using a web server, you can simulate one locally by running `python/pythonServer.py` (requires Python), after which you can access the test at `http://localhost:8000/index.html?url=tests/[testname].xml`

3 Test setup

3.1 Sample rate

Depending on how the experiment is set up, audio is resampled automatically (the Web Audio default) or the sample rate is enforced. In the latter case, you will need to make sure that the sample rate of the system is equal to the sample rate of these audio files. For this reason, all audio files in the experiment will have to have the same sample rate.

Always make sure that all other digital equipment in the playback chain (clock, audio interface, digital-to-analog converter, ...) is set to this same sample rate.

Note that upon changing the sampling rate, the browser will have to be restarted for the change to take effect.

3.1.1 Mac OS X

To change the sample rate in Mac OS X, go to **Applications/Utilities/Audio MIDI Setup** or find this application with Spotlight (see Figure 1). Then select the output of the audio interface you are using and change the 'Format' to the appropriate number. Also make sure the bit depth and channel count are as desired. If you are using an external audio interface, you may have to go to the preference pane of that device to change the sample rate.

Also make sure left and right channel gains are equal, as some applications alter this without changing it back, leading to a predominantly louder left or right channel. See Figure 1 for an example where the channel gains are different.

3.1.2 Windows

To change the sample rate in Windows, right-click on the speaker icon in the lower-right corner of your desktop and choose 'Playback devices'. Right-click the appropriate playback device and click 'Properties'. Click the 'Advanced' tab and verify or change the sample rate under 'Default Format'. If you are using an external audio interface, you may have to go to the preference pane of that device to change the sample rate.

3.2 Local test

If the test is hosted locally, you will need to run the local webserver provided with this tool.
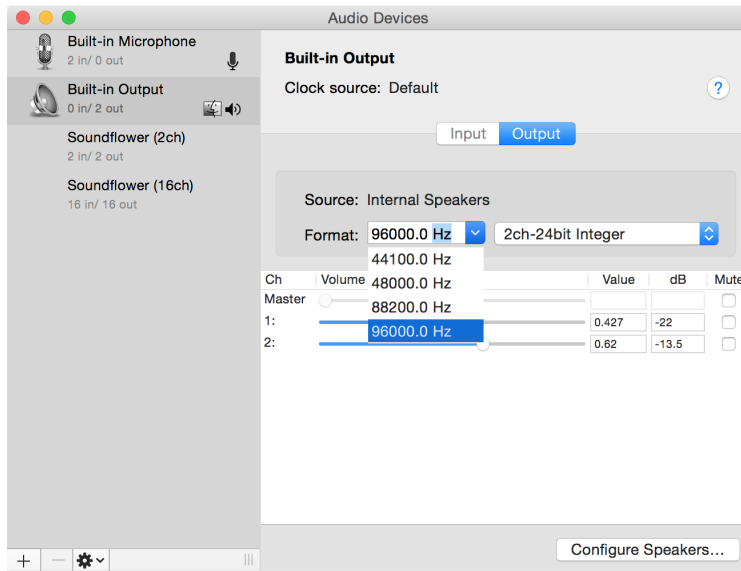
Figure 1: The Audio MIDI Setup window in Mac OS X

3.2.1   Mac OS X & Linux

On Mac OS X, Python comes preinstalled, as with most Unix/Linux distributions.

Open the Terminal (find it in **Applications/Terminal** or via Spotlight), and go to the folder you downloaded. To do this, type `cd [folder]`, where `[folder]` is the folder where to find the `pythonServer.py` script you downloaded. For instance, if the location is `/Users/John/Documents/WebAudioEvaluationToolbox/`, then type

`cd /Users/John/Documents/WebAudioEvaluationToolbox/`

Then hit enter and run the Python script by typing

`python python/pythonServer.py`

and hit enter again. See also Figure 2.

Alternatively, you can simply type `python` (followed by a space) and drag the file into the Terminal window from Finder.

You can leave this running throughout the different experiments (i.e. leave the Terminal open). Once running the terminal will report the current URL to type into your browser to initiate the test, usually this is http://localhost:8000/. On OSX 10.10 or newer, you may get a dialogue asking if Python can accept incomming connections, click yes.

To start the test, open the browser and type
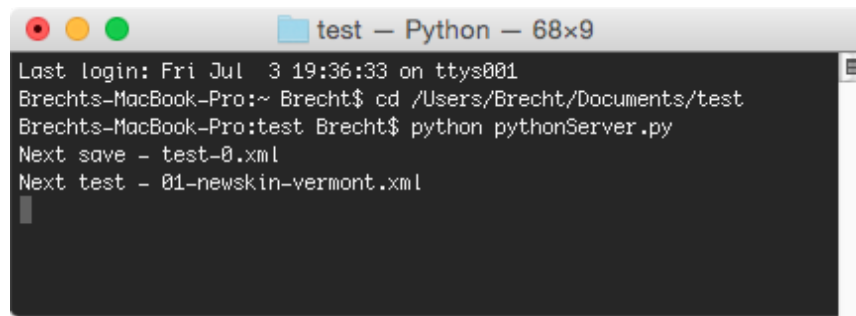
Figure 2: Mac OS X: The Terminal window after going to the right folder (`cd [folder_path]`) and running `pythonServer.py`.

`localhost:8000`

and hit enter. The test should start (see Figure 5).

To quit the server, either close the terminal window or press Ctrl+C on your keyboard to forcibly shut the server.

3.2.2 Windows

On Windows, Python is not generally preinstalled and therefore has to be downloaded[2] and installed to be able to run scripts such as the local webserver, necessary if the tool is hosted locally.

Once installed, simply double click the Python script `pythonServer.py` in the folder you downloaded.

You may see a warning like the one in Figure 3. Click 'Allow access'.

The process should now start, in the Command prompt that opens - see Figure 4.

You can leave this running throughout the different experiments (i.e. leave the Command Prompt open).

To start the test, open the browser and type

`localhost:8000`

and hit enter. The test should start (see Figure 5).

