W3C WebRTC WG Meeting

June 22, 2021
8:00 AM - 10:00 AM Pacific Time

Chairs: Bernard Aboba
Harald Alvestrand
Jan-Ivar Bruaroey

W3C WG IPR Policy

- This group abides by the W3C Patent Policy https://www.w3.org/Consortium/Patent-Policy/
- Only people and companies listed at https://www.w3.org/2004/01/pp-impl/47318/status are allowed to make substantive contributions to the WebRTC specs

Welcome!

- Welcome to the June 2021 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Streams versus Callback APIs
  - Mediacapture-transform
  - Screen Capture

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/June_22_2021
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - https://w3c.github.io/mediacapture-image/
  - https://w3c.github.io/mediacapture-output/
  - https://w3c.github.io/mediacapture-screen-share/
  - https://w3c.github.io/mediacapture-record/
  - https://w3c.github.io/webrtc-pc/
  - https://w3c.github.io/webrtc-extensions/
  - https://w3c.github.io/webrtc-stats/
  - https://w3c.github.io/mst-content-hint/
  - https://w3c.github.io/webrtc-priority/
  - https://w3c.github.io/webrtc-nv-use-cases/
  - https://github.com/w3c/webrtc-encoded-transform
  - https://github.com/w3c/mediacapture-transform
  - https://github.com/w3c/webrtc-svc
  - https://github.com/w3c/webrtc-ice
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is being recorded. The recording will be public.
- Volunteers for note taking?

Virtual Interim Meeting Tips

This session is being recorded

- **Type +q and -q in the Google Meet chat to get into and out of the speaker queue.**
- **Please use headphones when speaking to avoid echo.**
- **Please wait for microphone access to be granted before speaking.**
- **Please state your full name before speaking.**
- **Poll mechanism will be used to gauge the “sense of the room”.**

Issues for Discussion Today

- 08:10 AM - 08:30 AM Streams versus Callbacks (Dan Sanders)
- 08:30 AM - 09:10 AM Media-capture transform (Harald)
  - Slides by Harald (8:30 to 8:40)
  - Slides by Youenn (8:40 to 8:50)
  - Discussion (8:50 to 9:05)
  - Polls (9:05 to 9:10)
- 09:10 AM - 09:30 AM Screen Capture (Jan-Ivar)
- 09:30 AM - 09:35 AM getViewportMedia Compromise (Elad Alon)
- 09:35 AM - 09:45 AM Capture Handle and Alternatives (Elad Alon)
- 09:45 AM - 09:55 AM Conditional Focus (Elad Alon)
- 09:55 AM - 10:00 AM Wrap-up and Next Steps

Time control:

- A warning will be given 2 minutes before time is up.
- Once time has elapsed we will move on to the next item.

Streams versus Callbacks (Dan Sanders)

- WebCodecs considered both Streams and callback APIs.
- Early encode/decode APIs used Streams.
  - Subsequently moved to a callback model: https://docs.google.com/document/d/10S-p3Ob5snRMjBqpBf5oWn6eYij1vos7cujHoOCCCAw/view
- VideoTrackReader originally used Streams, too.
  - Changed to callbacks along with the rest of WebCodecs.
  - Replaced by mediacapture-transform API, using Streams: https://github.com/w3c/webcodecs/issues/131#issue-798838697

WebCodecs Encode/Decode

- WebCodecs didn’t fit Streams abstraction well.
  - Streams state transitions (close, abort) are destructive.
  - Method calls expressed our state transitions more clearly.
  - Synchronous callbacks happen in a known state.
- WebCodecs use of Streams was complex.
  - Depended on every detail of Streams API.
  - Required the same from apps.
  - Put pressure on Streams spec to add more features.

