W3C WebRTC WG Meeting

April 27, 2021
8:00 - 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 April 2021 interim meeting of the W3C WebRTC WG!
    - During this meeting, we will discuss WebRTC-PC extension process, Testing, WebRTC Extensions, MediaCapture Transform, WebRTC encoded transform, getViewportMedia, displayMediaCaptureHandle and Screenshot API.

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/April_27_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.

Issues for Discussion Today

- In Memoriam (Sergio)
- WebRTC-PC extension process (Harald)
- Testing (Bernard)
- WebRTC Extensions (Bernard)
- Media Capture Transform (Guido, Harald and Youenn)
- WebRTC encoded transform (Youenn)
- getViewportMedia (Elad, Jan-Ivar)
- Capture Handle (Elad)
- Screenshot API (Elad)
- Conditional Audio Suppression (Elad)

Dr. Alex Gouaillard

It is with a heavy heart that we must give you the news that our friend, colleague and leader Dr. Alex passed away on Thursday, April 8th 2021. He was involved in a motor vehicle accident in Thailand that took his life.

All of us are in shock. And for anyone who knew him, whether for weeks, months or years, he was a brilliant human being and as memorable a personality as you will meet in life. He will forever have a place in our hearts.

Dr. Alex was among the largest contributors and advocates for WebRTC technology. And although we are still mourning his loss, we will honour his memory by working to fulfill his vision for the future of WebRTC.

WebRTC-PC Extension Process (Harald)

- PR #2637 - creates a new document
- W3C REC process envisages “new features can roll in”
  - But only if they satisfy the criteria for REC: Implementation, consensus
- Features have to have somewhere to live while getting that far

Proposal:

- Webrtc-extensions and “small sharp documents” are “incubation places”
- Once an extension has shown consensus, test suite & two implementations, a merge is possible (if desired)
- Bug fixes are always appropriate!
- We should aim at a “merge round” no more often than 6 months or so

Test Proposal from Two Meetings Ago

- Build a content reflector that speaks the WebRTC stack - as minimal as possible
- https://github.com/jlaine/aiortc-wpt-demo/
  - only 60 lines of code
  - terminates STUN + DTLS, decrypts SRTP, echoes RTP/RTCP packets via WebSockets
  - uses RTCPeerConnection from aiortc
  - WPT already uses aioquic for quic tests
- Simple tests!
  - create a peerconnection, connect WebSocket
  - get raw packets
    - parsing RTP/RTCP/SCTP…
  - Moving backend to ORTC allows testing of STUN/DTLS as well
  - Example tests:
    - NACK (potential FF bug),
    - RTCP BYE (webrtc.org bug),
    - VP8-simulcast
    - See also https://github.com/sipsorcery/webrtc-echoes

Status Update

- Asked Jeremie Laine to address issue with crc32 library license
  - Need MIT license or equivalent
  - No objection from Jeremie, but work has not started
- Next steps (from last meeting)
  - Ask WPT folks to give us such a server (RFC Progress)
  - Start writing tests!
  - Scope of tests? What is “web”, what is IETF

Issues for Discussion Today

- WebRTC Extensions
  - PR 37: requestKeyFrame API (Bernard)
- Media Capture Transform
  - Update (Guido and Harald)
  - Feedback on current approach (Youenn)
- WebRTC encoded transform
  - PR 64:Remove redundant API (Youenn)
- getViewportMedia (Elad, Jan-Ivar)
- displayMediaCaptureHandle (Elad)
- Screenshot API (Elad)

PR 37: requestKeyFrame API (Bernard)

- Existing mechanisms used to cause an RTCRtpSender to generate a keyframe have limitations:
  - No mention of keyframe generation in WebRTC-PC (and therefore, no tests)
  - sender.replaceTrack(existingTrack)
    - Not reliable (due to race?)
  - RTCRtpSendParameters.encodings[*i*].active
    - Even if setting active=false, then active=true causes a keyframe to be generated, it will likely cause a stutter.
- Proposal
  - sender.generateKeyFrame(encodings)
    - More appropriate name since a keyframe is being generated, not requested.
    - Encodings identifies the streams to be reset (e.g. by RIDs)
    - Effect is the same as receiving an FIR with SSRCs corresponding to the target streams.
      - May cause all streams to be reset: 10107 - Upon PLI for a specific SSRC, it should generate a keyframe just for that stream - webrtc (chromium.org)

MediaCapture Transform: Proposal to adopt

Status of Implementation

- Implemented in Chrome / Edge
- Available since Chrome M90 behind an origin trial (WebCodecs)
  - Also as an experimental Web feature
- Origin Trial has 50+ signups
  - 8+ are experimenting with Breakout Box
  - Positive feedback from developers
- Apps in active development by Zoom and Google Meet

MediaCapture Transform Issue 4

Is Readable/WritableStream the right approach?

We believe it is.

