# W3C WebRTC WG Meeting

December 2, 2020
11:00 AM - 12:30 PM 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 December interim meeting of the W3C WebRTC WG!
    - During this meeting, we will talk about Insertable Streams, Capabilities, WebRTC-NV use cases, getCurrentBrowsingContextMedia and HTML capture.
- This is the last virtual interim of 2020, so enjoy the Holidays, and stay safe!

# About this Virtual Meeting

- Meeting info:
    - https://www.w3.org/2011/04/webrtc/wiki/December 02_2020
- 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://w3c.github.io/webrtc-dscp-exp/
    - https://github.com/w3c/webrtc-insertable-streams
    - 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

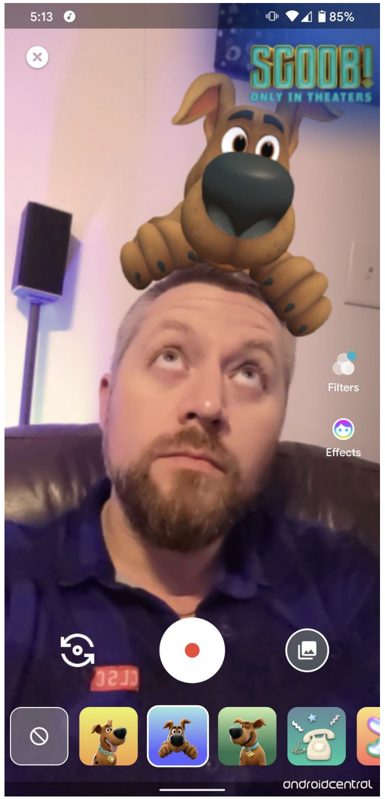
- Insertable Streams Raw Media (Harald): 11:05 AM - 11:15 AM
- WebRTC Insertable Streams (Youenn): 11:15 AM - 11:25 AM
- WebRTC Capabilities: 11:25 AM - 11:40 AM
  - Issue #2460 - **getCapabilities seems to leak hardware capabilities w/o permission (Jan-Ivar)**
  - **Issue #22 - getCapabilities seems to leak hardware capabilities w/o permission (Jan-Ivar)**
  - **#550 - Stats API should require additional permission (Jan-Ivar)**
  - **PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)**
  - **MC API #159 - MediaCapabilities API extended to WebRTC - (Johannes Kron)**
- Media Capture Extensions (Youenn): 11: 55 AM - 12:05 PM
  - **Issue #12 - Expose camera presets to web pages (Youenn)**
- WebRTC-NV Use Cases (Tim Panton): 11:40 AM - 11:55 AM
  - **PR 68: First pass at new and updated use cases**
- Capture This Tab (Elad): 12:05 PM - 12:15 PM
- First Class HTML Capture (Jan-Ivar): 12:15 PM - 12:25 PM

## Issues for Discussion Today

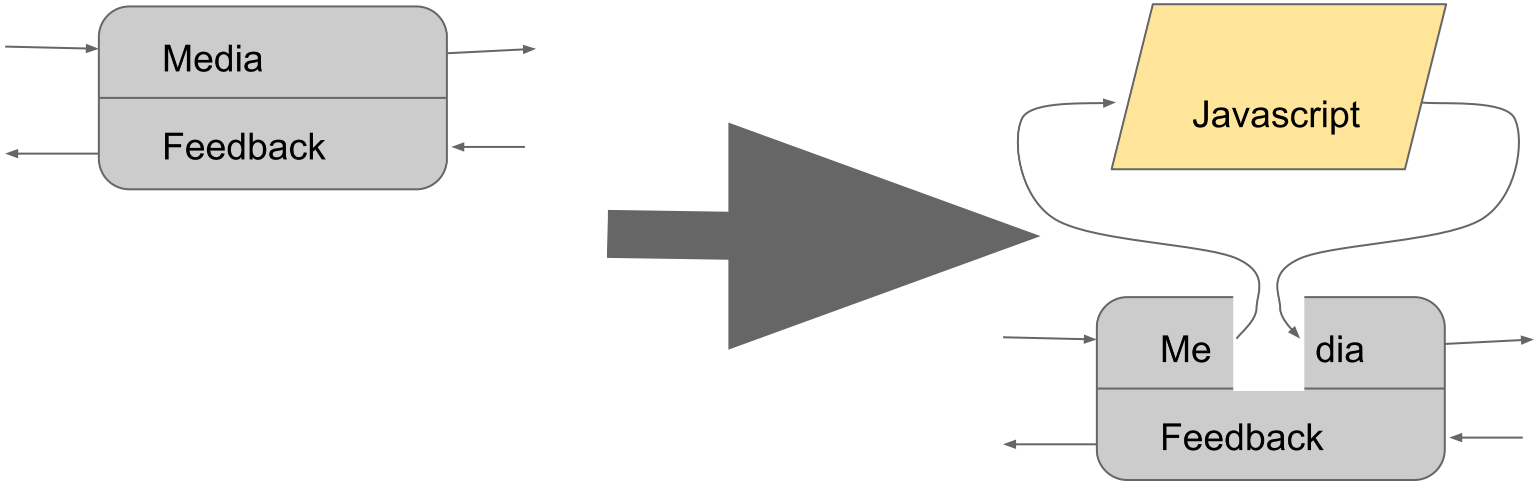
- Insertable Streams Raw Media (Harald, 10 minutes)
- WebRTC Insertable Stream (Youenn, 10 minutes)
  - **Issue #48 - WebRTC insertable streams as transform (Youenn)**

# Insertable Streams for Raw Media

- Open issues
  - #2 Is Phase 2 needed?
  - #1 Does face recognition belong?



