Proposal: AudioDeviceClient API (tentative title)

Author: hongchan@chromium.org

Last updated: 10/28/2018
Status: Draft

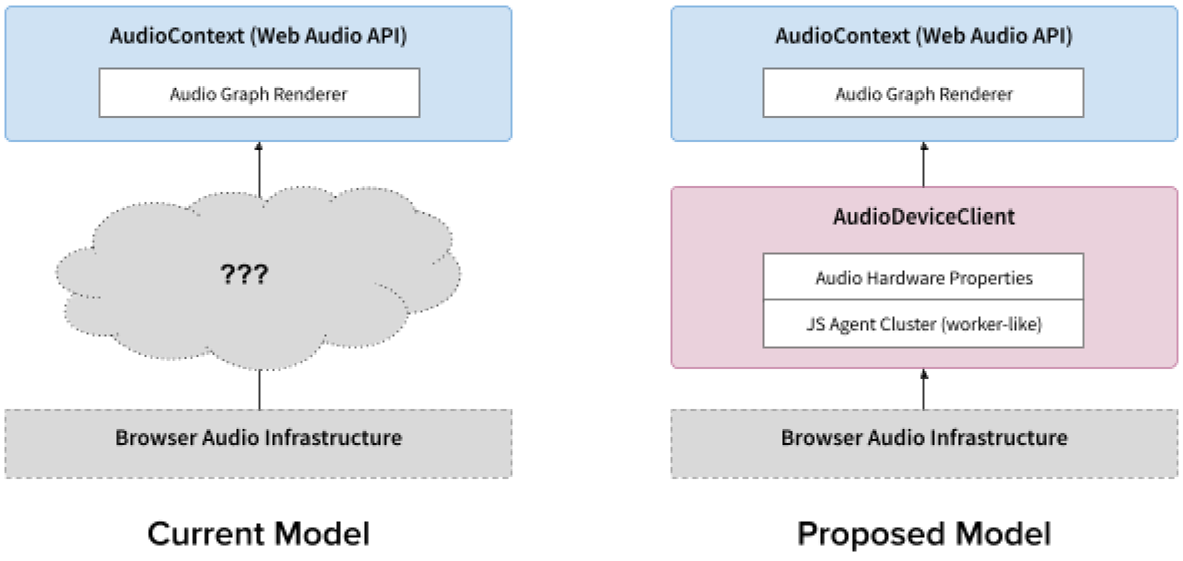
Rationale

Web Audio API V1 lacks of low-level functionality for serious audio processing. Although the Audio Worklet interface paved a path to the low-level audio programming, but it is still confined by the boundary of Web Audio API graph mechanism.

As demonstrated in this article, the combination of Audio Worklet, SharedArrayBuffer and Web Assembly on Worker thread becomes an affordable path for developers who have existing native applications. However, this is a complicated set up for a simple task from the perspective of conventional audio programming.

To facilitate this trend by reducing the complexity of such setup while maintaining backward compatibility toward Web Audio API V1, the newly designed AudioDeviceClient interface is proposed here.

The new API will be an intermediate layer between Web Audio API and an actual audio device used by the browser. It exposes various low-level functionalities such as variable callback buffer size, implicit resampling and device selection, which have been completely hidden from developers.


AudioContext (Web Audio API)
Audio Graph Renderer
???
Browser Audio Infrastructure
Current Model
AudioContext (Web Audio API)
Audio Graph Renderer
AudioDeviceClient
Audio Hardware Properties
JS Agent Cluster (worker-like)
Browser Audio Infrastructure
Proposed Model

Key Features

- Provides a dedicated global scope on a seperate thread (high-priority when permitted) for audio processing purpose in JavaScript.
    - Blink Architecture team and Web Audio team decided not to use the real-time priority thread for Audio Worklet operation due to security reasons. However, with this simplified API design we can detect a malicious loop in the callback function and prevent the system from hang. This mechanism was difficult to enforce on Web Audio API because it is too dynamic.
- Audio device selection via MediaTrackConstraints pattern.
    - We can piggyback a proven solution to workaround the fingerprinting issue.
- Variable callback buffer size. (as opposed to 128-frames for Web Audio API)
    - The fixed buffer size has been an "unsolvable" issue for Web Audio API V1.
- Implicit sample rate conversion when a given rate is different from the current hardware rate
    - Resampling is also supported by AudioContext, but the feature is natural to have in this intermediate layer.
- Direct access to inputs and outputs of an audio hardware within a callback function
    - Asymmetric API shape has been an issue; getUserMedia() for the microphone and Audio Output Devices API for the output device. This inherently means the disconnection between input and output - which leads to problems like

    synchronization and clock drift. Having a single callback for both input and output is much cleaner and easier to work with.

- Being able to avoid unnecessary overhead by opting out AudioContext
    - A lot of legacy audio-oriented projects cannot be easily ported to Web Audio API (The cost of rewriting is simply too high) and a simple callback-based API is significantly affordable to them.
    - Web Audio API's rendering system can be redundant for some use cases.
- Can report "render capability" so the application can dynamically adjust the load based on the system performance.
    - Web Audio API V1 doesn't have a mechanism to handle buffer underruns at all. With this intermediate layer, it is possible to give the control back to developer so they can adjust the application based on rendering capability.