- Using streams allows us to leverage mechanisms proven in production (e.g., direct support for workers, familiar interface, well-known programming model).
  - WebCodecs removed VideoTrackReader in favor of Breakout Box
- Separating readable and writable allows us to better support more use cases than a transform-only approach:
  - ***Custom sinks***: It allows using WebCodecs+WebTransport to implement custom protocols (Zoom is currently testing this).
  - ***Custom sources***: symmetric with custom sink.
  - ***Multiple-source tracks***: Weather report (mix webcam + other video/presentation)

MediaCapture Transform Issue 6

Memory management for incoming frames.

What if the upstream track produces more frames than can be consumed?

Existing mechanisms:

- ***Application level***: VideoFrame and AudioFrame have a close() method that allows precise management by the application. MediaStreamTrackProcessor also has an input buffer whose maximum size can be set by the application.
- ***UA level***: this problem may already exist with platform sinks and sources, the same mechanisms UAs have in place apply in this case

MediaCapture Transform Issue 20

Add "real-time" warning/note to MediaStreamTrackGenerator

What if a MediaStreamTrackGenerator (which is a track) generates more frames than the sinks can consume. How to notify the application about it?

Potential solutions:

- Platform sinks can use existing error-reporting mechanisms (e.g., error event in media elements and maybe peer connections)
- We can define a signal that can be exposed in MediaStreamTrackGenerator.readableControl

MediaCapture Transform Issue 1

Add a high-quality face/body tracking API to discourage poor/discriminatory implementations

- We can adopt a similar approach to the one we took on webrtc-encoded-transform and define a standard transformation for this
- Perhaps specify it on an extension spec, since the details might not be trivial and look reasonably orthogonal to the core of the spec

Proposal: Adopt the spec as a WG document

What WG Adoption Means

- WG agrees that this problem needs solving, and that the given spec is a starting point
- WG agrees that further changes will be done under WG auspices
- WG has the right and the duty to make changes to the API if it is in the interest of the Web community to do so

Feedback on Transform (Youenn)

A transform for both audio and video?

- For video, we need a better solution than Canvas
- For audio, we already have Audio Worklet

Media capture transform vs. Audio worklet

- Audio worklet more reliable (high priority audio thread)
- Audio worklet more resilient (no sync XHR)
- Audio worklet does not have control signals
    - Use is unclear for audio, can probably be extended or shimed

Proposal: focus solely on raw video

- Simpler to define and potentially better API

Feedback on Transform (Youenn)

Processing ideally done off the main thread

- Current API is main thread centric
  - If main thread is blocked, large potential video memory increases
  - Current API difficult to implement without main thread blocking risk
    - Streams spec algorithm & Chrome blocked on main thread
    - Edge cases hard to optimize

Proposal: envision off-the-main thread alternatives, for instance:

- Allow transferring MediaStreamTrack in/from workers
  - Consensus at last interim to investigate this / Shimable in Chrome
- Add API to expose frames in workers/lets
  - Processing of MediaStreamTrack content in workers/lets only

Feedback on Transform (Youenn)

How could it look like?

```
// main-page.js
const worker = new Worker('my-head-detection-worker.js');
const stream = await navigator.mediaDevices.getUserMedia({ video: true});
const track = stream.getVideoTracks()[0].clone();
worker.postMessage({type:'head', track: track}, [track]);
worker.onmessage = (event) => console.log(event.data));

// my-head-detection-worker.js
async function computeHeadPosition(frame)
{
  ...
}
onmessage = (e) => {
  if (e.data.type === 'head+encode') {
    const encoder = new VideoEncoder({ output : (data) => { ... } });
    const track = e.data.track;
    track.onmute = (event) => { self.postMessage('head detection disabled');
    track.onunmute = (event) => { self.postMessage('head detection enabled');
    track.onframe = (event) => {
        const position = await computeHeadPosition(event.frame);
        if (position) self.postMessage('we got a head at ' + position);
        encoder.encode(event.frame);
    };
  }
};
```

Frame hopping from camera thread to worker thread without ever going through main thread

Feedback on Transform (Youenn)

Existing APIs heavily based on MediaStreamTrack

- WebAudio, RTCPeerConnection, MediaRecorder, HTMLMediaElement
- Optimized pipeTo implementation between all of these, input & output

MediaStreamTrack and WebCodecs

- JS executed for every audio/video frame
    - Not always useful, potentially suboptimal
- ReadableStream.pipeTo to the rescue?
    - Not really, WebCodec does not take streams as input/output
    - Not always easy to optimize
- WebCodec to support MediaStreamTrack as input/output?

