HTML Web Audio Elements: Easy Interaction with Web Audio API Through HTML

Stephanus Volke
Jade Hochschule
Wilhelmshaven/Oldenburg/
Elsfleth
stephanus.volke@jade-hs.de

Bastian Bechtold
Jade Hochschule
Wilhelmshaven/Oldenburg/
Elsfleth
bastian.bechtold@jade-hs.de

Joerg Bitzer
Jade Hochschule
Wilhelmshaven/Oldenburg/
Elsfleth
joerg.bitzer@jade-hs.de

ABSTRACT

The JavaScript Web Audio API has a powerful but low-level and complicated structure. Therefore, many JavaScript-based wrapper libraries exist, which are intended to simplify its usage. This paper presents a completely new approach, which translates the API into HTML Custom Elements and allows definition, usage and control of complex audio scenarios using only normal HTML elements.

CCS Concepts

• **Software and its engineering** → **Data flow architectures; Frameworks;**

1. INTRODUCTION

JavaScript has a foundational influence on modern web application development. This is in particular due to the many new features and APIs which were added to the language over the last years [4, 9]. One of these features is the Web Audio API that enables production, playback, editing and analyses of audio files in a web browser [1]. Because of its low level and partially complicated interface, there are already several libraries that provide simplified wrappers [5, 7]. However, one crucial obstacle remains: the interaction of user interaction layer written in HTML and the processing layer written in JavaScript. In order to simplify this interaction, a relatively new approach is to represent imperative JavaScript objects as interactive, declarative HTML elements. With HTML Custom Elements, which are currently being standardized, new and valid HTML elements can be defined that allow description and controlling of functionality through attributes with normal HTML syntax [2]. By using this concept, our new HTML Web Audio Elements enables purely declarative generation and control of complex audio processing within pure HTML markup and without additional JavaScript interaction.

2. HTML WEB AUDIO ELEMENTS

The main idea of HTML Web Audio Elements is to simplify the interaction between the Web Audio API-driven, JavaScript-controlled audio processing layer and the corresponding HTML-based user interaction layer in an easy and clean way. To achieve this, the HTML Web Audio Elements are divided into two main groups: non-visual elements, which define the functional behaviour and visual elements that expose this functionality to the user. Source code and working usage examples are available at the project homepage www.pub.tgm.io/webaudio-elements.

2.1 Non-Visible Elements

The non-visible elements are intended to translate the Web Audio API core nodes into HTML DOM nodes, which keep the general API's audio graph structure unchanged. JavaScript properties are represented as HTML attributes and methods as attribute changes. This structure allows communication between different elements by listening for attribute changes via mutation observers [8]. A usage example is given in Figure 1.

Loading Audio Data

The most basic element is the `<webaudio-context>`. Using this element automatically instantiates a Web Audio API AudioContext, which all other audio-elements rely on. It should be used as the parent for all associated elements.

Audio data can be loaded and decoded by specifying a `<webaudio-buffer>` element. Its read-only `state` attribute reflects the current status: *waiting*, *downloading*, *decoding*, *ready* or *error*. Once audio data has been loaded, it can not be changed anymore. Instead, a new element must be created.

Playback

Playback is managed by the `<webaudio-source>` element, which holds a reference to the corresponding `<webaudio-buffer>` through its `buffer-id` attribute. Changing the `state` attribute to one of *playing*, *paused* or *stopped* controls playback. At the same time, the current audio position is available in the `pos` attribute. Creating more than one source element referencing the same buffer is possible and provides features such as simultaneously playing the same audio material at different positions.

Audio Processing and Routing

The elements discussed so far only provide functionality for handling, but not for processing audio data. For filtering, gain control etc., processing elements exist, which can be used either in a global context or local source mode. The