# Open up the MediaStreamTrack

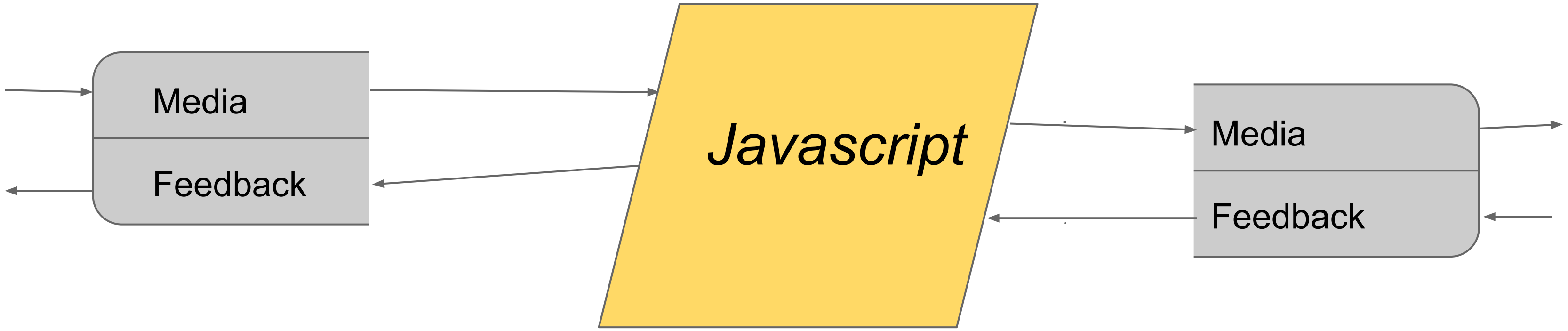


Breakout Box stage 1 -> 2

# Stage Two Track Processor

```
function addMoustache(videoFrame) {
  let facePosition = detectFace(videoFrame.data);
  return addMoustache(videoFrame.data, facePosition);
}

processingTrack = new ProcessingMediaStreamTrack(videoTrack);
Transformer = new TransformStream({
   Transform: (videoBuffer) => {
      videoFrame.modifyData(addMoustache(videoFrame));
  }
});
processingTrack.readable
     .pipeThrough(transformer)
     .pipeTo(processingTrack.writable);
```

# Break Apart the MediaStreamTrack



Breakout Box stage 3 - allows for generating and consuming tracks directly

## #2 Is Phase 2 needed?

- Implementation of TrackGenerator and TrackConsumer turned out to be simple
- ProcessingMediaStreamTrack can be shimmed in O(10) lines of JS
- Less is more?

Suggested resolution: Delete phase 2, add shim to examples.

## #1 Specify a face-tracking API?

- Is this a separable concern?
- Is someone else addressing it?
    - https://blog.tensorflow.org/2020/03/face-and-hand-tracking-in-browser-with-mediapipe-and-tensorflowjs.html
    - https://wicg.github.io/shape-detection-api/
- If “no” and “no” - how do we start?

## Status and Next Steps

- Experimental implementation has landed
- Start of specification available
    - https://github.com/w3c/mediacapture-insertable-streams

## Issue #48 - WebRTC insertable streams as transform (Youenn)

Ongoing Safari experiment

- Support of SFrame transform, JS transforms and combos
    - JS transforms running in background threads as a default
- Available in Safari nightly behind a feature flag

Experimental API allows attaching transforms to senders and receivers

```
typedef (SFrameTransform or RTCRtpScriptTransform) RTCRtpTransform;
partial interface RTCRtpSender {
    attribute RTCRtpTransform? transform;
};
partial interface RTCRtpReceiver {
    attribute RTCRtpTransform? transform;
};
```

## #48 - WebRTC insertable streams - SFrame transform

### Example

```
// Sender
const stream = await getStream();
const pc = new RTCPeerConnection();
const sender = pc.addTrack(
         stream.getVideoTracks()[0], stream);

const key = await keyManagement.key();
sender.transform = new SFrameTransform();
sender.transform.setEncryptionKey(key);


// Receiver
const pc = new RTCPeerConnection();
const key = await keyManagement.key();
pc.ontrack = e => {
  e.receiver.transform = new SFrameTransform();
  e.receiver.transform.setEncryptionKey(key);
};
```

### WebIDL

```
dictionary SFrameOptions {
   ...
};
interface SFrameTransform {
    constructor(optional SFrameOptions options);

    Promise<undefined> setEncryptionKey(
        CryptoKey key,
        optional unsigned long long keyID);
    Promise<undefined> ratchetEncryptionKey();
};
SFrameTransform includes GenericTransformStream;
```

## #48 - WebRTC insertable streams - JS+SFrame transform

SFrame transform combined with JS transform

- ReadableStream/WritableStream pair
- Encoded audio/video frames or ArrayBuffers

```
class mySFrameDecryptionTransformer {
    start(readable, writable) {
        this.sframeTransform = new SFrameTransform({ role : "decrypt" });
        this.postDecryptTransform = new TransformStream(...);
        readable.pipeThrough(sframeTransform)
                .pipeThrough(postDecryptTransform)
                .pipeTo(writable);
    }
    transformPostDecryption(frame, controller) => {
        // Handle decryption error case.
        if (!frame.data.byteLength)
            return;
        // Do specific decrypted frame processing before going to decoder.
        ...
        controller.enqueue(frame);
    }
}
```

## #48 - WebRTC insertable streams - Transform model

Can we update current insertable streams API to using transform model?

- Yes, using WhatWG TransformStream

```
dictionary ReadableWritablePair {
    required ReadableStream readable;
    required WritableStream writable;
};
typedef (SFrameTransform or ReadableWritablePair) RTCRtpTransform;
```

Questions to the Working Group

- Add SFrameTransform to insertable streams draft spec?
- Update insertable streams draft spec to transform model?