---

[2]`https://www.python.org/downloads/windows/`

11

Figure 3: Windows: Potential warning message when executing `pythonServer.py`.
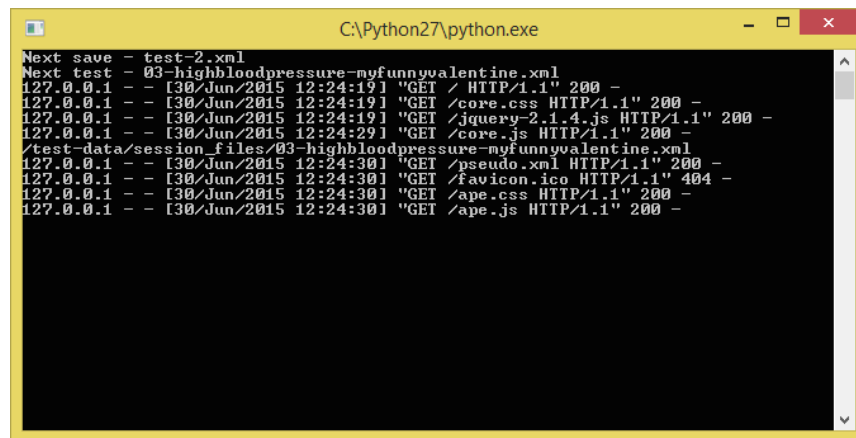
Figure 4: Windows: The Command Prompt after running `pythonServer.py` and opening the corresponding website.
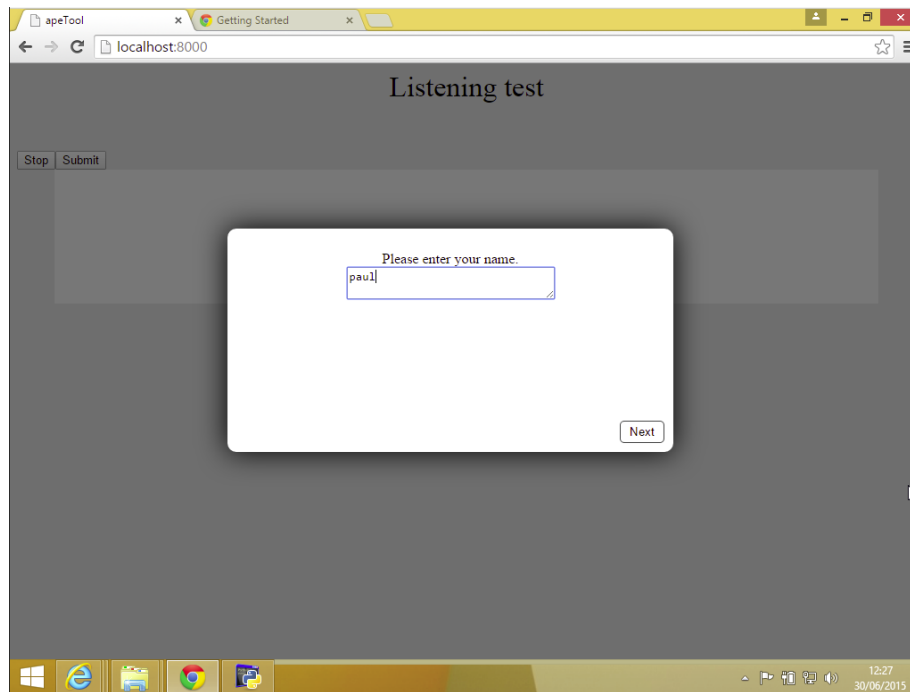
Figure 5: The start of the test in Google Chrome on Windows 7.

If at any point in the test the participant reports weird behaviour or an error of some kind, or the test needs to be interrupted, please notify the experimenter and/or refer to Section 9.

When the test is over (the subject should see a message to that effect), the output XML file containing all collected data should have appeared in 'saves/'. The names of these files are 'test-0.xml', 'test-1.xml', etc., in ascending order. The Terminal or Command prompt running the local web server will display the following file name. If such a file did not appear, please again refer to Section 9.

It is advised that you back up these results as often as possible, as a loss of this data means that the time and effort spent by the subject(s) has been in vain. Save the results to an external or network drive, and/or send them to the experimenter regularly.

To start the test again for a new participant, you do not need to close the browser or shut down the Terminal or Command Prompt. Simply refresh the page or go to `localhost:8000` again, a new session will be created.

13

3.3 Remote test

Put all files on a web server which supports PHP. This allows the 'save.php' script to store the XML result files in the 'saves/' folder.

Ensure that the `saves/` directory has public read-write access. On most linux servers this can be achieved using the command `sudo chmod 777 ./saves`.

Make sure the `projectReturn` attribute of the `setup` node is set to the `save.php` script.

Then, just go to the URL of the corresponding HTML file, e.g. `http://server.com/path/to/WAET/index.htm` If storing on the server doesn't work at submission (e.g. if the `projectReturn` attribute isn't properly set or PHP does not have the correct permissions), the result XML file will be presented to the subject on the client side, as a 'Save file' link.

3.4 Load a test / Multiple test documents

By default the index page will load an empty page. To automatically load a test document, you need to append the location in the URL. If your URL is normally http://localhost:8000/index.html you would append the following: `?url=/path/to/your/test.xml`. Replace the fields with your actual path, the path is local to the running directory, so if you have your test in the directory `example_eval` called `project.xml` you would append `?url=/example_eval/project.xml`.

4 Interfaces

The Web Audio Evaluation Tool comes with a number of interface styles, each of which can be customised extensively, either by configuring them differently using the many optional features, or by modifying the JavaScript files.

To set the interface style for the whole test, set the attribute of the `setup` node to `interface="APE"`, where `"APE"` is one of the interface names below.

4.1 Templates

This section describes the different templates available in the Interfaces folder (`./interfaces`),

Blank Use this template to start building your own, custom interface (JavaScript and CSS).

AB Performs a pairwise comparison, but supports n-way comparison (in the example we demonstrate it performing a 7-way comparison).

ABX Like AB, but with an unknown sample X which has to be identified as being either A or B.

APE The APE interface is based on [3], and consists of one or more axes, each corresponding with an attribute to be rated, on which markers are placed. As such, it is a multiple stimulus interface where (for each dimension or attribute) all elements are on one axis so that they can be maximally compared against each other, as opposed to rated individually or with regards to a single reference. It also contains an optional text box for each element, to allow for clarification by the subject, tagging, and so on.

Discrete Each audio element is given a discrete set of values based on the number of slider options specified. For instance, Likert specifies 5 values and therefore each audio element must be one of those 5 values.

Horizontal sliders Creates the same interfaces as MUSHRA except the sliders are horizontal, not vertical.

MUSHRA This is a straightforward implementation of [4], especially common for the rating of audio quality, for instance for the evaluation of audio codecs. This can also operate any vertical slider style test and does not necessarily have to match the MUSHRA specification.