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

```
<body>
  <webaudio-context id="ctx">
    <webaudio-buffer id="buffer1"
                     context-id="ctx"
                     src="/url/to/file.wav"
                     state="ready">
    </webaudio-buffer>
    <webaudio-buffer id="buffer2"
                     context-id="ctx"
                     src="/url/to/another/file.wav"
                     state="decoding">
    </webaudio-buffer>
    <webaudio-source id="src1"
                     buffer-id="buffer1"
                     state="playing"
                     pos="23.24">
      <webaudio-biquad freq="2000"
                       value="12"
                       type="lowpass"
                       disabled>
      </webaudio-biquad>
    </webaudio-source>
    <webaudio-source id="src2"
                     buffer-id="buffer2"
                     state="waiting">
      <webaudio-gain src-id="src"
                     scale="lin"
                     value="0.8">
      </webaudio-gain>
    </webaudio-source>
    <webaudio-gain src-id="src"
                   scale="lin"
                   value="0.8">
    </webaudio-gain>
  </webaudio-context>
  <button class="is-blaybtn" data-source="src1"
          data-state="playing">Pause</button>
  <button class="is-playbtn" data-source="src2"
          data-state="waiting" disabled>Play</button>
</body>
```

Figure 1: Example audio setup with two audio files. The first buffer element with id=”buffer1” is ready for playback, the second one is currently decoding. Therefore, the source element with id=”src2” is in waiting state and the corresponding *play* button disabled. Source element src1 is in playing state and its playback position gets continuously updated in the pos attribute. Filtering of source src1 is bypassed because of the disabled attribute of the <webaudio-biquad>, but the global gain element is active.

latter means that only data of one single source element gets processed, while the former takes all predefined sources and uses their sum. Thereby different sources can be processed independently. This behavior can be defined by appropriate element arrangement: Children of <webaudio-source> elements only affect their parents while neighbor elements get processed in the order they are defined. Additionally, each element has a disabled attribute which can temporarily remove itself or its children from audio processing.

2.2 Visible Elements

In terms of the current W3C HTML Custom Element working draft, the controlling elements are Extended Built-in HTML elements [2], which were expanded by the functionality required to control the functional elements. Because of Apple’s announcement to not support this feature [6], the current implementation relies on a class-based approach, in which elements with a specific class get additional features. The <button> elements in Figure 1, for example get extra data attributes for the playback status of their corresponding <webaudio-source> elements.

3. BROWSER SUPPORT

Even though both the Web Audio API and especially HTML Custom Elements offer completely new implementation approaches, their underlying specifications are neither standardized nor fully implemented in any browsers to date. Therefore, the current HTML Web Audio Elements implementation is a trade-off between proposed specifications, browser vendor announcements and available implementations. Currently, only Chromium-based browsers meet all requirements for non-polyfilled, dependency-free execution. However, only features are used which have nearly passed the standardization process and whose realization have already been started by all major browser vendors.

4. CONCLUSIONS

The HTML Web Audio Audio Elements offer a completely new way of creating audio-related web applications. Because of their standardized HTML implementation, all necessary steps to load, play and even process audio data can be performed without writing a single line of JavaScript code. The declarative and hierarchical structure is easy to read and to understand, which makes them accessible to less-experienced programmers. However for widespread usage, browser support is essential. It is expected that community-driven development will lead to many more freely available processing elements.

5. REFERENCES

[1] Paul Adenot and Chris Wilson. Web Audio API. W3C Working Draft 08 December 2015, W3C, December 2015. https://www.w3.org/TR/webaudio/.

[2] Domenic Denicola. Custom Elements. W3C Workin Draft, W3C, October 2016. https://www.w3.org/TR/2016/WD-custom-elements-20161013/.

[3] Ecma International. ECMAScript 2015 Language Specification. Standard ECMA-262, Ecma International, June 2015. http://www.ecma-international.org/ecma-262/6.0/.

[4] Alejandro Mantecon Guillen. pizzicato.js. https://alemangui.github.io/pizzicato/, March 2017. Accessed: 2017-03-28.

[5] Ryosuke Niwa. The is attribute is confusing? Maybe we should encourage only ES6 class-based extension. https://github.com/w3c/webcomponents/issues/509#issuecomment-222860736, June 2016. Accessed: 2017-03-27.

[6] James Simpson. Howler.js. https://howlerjs.com/, March 2017. Accessed: 2017-03-28.

[7] Anne van Kesteren, Aryeh Gregor, Alex Russell, and Robin Berjon. W3C DOM4. W3C Recommendation, W3C, November 2015. https://www.w3.org/TR/dom/.

[8] W3C. JAVASCRIPT APIS CURRENT STATUS. https://www.w3.org/standards/techs/js, March 2017. Accessed: 2017-03-27.