VideoTrackReader Callback API

```
interface VideoTrackReader {
  constructor(MediaStreamTrack track);
  void start(OutputCallback callback);
  void stop();
};
```

Decision to Move to WHATWG-Streams

- mediacapture-transform API proposal already existed.
  - Reduced scope of WebCodecs.
- TransferableStreams solved capture in workers for us.
- MediaStreamTrack *does* fit Streams abstraction.
  - Sequencing and state changes map well.
  - Consistent API for developers.
    - “we did not find that the code created by streams MSP looked particularly complicated (e.g. demo #1 or demo #2)”

Streams versus Callbacks

- Streams implement buffering and backpressure.
  - With callbacks, all apps have to implement this.
  - Chrome implementation drops stale frames from full buffer as new frames come in.
- TransferableStreams do not transfer Chunks.
  - Could be fixed in Streams spec.
  - Chrome Source implementation works around this.
- Callback API has to happen on capture (typically main) thread.
  - TransferableStreams can go directly to worker.

WebCodecs Issue 131: Depreciate VideoTrackReader

- An initial investigation into whether or not there were advantages that VideoTrackReader (VTR) with no equivalents in MediaStreamProcessor (MSP) did not find anything.
  - Setting a queue size was added to the MSP design as a result this investigation, to allows web applications to prioritize latency with small queues, or to prevent dropping frames with a larger queue.
- VTR's advantage was the simplicity of its API.
  - Callback-based design that doesn't require users to know anything about the Streams API.
  - However, we did not find that the code created by streams MSP looked particularly complicated (e.g. demo #1 or demo #2).
  - The ability to transfer a stream endpoint (rather than posting each frame) makes for clean code in MSP's case (e.g demo #1 in workers).

WebCodecs Issue 131: Depreciate VTR (cont'd)

- Keeping VideoTrackReader for the sole reason of offering a callback alternative to MediaStreamProcessor is untenable.
- MSP offers a distinct opportunity for platforms to implement under-the-hood optimizations, not available to VTR. Specifically, VTR must receive frames on the main thread and then post them to a worker.
  - MSP can transfer a stream endpoint to a worker, and receive frames directly on the worker thread, thus helping main thread contention.

WebCodecs Issue 131: Depreciate VTR (excerpt)

In short,

- VTR (and co.) doesn't offer any unique advantages over MSP (and co.), while MSP has unique better performance potential with workers.
- MSP's use of streams is justified and shouldn't be much harder to use compared to VTR's callback.
- Only having one way to do things on the web platform is healthier for the platform, and less maintenance overall.

For these reasons, we are deprecating VideoTrackReader (and not developing VideoTrackWriter + audio equivalent), and instead helping out with the development of MediaStreamTrackProcessor / MediaStreamTrackGenerator.

Discussion

- Questions?
- Opinions?

Mediacapture-transform issues (Harald)

- Main issues under active discussion
  - #4 Is using ReadableStream/WritableStream the right approach?
  - #23 Out-of-main-thread processing by default
  - #29 Is Audio MSTG/MSTP necessary?

#4 Is using ReadableStream/WritableStream the right approach?

These slides will cover:

- The requirements
- Properties of Media (in general)
- Properties of Streams
- The problem of non-media events
- The proposal under consideration
- A proposal for WG decision

The requirements

- Efficient media processing in user-managed code (JS/WASM)
- Minimum disruption of existing WebRTC APIs
- Possibility to do processing off the main thread
  - On-thread was already somewhat possible by bouncing video streams via Canvas

The Media Properties (as seen from WebRTC)

- Audio and Video are inherently sequenced data
  - Audio has samples (small); video has frames (large). Usually audio is done in blocks.
- In the WebRTC model, timing is carried in stamps, not wall clock
  - Except at playback time, there’s no inherent need to sync to the system clock
- WebRTC media processing is adaptive, “soft” real time
  - We drop frames, cover up missing samples and do other tricks as a matter of routine

The MediaStreamTrack API is a control surface that allows the user to control how the platform processes the media data. It does not interact directly with the media data; media processing never touches the Javascript thread.