Feedback on Transform (Youenn)

How could it look like?

```
// encode path
const stream = await navigator.mediaDevices.getUserMedia({ video: true });
const encoder = new VideoEncoder({ output : (data) => {
  transferEncodedData(data);
} });
encoder.encode(stream.getVideoTracks()[0]);
```

Frame hopping from camera thread to web codec thread without ever going through main thread

```
// decode path
const decoder = new VideoDecoder({ type: 'track', output : (track) => {
  video.srcObject = new MediaStream([track]);
});
```

Frame hopping from web codec thread to video rendering thread without ever going through main thread

Feedback on Transform (Youenn)

The controlling channel

- Usage and meaning of web-facing signals is unclear
  - Should specs define when native sinks create such signals?
  - Should specs define what native sources do with these signals?
- Internal signals are unclear as well
  - Potentially User Agent specific signals?
- Difficult to evaluate whether current API surface is well suited or not
  - What happens if web app breaks either web-facing or internal signals?

- Proposal
  - Identify/Document use cases before talking about API

Feedback on Transform (Youenn)

Potential issues with control signals current API

- Implicit signaling should be the default for all signals
    - People will forget to properly setup the controlling channel
- Keeping the order of signals is not guaranteed
    - Within web-facing signals and with internal signals as well
- Web-facing signals and internal signals seem to be two different things
    - Might be easier to split in two different concepts/two APIs
- Spec seems to assume a single source -> transformer -> sink pipe
    - Track signal target for several sinks/Multiple tracks as input of a processor
    - Signal source is not exposed which might be useful in case of one track with multiple sinks

Feedback on Transform (Youenn)

What about additional downlink signals

- Muted/ended tracks

Example

- RTCPeerConnection sends a capture track
- RTCPeerConnection sends a generated track that takes capture track as input
- Capture track gets muted/disabled
  - Black frames will be sent for first track
  - No frames sent for second track
    - Except if transformer track gets muted/disabled based on its related capture track

Feedback on Transform (Youenn)

Should we expose frames using a ReadableStream?

- No clear benefit with regards to native transforms
  - Simpler for native transforms to directly use MediaStreamTrack
    - Possibility to handle muted/enabled states
- Need to be cautious of the memory model
- maxBufferSize
  - Interesting idea
    - Probably useful to document/identify use cases
  - Possibility to retrofit this feature
    - In a callback-based (VideoTrackReader) or event-based API

Feedback on Transform (Youenn)

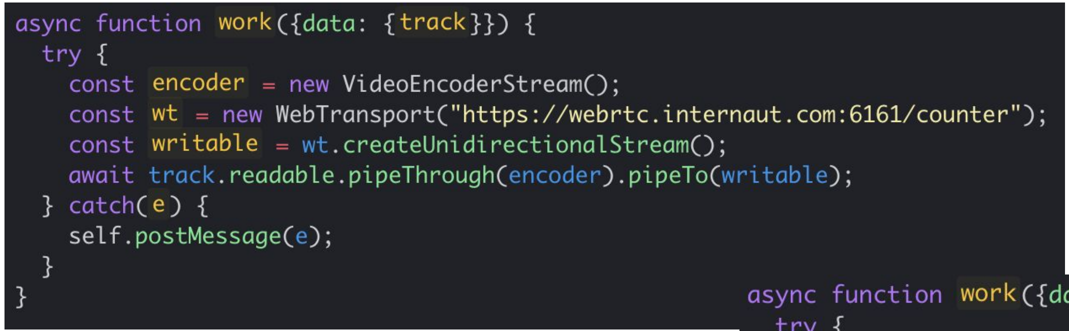
Tentative summary

- A raw video media access API is a great idea
- WebRTC WG is the right place to do this work
- Current proposal needs more work
    - Let's focus on video and core functionality in V1
    - In parallel, let's document additional requirements and use cases
        - Might lead to introducing control signals

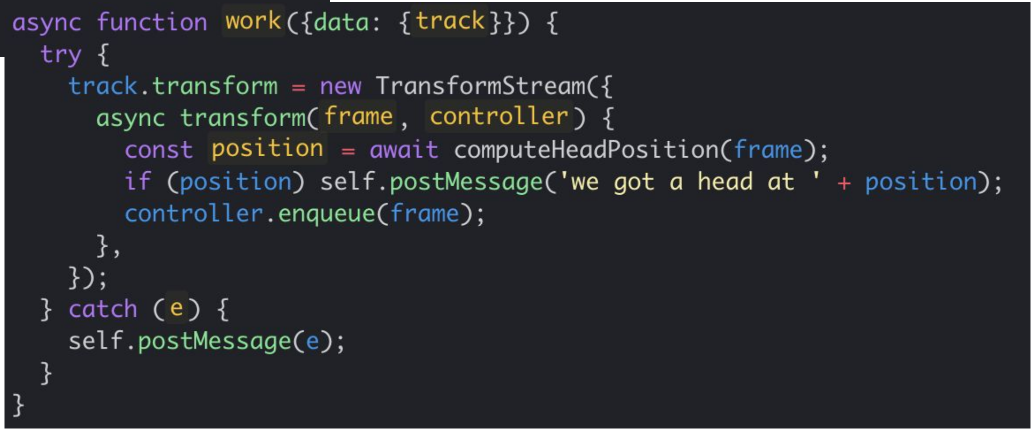
Feedback on Transform (Jan-Ivar)

What Youenn said, except adding streams to his examples, for back-pressure.