4.2 Examples

Below are a number of established interface types, which are all supported using the templates from the previous section. From [2].

- AB Test / Pairwise comparison [5,6]: Two stimuli presented simultaneously, participant selects a preferred stimulus.

- ABC/HR (ITU-R BS. 1116) [7] (Mean Opinion Score: MOS): each stimulus has a continuous scale (5-1), labeled as Imperceptible, Perceptible but not annoying, Slightly annoying, Annoying, Very annoying.

- -50 to 50 Bipolar with Ref: each stimulus has a continuous scale -50 to 50 with default values as 0 in middle and a reference.

- Absolute Category Rating (ACR) Scale [8]: Likert but labels are Bad, Poor, Fair, Good, Excellent

- ABX Test [9]: Two stimuli are presented along with a reference and the participant has to select a preferred stimulus, often the closest to the reference.

- APE [3]: Multiple stimuli on one or more axes for inter-sample rating.

- Comparison Category Rating (CCR) Scale [8]: ACR & DCR but 7 point scale, with reference: Much better, Better, Slightly better, About the same, Slightly worse, Worse, Much worse.

- Degredation Category Rating (DCR) Scale [8]: ABC & Likert but labels are (5) Inaudible, (4) Audible but not annoying, (3) Slightly annoying, (2) Annoying, (1) Very annoying.

- ITU-R 5 Point Continuous Impairment Scale [10]: Same as ABC/HR but with a reference.

- Likert scale [11]: each stimulus has a five point scale with values: Strongly agree, Agree, Neutral, Disagree and Strongly disagree.

- MUSHRA (ITU-R BS. 1534) [12] Multiple stimuli are presented and rated on a continuous scale, which includes a reference, hidden reference and hidden anchors.

- Pairwise Comparison (Better/Worse) [6]: every stimulus is rated as being either better or worse than the reference.

- Rank Scale [13]: stimuli ranked on single horizontal scale, where they are ordered in preference order.

- 9 Point Hedonic Category Rating Scale [14]: each stimulus has a seven point scale with values: Like extremely, Like very much, Like moderate, Like slightly, Neither like

nor dislike, Dislike extremely, Dislike very much, Dislike moderate, Dislike slightly. There is also a provided reference.

4.3 Building your own interface

4.3.1 Nodes to familiarise

Core.js handles several very important nodes which you should become familiar with. The first is the Audio Engine, initialised and stored in variable 'AudioEngineContext'. This handles the playback of the web audio nodes as well as storing the 'AudioObjects'. The 'AudioObjects' are custom nodes which hold the audio fragments for playback. These nodes also have a link to two interface objects, the comment box if enabled and the interface providing the ranking. On creation of an 'AudioObject' the interface link will be nulled, it is up to the interface to link these correctly.

The specification document will be decoded and parsed into an object called 'specification'. This will hold all of the specifications various nodes. The test pages and any pre/post test objects are processed by a test state which will proceed through the test when called to by the interface. Any checks (such as playback or movement checks) are to be completed by the interface before instructing the test state to proceed. The test state will call the interface on each page load with the page specification node.

4.3.2 Modifying `core.js`

Whilst there is very little code actually needed, you do need to instruct core.js to load your interface file when called for from a specification node. There is a function called 'loadProjectSpecCallback' which handles the decoding of the specification and setting any external items (such as metric collection). At the very end of this function there is an if statement, add to this list with your interface string to link to the source. There is an example in there for both the APE and MUSHRA tests already included. Note: Any updates to core.js in future work will most likely overwrite your changes to this file, so remember to check your interface is still here after any update that interferes with core.js. Any further files can be loaded here as well, such as css styling files. jQuery is already included.

4.3.3 Building the Interface

Your interface file will get loaded automatically when the 'interface' attribute of the setup node matches the string in the 'loadProjectSpecCallback' function. The following functions

17

must be defined in your interface file. A template file is provided in `interfaces\blank.`
`js`.

- **loadInterface** - Called once when the document is parsed. This creates any necessary bindings, such as to the metric collection classes and any check commands. Here you can also start the structure for your test such as placing in any common nodes (such as the title and empty divs to drop content into later).

- **loadTest(audioHolderObject)** - Called for each page load. The audioHolderObject contains a specification node holding effectively one of the audioHolder nodes.

- **resizeWindow(event)** - Handle for any window resizing. Simply scale your interface accordingly. This function must be here, but can me an empty function call.

loadInterface

This function is called by the interface once the document has been parsed since some browsers may parse files asynchronously. The best method is simply to put 'loadInterface()' at the top of your interface file, therefore when the JavaScript engine is ready the function is called.

By default the HTML file has an element with id "topLevelBody" where you can build your interface. Make sure you blank the contents of that object. This function is the perfect time to build any fixed items, such as the page title, session titles, interface buttons (Start, Stop, Submit) and any holding and structure elements for later on.

At the end of the function, insert these two function calls: testState.initialise() and testState.advanceState();. This will actually begin the test sequence, including the pre-test options (if any are included in the specification document).

loadTest(audioHolderObject)

This function is called on each new test page. It is this functions job to clear out the previous test and set up the new page. Use the function audioEngineContext.newTestPage(); to instruct the audio engine to prepare for a new page. "audioEngineContext.audioObjects = [];" will delete any audioObjects, interfaceContext.deleteCommentBoxes(); will delete any comment boxes and interfaceContext.deleteCommentQuestions(); will delete any extra comment boxes specified by commentQuestion nodes.

This function will need to instruct the audio engine to build each fragment. Just passing the constructor each element from the audioHolderObject will build the track, audioEngineContext.newTrack(element) (where element is the audioHolderObject audio element). This will return a reference to the constructed audioObject. Decoding of the audio will happen asynchronously.

You also need to link audioObject.interfaceDOM with your interface object for that audioObject. The interfaceDOM object has a few default methods. Firstly it must start

disabled and become enabled once the audioObject has decoded the audio (function call: enable()). Next it must have a function exportXMLDOM(), this will return the xml node for your interface, however the default is for it to return a value node, with textContent equal to the normalised value. You can perform other functions, but our scripts may not work if something different is specified (as it will breach our results specifications). Finally it must also have a method getValue, which returns the normalised value.

It is also the job the interfaceDOM to call any metric collection functions necessary, however some functions may be better placed outside (for example, the APE interface uses drag and drop, therefore the best way was to call the metric functions from the dragEnd function, which is called when the interface object is dropped). Metrics based upon listening are handled by the audioObject. The interfaceDOM object must manage any movement metrics. For a list of valid metrics and their behaviours, look at the project specification document included in the repository/docs location. The same goes for any checks required when pressing the submit button, or any other method to proceed the test state.