Streams: The properties

- Streams offer sequenced JS processing of arbitrary objects
- Streams offers transferability - the ability to move one end of a stream to a different processing object (usually a worker)
- Streams offers a very simple pushback mechanism: Don’t read!
  - Buffering can be made controllable.
- Streams make it easy to define “processing elements” that can be chained together to make more complex systems (TransformStream)
- Streams are not tied to the system clock
- Streams are an existing feature of the Web platform

The Problem of Non-Media Events

Stuff happens around a media stream. For example:

- Downstream congestion may indicate the need to downscale
- Errors and disconnects may indicate the need to stop or restart processing
- Upstream changes of state (for example mute) may not be inferrable from the media
- The JS code may want to reconfigure its upstream or downstream

At the moment, we do not have a good model that covers all of these cases well (the “feedback stream” proposed is not well defined; ad-hoc callback interfaces like those proposed for the SframeTransform are another option).

Suggest not solving this problem in this iteration.

The Proposal Under Consideration

- Make an interface to generate a Stream from a MediaStreamTrack
- Make an interface to generate a MediaStreamTrack from a Stream
- Make these as simple as possible (but no simpler!)
    - We know we don’t know what controls beyond backpressure we need. So don’t lock those things down now.

The first proposal was presented to the WG in November 2020 (7 months ago)

The proposal has been simplified quite a bit since then, and can be simplified even more.

(minimizing the feedback mechanisms until we are surer what we need seems logical)

A Proposal for WG Decisions

The WG should agree (in separate decisions):

1. Building an interface for JS/WASM processing of raw media is a reasonable thing for this WG to work on (already in the charter)
2. Building this interface on the Streams API is a sensible thing
3. The chairs should send out a call for adoption of mediacapture-transform as a FPWD for this WG to work on

#23 Streams vs Callbacks (counterpoint) (Youenn)

- Compare streams and callbacks for MediaStreamTrack media flow
  - Processing of incoming frames
  - Video track only

- What bring WhatWG streams?
  - Backpressure
  - Buffering
  - Locking
  - Transferability
  - Piping

- Let's look at each of these criteria

#23 WhatWG Streams - backpressure (Youenn)

- Backpressure unused currently
  - getUserMedia, getDisplayMedia, RTCPeerConnection, canvas
  - Future sources may use backpressure
    - Web Codecs for non-realtime use case?

- Backpressure is good to have
  - Streams have built-in backpressure
  - Backpressure easily supported by promise-based callbacks
    - Service worker events
    - TransformStream transform callback

```
async function computeHeadPosition(frame)
{
  ...
}
track.onframe = async (frame) => computeHeadPosition(frame);
```

#23 WhatWG Streams - buffering (Youenn)

- Streams are good at buffering
  - Very easy to buffer with streams without really noticing it
  - Usually fine but not great for realtime processing & buffer pools

```
const processor = new MediaStreamTrackProcessor(track, 10); // Camera has a buffer pool of 10 frames.
await new Promise(resolve => setTimeout(resolve, 10000)); // Wait 10 seconds.
const stream = processor.readable;
const reader = stream.getReader(); // Read frame.
const chunk = await reader.read();
if (!chunk.done)
   alert(chunk.value.timestamp); // Timestamp shows this is a 10 seconds old frame.
...

reader.releaseLock();
stream.pipeThrough(new TransformStream(..., { highWaterMark: 2 }); // How many frames buffered in pipeline?
```

- Buffering is a potential footgun
  - Not very useful for video processing
  - Preferable to have explicit buffering done by the application itself
    - As done by WebAudio

#23 WhatWG Streams - locking (Youenn)

- Streams require locking to read data
  - Only one reader per stream at a time
  - MediaStreamTrack has no such notion: sources can fuel multiple sinks

- Streams can tee to allow reading in parallel
  - Does not match MediaStreamTrack cloning

```
const p = new MediaStreamTrackProcessor(track);
const teed = p.readable.tee();
const r1 = teed[0];
const r2 = teed[1];

const reader1 = r1.getReader();
const chunk = await reader1.read();
if (!chunk.done)
   alert(chunk.value.timestamp);
```

```
const p1 = new MediaStreamTrackProcessor(track);
const p2 = new MediaStreamTrackProcessor(track);
const r1 = p1.readable;
const r2 = p2.readable;
...
```