Back-pressure != buffering. For real-time too regardless of drop vs buffer strategy.



```
async function work({data: {track}}) {
  try {
    const encoder = new VideoEncoderStream();
    const wt = new WebTransport("https://webrtc.internaut.com:6161/counter");
    const writable = wt.createUnidirectionalStream();
    await track.readable.pipeThrough(encoder).pipeTo(writable);
  } catch(e) {
    self.postMessage(e);
  }
}
```



```
async function work({data: {track}}) {
  try {
    track.transform = new TransformStream({
      async transform(frame, controller) {
        const position = await computeHeadPosition(frame);
        if (position) self.postMessage('we got a head at ' + position);
        controller.enqueue(frame);
      },
    });
  } catch (e) {
    self.postMessage(e);
  }
}
```

PR 64: Remove redundant API (Youenn)

- Initial proposal
  - RTCRtpSender/RTCRtpReceiver createEncodedStreams
    - Makes it easier to do things in the main thread
      - Not suitable for realtime processing (e.g. ScriptProcessorNode)
    - Notion of transform not very clear
- WG Consensus: move to a transform model
  - Define RTCRtpScriptTransform & SFrameTransform
  - Off-the-main thread processing by default
- We now have two APIs for the same purpose in the spec

- Proposal: remove createEncodedStreams from the specification
  - Complete the shift to a transform model
  - Bonus: more precise algorithms that do not resort on transferring streams

Issue 99: Use WebCodec APIs (Youenn)

Two APIs for very similar concepts

- RTCEncodedAudioFrame and RTCEncodedVideoFrame
- EncodedAudioChunk and EncodedVideoChunk

Main differences

- WebCodec uses immutable encoded frames
  - WebRTC encoded transform expect scripts to change frames
    - Data ownership transferred at write step
- RTC specific metadata exposure

Two options

- Use two different APIs
- Merge the two APIs
  - Select a common 'data' handling
  - Extend WebCodec frame metadata for RTC

#155 getViewportMedia (Jan-Ivar)

Agreement! (Consensus?)

- Only expose if `window.crossOriginIsolated` (a.k.a. COOP+COEP)
- Resources are on their own (vulnerable to Spectre anyway) 😜
- Failure mode: Block loading of non-opt-in iframes (generalizes well)

Agree on: Shape of opt-in-to-capture doc header.

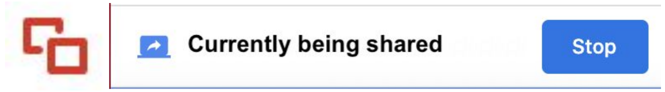
Problems w/`COEP: require-corp; html-capture`. Opts into riskier (not safer) profile (confusing). Requires Fetch Metadata request header as well (Sec-Fetch-COEP)

Proposal from @camillelamy: Off-by-default Document policy (WICG)

- Top-level doc says: `"Require-Document-Policy: html-capture"`
- Iframe docs must say: `"Document-Policy: html-capture"` to load
- Avoids having to define separate Fetch Metadata request header

getViewportMedia: Mitigate User information harvesting

- Require (a subset of) existing **mediacapture-screen-share** protections:
  - MUST require User Gesture (transient activation)
  - MUST NOT store a "granted" permission entry
  - MUST follow Privacy Indicator Requirements