5   Project XML

Each test is defined by its project XML file, examples of these can be seen in the ./tests/examplesl/ directory.

In the XML there are several nodes which must be defined:

- `<waet>`: The root node.
- `<setup>`: The first child node, defines whole-test parameters
- `<page>`: Specifies a test page, attached *after* the `<setup>` nodes.
- `<audioelement>`: Specifies an audio element.

The test uses XML validation, so the ordering of nodes is important to pass this validation. Some nodes also have specific attributes which must be set and may even have a certain format to apply them. This is done so error checking can be performed to catch easy to find errors before loading and running a test session. If your project XML fails this validation, all the errors will be listed.

Before identifying any features, this part will walk you through the available nodes, their function and their attributes.

5.1   Root

The root node is `<waet>`, it must have the following attributes:

`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`

`xsi:noNamespaceSchemaLocation="test-schema.xsd"`.

This will ensure it is checked against the XML schema for validation.

5.2   Set up

The first child node, `<setup>` specifies any one time and global parameters. It takes the following attributes:

- `interface`: String, mandatory, specifies the interface to load
- `projectReturn`: URL, mandatory, specifies the return point. Can be a 3rd party server or the local server. Set to null to disable automatic saving. Specifying "save.php" will trigger the return if either the PHP or python servers are used. On error, it will always default to presenting the save on page.

- **randomiseOrder**: Boolean, optional, if true it will randomise the order of the test pages. Default is false.

- **poolSize**: non-negative integer, optional. Specifies the number of test pages to actually test with. Combined with randomiseOrder being true will give a random set of test pages per participant from the given pool of `<page>` nodes. Specifying 0 disables this option, default is 0.

- **loudness**: non-positive integer, optional. Set the default LUFS target value. See Section 6.10 for more.

- **sampleRate**: positive integer, optional. If set, the sample rate reported by the Web Audio API must match this number. See Section 6.4.

- **calibration**: boolean, optional. If true, a simple hearing test is presented to user to gather system frequency response (DAC, listening device and subject hearing). Default is false.

- **crossFade**: decimal greater than or equal to 0.0, optional. Define the crossFade between fragments when clicked in seconds. Default is 0.0s.

- **preSilence**: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the test at the beginning of the buffer in seconds. Default is 0.0s

- **postSilence**: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the test at the end of the buffer in seconds. Default is 0.0s

The `<setup>` node takes the following child nodes, note these must appear in this order:

- **`<survey>`**: Min of 0, max of 2 occurences. See Section 5.4

- **`<metric>`**: Must appear only once.

- **`<interface>`**: Must appear only once.

5.3 Page

The only other first level child nodes, these specify the test pages. It takes the following attributes:

- **id**: ID, mandatory. A string which must be unique across the entire XML. It is used to identify the page on test completion as pages are returned in the results in the order they appeared, not specified.

- `hostURL`: URL, mandatory. Used in conjuction with the `<audioelement>` url to specify where the audio files are located. For instance if all your files are in the directory `./test/` you can set this attribute to "/test/" and the `<audioelement>` url attribute only needs to file name. Set to "" if no hostURL prefix desired.

- `randomiseOrder`: Boolean, optional. If true the audio fragments are presented randomly rather than the order specified. See Section 6.2. Default is false.

- `repeatCount`: non-negative integer, optional. Specify the number of times to repeat the test page (re-present). Each presentation will appear as an individual page in the results. Default is 0.

- `loop`: Boolean, optional. If true, the audio elements will loop synchronously with each other. See 6.3. Default is false.

- `showElementComments`: Boolean, optional. If true then there will be a comment box on the test page for each audio element presented, see Section 6.11.

- `loudness`: non-positive integer, optional. Set the LUFS target value for this page. Supersedes the `<setup>` loudness attribute for this page. See Section 6.10 for more.

- `label`: enumeration, optional. Set the label to one of the following

    - `default`: The default by the interface (Default if undefined)

    - `none`: Show no labels

    - `number`: Show natural numbers starting at index 1

    - `letter`: Show letters starting at 'a'

    - `capital`: Show letters starting at 'A'

- `poolSize`: non-negative integer, optional. Determine the number of `<audioelement>` nodes to take from those defined. For instance if `poolSize=3` and there are 4 audio elements, only 3 will actually be loaded and presented to the user.

- `alwaysInclude`: boolean, optional. If the parent `<setup>` node has poolSize set, you can enforce the page to always be selected by setting alwaysInclude to true. Default is false

- `preSilence`: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the page at the beginning of the buffer in seconds. Supercedes any value set in `<setup>`. Default is 0.0s

- `postSilence`: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the test at the end of the buffer in seconds. Supercedes any value set in `<setup>`. Default is 0.0s

The `<page>` node takes the following child, nodes note these must appear in this order:

- `<title>`: Appear once or not at all. The text content of this node specifies the title of the test page, for instance `<title>John Doe's Test</title>`

- `<commentboxprefix`: Appear once or not at all. The text content specifies the prefix of the comment boxes, see 6.11.

- `<interface>`: Must appear only once.

- `<audioelement>`: Minimum of one. Specifies an audio element, see Section 5.6.

- `<commentquestion>`: Min of 0, max unlimited occurences. See Section 6.11.

- `<survey>`: Min of 0, max of 2 occurences. See Section 5.4

5.4 Survey

These specify any survey items to be presented. The must be a maximum of two of these per `<setup>` and `<page>` nodes. These have one attribute, location, which must be set to one of the following: before, pre, after or post. In this case before == pre and after == post. This specifies where the survey must appear before or after the node it is associated with. When a child of `<setup>` then pre/before will be shown before the first test page and after/post shown after completing the last test page. When a child of `<page>` then pre/before is before the test commences and after/post is once the test has been submitted.

The survey node takes as its only set of childs the `<surveyentry>` node of which there can be any number.

5.4.1 Survey Entry

These nodes have the following attributes, which vary depending on the survey type wanted:

- `id`: ID, mandatory. Must be unique across the entire XML, used to identify the response in the results.

- `type`: String, mandatory. Must be one of the following: statement, question, checkbox, radio or number. This defines the type to show.

- `mandatory`: Boolean, optional. Defines if the survey must have a response or not. Does not apply to statements. Default is false.

- **min**: Number, optional. Only applies when `type="number"`, the minimum valid response.