## #48 - WebRTC insertable streams - JS transform

### Example

```
// HTML page
const pc = new RTCPeerConnection();
const worker = new Worker("worker-module.js");

const sender = pc.addTrack(track, stream);
sender.transform = new
    RTCRtpScriptTransform(worker, "myNoopTransformer");
sender.transform.port.postMessage("Hello Transformer");
sender.transform.onmessage = (e) => console.log(e.data);
```

```
// worker-module.js
class myNoopTransformer extends RTCRtpScriptTransformer {
    constructor() {
        this.port.postMessage("Hello Transform");
    }
    start(readable, writable, context) {
        this.port.postMessage("Starting");
        readable.pipeTo(writable);
    }
};
registerRTCRtpScriptTransformer(
    "myNoopTransformer", myNoopTransformer);
```

### WebIDL

```
[Exposed=Window]
interface RTCRtpScriptTransform {
    constructor(Worker worker, DOMString name);
    readonly attribute MessagePort port;
    attribute EventHandler onerror;
};

[Exposed=DedicatedWorker]
interface RTCRtpScriptTransformer {
    constructor();
    readonly attribute MessagePort port;
    undefined start(
        ReadableStream readable,
        WritableStream writable,
        RTCRtpScriptTransformerContext context);
};

[Exposed=DedicatedWorker]
partial interface DedicatedWorkerGlobalScope {
    undefined registerRTCRtpScriptTransformer(
        DOMString name,
        RTCRtpScriptTransformerConstructor constructor);
};
```

## #48 - WebRTC insertable streams - JS transform variations

Make RTCRtpScriptTransformer deal either with frames or streams

```
// Stream version
class myNoopTransformer extends RTCRtpScriptTransformer {
    start(readable, writable, context) {
        readable.pipeTo(writable);
    }
};

// Frame version
class myNoopTransformer extends RTCRtpScriptTransformer {
    transform(frame, controller) {
        controller.enqueue(frame);
    }
}
```

## #48 - WebRTC insertable streams - JS Transform Worklet

Run transform in RTCPeerConnection Worklet

- No need to manage a dedicated worker
- Control of API exposed in worklet

Relatively cheap to specify and implement

- Direct reuse of worklet infrastructure

```
partial interface RTCPeerConnection {
    undefined addModule(UVString url);
};
partial interface RTCRtpScriptTransform {
    constructor(RTCPeerConnection connection, DOMString name);
};
```

## Issues for Discussion Today

- WebRTC Capabilities
  - Issue **#2460 - getCapabilities seems to leak hardware capabilities w/o permission (Jan-Ivar)**
  - **Issue #22 - getCapabilities seems to leak hardware capabilities w/o permission (Jan-Ivar)**
  - **PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)**
  - **#550 - Stats API should require additional permission (Henrik)**
  - **MC API #159 - MediaCapabilities API extended to WebRTC - (Johannes Kron)**

# State of privacy in WebRTC-PC / Stats / SVC

**Thanks to PING for reviewing these APIs!**

```
RTCRtpSender.getCapabilities("audio");
RTCRtpSender.getCapabilities("video");
RTCRtpReceiver.getCapabilities("audio");
RTCRtpReceiver.getCapabilities("video");

const pc = new RTCPeerConnection();
const offer = await pc.createOffer();
const stats = await pc.getStats();
```

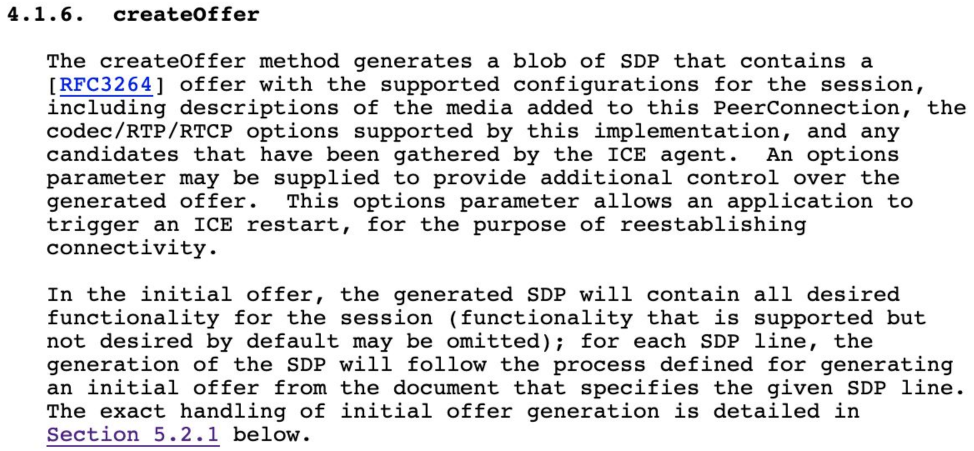
**3 issues were filed + a PR (open):**

- **#2460 - getCapabilities seems to leak hardware capabilities w/o permission**
- **#22 - getCapabilities seems to leak hardware capabilities w/o permission**
- **#550 - Stats API should require additional permission**
- **PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)**

## #2460/#22 - getCapabilities leaks hardware capabil w/o permission

A site can learn about the visitor's underlying hardware capabilities w/o a permission prompt or some other positive, affirmative action by the visitor.

Most of the same information is available in the SDP offer from `pc.createOffer()` which inherently needs to be signaled by JS to form a peer-to-peer connection, as described in JSEP (IETF):



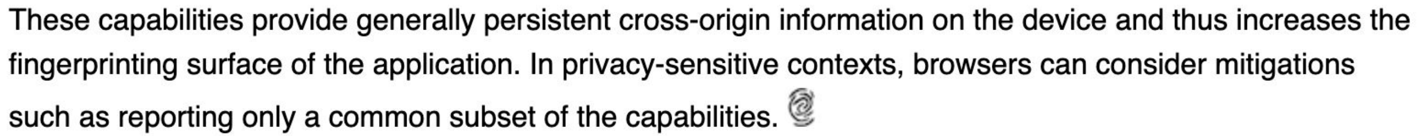
```
4.1.6.  createOffer

   The createOffer method generates a blob of SDP that contains a
   [RFC3264] offer with the supported configurations for the session,
   including descriptions of the media added to this PeerConnection, the
   codec/RTP/RTCP options supported by this implementation, and any
   candidates that have been gathered by the ICE agent.  An options
   parameter may be supplied to provide additional control over the
   generated offer.  This options parameter allows an application to
   trigger an ICE restart, for the purpose of reestablishing
   connectivity.

   In the initial offer, the generated SDP will contain all desired
   functionality for the session (functionality that is supported but
   not desired by default may be omitted); for each SDP line, the
   generation of the SDP will follow the process defined for generating
   an initial offer from the document that specifies the given SDP line.
   The exact handling of initial offer generation is detailed in
   Section 5.2.1 below.
```