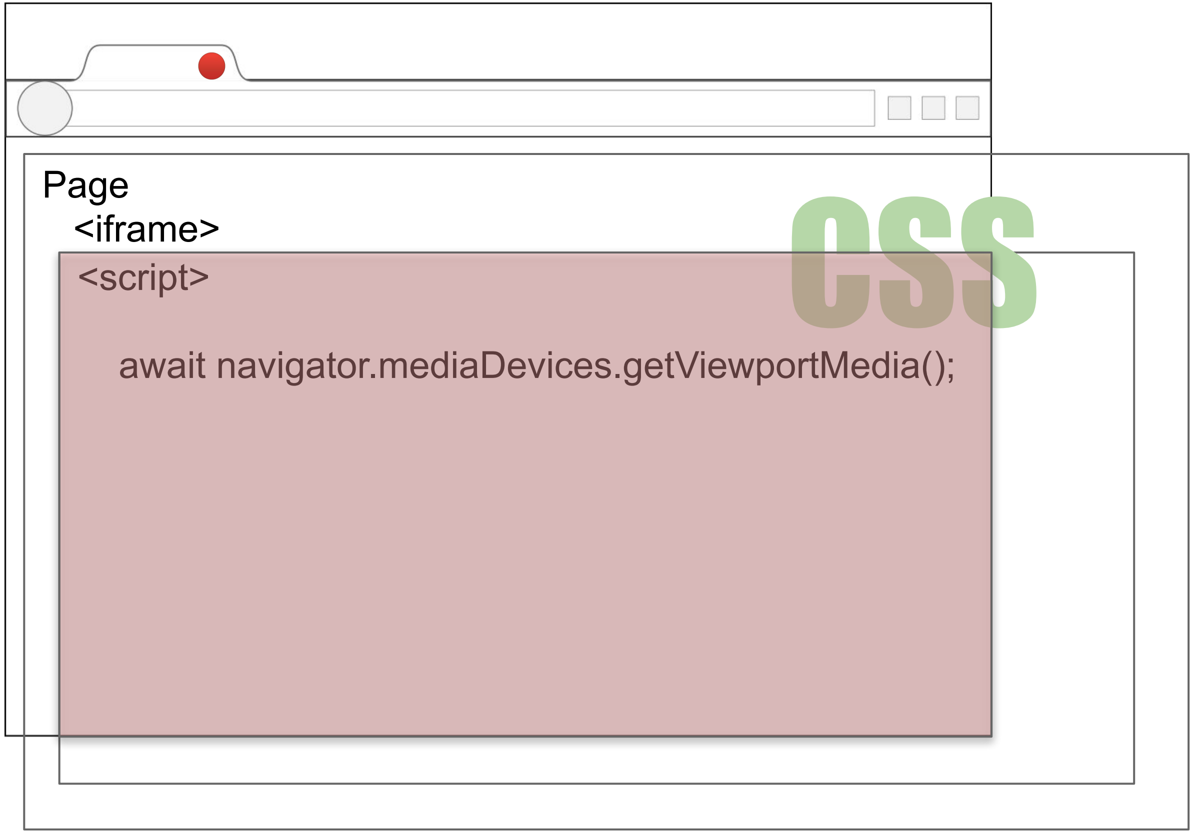
- Require additional protections:
  - MUST require permission to use (*“Allow site to record its rendering?”*)
  - MUST attempt to explain the risk to the user (*“This may fingerprint you”*)
  - MUST require current global object == relevant global object, to reject `iframe.contentWindow.navigator.mediaDevices.getDocumentMedia()`
  - MUST NOT disable extensions (avoid Ad-blocker blocker mode)
- Considering dropping:
  - MUST mute/pause while `document.visibilityState != “visible”`

getViewportMedia: cropping

Proposal: Capture intersection of ***viewports*** of TLBC's active document & iframe.

Transfer MediaStreamTrack instead of allowing capture by others. Lets top-level page delegate **iframe capture** to iframes without opting into being captured itself. Seems most conservative (to start).

Solves cropping at the target. Avoids passing IDs or cropping coordinates. Fine-grained enough.



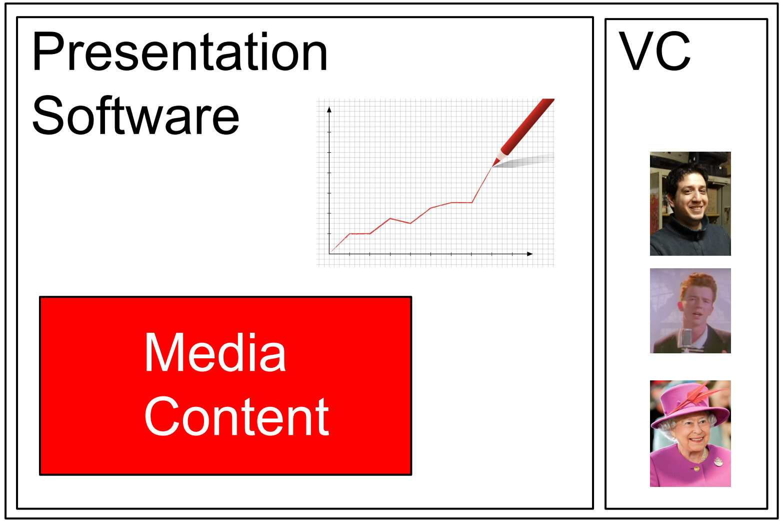
Benefits of getViewportMedia from Arbitrary Frame (Elad)

Consider an application that combines presentations and VC. One frame each.

The VC frame captures the entire tab and crops itself away.

This allows the VC app to use restrictOwnAudio to exclude its own audio from the capture.

The VC can now transmit the captured media content without creating an audio feedback loop.