- **max**: Number, optional. Only applies when `type="number"`, the maximum valid response.

- **boxsize**: String, optional. Only applies when `type="question"` and must be one of the following: normal (default), small, large or huge.

The nodes have the following children, which vary depending on the survey type wanted.

- **<statement>**: Must appear only once. Its text content specifies the text to appear as the statement or question for the user to respond to.

- **<option>**: Only valid if the parent node has the attribute `type` set to checkbox or radio. Has attribute `name` to identify the selected option in the results. The text content is the text to show next to the radio/checkbox.

5.5 Interface

This node specifies any interface specific options and test parameters. It has an optional `name` attribute used to set the axis name (where applicable), such as the multi-axis APE interface. Specifying multiple interface nodes in a `<page>` node will trigger multiple axis where applicable, otherwise only the *first node* will be used and the rest ignored.

The node has the following children, note the order these must appear in is as follows:

- **title**: Min 0, max 1 occurence. The text content specifies the name of the axis as shown to the user.

- **interfaceoption**: Min 0, max unbounded. Specifies the interface options. See Section 6.1.

- **scales**: Min 0, max 1 occurence. Contains `<scalelabel>` nodes which define the displayed scales. See Section **??**.

5.6 Audio Element

Appear as children of the `page` node. Each of these specify an individual interface fragment to display. Multiple fragments can reference the same file (allowing for repetition with different parameters or blind-doubles). The node has the following attributes:

- **id**: ID, mandatory. Must be unique across the test page. Used to identify the specific fragment in the results.

- **name**: String, optional. If you wish to group fragment across pages when performing result analysis, set the group name here.

- **url**: URL, mandatory. Used with the parent `page` nodes' `hostURL` attribute to get the full url of the audio file to load.

- **gain**: Float, optional. Specify the gain in decibels to apply to the node after loudness normalisation. Default is 0.

- **type**: String, optional. Must be one of the following: normal (default when not specified), anchor, reference or outside-reference. Normal, anchor and reference are presented as normal, outside-reference presents the node as a separate interface option.

- **marker**: Integer between 0 and 100, optional. Only used when `type="anchor"|"reference"`. See Section 6.6.

- **loudness**: Set the loudness of this element in LUFS. Supercedes all other set values. See Section 6.10.

- **alwaysInclude**: boolean, optional. If the parent `<page>` node has poolSize set, you can enforce the element to always be selected by setting alwaysInclude to true. Default is false

- **preSilence**: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the page at the beginning of the buffer in seconds. Supercedes any other value. Default is 0.0s

- **postSilence**: decimal greater than or equal to 0.0, optional. Add a portion of silence to all elements in the test at the end of the buffer in seconds. Supercedes any other value. Default is 0.0s

6   Features

This section covers the different features implemented in the Web Audio Evaluation Tool, how to use them, and what to know about them.

Unless otherwise specified, *each* feature described here is optional, i.e. it can be enabled or disabled and adjusted to some extent.

As the example project showcases (nearly) all of these features, please refer to its configuration XML document for a demonstration of how to enable and adjust them.

6.1   Interface options

The interface node has children of interface options which are used to specify modifications to the test environment. These are divided into two catagories: check and show. Check are used to specify conditions which must be met before a page can be completed, these include checking all fragments have been played or checking all fragments have a comment and so on. Show is used to show an optional on page element or control, such as the playhead or master volume.

Check items have the attribute "type" set to "check". The following list gives the string to give the "name" attribute along with a description of the check.

- `fragmentPlayed`: Checks that all fragments have been at least partially played

- `fragmentFullPlayback`: Checks that all fragments have been fully played. *NOTE:* This will always clear if the page is looping as it is not possible to know every sample has been played.

- `fragmentMoved`: Checks that all fragments have been moved. This is interface dependent, for instance on AB this will always clear as there is no movement.

- `fragmentComments`: Cheks that all fragments have a comment. Will clear if there are no on page comments but with a console warning.

- `scalerange`: Has two extra attributes "min" and "max". Checks that at least one element is below the min value and one element is above the max value.

Show items have the attribute "type" set to "show". The following list gives the string to give the "name" attribute along with a description.

- `playhead`: Shows the playhead to the end user indicating where in the file they are currently listening

- `page-count`: Shows the current test page number and the total number of test pages.

- `volume`: Shows a master volume control to the user to manipulate the output gain of the page. This is tracked.

6.1.1 Multiple scales

In the case of multiple rating scales, e.g. when the stimuli are to be rated in terms of attributes 'timbre' and 'spatial impression', multiple interface nodes will have to be added, each specifying the title and annotations.

This is where the `interface`'s `name` attribute is particularly important: use this to retrieve the rating values, comments and metrics associated with the specified interface. If none is given, you can still use the automatically given `interface-id`, which is the interface number starting with 0 and corresponding to the order in which the rating scales appear.

6.2 Randomisation

[WORK IN PROGRESS]

6.2.1 Randomisation of configuration XML files

The python server has a special function to automatically cycle through a list of test pages. Instead of directly requesting an XML, simply setting the url item in the browser URL to `pseudo.xml` will cycle through a list of XMLs. These XMLs must be in the local directory called `./pseudo/`.

6.2.2 Randomsation of page order

The page order randomisation is set by the `<setup>` node attribute `randomise-order`, for example `<setup ...  randomise-order="true">...</setup>` will randomise the test page order. When not set, the default is to **not** randomise the test page order.

6.2.3 Randomisation of axis order

6.2.4 Randomisation of fragment order

The audio fragment randomisation is set by the `<audioholder>` node attribute `randomise-order`, for example `<audioholder ...  randomise-order="true">...</audioholder>` will ran-

domise the test page order. When not set, the default is to **not** randomise the test page order.

6.2.5 Randomisation of initial slider position

By default slider values are randomised on start. The MUSHRA interface supports setting the initial values of all sliders throught the `<audioholder>` attribute `initial-position`. This takes an integer between 0 and 100 to signify the slider position.

6.3 Looping

Looping enables the fragments to loop until stopped by the user. Looping is synchronous so all fragments start at the same time on each loop. Individual test pages can have their playback looped by the `<page>` attribute `loop` with a value of "true" or "false". If the fragments are not of equal length initially, they are padded with zeros so that they are equal length, to enable looping without the fragments going out of sync relative to each other.

Note that fragments cannot be played until all page fragments are loaded when in looped mode, as the engine needs to know the length of each fragment to calculate the padding.