Use cases:

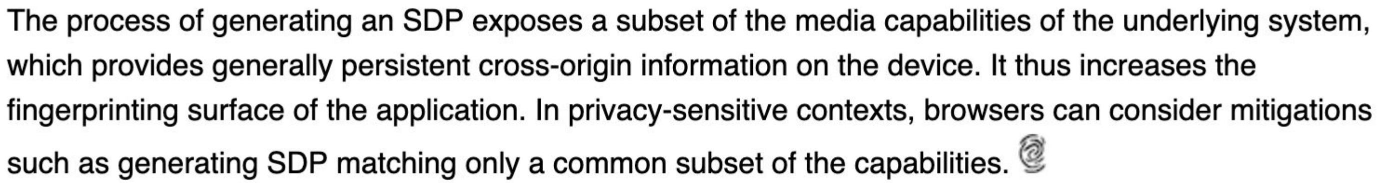
- Data channels
- Receive media
- Send media other than cam/mic/screen, e.g. canvas/elem.captureStream()

## #2460/#22 - getCapabilities leaks hardware capabil w/o permission

getCapabilities:


These capabilities provide generally persistent cross-origin information on the device and thus increases the fingerprinting surface of the application. In privacy-sensitive contexts, browsers can consider mitigations such as reporting only a common subset of the capabilities.

createOffer says:


The process of generating an SDP exposes a subset of the media capabilities of the underlying system, which provides generally persistent cross-origin information on the device. It thus increases the fingerprinting surface of the application. In privacy-sensitive contexts, browsers can consider mitigations such as generating SDP matching only a common subset of the capabilities.

**Conclusion** from Graphics Hardware Fingerprinting document linked in issue:

- “Information relating to graphics hardware capabilities provided by [WEBRTC], [WebRTC-Stats], [WebRTC-SVC] ... may also be inferred from other sources such as **Web-GPU, Web-GL and performance API**.”
- “...graphics hardware fingerprinting concerns are not WebRTC-specific. ...consider adding a permission relating to *“whether the page is permitted to know what graphics hardware the user is running”* (outside WebRTC)”

**Proposed resolution** is to include a note relating to implementation issues with hardware capabilities.

## #550 - Stats API should require additional permission

Two privacy harms / risks reported:

1. Leaking communication / plain text
    - A useful consideration for isolated streams (WebRTC-identity), but for regular streams, the web page has prior access to audio and text content.

2. Hardware fingerprinting (`decoderImplementation` & `codec`)
    - Similar to **#2460** (covered in previous slide)

## PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)

**Problem:** It’s not clear what the requirements are of getCapabilities().

1. As previously noted, for privacy reasons *"browsers can consider mitigations such as reporting only a common subset of capabilities"*.
2. getCapabilities() is a synchronous API, but querying HW encoder/decoder support might block JS.

**Intended use case (example):**

- Check codec capabilities and do RTCRtpTransceiver.setCodecPreferences() *before or during* O/A exchange.

**Is our intended use-case fulfilled if only a subset is reported?**

- Only if getCapabilities() and createOffer()/createAnswer() match…
- Would an implementation where they don’t match be spec-compliant?

## PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)

**Proposal 1:**

- Clarify that if a capability was exposed in createOffer() or createAnswer() it must also be exposed in getCapabilities() if later called in the same document.
  - ✔️ Prevents having to parse SDP to obtain the full set of capabilities.
  - ❌ Does NOT prevent getCapabilities() from being HW-agnostic prior to createOffer()/createAnswer(). **Proposal 2:** ✔️ Require this to also match?
- Clarify that if a capability was exposed in getCapabilities() it MUST be returned by future getCapabilities() calls later called in the same document.
  - ✔️ Prevents setCodecPreferences() from throwing InvalidModificationError due to invalid codec.

Hopefully no implementation needs to be updated to fulfill these requirements.

- Revisit async API in an extension spec.

## MC #159: MediaCapabilities API extended to WebRTC (Johannes)

- “Enable web sites to select an efficient configuration before the stream starts.”
  *(codec, resolution, frame rate, etc)*
- Currently in WebRTC:
  {RTCRtpReceiver, RTCRtpSender}.getCapabilities()
  => List of codecs, no information about expected performance.
- Existing MediaCapabilities API ('media-source', 'file', or 'record')
  Query for a specific configuration => (supported, smooth, powerEfficient)
- Proposed spec change PR:
  - Adds **webrtc** as **MediaDecodingType** and **MediaEncodingType.**
  - Adds **scalabilityMode** to **VideoConfiguration** to support querying for SVC support.
  - Clarifies MIME types for WebRTC.
    ‘video/VP9;profile-id=0’ vs ‘video/webm; codecs="vp09.00.10.08"’

## MC #159: MediaCapabilities API extended to WebRTC

- Example usage:

```
const mediaConfig = {
  type: 'webrtc',
  audio: {
   contentType: 'audio/opus',
   channels: '2',
   bitrate: 132266,
   samplerate: 48000
  },
  video: {
    contentType: 'video/VP9;profile-id=0',
    width: 1920,
    height: 1080,
    bitrate: 2646242,
    framerate: '60'
  }
};

var result = await navigator.mediaCapabilities.decodingInfo(mediaConfig);
```

## MC #159: MediaCapabilities API extended to WebRTC

- Possible implementations:
  - Deterministic model based on the hardware/software configuration.
  - Prediction based on historical performance.