- Streams locking/teeing not well suited for MediaStreamTrack
  - Additional complexity for developer, no clear benefit

#23 WhatWG Streams - transferability (Youenn)

- WhatWG streams are transferable
  - So are MediaStreamTracks
- Keeping MediaStreamTrack close to frame processing is useful
  - Ended, muted, unmuted events, applyConstraints

```
async function computeHeadPosition(frame)
{
  ...
}
onmessage = (e) => {
  const track = e.data.track;
  track.onmute = (event) => self.postMessage('head detection disabled');
  track.onunmute = (event) => self.postMessage('head detection enabled');
  track.onframe = async (event) => self.postMessage('we got a head at ' + await
computeHeadPosition(event.frame));
};
```

- Easier to optimize transfering of MediaStreamTrack
  - Hop from thread producing frames to thread processing frames
- Optimizing streams transferability should not be a must-have

#23 WhatWG Streams - piping (Youenn)

- Stream piping is good for native transforms
  - Optimize the flow from a native source to a native sink (no JS)
    - RTCRtpScriptTransformer to SFrameTransform

- MediaStreamTracks have piping
  - video.srcObject/pc.addTrack/MediaRecorder
- Better for native transforms to manipulate tracks directly
  - Transform frames + handle states like muted/unmuted

```
myTransform = new BackgroundBlurTransformStream();
processor = new MediaStreamTrackProcessor(captureTrack);
generator = new MediaStreamTrackGenerator();
processor.readable.pipeThrough(myTransform)
                  .pipeTo(generator.writable);
transformedTrack = processor.track;
```

```
myTransform = new BackgroundBlurTransform();
transformedTrack = captureTrack.transform(myTransform)
```

- Streams piping is redundant with MediaStreamTrack model

#23 WhatWG Streams - tentative conclusion (Youenn)

- Streams main benefit is backpressure
  - Other stream features are not particularly well suited
  - Backpressure can easily be implemented with a callback-based API

- Streams overlap with MediaStreamTracks
  - Increase complexity without clear benefits
    - tee() vs. clone() or closed vs. onended

- Callback-based APIs already successfully deployed
  - requestVideoFrameCallback

- Proposal: do a full comparison of stream vs. callback-based proposals
  - Both processing and generation

Discussion

- Are the statements made accurate?
- Are the arguments made compelling in one direction or the other?
- Are there concerns by Web developers that haven’t been addressed?
-

Poll Mechanism: Warmup

Let's make sure everyone knows how to respond to polls. Which of these two flavors of ice cream do you prefer?

- A: Vanilla
- B: Chocolate

Poll #1: To Stream or not to Stream

- Alternatives:
  - We should adopt a proposal based on Streams
  - We should wait for a proposal based on Callbacks
  - We don’t have enough information to decide

#23 Out-of-main-thread processing by default

Two trains of thought:

- Worker processing should be easy. Main-thread processing should be possible. (Initial proposal)
- Worker processing should be mandatory. Main-thread processing should be hard or impossible. (Proposals for making MSTG/MSTP worker-only)

There is no precedent in the platform for worker-only APIs, except for APIs that control workers. The Worker API set is still lagging behind the Window API set.

Web developers have mostly said “don’t care”, or “want both options”.

Proposed decision: Make APIs available in both contexts. Encourage recording usage.

#29 MediaStreamTrackProcessor/Generator for Audio

- Alternatives:
  - AudioWorklet for audio processing, MSTP/MSTG for video
  - MSTP/MSTG defined for both audio and video
- Advantages of making MSTP/MSTG available
  - One API for joint processing of audio and video (sync, look-to-listen…)
  - Soft realtime apps don’t have to run on the system clock (echo / howl detector)
- Advantages of not making MSTP/MSTG available
  - Forces people to learn to use AudioWorklet
  - Runs all audio processing on a high priority thread (even if it doesn’t need it)
  - Reduces implementation effort, since MSTP/MSTG doesn’t need audio code
- Proposed decision: Specify MSTP/MSTG for both Audio and Video