6.4 Sample rate

If you require the test to be conducted at a certain sample rate (i.e. you do not tolerate re-sampling of the elements to correspond with the system's sample rate), add `sampleRate="96000"` - where "96000" can be any support sample rate (in Hz) - so that a warning message is shown alerting the subject that their system's sample rate is different from this enforced sample rate. This is checked immediately after parsing and stops the page loading any other elements if this check has failed.

6.5 Metrics

The `Metric` node, which contains the metrics to be tracked during the complete test, is a child of the `setup` node, and it could look as follows.

```
<Metric>
        <metricEnable>testTimer</metricEnable>
        <metricEnable>elementTimer</metricEnable>
        <metricEnable>elementInitialPosition</metricEnable>
```

```
        <metricEnable>elementTracker</metricEnable>
        <metricEnable>elementFlagListenedTo</metricEnable>
        <metricEnable>elementFlagMoved</metricEnable>
        <metricEnable>elementListenTracker</metricEnable>
</Metric>
```

When in doubt, err on the inclusive side, as one never knows which information is needed in the future. Most of these metrics are necessary for post-processing scripts such as timeline_view_movement.py.

6.5.1 Time test duration

`testTimer`
One per test page. Presents the total test time from the first playback on the test page to the submission of the test page (exculding test time of the pre-/post- test surveys). This is presented in the results as `<metricresult id="testTime"> 8.60299319727892 </metricresult>`. The time is in seconds.

6.5.2 Time fragment playback

`elementTimer`
One per audio fragment per test page. This totals up the entire time the audio fragment has been listened to in this test and presented `<metricresult name="enableElementTimer"> 1.0042630385487428 </metricresult>`. The time is in seconds.

6.5.3 Initial positions

`elementInitialPosition`
One per audio fragment per test page. Tracks the initial position of the sliders, especially relevant when these are randomised. Example result `<metricresult name="elementInitialPosition"> 0.8395522388059702 </metricresult>`.

6.5.4 Track movements

`elementTracker`
One per audio fragment per test page. Tracks the movement of each interface object. Each movement event has the time it occured at and the new value.

6.5.5 Which fragments listened to

`elementFlagListenedTo`
One per audio fragment per test page. Boolean response, set to true if listened to.

6.5.6 Which fragments moved

`elementFlagMoved`
One per audio fragment per test page. Binary check whether or not a the marker corresponding with a particular fragment was moved at all throughout the experiment.

6.5.7 elementListenTracker

`elementListenTracker`
One per audio fragment per test page. Tracks the playback events of each audio element pairing both the time in the test when playback started and when it stopped, it also gives the buffertime positions.

6.6 References and anchors

The audio elements, `<audioelement>` have the attribute `type`, which defaults to normal. Setting this to one of the following will have the following effects.

6.6.1 Outside Reference

Set type to 'outside-reference'. This will place the object in a separate playback element clearly labelled as an outside reference. This is exempt of any movement checks but will still be included in any listening checks.

6.6.2 Hidden reference

Set type to 'reference'. The element will still be randomised as normal (if selected) and presented to the user. However the element will have the 'reference' type in the results to quickly find it. The reference can be forced to be below a value before completing the test page by setting the attribute 'marker' to be a value between 0 and 100 representing the integer value position it must be equal to or above.

6.6.3   Hidden anchor

Set type to 'anchor'. The element will still be randomised as normal (if selected) and presented to the user. However the element will have the 'anchor' type in the results to quickly find it. The anchor can be forced to be below a value before completing the test page by setting the attribute 'marker' to be a value between 0 and 100 representing the integer value position it must be equal to or below.

6.7   Checks

These checks are enabled in the `interface` node, which is a child of the `setup` node.

6.7.1   Playback checks

Enforce playing each sample at least once, for at least a little bit (e.g. this test is satisfied even if you only play a tiny portion of the file), by alerting the user to which samples have not been played upon clicking 'Submit'. When enabled, one cannot proceed to the next page, answer a survey question, or finish the test, before clicking each sample at least once.

Alternatively, one can check whether the *entire* fragment was listened to at least once.

Add `<check name="fragmentPlayed"/>` to the `interface` node.

6.7.2   Movement check

Enforce moving each sample at least once, for at least a little bit (e.g. this test is satisfied even if you only play a tiny portion of the file), by alerting the user to which samples have not been played upon clicking 'Submit'. When enabled, one cannot proceed to the next page, answer a survey question, or finish the test, before clicking each sample at least once. If there are several axes, the warning will specify which samples have to be moved on which axis.

Add `<check name="fragmentMoved"/>` to the `interface` node.

6.7.3   Comment check

Enforce commenting, by alerting the user to which samples have not been commented on upon clicking 'Submit'. When enabled, one cannot proceed to the next page, answer a

survey question, or finish the test, before putting at least one character in each comment box.

Note that this does not apply to any extra (text, radio button, checkbox) elements, unless these have the 'mandatory' option enabled.

Add `<check name="fragmentComments"/>` to the `interface` node.

6.7.4 Scale use check

It is possible to enforce a certain usage of the scale, meaning that at least one slider needs to be below and/or above a certain percentage of the slider.

Add `<check name="scalerange" min="25" max="75"/>` to the `interface` node.

6.7.5 Note on the use of multiple rating axes

I.e. what if more than one axis? How to specify which axis the checks relate to?

6.8 Platform information

For troubleshooting and usage statistics purposes, information about the browser and the operating system is logged in the results XML file. This is especially useful in the case of remote tests, when it is not certain which operating system, browser and/or browser were used. Note that this information is not always available and/or accurate, e.g. when the subject has taken steps to be more anonymous, so it should be treated as a guide only.

Example:

```
<navigator>
        <platform>MacIntel</platform>
        <vendor>Google Inc.</vendor>
        <uagent>Mozilla/5.0 ... </uagent>
        <screen innerHeight="1900px" innerWidth="1920px"/>
</navigator>
```

6.9 Gain

It is possible to set the gain (in decibel) applied to the different audioelements, as an attribute of the `audioelement` nodes in the configuration XML file:

```
<audioElements url="sample-01.wav" gain="-6" id="sample01quieter" />
```
Please note, there are no checks on this to detect if accidentaly typed in linear. This gain is applied *after* any loudness normalisation.