- Privacy concerns
  - Too specific information may expose information that can be used for fingerprinting.
  - Either through perfect knowledge of the hardware/software configuration, or if the historical data is too specific so that cross-site tracking is possible. (e.g., play video with odd resolution at web site A, check the response at web site B),

## Issues for Discussion Today

- Media Capture Extensions
  - **Issue #12 - Expose camera presets to web pages (Youenn)**

## Issue #12 - Expose camera presets to web pages (Youenn)

Web pages tend to favor using camera presets

- Good for performances and image quality

Web pages have difficulties selecting camera presets

- resizeMode, frame rate, ideal constraints
    - Hard to master, cross-browser difficulties
- No easy way to know whether there are more suitable presets
    - Things might get more complex if exposing pixel format

Proposal: define a more straightforward API

- Expose camera native presets similarly to what native APIs do
- Expose API to MediaStreamTrack
    - API available after getUserMedia is granted

## #12 - Expose camera presets to web pages (Youenn)

### Potential API

```
dictionary MediaDevicePreset {
     required unsigned long width;
     required unsigned long height;
     DoubleRange frameRate;
};
partial interface MediaStreamTrack {
     readonly attribute sequence<MediaDevicePreset> presets;
};
```

### Example

```
const stream = await navigator.mediaDevices.getUserMedia({ video : true });
const track = stream.getVideoTracks()[0];
const preset = selectBestVideoPreset(track. presets);
track.applyConstraints({width: preset. width, height: preset.height})
```

## #12 - Expose camera presets to web pages (Youenn)

Reuse the same API in MediaCapture Automation

```
dictionary MockCameraConfiguration : MockCaptureDeviceConfiguration
{
   double defaultFrameRate = 30;
   DOMString facingMode = "user";
   sequence<MediaDevicePreset> presets;
};
```

Purpose

- Enable WPT tests validating constraints implementations
- Enable web sites to test against more realistic mock devices

## WebRTC-NV Use Cases

- PR 68: First pass at new and updated use cases

# WebRTC-NV Use Cases

## PR 68: First pass at new and updated use cases (Tim)

- PR is here : https://github.com/w3c/webrtc-nv-use-cases/pull/68/files
- Based on TPAC presentation
- All things that _nearly_work_ in WebRTC 1.0
- PR Addresses the “These aren’t all new” - by extending existing use cases
- Extend 4 existing
- Add 3 new

## Tldr; Extend existing use cases

- Improve remote UX for pre-empted media calls
- Extend IoT use-case to work in isolated networks
- Non-discriminatory FunnyHats
- Add local in-browser SFU/MCU for conferencing

## pre-empted media calls on mobile

Mobile Browsers drop media when GSM call arrives.
The remote user has no idea what happened.

- The user agent must provide the ability to play selected media to the remote party during an interruption (c.f. on-hold music)

- The user agent must provide the ability to 'park' a connection such that it can be retrieved and continued by a newly loaded page to prevent accidental 'browsing away' from dropping a call irretrievably

## IoT sensor connection resilience

IoT sensors often communicate with LAN users (e.g. baby monitor)
Such connections should work even in event of cloud/internet outage

- A 'long-term connection' must be able to be re-established without access to external services in the event of the local network becoming isolated from the wider network without compromising e2e security.

## Non-discriminatory FunnyHats

FunnyHats depends on face tracking.
Simplistic facetrack AI’s have been shown to be discriminatory

- The user agent must provide Non discriminatory implementations of facetracking and body tracking algorithms that can be efficiently used by the application.

(Not mentioned at TPAC - also see Harald’s slides)

## Browser SFU/MCU - Cloud Conferencing -

A Browser could act as an SFU/MCU for small groups

- A group member can encrypt and send copies of the encoded media directly to multiple group members without the intervention of the media server

## Tldr; New use cases

- Decentralized web over datachannel for Matrix, |pipe| etc
- Low latency P2P broadcast for Auctions, Betting etc.
- Reduced Complexity Signalling

## Decentralized internet

New decentralized applications provide P2P services and client-server services for consumption in a browser.
The differences in transport layer semantics make it difficult to share code between the 2 modes.

- Ability to intercept the fetch API and service it over a P2P link. (one way to do this would be to support Datachannels in Service Workers - which can already intercept fetch)

(Matrix and |pipe| as examples.)

## Low latency P2P broadcast

There are 'broadcast' applications that require low latency realtime media - Distributed auctions or betting for example.
The same live video and audio (+data) should be sent to hundreds of recipients
WebRTC 1.0 can do this, but it lacks some features that those industries require and have in higher latency streaming technologies.

- Predictable auto-play for media elements that works for first time users and is testable.
- Ability to reuse DRM assets streamed over the data channels
- Ability to reuse subtitle assets streamed over the data channels

## Reduced complexity signalling

Some (simpler) media/data sources/sinks do not require the full array of webRTC's optionalities. In such cases the full SDP O/A could be replaced with offline static configuration and a simple URI at runtime.

- A URI format that defines the remaining transport related fields eg service address/port, ICE credentials, DTLS fingerprint

## Summary

Existing

- Improve remote UX for pre-empted media calls
- Extend IoT use-case to work in isolated networks
- Non-discriminatory FunnyHats
- Add local in-browser SFU/MCU for conferencing

New

- Decentralized web over datachannel for Matrix, |pipe| etc
- Low latency P2P broadcast for Auctions, Betting etc.
- Reduced Complexity Signalling

## getCurrentBrowsingContextMedia

(Elad Alon)

Save users from accidentally sharing their tentative job-search during their 1:1 with their current manager.

State of the art:

- getDisplayMedia allows the app to call upon the user to pick a share-source.
- The source cannot be restricted by the app; i.e. the app cannot specify that it's interested in tab-sharing.
- Some risk of user sharing wrong thing; e.g. whole screen or a different tab .