Capture Handle - Exposition

- Introducing VC-MAX, your one stop shop for all your fictional-VC needs.
  - Fictive video-conferencing software specializing in illustrating points.
  - FaaS - fiction as a service.
  - Any resemblance to real software project, living or dead, is purely coincidental.
- VC-MAX exposes a “Share” button that calls getDisplayMedia and transmits the resulting tracks to remote participants.
- VC-MAX wants to collaborate with various presentation software. When the user chooses to share MS ForcePoint, OpenOffice-365 or Doodle Slides, VC-MAX would like to embed user-facing controls for navigating the presentation - previous slide, next slide, etc.

Capture Handle - Puzzle Pieces

- Assume MS ForcePoint, OpenOffice-365 and Doodle Slides all define their own APIs for navigating a presentation, subject to appropriate authentication, access-control, etc.
  - The shapes of these APIs are out of scope.
  - It’s virtually guaranteed that the API includes some **session ID**.
- VC-MAX needs to discover:
  - The captured application’s identity. (ForcePoint? Doodle?)
  - The session ID.
- Our **discussion’s scope** is discovery of these generic bare essentials. Their use is left as an exercise to the reader. :-)

Capture Handle - Possible Hack

What could we do with no changes to the browser?

- The presentation software could embed a QR code[1].
- VC-MAX could scan for that QR code.

Problems?

- Unergonomic.
- Inefficient.
  - VC-MAX must continuously scan for the QR code and remove the user-facing controls if no longer capturing the presentation.

---

(1) QR code used to keep the illustration simple. Similar solutions yield similar problems.

Capture Handle - Suggested Solution

On the captured app:

- Allow opt-in exposure of the origin.
- Allow a custom handle to be set.
    - Let it be a string. That string could be a serialized JSON carrying an arbitrary amount of data.
- Call these the **capture-handle**.

On the capturing app:

- Allow checking the current capture-handle.
- Allow registering a handler for capture-handle-update events.

Capture Handle - Suggested Solution (Captured App)

```
// Code on Slides-3000
navigator.mediaDevices.setCaptureHandleConfig({
  exposeOrigin: true,
  handle: JSON.stringify({
    description: "See slides-3000.com for our API description. " +
                 "It supports such amazing actions as nextSlide, " +
                 "prevSlide, goFullScreen, etc.",
    sessionId: mySessionId(),
  })
});
```

Capture Handle - Suggested Solution (Capturing App)

```
// Code on VC-MAX
const mediaStream = await navigator.mediaDevices.getDisplayMedia();
const captureHandle =
    mediaStream.getVideoTracks()[0].getSettings().captureHandle;
if (captureHandle.origin == "preso.slides-3000.com") {
  const sessionId = JSON.parse(captureHandle.handle).sessionId;
  Let commsChannel = ...;  // Specific to Slides-3000.
  commsChannel.postMessage(sessionId, "goFullScreen");
  ...
} else if (captureHandle.origin == “avoid.doodle.com”) {
  ...
}
```

Capture Handle - Interesting Note

Application don’t actually have to expose their origin. If the **session ID** is sufficiently unguessable, a separate API could be set up exposing **validate()**.

```
const captureHandle =
  mediaStream.getVideoTracks()[0].getSettings().captureHandle;
if (slides3000Helper.validate(captureHandle)) {
  ...
}
```

If a site exposes its origin, though, an RTT to the identification service could be saved.

Note that the captured site chooses to <u>allow</u> exposing the origin; the browser <u>sets</u> the actual value to be read on the capturing side.

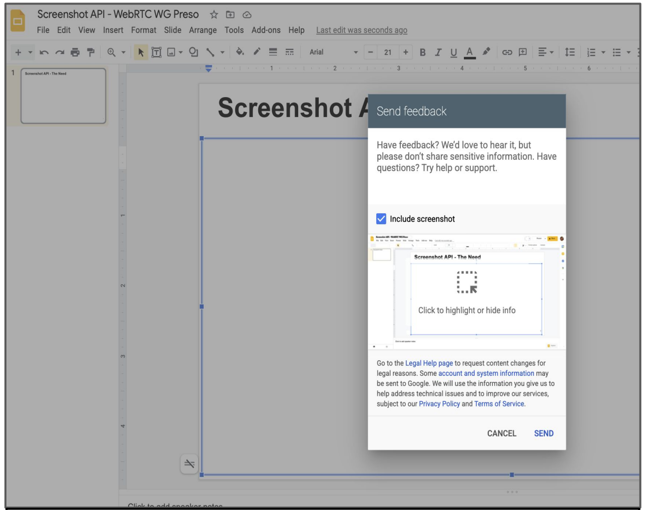
Capture Handle - Stop Worrying, Love the Bomb.

Privacy/Security concerns? Probably not.

- Opt-in
- User-driven
- Theoretically no-op (see QR-hack - already possible!)
- Captured app only discovers it’s captured if capturer informs it.
- App-controlled handle opaque
- Excessive events not a viable attack

Screenshot API - Who? Why?

Who takes screenshots of the current tab's viewport?