#182 Safer integrated web presentations in getDisplayMedia (Jan-Ivar)

Give preferential placement to “isolated-browser” display surfaces:

- A **browser** display surface is the rendered form of a browsing context. This is not strictly limited to HTML [HTML] documents, though the discussion in this document will address some specific concerns with the capture of HTML.
- An **isolated-browser** display surface is the rendered form of a browsing context where the current top-level document is cross-origin isolated and has opted into html capture. Capture survives navigation to other pages that are also site-isolated and opted into capture. But upon any other navigation, the user agent *MUST* freeze capture on the last safe frame, and *MAY* prompt the user with a warning and option to allow capture of the unsafe content. Capture will resume once the browsing context is navigated back to safety, unless the user answered affirmatively to the prompt, in which case the source becomes a browser display surface.

Screen Capture Slide controls (Jan-Ivar)

More better **web presentations**. Capture & control the app, not the tab.

“IDs are bad” — Whom do they serve? What about the rest of us?

Feedback: ”I understand the value of an API like that, but I think it should be a benefit for all, not just for the ones that control both the content and the conferencing services. I really hope that the API can be modified so this can happen. — Best regards Sergio”

Products don’t interop = Users lose.
What use cases do they enable? **Slide controls in the VC meeting**.

Should we **standardize** this baseline use case instead?

- Are we a working group in the position to do that? Yes
- Is it our obligation to attempt to standardize a baseline here? Yes
- Should we do that? Yes

mediaDevices.getSlidesController() (Jan-Ivar)

```
// Capturee (the presentation being captured). Site-isolated and opted in to 🔴

try {
  const controller = await navigator.mediaDevices.getSlidesController({
    slide: 1,
    slides: 10,
    onslidechange: ({slide}) => advanceToSlide(controller.slide = slide),
  });
  nextSlideButton.onclick = () => advanceToSlide(++controller.slide); // triggers captur-
                                                                      // er onslidechange

  // console.log(controller.slide);  // 1 (mutable 1-10)
  // console.log(controller.slides); // 10 (read-only)
  // console.log(controller.id);     // rotated on navigation (read-only)
} catch (e) {
  if (e.name == "SecurityError") return; // not site-isolated & opted in to capture
}
```

API is SecureContext & throttled. Only the last controller returned is alive. Iframe hijacking.

mediaDevices.getSlidesController() (Jan-Ivar)

```
// Capturer (the VC conference doing the capturing)  👁

try {
  const stream = await navigator.mediaDevices.getDisplayMedia();
  const [track] = stream.getVideoTracks();

  const controller = await navigator.mediaDevices.getSlidesController({
    track,
    onslidechange: () => updateProgressBar(controller.slide, controller.slides),
  });
  nextSlideButton.onclick = () => controller.slide++; // triggers capturee onslidechange


  // console.log(controller.slide);  // 1 (mutable 1-10)
  // console.log(controller.slides); // 10 or 0 if capturee was nav'ed away (read-only)
  // console.log(controller.id);     // may be used to find capturee, or "" (read-only)
} catch (e) {
  if (e.name == "NotFoundError") return; // track has no presentation controller
}
```

#158 getViewportMedia() cropping (Jan-Ivar)

MST is already transferable.

WebIDL

```
[Exposed=(Window,Worker), Transferable]
partial interface MediaStreamTrack {
};
```

So cropping to an arbitrary iframe B is easily solved (without ids):

```
A.html:
  <iframe src="B.html">
  <script>
    const {data: track} = await new Promise(r => window.onmessage = r);
    const pc = new RTCPeerConnection();
    pc.addTrack(track);

B.html:
  <script>
    const stream = await navigator.mediaDevices.getViewportMedia(); // crops to self
    const [track] = stream.getVideoTracks();
    parent.postMessage(track, [track]);
```

#158 getViewportMedia() cropping (Jan-Ivar)

We thought we had progress in the issue. To crop or not. A compromise proposal for a common case? 👎

Capture full viewport from iframe (important? Seemed so on github...)

```
await navigator.mediaDevices.getViewportMedia({viewport: "tab"});  // default is "self"
```

Details:

1. "self" — iframe gets its intersection with the top viewport.
2. "tab" — iframe gets the top-level viewport.
3. ~~"node"~~ — seemed to have agreement this is unnecessary (for now)?
4. Should we default to "self" or "tab" ?