6.10   Loudness

Each audio fragment on loading has its loudness calculated. The tool uses the EBU R 128 recommendation following the ITU-R BS.1770-4 loduness calculations to return the integreated LUFS loudness. The attribute `loudness` will set the loudness from the scope it is applied in. Applying it in the `<setup>` node will set the loudness for all test pages. Applying it in the `<page>` node will set the loudness for that page. Applying it in the `<audioelement>` node will set the loudness for that fragment. The scope is set locally, so if there is a loudness on both the `<page>` and `<setup>` nodes, that test page will take the value associated with the `<page>`. The loudness attribute is set in LUFS

6.11   Comment Boxes

There are two types of comment boxes which can be presented, those linked to the audio fragments on the page and those which pose a general question. The audio fragment boxes are shown by setting the attribute `showElementComments` to true of the page in question. This will then show a comment box below the main interface for every fragment on the page. There is some customisation around the text that accompanies the box, by default the text will read "Comment on fragment" followed by the fragment identifier (the number / letter shown by the interface). This 'prefix' can be modified using the page node `<commentboxprefix>`, see Section 5.3 for where to place this node in the document. The comment box prefix node takes no attribute and the text contained by the node represents to the prefix. For instance if we have a node `<commentboxprefix> Describe fragment </commentboxprefix>`, then the interface will show "Describe fragment" followed by the identifier.

The second type of comment box is slightly more complex because it can handle different types of response data. These are called comment questions because they are located in the comment section of the test but pose a specific question.

7   Using the test create tool

We provide a test creation tool, available in the directory test_create. This tool is a self-contained web page, so doubling clicking will launch the page in your system default browser.

The test creation tool can help you build a simple test very quickly. By simply selecting your interface and clicking check-boxes you can build a test in minutes.

Include audio by dragging and dropping the stimuli you wish to include.

The tool examines your XML before exporting to ensure you do not export an invalid XML structure which would crash the test.

This guide will help you to construct your own interface on top of the WAET (Web Audio Evaluation Tool) engine. The WAET engine resides in the core.js file, this contains prototype objects to handle most of the test creation, operation and data collection. The interface simply has to link into this at the correct points.

8 Analysis and diagnostics

8.1 In the browser

See 'analysis.html' in the main folder: immediate visualisation of (by default) all results in the 'saves/' folder.

8.2 Python scripts

The package includes Python (2.7) scripts (in 'python/') to extract ratings and comments, generate visualisations of ratings and timelines, and produce a fully fledged report.

Visualisation requires the free matplotlib toolbox (http://matplotlib.org), numpy and scipy. By default, the scripts can be run from the 'python' folder, with the result files in the 'saves' folder (the default location where result XMLs are stored). Each script takes the XML file folder as an argument, along with other arguments in some cases. Note: to avoid all kinds of problems, please avoid using spaces in file and folder names (this may work on some systems, but others don't like it).

8.2.1 comment_parser.py

Extracts comments from the output XML files corresponding with the different subjects found in 'saves/'. It creates a folder per 'audioholder'/page it finds, and stores a CSV file with comments for every 'audioelement'/fragment within these respective 'audioholders'/pages. In this CSV file, every line corresponds with a subject/output XML file. Depending on the settings, the first column containing the name of the corresponding XML file can be omitted (for anonymisation). Beware of Excel: sometimes the UTF-8 is not properly imported, leading to problems with special characters in the comments (particularly cumbersome for foreign languages).

8.2.2 evaluation_stats.py

Shows a few statistics of tests in the 'saves/' folder so far, mainly for checking for errors. Shows the number of files that are there, the audioholder IDs that were tested (and how many of each separate ID), the duration of each page, the duration of each complete test, the average duration per page, and the average duration in function of the page number.

8.2.3 generate_report.py

Similar to 'evaluation_stats.py', but generates a PDF report based on the output files in the 'saves/' folder - or any folder specified as command line argument. Uses pdflatex to write a LaTeX document, then convert to a PDF.

8.2.4 score_parser.py

Extracts rating values from the XML to CSV - necessary for running visualisation of ratings. Creates the folder 'saves/ratings/' if not yet created, to which it writes a separate file for every 'audioholder'/page in any of the output XMLs it finds in 'saves/'. Within each file, rows represent different subjects (output XML file names) and columns represent different 'audioelements'/fragments.

8.2.5 score_plot.py

Plots the ratings as stored in the CSVs created by score_parser.py Depending on the settings, it displays and/or saves (in 'saves/ratings/') a boxplot, confidence interval plot, scatter plot, or a combination of the aforementioned. Requires the free matplotlib library. At this point, more than one subjects are needed for this script to work.

8.2.6 timeline_view_movement.py

Creates a timeline for every subject, for every 'audioholder'/page, corresponding with any of the output XML files found in 'saves/'. It shows the marker movements of the different fragments, along with when each fragment was played (red regions). Automatically takes fragment names, rating axis title, rating axis labels, and audioholder name from the XML file (if available).

8.2.7 timeline_view.py

Creates a timeline for every subject, for every 'audioholder'/page, corresponding with any of the output XML files found in 'saves/'. It shows when and for how long the subject listened to each of the fragments.

9 Troubleshooting

9.1 Reporting bugs and requesting features

Thanks to feedback from using the interface in experiments by the authors and others, many bugs have been caught and fatal crashes due to the interface seem to be a thing of the past entirely.

We continually develop this tool to fix issues and implement features useful to us or our user base. See `https://code.soundsoftware.ac.uk/projects/webaudioevaluationtool/issues` for a list of feature requests and bug reports, and their status.

Please contact the authors if you experience any bugs, if you would like additional functionality, if you spot any errors or gaps in the documentation, if you have questions about using the interface, or if you would like to give any feedback (even positive!) about the interface. We look forward to learning how the tool has (not) been useful to you.

9.2 First aid

Meanwhile, if things do go wrong or the test needs to be interrupted for whatever reason, all data is not lost. In a normal scenario, the test needs to be completed until the end (the final 'Submit'), at which point the output XML is stored in the `saves/`. If this stage is not reached, open the JavaScript Console (see below for how to find it) and type

`createProjectSave()`

to present the result XML file on the client side, or

`createProjectSave(specification.projectReturn)`

to try to store it to the specified location, e.g. the 'saves/' folder on the web server or the local machine (on failure the result XML should be presented directly in the web browser instead)

and hit enter. This will open a pop-up window with a hyperlink that reads 'Save File'; click it and an XML file with results until that point should be stored in your download folder.

Alternatively, a lot of data can be read from the same console, in which the tool prints a lot of debug information. Specifically:

- the randomisation of pages and fragments are logged;
- any time a slider is played, its ID and the time stamp (in seconds since the start of the test) are displayed;

37

- any time a slider is dragged and dropped, the location where it is dropped including the time stamp are shown;

- any comments and pre- or post-test questions and their answers are logged as well.