Restrictions

- The AudioDeviceClient interface is activated only via HTTPS.
    - New feature should be on secure context by default.
- 1:1 mapping between an AudioDeviceClient and a corresponding AudioDeviceClientGlobalScope
    - Because AudioDeviceClientGlobalScope can run the WASM code with thread support, multi-threading is not blocked by this.
- The AudioDeviceClient does not autoplay by default.
- An AudioClient represent a single audio hardware; device aggregation is not supported.
    - **TBD**
- The bit resolution of the audio stream is fixed to 32-bit float.
    - It is unrealistic to have something else for the audio path than 32-bit float.
- The callback buffer size must be greater than 128-frames and will be rounded to the nearest multiple of 128.
    - **TBD**

Use cases

- Porting existing audio engines to the web platform with the minimum engineering cost (e.g. Wwise, FMOD, Soundation, Audiotools or retro game audio)
- A teleconference app with custom audio processing and direct hardware access (e.g. complex echo cancellation, audio spatialization, and auditory scene analysis/information retrieval)
- A hybrid audio application that uses Web Audio API's graph system (processing and synthesis) **and** customized hardware configuration (resampling, variable buffer size, multichannel input/output, or master effect DSP)

Tentative WebIDL

```
dictionary AudioDeviceClientConstraints {
  USVString deviceId;
  optional float sampleRate;
  unsigned callbackBufferSize = 128;
  unsigned inputChannelCount = 0;
  unsigned outputChannelCount = 2;
}

enum AudioDeviceClientState {
  "pending",
  "running",
  "stopped"
}

[Exposed=Window,
 SecureContext,
 NoConstructor]
interface AudioDeviceClient : EventTarget {
  readonly attribute USVString deviceId;
  readonly attribute float sampleRate;
  readonly attribute unsigned callbackBufferSize;
  readonly attribute unsigned inputChannelCount;
  readonly attribute unsigned outputChannelCount;
  readonly attribute AudioDeviceClientState state;
  readonly attribute MessagePort port;
  attribute EventHandler onstatechange;
  Promise<void> start();
  Promise<void> stop();
  [SameObject] AudioContext getContext();
}

[Exposed=Window]
partial interface MediaDevices : EventTarget {
  Promise<AudioDeviceClient> getAudioClient();
}

callback AudioDeviceCallback = void (sequence<Float32Array> input,
                                     sequence<Float32Array> output,
                                     AudioContextCallback callback);

[Exposed=AudioDeviceClient]
interface AudioDeviceClientGlobalScope {
  void postMessage(any message, sequence<object> transfer);
  void close();
  void setDeviceCallback(AudioDeviceCallback callback);
  attribute readonly DOMHighResTimeStamp now;
  readonly attribute float renderCapability;
  attribute EventHandler onerror;
  attribute EventHandler onmessage;
  attribute EventHandler onmessageerror;
}
```

Code Examples

Window Global Scope side

```
/* global async */

const devices = await navigator.mediaDevices.enumerateDevices();
const myAudioDeviceId = extractAudioDeviceId(devices);

const constraints = {
  deviceId: myAudioDeviceId,
  sampleRate: 8000,
  callbackBufferSize: 512,
  inputChannelCount: 2,
  outputChannelCount: 6,
};

// If the requested constraints is not accepted, this will be rejected
// with an error.
const client = await navigator.mediaDevices.getAudioClient(constraints);
await client.start('my-client.js');

// A client can instantiate an AudioContext.
const audioContext = client.getContext();
const oscillator = new OscillatorNode(audioContext);
oscillator.connect(audioContext.destination);
oscillator.start();
oscillator.stop(5);
oscillator.onended = () => { client.stop(); };
```

AudioDeviceClientGlobalScope side

```
import Module from './simple-wasm-kernel.js';
```

```
const kernel = new Module.SimpleKernel();

const inputBuffer = Module.getHeapAddress(bufferSize);
const outputBuffer = Module.getHeapAddress(bufferSize);

const process = (input, output, audioContextCallback) => {
  // |input| will be routed to Web Audio API's |context.source| node.
  const audioContextBuffer = audioContextCallback(input);

  // WASM kernel here works as a final DSP of AudioContext. It processes
  // the rendered buffer from AudioContext and put the result back to
  // |output| buffer.
  kernel.process(renderedBuffer, output);
};

setDeviceIOMemory(inputBuffer, outputBuffer);
setDeviceCallback(process);
```

- WG discussed several options on how to incorporate the audio data rendered by Web Audio API graph:
  - Provide another function argument, which is a rendered buffer from the graph.
  - Provide a render callback function itself as an argument, so user can explicitly call the web audio render function inside of device callback function.

Design Issues

1. Handshaking process between AudioClient and audio device
   a. Research what native AudioIO APIs do for handshaking; in most cases they simply fails when parameters are not supported. Being able to query the supported parameters does not work for the web due to the fingerprinting concern.
   b. Research WebRTC's MediaTrackConstraints and enumerateDevices().
2. Supporting multiple devices through a single AudioClient instance (aka Device Aggregation in MacOS)
   a. We already need two different callbacks for input (source) and output (sink).
   b. FireFox does "re-clock" to align two different audio streams.
3. The association between AudioClient and AudioContext.
   a. Chrome's position is 1:1 mapping between AudioClient and AudioContext; meaning getContext() method always returns a same object.
4. How to render the Web Audio API graph within AudioClient's callback function
   a. How to reconcile buffer size difference between AudioClient and AudioContext, especially when the buffer size is not multiple of 128 frames.
5. How to handle buffer underrun or blocking operation in ADCGlobalScope.
   a. How to report glitch or performance issues to developers.

6. Better WASM integration.

TODO

- Submit this proposal to W3C Audio Community Group for the review.