Screenshot API - WebRTC WG Preso
File Edit View Insert Format Slide Arrange Tools Add-ons Help
Last edit was seconds ago
Screenshot API - The Need
Screenshot A

Send feedback

Have feedback? We'd love to hear it, but please don't share sensitive information. Have questions? Try help or support.

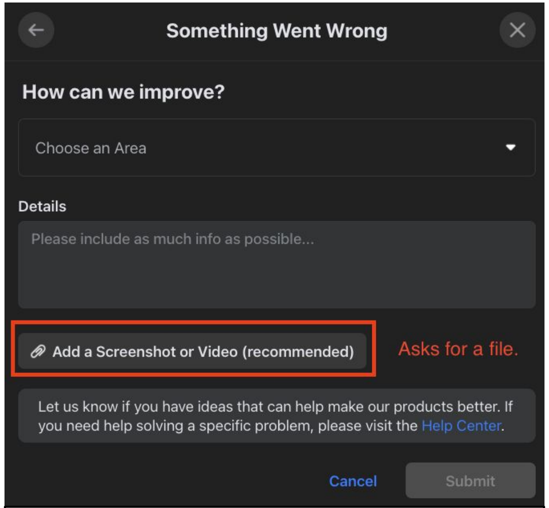
☑ Include screenshot

Click to highlight or hide info

Go to the Legal Help page to request content changes for legal reasons. Some account and system information may be sent to Google. We will use the information you give us to help address technical issues and to improve our services, subject to our Privacy Policy and Terms of Service.

CANCEL SEND

Most Web-based Google Products



Something Went Wrong

How can we improve?

Choose an Area

Details

Please include as much info as possible...

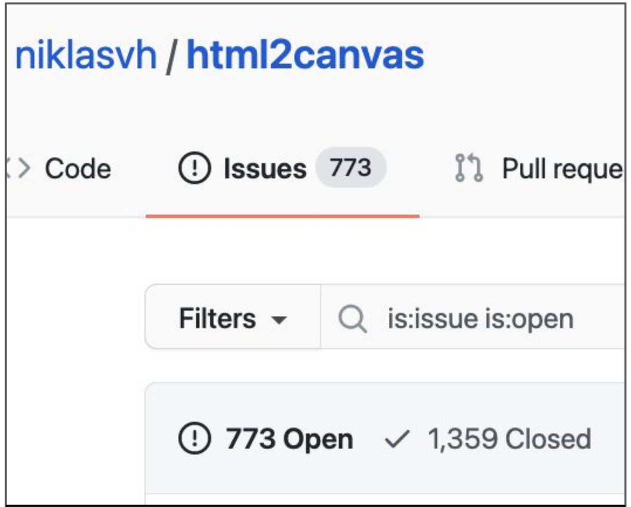
Add a Screenshot or Video (recommended)

Asks for a file.

Let us know if you have ideas that can help make our products better. If you need help solving a specific problem, please visit the Help Center.

Cancel Submit

Facebook



Sites using html2canvas

Why? Often - to gather user feedback on defects.
Is it important? Depends - is customer-retention important? (Yes it is!)

Screenshot API - State of the Art

- No JS-exposed API for grabbing a screenshot.
- Workarounds exist.
- No workaround is acceptable.

Screenshot API - Workaround #1 (gDM)

Idea: Use getDisplayMedia, grab a single frame.

Deficiencies:

- User compelled to grant permission to grab multiple frames.
- Permission can persist for many hours if user is not careful to revoke.
- User cannot see what they share. Application could mislead by presenting one frame, when it in fact surreptitiously recorded several.
- Bad lesson for the user about permissions on the Web.
- Savvy users would mistrust the app for asking for excessive permissions.
- User may choose wrong thing - even a good app could accidentally store inappropriate content on back-end, become liable. (Online video game bug-triager comes across user's banking info => **shareholders sad**.)

Screenshot API - Workaround #2 (gVM)

Idea: Use getViewportMedia, grab a single frame.

Deficiencies:

- All points from the previous slide except the last one.
- Also, it’s not yet been specified, let alone implemented. (But that’s a minor issue in comparison to the rest.)

Screenshot API - Workaround #3 (DOM Redrawing)

Idea: Walk the DOM and redraw it onto a Canvas.

Popular solution:

- Google products use something similar.
- Libraries such as html2canvas exist and are popular.

Deficiencies:

- Cross-origin content is a challenge.
- A big challenge.

Screenshot API - Workaround #4 (Extensions)

Idea: Extensions can take a screenshot...

Deficiencies:

- Awkward user flow requiring the installation of what essentially amounts to third-party software.
- Installing third-party software is **risky**. (Bad lesson to teach users.)
- No cross-compatibility between browsers.

Screenshot API - Workaround #5 (Manual)

Idea: Ask the user to upload their own screenshot.

Deficiencies:

- Awkward user flow.
- Not all users even know how to take a screenshot.
- Users might upload wrong thing; humans exposed to this information could abuse it.