Proposal:

- Add getBrowserContextMedia - allow app to prompt the user for permission to share the current tab.

Benefits:

- Apps can reduce the risk of the user sharing the wrong thing. Avoid users accidentally sharing their job-search with their manager.
- The app can intelligently process captured content, based on intimate knowledge with the placement of elements in the captured stream - cropping, annotations, etc.

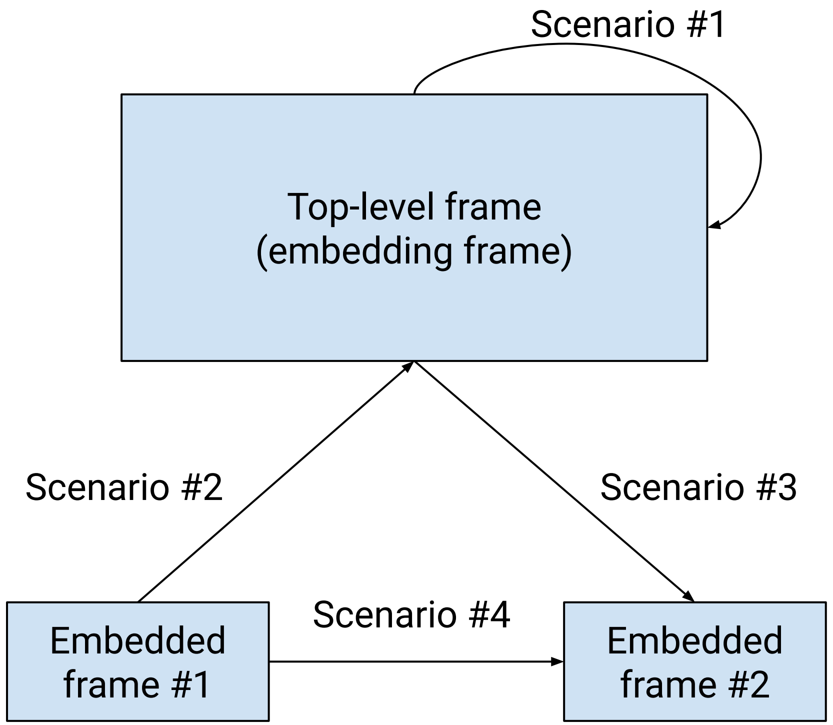
Use cases:

- Sharing of a document (e.g. Google Slides) to a meeting can be done via a button-click from the page, with a browser prompt to allow/disallow, rather than to choose the current tab.
  - **Also**, with getBrowserContextMedia, an app like Google Slides needs not worry about:
    1. Detecting if the user chose another tab by mistake (technically difficult).
    2. Explaining the mistake to the user and asking him to try again.
- Recording a [video-conferencing / gaming / etc.] session to a local file and/or streaming it (e.g. Twitch).
- Useful for embedding defect-reporting mechanisms in an application - can capture a video of the bug.

The main security issue is the ability of a page to capture output from other sites, breaking the origin-isolation model of Web security.

- Ancient Scandinavian proverb

## Risks


Scenario #1
Top-level frame
(embedding frame)
Scenario #2
Scenario #3
Scenario #4
Embedded frame #1
Embedded frame #2

## 1. Browsing context capturing itself

Browsers attempt to prevent applications from spying on the user. For example, to prevent applications from deducing whether a user has visited certain other websites, Chrome intentionally mis-reports to the application the color of links (unless they have not been set to a non-default color by the application itself); this prevents the application from finding out if links are purple, meaning that they have been visited.

Screen-capture allows an application to circumvent these protections - anything that's visible as purple to the user, will be seen as purple by the capture, including links.

When examining this particular example, some ways to preserve the privacy of the user's history come to mind, including:

- Forcing links' color to the default color if a screen-capture is active.
- Denying new screen-captures, and breaking off active screen-captures, if a default-color link is ever visible on the page.

Even when considering just this one example (link-purpling), we cannot think of any cure which would not be more bitter than the disease. We conclude that the only reasonable mitigation would be to make the user acutely aware of the risks of permitting screen-capture. The confirmation-dialog will remain our only mitigation of these risks as a whole, but individual issues could be given special attention as they come up. For example, if link-purpling is ever considered sufficiently dangerous, we could apply one of the aforementioned mitigations that are specific to that issue. We don't, however, have a general, scalable solution for concealing the user's history/identity from the application.

## 2. Embedded resource capturing the embedding resource

*Risk*

An embedded resource (e.g. iframe) might capture information from the embedding resource (e.g. top-level application).

*Mitigation*

The display-capture feature-policy solves this issue for getDisplayMedia, and can do so for getCurrentBrowsingContextMedia as well. Namely, code from a cross-origin resources  is not allowed to call either of these functions, unless the iframe-tag includes the relevant allow attribute, specifying allow="display-capture”.

## 3. Embedding resource capturing an embedded resource

*Risk*

An embedding resource (e.g. top-level application) might capture information from an embedded resource (e.g. iframe).

*Mitigation's Limitations*

1. Common and desirable scenario. The mitigation must allow approved captures to proceed uninterrupted - there should be no break in the capture while determining whether the capture is allowed. This rules out hypothetical mitigations that would pause the capture until the embedded resource fully/partially loads, and leaves us with a feature-policy communicated via an HTTP header as our only option.
2. Should not be so restrictive as to prevent real usage of the feature. An opt-in mechanism would be prohibitively restrictive - complex applications would require too many web-servers to be configured (and configuration maintained). We are forced to consider only mechanisms that default to allowing the capture. (Also helpful if applying the mitigation to getDisplayMedia.)
3. Existing mechanisms would be problematic to use, because they would tie together different functionalities that the embedded resource might not be interested in combining. We are compelled to introduce a new feature-policy.