You can select all this and save into a text file, so that none of this data is lost. You may to choose to do this even when a test was successful as an extra precaution.

If you encounter any issue which you believe to be caused by any aspect of the tool, and/or which the documentation does not mention, please do let us know!

Opening the JavaScript Console

- In Google Chrome, the JavaScript Console can be found in **View>Developer>JavaScript Console**, or via the keyboard shortcut Cmd + Alt + J (Mac OS X).

- In Safari, the JavaScript Console can be found in **Develop>Show Error Console**, or via the keyboard shortcut Cmd + Alt + C (Mac OS X). Note that for the Developer menu to be visible, you have to go to Preferences (Cmd + ,) and enable 'Show Develop menu in menu bar' in the 'Advanced' tab. **Note that as long as the Developer menu is not visible, nothing is logged to the console, i.e. you will only be able to see diagnostic information from when you switched on the Developer tools onwards.**

- In Firefox, go to **Tools>Web Developer>Web Console**, or hit Cmd + Alt + K.

9.3   Known issues and limitations

The following is a non-exhaustive list of problems and limitations you may experience using this tool, due to not being supported yet by us, or by the Web Audio API and/or (some) browsers.

- Issue **#1463**: **Firefox** only supports 8 bit and 16 bit WAV files. Pending automatic requantisation (which deteriorates the audio signal's dynamic range to some extent), WAV format stimuli need to adhere to these limitations in order for the test to be compatible with Firefox.

- Issues **#1474** and **#1462**: On occasions, audio is not working - or only a continuous 'beep' can be heard - notably in **Safari**. Refreshing, quitting the browser and even enabling Developer tools in Safari's Preferences pane ('Advanced' tab: "Show 'Develop' menu in menu bar") has helped resolve this. If no (high quality) audio can be heard, make sure your entire playback system's settings are all correct.

10 References

[1] N. Jillings, D. Moffat, B. De Man, and J. D. Reiss, "Web Audio Evaluation Tool: A browser-based listening test environment," in *12th Sound and Music Computing Conference*, July 2015.

[2] 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*, 2016.

[3] B. De Man and J. D. Reiss, "APE: Audio Perceptual Evaluation toolbox for MAT-LAB," in *136th Convention of the Audio Engineering Society*, April 2014.

[4] *Method for the subjective assessment of intermediate quality level of coding systems*. Recommendation ITU-R BS.1534-1, 2003.

[5] S. P. Lipshitz and J. Vanderkooy, "The Great Debate: Subjective evaluation," *Journal of the AES*, vol. 29, no. 7/8, pp. 482–491, 1981.

[6] H. A. David, *The method of paired comparisons*, vol. 12. DTIC Document, 1963.

[7] "Recommendation ITU-R BS. 1116-1: Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems," *International Telecommunication Union*, 1997.

[8] "Recommendation ITU-T P. 800: Methods for subjective determination of transmission quality," *International Telecommunication Union*, 1996.

[9] D. Clark, "High-resolution subjective testing using a double-blind comparator," *Journal of the AES*, vol. 30, no. 5, pp. 330–338, 1982.

[10] "Recommendation ITU-R BS. 562-3: Subjective assessment of sound quality," *International Telecommunication Union*, 1997.

[11] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, 1932.

[12] "Recommendation ITU-R BS.1534-1: Method for the subjective assessment of intermediate quality levels of coding systems," *International Telecommunication Union*, 2003.

[13] G. C. Pascoe and C. C. Attkisson, "The evaluation ranking scale: A new methodology for assessing satisfaction," *Evaluation and program planning*, vol. 6, no. 3, pp. 335–347, 1983.

[14] D. R. Peryam and N. F. Girardot, "Advanced taste-test method," *Food Engineering*, vol. 24, no. 7, pp. 58–61, 1952.

A   Legacy

The APE interface and most of the functionality of the first WAET editions are inspired by the APE toolbox for MATLAB [3]. See `https://code.soundsoftware.ac.uk/projects/ape` for the source code and `http://brechtdeman.com/publications/aes136.pdf` for the corresponding paper.

B Listening test instructions example

Before each test, show the instructions below or similar and make sure it is available to the subject throughout the test. Make sure to ask whether the participant has any questions upon seeing and/or reading the instructions.

- You will be asked for your name ("John Smith") and location (room identifier).

- An interface will appear, where you are asked to

    - click green markers to play the different mixes;

    - drag the markers on a scale to reflect your preference for the mixes;

    - comment on these mixes, using text boxes with corresponding numbers (in your **native language**);

    - optionally comment on all mixes together, or on the song, in 'General comments'.

- You are asked for your personal, honest opinion. Feel free to use the full range of the scale to convey your opinion of the various mixes. Don?t be afraid to be harsh and direct.

- The markers appear at random positions at first (which means some markers may hide behind others).

- The interface can take a few seconds to start playback, but switching between mixes should be instantaneous.

- This is a research experiment, so please forgive us if things go wrong. Let us know immediately and we will fix it or restart the test.

- When the test is finished (after all songs have been evaluated), just call the experimenter, do NOT close the window.

- After the test, please fill out our survey about your background, experience and feedback on the test.

- By participating, you consent to us using all collected data for research. Unless asked explicitly, all data will be anonymised when shared.

C   Terminology

As a guide to better understand the Instructions, and to expand them later, here is a list of terms that may be unclear or ambiguous unless properly defined.

Subject The word we use for a participant, user, ... of the test, i.e. not the experimenter who designs the test but the person who evaluates the audio under test as part of an experiment (or the preparation of one).

User The person who uses the tool to configure, run and analyse the test - i.e. the experimenter, most likely a researcher - or at least

Page A screen in a test

Fragment An element, stimulus or sample in a test; corresponds with an `audioelement`

Test A complete test which can consist of several pages; corresponds with an entire configuration XML file

Configuration XML file The XML file containing the necessary information on interface, samples, survey questions, configurations, ... which the JavaScript modules read to produce the desired test.

Results XML file The output of a successful test, including ratings, comments, survey responses, timing information, and the complete configuration XML file with which the test was generated in the first place.

Contact details

- Nicholas Jillings: `nicholas.jillings@mail.bcu.ac.uk`

- Brecht De Man: `b.deman@qmul.ac.uk`

- David Moffat: `d.j.moffat@qmul.ac.uk`