Screenshot API - Goals and Non-Goals

Non-Goals

- No need to support grabbing an image of other display-surfaces.
- No need to grab a screenshot of the page as visible beyond the tab’s viewport.
- No need to support **information-adding** mechanisms (annotation).

Goals

- Support taking a screenshot of the current tab’s viewport.
    - Open question: What about the current window?
    - Current monitor is probably off-limits for now; there could be more than a single “current monitor.” Can revisit later.

Screenshot API - Security Requirements 1/2

There is consensus that getViewportMedia is safe enough if gated by:

1. Cross-origin isolation
2. Opt-in header (TBD)

This is an upper-bound on the requirements for a screenshot API.

Is the bound tight? Can we relax any of these using replacements?

Screenshot API - Security Requirements 2/2

Common-sense requirements:

- Preview must be shown to the user.
- User must be allowed to crop and/or black-out content in-browser.
- Transient activation required.
- Responsible document must be fully active.
- Document must have some necessary permission.

Can adding any of the following stand in for COI and opt-in headers?

- Display a stern warning in the dialog shown to the user.
- Heuristics applied by user agent to avoid abuse. [Challenging]
- Random delay before screenshot taken. [Only handles edge case.]
- Introduce random noise to screenshot. [Ineffectual?]

The above mechanisms aren’t bulletproof, but maybe in combination…?

Conditional Audio Suppression

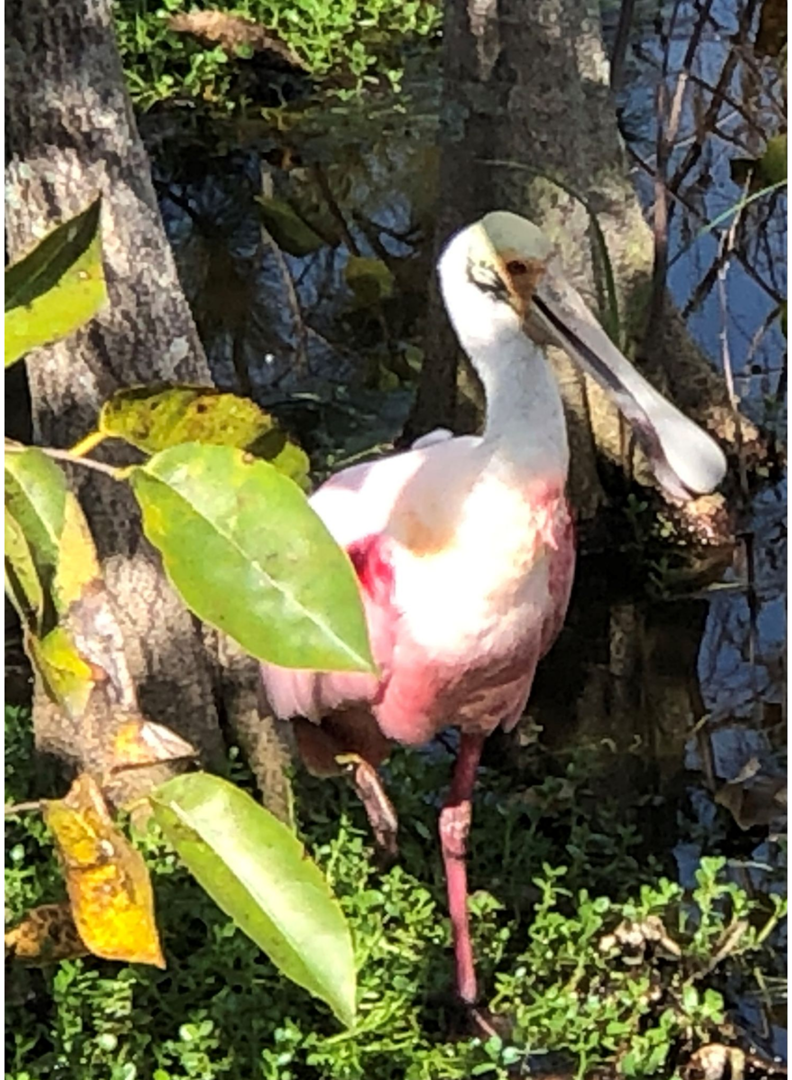
Suppose this meeting weren’t virtual. We’d all be sitting in a room, taking turns presenting from our laptops to a projector and a set of speakers. The next speaker presents a tab. Do they want the audio to flow out of both their laptop’s speakers as well as the PA system’s? Probably not.

Solution? Define a new constraint - suppressLocalAudioPlayback.

When this constraint is applied, audio from captured source is not played out over the local speakers.

This can also help with echo cancellation on a single-machine setup, as playing the locally captured audio in a VC as though it came from another participant simplifies things for the echo canceller.

For extra credit



Name the bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird