W3C WebRTC WG Meeting

July 22, 2021
10:00 AM - Noon 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 July 2021 interim meeting of the W3C WebRTC WG, at which we will cover:
  - MediaStreamTrack transfer
  - Mediacapture-transform alternative proposal
  - Screen Capture

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/July_22_2021
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - https://w3c.github.io/mediacapture-extensions/
  - 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?

Understanding Document Status

- Hosting within the W3C repo does ***not*** imply adoption by the WG.
  - WG adoption requires a Call for Adoption (CfA) on the mailing list.
- Editor's drafts do ***not*** represent WG consensus.
  - WG drafts ***do*** imply consensus, once they're confirmed by a Call for Consensus (CfC) on the mailing list.

W3C Code of Conduct

- This meeting operates under W3C Code of Ethics and Professional Conduct

- We're all passionate about improving WebRTC and the Web, but let's all keep the conversations cordial and professional

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 may be used to gauge the “sense of the room”.**

Issues for Discussion Today

- 10:10 AM - 10:25 AM MediaStreamTrack Transfer
  - Slides by Youenn (10:10 - 10:15)
  - Discussion (10:15 - 10:25 AM)
- 10:25 AM - 11:00 AM Mediacapture-transform Callback Proposal (Youenn)
  - Slides by Youenn (10:25 - 10:45)
  - Discussion (10:45 - 11:00)
- 11:00 AM - 11:30 AM Screen Capture (Jan-Ivar)
  - Slides by Jan-Ivar (11:00 - 11:15)
  - Discussion (11:15 - 11:30)
- 11:30 AM - 12:00 PM 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.

#29 MediaStreamTrack Transfer: Summary

- 05/21 interim meeting: 'transferable' option presented
  - 'transferable' option quote from 05/21 interim meeting
    - Default operation is “move” - original track is destroyed; if needed, must use “.clone()”
    - Lifetime strongly tied to original document
  - Conservative lifetime management
    - Works with all existing MediaStreamTrack sources
    - Works with existing security and privacy infrastructure
    - Works with all implementations
- How to complete specification?
  - Core algorithm already specified in mediacapture-extensions
    - PR 21 merged
    - Some pieces have not landed yet.

#29 MediaStreamTrack Transfer: editorial finalization

- Step 1: Clarify that, by default, all sources are tied to their creation context
  - Existing User Agent behavior
  - Leave the door open to new types of source with a different lifetime
  - PR#805 in mediacapture-main

- Step 2: Clarify MediaStreamTrack transfer behavior derived from step 1
  - If original document goes away, transferred track gets ended
  - Leave the door open to new types of source with a different lifetime
  - PR#30 in mediacapture-extensions

Discussion (10:15 to 10:25)

- How to determine consensus?
  - Slides at a meeting do ***not*** imply consensus.
  - Can issue a CfC on individual issues, or for promotion of MediaCapture-Extensions to WG draft.
- Questions?
- Opinions?

Mediacapture-transform Alternative Proposal (10:25 - 10:45)

#23 Towards more JS in MediaStreamTrack

- Goal 1: Enable safe & efficient access to MediaStreamTrack VideoFrame(s)
  - Head detection, object tracking…

- Goal 2: Enable transforming a native video MediaStreamTrack
  - CameraTrack → background blur → TransformedTrack
  - Use TransformedTrack as if it was CameraTrack
    - Including CameraTrack behavior
      - Muted, enabled, applyConstraints…

- Proposed API is illustrative
  - Explainer: https://github.com/youennf/mediacapture-extensions/blob/main/video-raw-transform.md
  - Suggestions highly welcomed!

#23 Why only focusing on video?

- Audio processing best practice != ReadableStream<AudioBuffer>
  - Store audio data in a ring buffer (SharedArrayBuffer)
  - I/O in worklet, audio processing in worker
    - The same worker can do video processing for synchronized A+V applications
  - Sources
    - https://developers.google.com/web/updates/2018/06/audio-worklet-design-pattern
      - "It uses the Audio Worklet as a simple "audio sink" and does everything in the Worker"
    - https://github.com/padenot/ringbuf.js/
      - "Sending audio from a non-real-time thread to a real-time thread is sometimes useful"
      - "The opposite is very useful"
- Uncertainty in exposing a built-in audio/video generic mechanism
  - Chances are high it will be suboptimal compared to app-specific
  - This proposal supports app-specifc audio/video mechanisms
- We can add a generic mechanism based on this proposal later on
  - If we find out we need it
  - In the meantime, worklet fills the gap as it can shim MSTP (& more)

#23 Why not using WhatWG streams?

- MediaStreamTrack is an augmented ReadableStream<VideoFrame>
  - Manages more than a queue of VideoFrame(s)
    - muted/unmuted, enabled, constraints, capabilities, settings...
  - **BUT MediaStreamTrack lacks a reader and a JS constructor**
  - Let's add this missing API!
    - It is best to add direct API instead of lossy converters
      - App-specific lossy converters can be written in JS

- Avoid WhatWG Streams edge cases
  - Identified by WebCodec team for WebCodec API
  - Identified by WebRTC WG members for MediaStreamTrack
    - buffering, teeing, 'optimized-but-underspecified' transfering
- Stay consistent with tightly related APIs
  - WebCodec and WebAudio

#23 Goal 1: Reading a video MediaStreamTrack (1/3)

- Do we want an equivalent to ReadableStream Reader?
  - Not really, reader.closed ↔ track.onended, cancel/stop
  - We only need reader.read()
    - Or an HTMLVideoElement.requestVideoFrameCallback equivalent

- Proposal: add 1 new callback to get video frames
  - A la requestVideoFrameCallback
  - Keep using existing MediaStreamTrack API for state management

```
async function transform(frame) {
  ...
}
// start processing
track.processVideoFrame = transform(frame)
// finish processing
track.processVideoFrame = null;
```

```
partial interface MediaStreamTrack {
  [Exposed=Worker] attribute VideoFrameCallback
processVideoFrame;
};
callback VideoFrameCallback = Promise<undefined>(VideoFrame);
```

#23 Goal 1: Reading a video MediaStreamTrack (2/3)

- Callback returns a promise. Once promise is settled:
  - User Agent can call the callback again
    - Enable backpressure
  - VideoFrame is closed automatically
    - Control lifetime of exposed VideoFrame

```
async function transform(f) {
  // Use 'f' as long as you are in
  ...
  // 'f' will be closed when exiting
}

track.processVideoFrame = (f) => {
   // transform returns a promise
  return transform(f);
}
```

- Consequences
  - If promise takes too much time to resolve, frames are dropped
  - If application needs to store a frame, it has to clone it explicitly

- Easy synchronization with MediaStreamTrack events
  - Muted event → callback no longer called until unmuted event fires
  - Ended event → callback no longer called

#23 Goal 1: Reading a video MediaStreamTrack (3/3)

- Comparison between callback vs. stream based approaches

```
async function transform(frame) {
  webCodecEncoder.encode(frame);
}

track.processVideoFrame = transform;
```

```
async function transform(frame) {
  webCodecEncoder.encode(frame);
}

const processor = new MediaStreamTrackProcess(track);
const reader = processor.readable.getReader();
let chunk = await reader.read();
while (!chunk.done) {
  await transform(chunk.data);
  // Probably need try/catch to ensure frame gets always
closed
  chunk.data.close();
  chunk = await reader.read();
};
```

#23 Goal 2: MediaStreamTrack JS Source - level 1

- Construct a MediaStreamTrack from a JS source
  - Reuse ReadableStream JS constructor model

```
const stream = await getUserMedia({video: true});
const originalTrack = stream.getVideoTracks()[0];

// originalTrack --> transform --> transformedTrack
const source = {
  start: (controller) => {
    originalTrack.processVideoFrame = async (frame) => {
      // Tranform frame from originalTrack
      const newFrame = await transform(frame);
      // Enqueue newFrame in transformedTrack pipeline
      controller.enqueue(newFrame);
    };
    // Stop track when originalTrack is stopped
    originalTrack.onended = () => controller.stop();
  },
  // When track is stopped, stop frame transforming
  stop: () => originalTrack.processVideoFrame = null
};
const transformedTrack = MediaStreamTrack.createVideoTrack(source);

// Use transformedTrack instead of originalTrack
pc.addTrack(transformedTrack, stream);
```

```
partial interface MediaStreamTrack {
  [Exposed=Worker] static MediaStreamTrack
        createVideoTrack(VideoSource source);
};

dictionary VideoSource {
  // Called synchronously by createVideoTrack
  VideoSourceStartCallback start;
  // Called when track.stop is called
  VideoSourceStopCallback stop;
};

[Exposed=Worker]
interface VideoSourceController {
  undefined enqueue(VideoFrame frame);
  undefined stop();
};

callback VideoSourceStartCallback =
                any(VideoSourceController);
callback VideoSourceStopCallback = any();
```

#23 MediaStreamTrack & WhatWG streams

- Integration with WhatWG streams is easy!
  - Just a few lines of tweakable code

```
// MediaStreamTrackProcessor equivalent

class MyMediaStreamTrackProcessor {
  constructor(track) {
    this.readable = new ReadableStream({
      start : (c) => {
        track.onended = () => c.close();
        const p = new Promise(r => this.firstFrame = r);
        track.processVideoFrame = (frame) => {
          c.enqueue(frame.clone());
          this.firstFrame();
          return new Promise(r => this.nextFrame = r);
        };
        return p;
      },
      pull : (c) => this.nextFrame(),
      close : () => track.processVideoFrame = null,
    });
  }
}
```

```
// MediaStreamTrackGenerator equivalent

function createMyMediaStreamTrackGenerator() {
  const controllers = { };
  // Create a MediaStreamTrack object
  const generatedTrack = MediaStreamTrack.createVideoTrack({
    start: (c) => controllers.track = c,
    stop:   () => controllers.stream.error()
  });
  // Create a WritableStream
  generatedTrack.writable = new WritableStream({
    start : (c) => controllers.stream = c,
    write : (f) => controllers.track.enqueue(f),
    close :  () => controllers.track.stop(),
    abort :  () => controllers.track.stop()
  });
  return generatedTrack;
}
```

#23 Goal 2: MediaStreamTrack JS Source - level 1

- What about directly passing a WritableStream to createVideoTrack?

```
MediaStreamTrack.createVideoTrack(VideoTrackSource source)

vs.

MediaStreamTrack.createVideoTrack(WritableStream stream)
```

- Passing a WritableStream to createVideoTrack is not enough
  - Need to handle more than VideoFrame(s)
    - For instance muted/unmuted MediaStreamTrack state
      - Let's go to level 2!

#23 Goal 2: MediaStreamTrack JS Source - level 2

- Support for muted/unmuted
  - Use case: if original track gets muted, mute the transformed track

```
const transformedTrack = MediaStreamTrack.createVideoTrack({
  start: (c) => {
    originalTrack.processVideoFrame = transform;
    // Handle camera track muted state.
    originalTrack.onmuted   = () => c.muted = true;
    originalTrack.onunmuted = () => c.muted = false;
  }
});
...
transformedTrack.onmuted = () => { ... };
pc.addTrack(transformedTrack, stream);
```

```
partial interface VideoSourceController {
  attribute boolean muted;
};
```

- A single control place, natural to use
  - Controller enqueues frames
  - Controller can mute/unmute its track
  - Controller is muted → controller does not enqueue frames

#23 Goal 2: MediaStreamTrack JS Source - level 3

- Let's start simple, but make sure we can easily extend the API

- Support for track.enabled changing to true or to false
  - Use case: if web app disables transformed track, disable the original track

- Support for capabilities, constraints and settings
  - Use case: if calling applyConstraints on the transformed track, apply the constraints to the original track

#23 MediaStreamTrack JS Source - the whole thing!

```
// video-call-with-a-dragon-head.html
async function getCameraTrackWithDragonHead()
{
  const stream = await
      navigator.mediaDevices.getUserMedia({video:true});
  const source = stream.getVideoTracks()[0];
  const worker = new Worker("dragon-head-worker.js");
  let resolve;
  const promise = new Promise(r => resolve = r);
  worker.onmessage = event => resolve(event.data.track);
  worker.postMessage({track:source}, [source]);
  return promise;
}

const track = await getCameraTrackWithDragonHead();

// Use track as if coming from getUserMedia, including:
// - Using clone, applyConstraints et al
// - Receiving muted/unmuted events from source
// - Set enabled=true/false, propagating to source
...
track.onmuted = () => { ... };
```

```
// dragon-head-worker.js
onmessage = (event) => {
  const track = event.data.track;
  const transformed = MediaStreamTrack.createVideoTrack({
    start: (c) => {
      this.controller = c;
      track.onended = () => c.stop();
      // transform then enqueue each camera frame
      track.processVideoFrame =
          async (f) => c.enqueue(await transformHead(f));
      // propagate camera muted state to track
      track.onmuted = () => c.muted = true;
      track.onunmuted = () => c.muted = false;
      // propagate camera settings to track
      c.capabilities = track.capabilities();
      c.constraints = track.constraints();
      c.settings = track.settings();
    },
    applyConstraints: async (constraints) => {
      // propagate applyConstraints to camera
      await track.applyConstraints(constraints);
      this.controller.constraints = track.constraints();
      this.controller.settings = track.settings();
    },
    // propagate enabled to camera
    enabledChanged: (value) => track.enabled = value,
    stop: () => track.processVideoFrame = null
  });
  self.postMessage({track:transformed}, [transformed]);
});
```

#23 Beyond Goal 2

- Extend API to handle native sink backpressure signals
  - HTMLMediaElement, RTCPeerConnection, WebAudio
  - Maybe, we need to do further investigation
  - Easy to support should there be a need

- Extend API to handle transforms
  - In particular native transforms like synthetic backgrounds
  - MediaStreamTrack → MediaStreamTrack
  - WASM/WebGPU (or even JS) dedicated transforms
    - Help authoring & use of good practices
      - Remove boiler plate code by safe & efficient code
    - Possibility for User Agent optimizations
      - Zero memory copy

#23 Tentative conclusion

- Simple: 2 MediaStreamTrack methods + 1 new interface
  - Easy to learn & use, blends well with existing APIs
- Safe & efficient
  - No buffering by default, implicit VideoFrame closing
  - Worker by default, at least initially until we gather more experience
- Powerful
  - Reduced API set for most useful functionality
  - Full MediaStreamTrack shiming support
- Flexible
  - Clear path towards native transforms, zero copy...
  - WhatWG stream bridge with a few lines of code

Discussion (10:45 - 11:00)

- Questions?
- Opinions?

Screen Capture (Jan-Ivar)

- #182 - Recognize safer & better-integrated web presentations in getDisplayMedia
- #158 - Add ability to crop a MediaStream obtained through the share-this-tab API

Screen-capture

#1 use case:
"Screen sharing Using WebRTC" = VC Presentations

2021 = web presentations

#1 use case is unsafe!

<u>We must fix this!</u>

W3C Working Draft

Screen Capture

W3C Working Draft 10 June 2021

This version:
https://www.w3.org/TR/2021/WD-screen-capture-20210610/

Latest published version:
https://www.w3.org/TR/screen-capture/

Latest editor's draft:
https://w3c.github.io/mediacapture-screen-share/

§ 1. Introduction

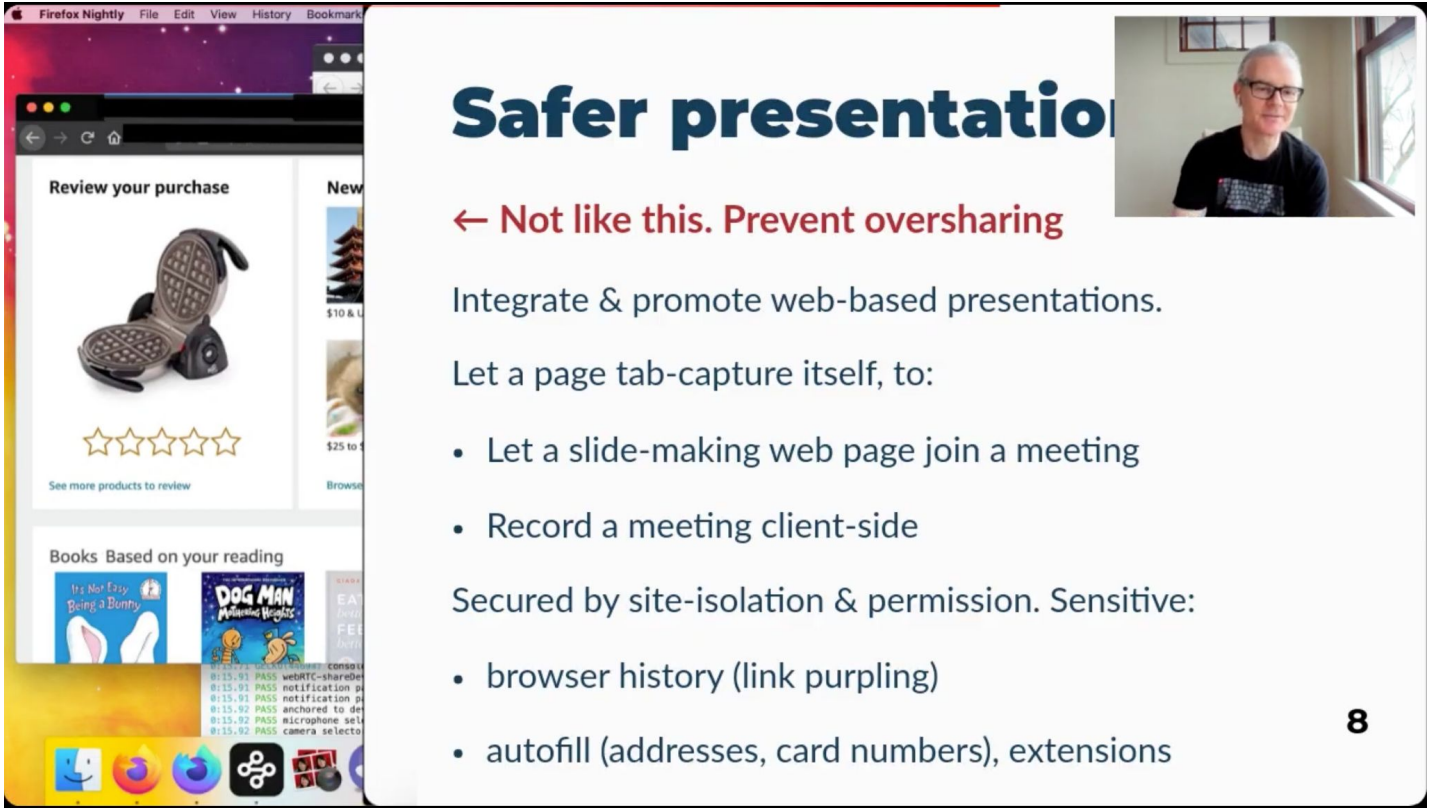
This section is non-normative.

This document describes an extension to the Media Capture API [GETUSERMEDIA] that enables the acquisition of a user's display, or part thereof, in the form of a video track. In some cases system, application or window audio is also captured which is presented in the form of an audio track. This enables a number of applications, including screen sharing using WebRTC [WEBRTC].

This feature has signficant security implications. Applications that use this API to access information that is displayed to users could access confidential information from other origins if that information is under the control of the application. This includes content that would otherwise be inaccessible due to the protections offered by the user agent sandbox.

This document concerns itself primarily with the capture of video and audio [GETUSERMEDIA], but the general mechanisms defined here could be extended to other types of media, of which depth [MEDIACAPTURE-DEPTH] is currently defined.