...Or simply postMessage the other way (from A.html to B.html) 👍

#158 getViewportMedia() cropping (Elad)

Transferability is only beginning to be specified. The road is long.

The use-cases for getViewportMedia(self) remain to be clarified.

The use-cases for getViewportMedia(arbitrary-frame) have previously been presented.

Is getViewportMedia(self) a compromise? Or is it a compatibility risk waiting to happen when some user agents are late to invest resources in implementing a use-case they don't deem interesting?

My own proposal for a compromise:

- getViewportMedia() can accept an enum value of SELF, an enum value of TAB, or an ID of an arbitrary iframe in the browsing context.
- The user agent MUST support SELF.
- The user agent MUST / SHOULD / MAY support TAB / arbitrary-iframe.
- The default is SELF.
- We can discuss a good mechanism for exposing these capabilities later.

Capture Handle - Insufficiency of SlidesController (Elad)

- There is benefit in Jan-Ivar's SlidesController. Particulars notwithstanding, there is room for more than one tool. An on-rails approach has its merits. Power tools have their merits. We should have both. We need both.
  - How does SlidesController help with anything other than slides? Should other applications wait patiently until we produce a single-use tool aimed specifically at them?
  - Can SlidesController hope to foresee all legitimate slides functionality? When additional features are required, is the Web to wait for us again?
  - How will SlidesController tackle the need for two-way communication?
  - What if the slides-application is not willing to accept messages from unauthenticated, unknown capturers?
  - Does SlidesController offer anything for self-capture, or should we create yet another single-use tool?

SlidesController is a great idea which does not replace Capture Handle. Let's pursue both.

Capture Handle - Proposed Alternatives (Elad)

Summarizing some objections to Capture Handle and their counters:

- Power tool vs. on-rails: Previous slide.
- Monopolistic concerns:
  - Alternatives are either limited on-rails solutions, or can equally limit collaboration.
  - Browsers are in the business of enabling collaboration, not forcing it.
  - There is nothing illegitimate in the use-case of a multi-tab application.
- “But you could also expose an is-self-capture bit and also…”: The argument that multiple tools have their place should not be carried ad-absurdum. If a single tool accomplishes multiple tasks, that is a point in its favor.
- Expose a MessagePort instead: Multiple objections. For example, that the capturing document must alert the captured document to the existence of the capture.

Conditional Focus - Rationale (Elad)

When screen-capture of a tab/window begins, should the captured tab/window be immediately focused?

Proposals are being discussed for capturer-capturee collaboration. When capturing a collaborating page, there might be no value in activating that tab/window, as controlling it remotely might be preferable.

(Shameless plug: It helps that with Capture Handle, when capture starts, the capturer knows immediately what it's capturing, and does not have to start exchanging messages over a MessagePort.)

It would be interesting to allow the capturing application to signal to the UA whether focus should be switched or not - immediately after capture starts.

Conditional Focus - Proposed API Shape (Elad)

Proposed API shape:

```
let [track] =
  (await navigator.mediaDevices.getDisplayMedia()).getVideoTracks();
track.focus(myPredicate(track));
```

Where myPredicate() is free to examine the track's settings and/or capture-handle (shameless plug).

Before getDisplayMedia's Promise is resolved, the user agent posts a microtask to close the focus-window. The application can therefore only call focus() immediately on the task where the Promise is resolved.

Failure to call focus() yields either the UA's default focus behavior, or a specified behavior of our choosing.

The window is closed after 1s either way, preventing delayed-focus-change attacks.

For extra credit

Name the mammal!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The mammal