Device Output API Proposal Outline

Introduction

The approach outlined here is essentially a reworking and extension of the Audio Output API draft found here:

http://w3c.github.io/mediacapture-output/

When looking at use cases from Web Audio as well as MSE/EME applications, it became clear that important features were missing from the current draft. Instead of relying on DOMStrings to represent output devices, a device abstraction interface is introduced along with APIs for device access and discovery that parallel the input device APIs already present in Media Capture.

Significant Use Cases

1. Selection of audio output devices that are physically paired with input devices (e.g. headset phones that go with headset mic)

2. Selection of the audio device to the one that best fits the nature of an application's output in terms of channel layout, sample rate, latency, etc.

3. Discovery of output capabilities in order to fetch the media rendition that will look/sound best on user's device. For instance, a movie may have a stereo mix and a 5.1 mix - if the user's system is configured for stereo, there is no value to taking the surround mix and doing a (typically non-optimal) downmix on the end device.

4. Affording the user of any audio application a straightforward choice of audio output device that best fits their needs of that moment

5. Extending the discovery, description and selection of output devices to cover video as well as audio.

6. Programmatically manipulating output devices, e.g. adjusting master volume or muting.

In particular case #4 requires some way to not only negotiate capabilities via constraints, but also to enumerate devices.

Note that there are lots of useful capabilities that we'd like to expose for output devices (latency, min/max sample rate, min/max channels, etc.). They aren't listed here because we're trying to establish an acceptable framework for working with output devices first.

Rationale

This proposal addresses several gaps in current Media Capture proposals.

First, though, let’s note that the Audio Output API is based on DOMString sinkIDs which, apart from a limited number of predefined identifiers, can only be obtained as device IDs from enumerateDevices(). Most of the gaps revolve around this fact in one way or another.

- The information provided by enumerateDevices() for output devices provides little useful semantic information. Output devices may be listed and their grouping with input devices may be known, but currently no output capabilities are exposed.

- It is not possible to obtain a the ID of an output device that satisfies specific constraints, e.g. has some minimum number of channels or sample rate.

- Due to the lack of an output device abstraction interface, it is not possible to programmatically act on devices.

- The Audio Output API is audio-only and does not treat video output. It seems more general to deal with output in a generic fashion, much as the Media Capture API does for input.

- The permissions grant model for output devices defined by the Audio Output API is only defined as a side effect of obtaining grants for input devices. However many audio applications do not make use of input devices.

Draft API Outline

MediaOutputDevice Interface

A new MediaOutputDevice interface will act as an output device abstraction and can generally be used as a sink, similarly to the way that sinkIDs were proposed to be used in the Audio Output API proposal. (Some of its signature should perhaps be refactored into a common super-interface that is also extended by MediaStreamTrack.)

```
interface MediaOutputDevice : EventTarget {
    readonly    attribute DOMString          kind;
    readonly    attribute DOMString          id;
    readonly    attribute DOMString          label;
    attribute boolean              enabled;
    attribute boolean              muted;
    attribute EventHandler         onmute;
    attribute EventHandler         onunmute;
    attribute float         volume;          // allow dynamic master volume change on audio devices?
    readonly    attribute boolean              remote;
    MediaOutputCapabilities getCapabilities ();
    MediaOutputConstraints  getConstraints ();
    MediaOutputSettings     getSettings ();
    Promise<void>         applyConstraints (MediaOutputConstraints constraints);
                attribute EventHandler         onoverconstrained;
};
```

Sink Specification

We propose to adapt the concepts in the following Audio Output API sections to accept MediaOutputDevices instead of Sink IDs;

* HTMLMediaElement Extensions
  supply MediaOutputDevice objects, not "sinkIDs"
* WebAudioAPI Extensions
  supply MediaOutputDevice objects, not sinkIDs
* Predefined IDs
  Instead of relying on a limited set of special IDs for “multimedia”, “communications”, etc, use getUserMediaOutput() with constraints to obtain the appropriate MediaOutputDevices for various situations. This also addresses the need for headset discovery, etc.

MediaOutputConstraints/Capabilities/Settings

A new set of output constraints/capabilities/settings will be defined that operate similarly to those already used by getUserMedia() and returned by getCapabilities() and getSettings(). Many of these will be identical to existing attributes for input, e.g. sampleRate, channelLayout, width, height, etc. Others will be new and output-specific, e.g. binaural.

In particular, it is desirable to discover codec types and properties in order for applications to request server content in the format and variant that can best be handled on the user’s device given its processing pipeline hardware and software. *(Jerry Smith of MS to provide more detail on this point.)*

Along with this come a family of output-oriented interfaces analogous to: MediaTrackSupportedConstraints, MediaTrackCapabilities, MediaTrackConstraints, MediaTrackConstraintSet and MediaTrackSettings. Since many constraints/capabilities are common to input and output, refactoring some super-interfaces of common attributes may make sense. These will bear analogous names MediaOutputXXX, where they correspond to input interfaces MediaTrackXXX.

MediaDevice interface extensions

We propose to extend the MediaDevices interface with these new methods:

```
partial interface MediaDevices {
   Promise<MediaOutputDevice>          getUserMediaOutput (MediaOutputConstraints constraints);
   MediaOutputSupportedConstraints getOutputSupportedConstraints ();
};
```

The new getUserMediaOutput() method is deliberately similar to getUserMedia(). An set of constraints lacking constraints other than the device kind will return the default MediaOutputDevice for that kind.

Obtaining a MediaOutputDevice implies granting the user permission to access it, so there are some UA permission grant interactions with the user that need to occur in certain situations, in order to gain access to devices. These will operate similarly to getUserMedia with respect to UA prompts, persistence, etc. except that access to default devices will not require an explicit permission grant.

Capabilities Discovery

The enumerateDevices() method will return an instance of MediaOutputCapabilities in the new capabilities member of a MediaDeviceInfo for any default output device, in all cases (whether the results are filtered or unfiltered as per the current definition of enumerateDevices()). Additionally, filtering will be disabled if any non-default output device has been granted access permission by the user.

Two new parameters to enumerateDevices will further support capability discovery:

- A constraints dictionary, analogous to those used by getUserMediaOutput(), will narrow the set of enumerated devices to those which match the constraints, and will also cause the capabilities returned for each device to reflect those available under the given constraints. For example, if a codec format is supplied as a constraint, a device's frame or sample rate would reflect the frame rate achievable under that codec format.
- A requestPermission flag will explicitly request the user's permission to allow unfiltered capabilities to be returned for all media devices (input and output).

TODO

Study Permissions API
Develop own output-oriented permissions model
Find a way to work with enumerateDevices() results and avoid getUserMediaOutput + constraints
Look at what really needs to be in MediaOutputDevice
Describe list of properties in enumerated devices (including semantic purpose like communications, multimedia…)