*Mitigation*

Define a new feature-policy, allow-capture-by-embedder, settable via an HTTP response header, but not via an iframe's allow attribute. A resource served with this HTTP response header may only be captured by allowlisted embedders.

## <u>allow-capture-by-embedder</u>

When <u>allow-capture-by-embedder</u> is specified for a resource:

1. Attempts to capture the display by the embedder are only allowed if the embedder is allowlisted (or is same-origin).
2. Any running display-capture by the embedder is <u>broken off</u>.

The policies supported by <u>allow-capture-by-embedder</u> will be:

- <u>‘*’:</u> Capturing is allowed by any embedder. This is the default, and applies either if the policy is not specified at all, or if it’s specified without an explicit policy value.
- <u>‘self’:</u> Capturing is allowed by embedders from the same origin only. (Note that same-origin embedders may nevertheless transfer this permission to resources from other origins.)
- <u>‘none’:</u> No embedders are allowed to capture (until they stop embedding this resource).
- <u>‘&lt;origin(s)&gt;:</u> This policy can be added to ‘self’. It specifies specific origins which are still allowed to capture the current resource. (This permission is still transitive.)

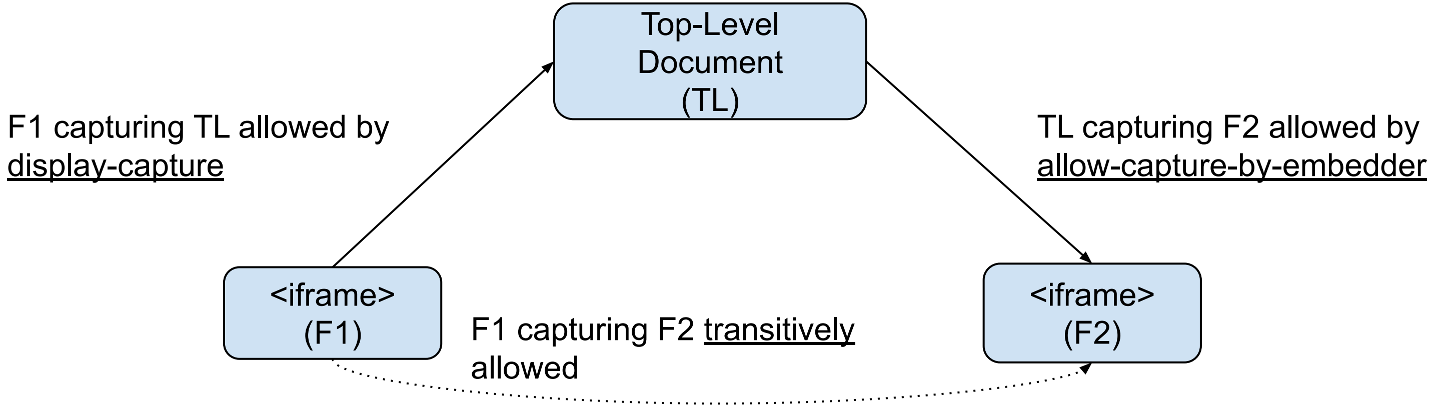
# 4. One embedded resource capturing another embedded resource

*Risk*

If multiple resources are embedded, any one of them could end up capturing any of the others.

*Mitigation*

We define the mitigation of scenario #2 as transitive - any embedded resource that allows its embedder to capture it, assumes this permission to be inherited by any resource that is allowed to capture the embedder itself.


Top-Level Document (TL)
F1 capturing TL allowed by display-capture
TL capturing F2 allowed by allow-capture-by-embedder
<iframe> (F1)
F1 capturing F2 transitively allowed
<iframe> (F2)

That is, assume TL is the top-level document, and it embeds to iframes, F1 and F2. If F2 allows itself to be captured by TL (managed by allow-capture-by-embedder), and TL allows itself to be captured by F1 (using display-capture), then F2 transitively allows itself to be captured by F1.

## First-class HTML capture

**navigator.mediaDevices.getDocumentMedia()**

**or alternatively named**

**document.captureStream()**

**(Jan-Ivar)**

## First-class HTML capture Goals

**Goals:**

1. Solve prohibitive UX flow for “record this meeting” & “Present Google Slides”.
2. Promote web (presentations) over native, using safer integrated HTML capture.

**Stream thyself!**

Google’s neat idea: Google Slides streaming itself into an ongoing meeting using existing tech like RTCPeerConnection. Appealing: page only needs to capture itself; requiring code in the target ensures buy-in; no outside capture threat.

**Stream thyself!**

Record web conference in progress from client perspective, including web layout.

## Stream thyself! = document

**Stream thyself!**

**=**

**capture document and down (document’s iframes and sub-iframes)**

**but not up**

**(i.e. an iframe cannot capture its embedder)**

**Lets sites use iframes to capture only what they want (no cropping needed)**

## Security Risks (even with Stream thyself)

1. **Cross-origin iframe capture** (rendered with user’s cookies from other site) in a manner that can be automated in active attacks on multiple sites.

2. **User information harvesting** from same-origin & cross-origins:
   a. Link purpling (browser history)
   b. Form autofill (address, credit card info)
   c. Extensions (e.g. LastPass)
   d. Font size (vision, age), coloring, and other preferences
   e. File input element may contain private info

## Poor mitigation for Cross-origin iframe capture

Simple HTML header easily defeated by wrapping an iframe in another iframe 👎

```
// A.com/index.html
// Headers: none
<iframe src="https://B.com/wrapper_exploit.html" allow="html-capture">

// B.com/wrapper_exploit.html
// Headers: Allow-Capture-By-Embedder: *
<iframe src="https://C.com/index.html" allow="html-capture"> // 🔴 captures C!


// C.com/index.html
// Headers: none
```

...unless we continuously check headers all the way down, but we're chasing DOM changes (iframes added, iframe navigation, redirects etc. 😮) = Poor man's COEP.