From presentation I gave @ W3C 2021 AC Meeting



“W3C WebRTC Working Group - April 2021 Update” youtu.be/watch?v=PQ5dGmqeuQI&t=419s

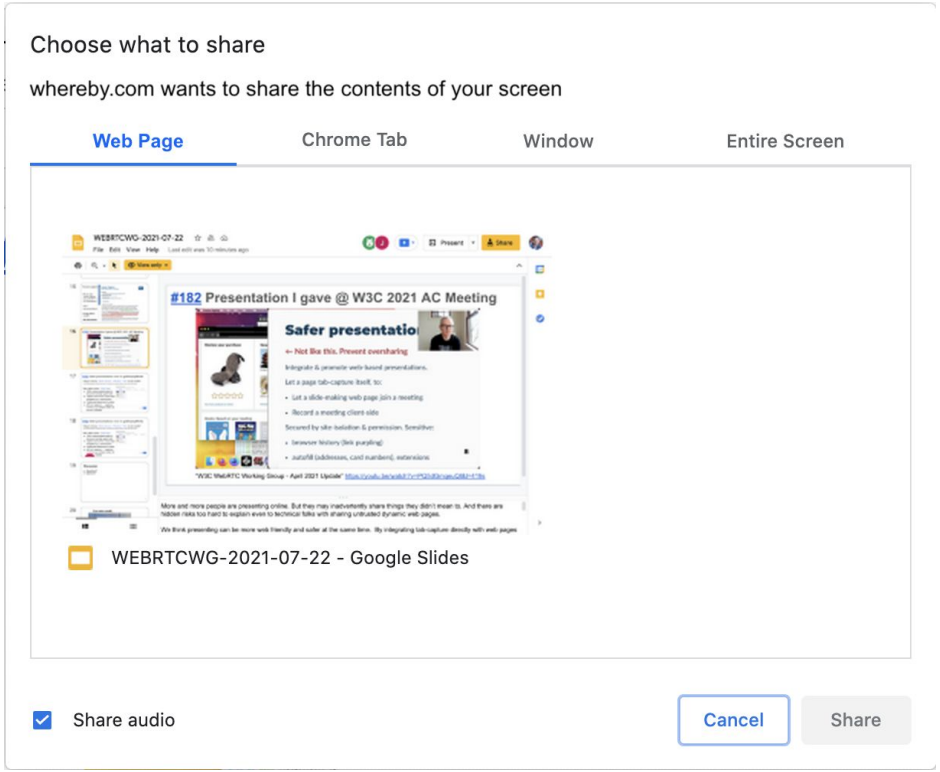
#182 Safer presentations even in getDisplayMedia

Today's choices, "Entire Screen", "Window", "Tab" are all unsafe!*

*Over-sharing (desktop, accidental back button or tab flip), even active malicious attacks on web's same-origin policy

New (safe) choice: "Web Page"

- TLBC w/site-isolated + opted-in document
- Capture turns off on cross-origin navigation
- Preferential placement in picker →
- 🍬 APIs for capturer↔capturee comms (e.g. next/prev slide, id) become available in safe garden



#182 Safer presentations even in getDisplayMedia

What does the spec need to facilitate this new choice in User Agents?

- Editorial: Name this new source as a concept. “web page”/site/app?
- Loosen Elevated Permission language in spec for these sources
- Encourage preferential placement of them vs tabs, w/SHOULD
- Specify requirements on *sites* to qualify as this new source:
  - window.crossOriginIsolated +
    - `Document-Policy: html-capture`
    - `Require-Document-Policy: html-capture`
- Editorial: Name these site requirements as a concept (“opted into html capture”?), bc may be same requirements as getViewportMedia

#158 crop MediaStream from the share-this-tab API

Proposal from Youenn:

```
navigator.mediaDevices.getTabViewportMedia() // tab viewport

document.getViewportMedia() // captures document viewport

iframe.getViewportMedia()   // captures iframe viewport
```

All 3 call the same underlying algorithm taking a viewport as internal parameter, which is responsible of permission policy, prompting, creation of track.

With this approach, there is no default option to select in the spec and no surprise from web developers on what they will get.

Easy feature detection of gradual support. Follow-up Q: Permissions policy? Has site-isolation & capture opt-in been resolved?

Discussion (11:15-11:30)

- Questions?
- Opinions?

Wrap-up and Next Steps (11:30-12:00)

For extra credit

Name the bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The mammal