## No mitigation for Cross-origin video & image capture

Simple HTML header doesn’t protect non-iframe cross-origin resources at all 👎

```
// A.com/index.html
// Headers: none
<video src="https://C.com/video1234567890.webm"> // 🔴 captures video from C!
<iframe src="https://B.com/iframe.html" allow="html-capture">

// B.com/iframe.html
// Headers: Allow-Capture-By-Embedder: *
<video src="https://C.com/video1234567890.webm"> // 🔴 captures video from C!

// C.com/video1234567890.webm
// Headers: none
```

Affects both videos and images (<img>)

## Mitigate Cross-origin iframe capture with COEP

We can leverage Cross-Origin-Embedder-Policy to solve this (mockup) 👍

```
// A.com/index.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
<iframe src="https://B.com/wrapper_exploit.html" allow="html-capture">

// B.com/wrapper_exploit.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
//          Cross-Origin-Resource-Policy: capture ←------- new
<iframe src="https://C.com/index.html" allow="html-capture"> // C not loaded


// C.com/index.html
// Headers: none
```

COEP ensures iframes are origin clean or (mockup) explicitly opted-in to capture.

This being a powerful feature, a higher bar of entry = a good thing.

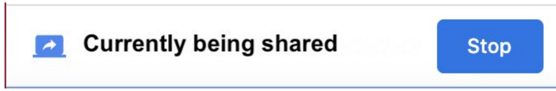
## Mitigate Cross-origin video & image capture with COEP

COEP already requires videos & images be either CORP or CORS 👍

```
// A.com/index.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
<video src="https://C.com/video1234567890.webm"> // C not loaded
<iframe src="https://B.com/iframe.html" allow="html-capture">

// B.com/iframe.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
//          Cross-Origin-Resource-Policy: capture
<iframe src="https://C.com/index.html" allow="html-capture"> // C not loaded


// C.com/video1234567890.webm
// Headers: none

// D.com/publicVideo.webm
// Headers: Cross-Origin-Resource-Policy: cross-origin (or Access-Control-Allow-Origin: *)
```

# Mitigate User information harvesting

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



- 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 mute/pause while `document.visibilityState != “visible”`
  - MUST require current global object == relevant global object, to reject `iframe.contentWindow.navigator.mediaDevices.getDocumentMedia()`

SHOULD allow User Agents to further attempt to mitigate information leakage

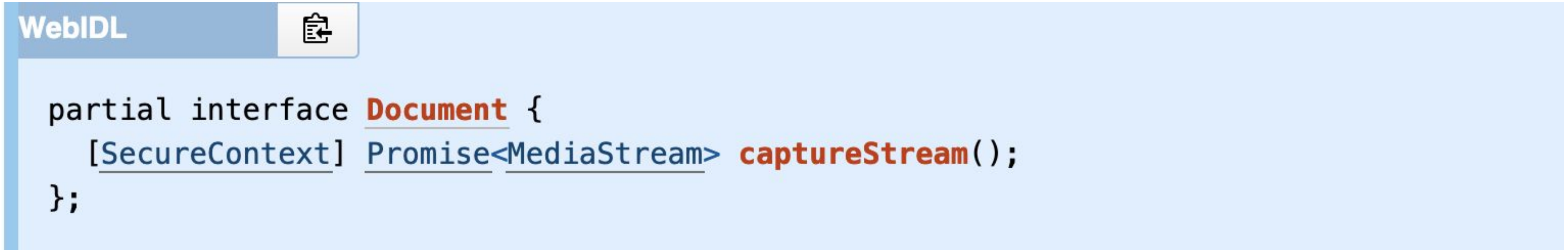
## Bikeshed on API

Good idea to define API under **mediacapture-screen-share**, to leverage existing protections there like the Privacy Indicator Requirements, but we still can choose

```
    await navigator.mediaDevices.getDocumentMedia();
```

Or

```
    await document.captureStream();
```



```
partial interface Document {
  [SecureContext] Promise<MediaStream> captureStream();
};
```

Ask: Let’s try to keep any bikeshedding orthogonal to discussions over functionality.

## Bikeshed on Permissions policy

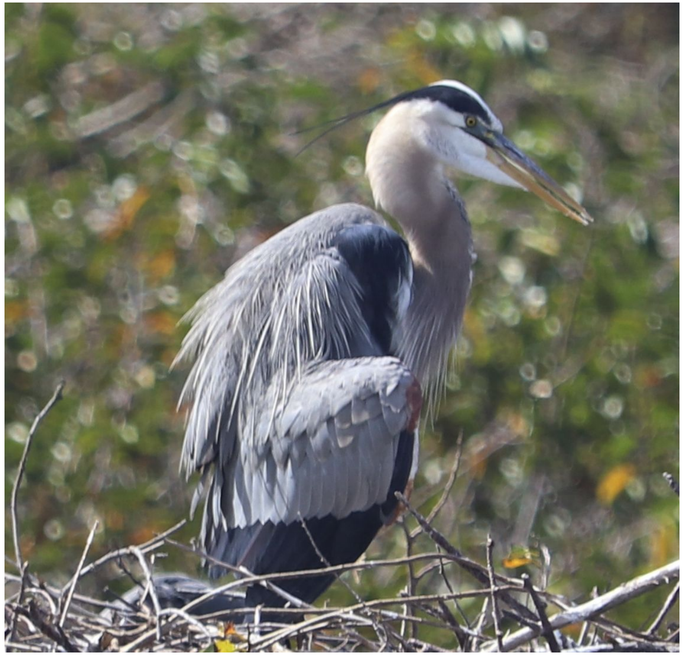
Sites use "display-capture" today to enable the highly-user-driven `getDisplayMedia` picker API:

```
<iframe src="https://B.com/index.html" allow="display-capture">
```

Given the security risks are slightly different, should we define a new policy?

```
<iframe src="https://B.com/index.html" allow="html-capture">
```

## For extra credit



**Name that Bird!**

# Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird