W3C WebRTC WG Meeting

November 15, 2022
8 AM - 10 AM

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 November 2022 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Encoded-transform
  - Timing Model
  - Face Detection, Background Blur, etc.
  - Message Port on Capture Handle
  - mediacapture-main
- Future meetings:
  - December 7, 2022
  - January 17, 2023

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/November_15_2022
- 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 (still) being recorded. The recording will be public.
- Volunteers for note taking?

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 (still) 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”.**

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.
  - Possible to merge PRs that may lack consensus, if a note is attached indicating controversy.

Topics for Discussion Today

- 08:10 - 08:30 AM Encoded-transform (Harald)
- 08:30 - 08:50 AM WebRTC-PC
- 08:50 - 09:10 AM Face Detection, BBlur, etc. (Riju)
- 09:10 - 09:20 AM Timing Model (Bernard)
- 09:20 - 09:40 AM Message Port on Capture Handle
- 09:40 - 10:00 AM mediacapture-main (Jan-Ivar)

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.

Encoded-Transform (Harald)

Start Time: 08:10 AM

End Time: 08:30 AM

IETF Hackathon Report

- Declared a party at IETF 115's hackathon event
- A few people said "interested, but can't make it". Nobody came.
- I did some experimentation anyway.

Repository: https://github.com/alvestrand/hackathon-encoded-media

Suggested API proposal

- Modeled after Breakout Box (Producer/consumer)
- Add explicit signals in an API definition
- Followed pre-WG model of enabling frames, but with an API doing extraction and insertion in separate API calls
- Described in the repository

Demos written at hackathon

- Frame counter demo (“bump in the stack”)
  - Fully supported by shim that emulates proposed API
- Pass incoming track to outgoing track
  - Shim allows connecting the pieces, but doesn’t convert frame types (yet)

No demos using signals were produced.

API learnings

- We may need only producers
- Processors can be written as pass-through producers
- End consumers can be extended with an “insert frames” function that takes a producer
- Experimentation is good for API design

Example code - “frame counter”

```
const frameSource = audioPassingPc.getSenders()[0].divertFrames();

class CountingFrameSource extends EncodedFrameSource {
   constructor(firstSource) {
     super();
     const frameCounter = new TransformStream({
       transform: (frame, controller) => {
         frameCount += 1;
         audioCounterDiv.innerText = frameCount;
         controller.enqueue(frame);
       }
     });
     this.readable = frameSource.readable.pipeThrough(frameCounter);
   }
 };

 const outgoingFrameSource = new CountingFrameSource(frameSource);
 audioPassingPc.getSenders()[0].insertFrames(outgoingFrameSource);
```

Peter’s Hackathon

- Can hack existing “two way” encoded-media API into a “one way transport” API
- Can hack existing “two way” encoded-media API into a “one way codec” API
- Test “one way transport” from client to server at https://github.com/pthatcher/hackathon-encoded-media
- API lacks a few things, but I have some ideas for more hacks

Example (Send Transport)

```
const encodedStreams = transceiver.sender.createEncodedStreams();
const encodedFrameWriter = encodedStreams.writable.getWriter();
for (dataToSend) {
  // This is where we need a constructor
  const encodedFrame = new RTCEncodedAudioFrame(...);
  encodedFrame.data = dataToSend;
  encodedFrameWriter.write(encodedFrame);
  // But how do we know how much we can send?  Congestion control on top?
}
```

Example (Receive Transport)

```
const encodedStreams = transceiver.receiver.createEncodedStreams();
const encodedFrameReader = encodedStreams.readable.getReader();
(async() => {
  while (true) {
    const encodedFrame = (await encodedFramesReader.read()).value;
    // Do something useful with encodedFrame.data
  }
})();
```

Example (Encode)

```
const encodedStreams = transceiver.sender.createEncodedStreams();
const encodedFrameReader = encodedStreams.readable.getReader();
(async() => {
  while(true) {
    const encodedFrame = (await encodedFramesReader.read()).value;
    // Do something useful with encodedFrame.data
    // But how do you control bitrates and keyframes?
  }
})();
```

Example (Decode)

```
const encodedStreams = transceiver.sender.createEncodedStreams();
const encodedFrameWriter = encodedStreams.writable.getWriter();
for (dataToSend) {
  // This is where we need a constructor
  const encodedFrame = new RTCEncodedVideoFrame(...);
  encodedFrame.data = dataToSend;
  encodedFrameWriter.write(encodedFrame);
  // How do we know when a key frame is needed?
}
```

Things the API lacks

- A constructor for RTCEncodedAudioFrame/RTCEncodedVideoFrame
- A bandwidth estimate from “sender”
- A bandwidth control for “encoder”
- Generate key frame requests for “encoder”
- A “needs key frame” signal from “decoder”

Discussion (End Time: 08:30)

-

WebRTC-PC

Start Time: 08:30 AM

End Time: 08:50 AM

Issues for Discussion Today

- Issue 2795: Missing URL in RTCIceCandidateInit
- Issue 2796: A simulcast transceiver saved from rollback by addTrack doesn’t re-associate, but unicast does (Jan-Ivar)
- Issue 2724: The language around setting a description appears to prohibit renegotiation of RIDs (Jan-Ivar)

Issue 2795: Missing URL in RTCIceCandidateInit (1/4 Youenn)

- Spec excerpt:

  This interface describes an ICE candidate, described in [RFC5245] Section 2. Other than `candidate`, `sdpMid`, `sdpMLineIndex`, and `usernameFragment`, the remaining attributes are derived from parsing the `candidate` member in candidateInitDict, if it is well formed.

- RTCIceCandidate.toJSON provides full fidelity

```
const candidate = await getFirstCandidate();
const candidateJSON = candidate.toJSON();
const candidateCopy = new RTCIceCandidate(candidateJSON);
for (let v in candidate)
     assert_equals(candidate[v], candidateCopy[v]);
```

- Past discussion on exposing srflx/relay server URL on RTCIceCandidate did not address what we should do about this

Issue 2795: Missing URL in RTCIceCandidateInit (2/4 Youenn)

1. Do we want to share srflx/relay server URL to remote parties?
   - Or should we omit this information by default?
2. Do we want to keep RTCIceCandidate current model?
   - Or should we instead have a hybrid approach where some fields are not preserved?

Personal position

- Let's not expose srflx/relay server URL via toJSON
- The current full-fidelity model is simple and consistent, let's keep it
- We can keep server URL in RTCPeerConnectionIceEvent
  - Applications wanting the hybrid approach can easily implement it

```
pc.onicecandidate = event => {
    if (event.candidate)
        event.candidate.url = event.url;
    processCandidate(event.candidate);
}
```

Issue 2795: Missing URL in RTCIceCandidateInit (3/4 Philipp)

- the information in toJSON is what is is needed for ICE
  - sdpMid/sdpMLineIndex and candidate string
  - usernameFragment came later
- most can be extracted from the candidate line
  - relayProtocol can not
  - url can not
  - ufrag, generation or any extension attributes are not translated to the object

Issue 2795: Missing URL in RTCIceCandidateInit (4/4 Philipp)

- Why do we have extra properties on RTCIceCandidate?
  - to stop developers from parsing the candidate string
- Why are developers doing this?
  - learn about the network topology (e.g. is UDP blocked and fallback to TCP is required) or configuration (e.g TURN server which returns internal IP address)
- RTCIceTransport candidatepairchange event
  - when changing candidate-pair, you may want to know which server you ended up on (typically also for first connectionstatechange to connected)
  - currently done using getStats
  - exposing on candidate object automagically makes it available?

Issue 2796: A simulcast transceiver saved from rollback by addTrack doesn't re-associate, but unicast does

RFC8829 section 5.7 on rollback: *"... an RtpTransceiver **MUST NOT be removed if a track was attached** to the RtpTransceiver **via the addTrack method**. This is so that an application may call addTrack, then call setRemoteDescription with an offer, then roll back that offer, then call createOffer and have an "m=" section for the added track appear in the generated offer."*

IOW, addTrack saves 🎣 an sRD-created transceiver from the 🦈 jaws of rollback. 👏👏

But in Chrome, simulcast is rolled back to ridless unicast for transceivers created by addTrack, but not for transceivers created by sRD.

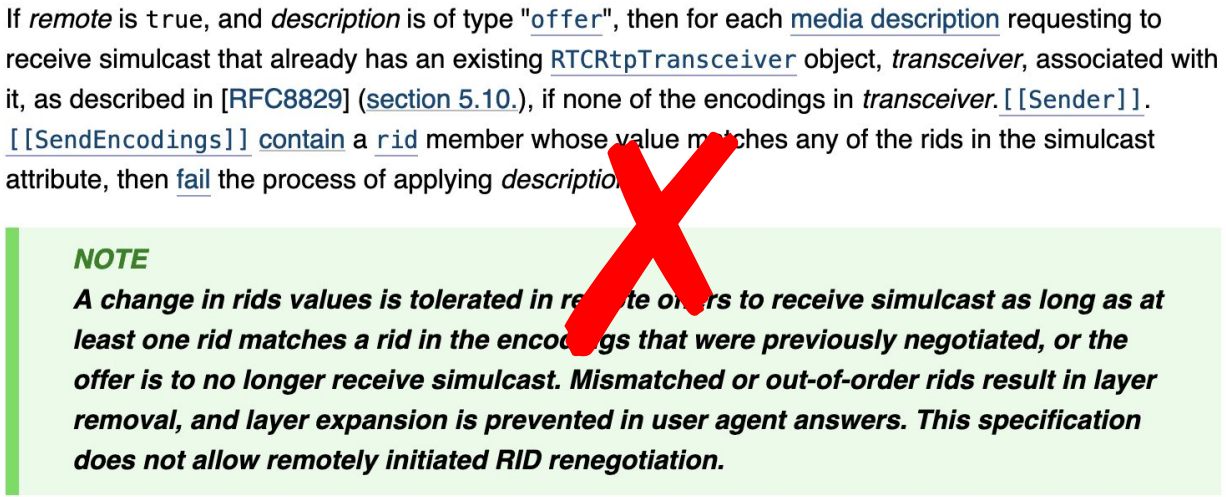
Proposal: roll back simulcast to ridless unicast for transceivers created by sRD as well.

Makes addTrack behave the same before and after sRD, and avoids the unassociated simulcast addTrack transceiver 🦄 (how would re-associating it even work?)

This seems a defensible interpretation of RFC 8829 (section 4.1.10.2) which says rollback *"discards any proposed changes to the session, returning the state machine to the "stable" state"*.

Issue 2724 / PR 2794: Don't fail sRD(offer) over simulcast rid mismatch, just answer with unicast. (1/2)

Last month left 1 remaining known rid as requirement to not fail sRD(simulcastReoffer):



5. If *remote* is `true`, and *description* is of type "`offer`", then for each media description requesting to receive simulcast that already has an existing `RTCRtpTransceiver` object, *transceiver*, associated with it, as described in [RFC8829] (section 5.10.), if none of the encodings in *transceiver*.`[[Sender]]`.`[[SendEncodings]]` contain a `rid` member whose value matches any of the rids in the simulcast attribute, then fail the process of applying *description*.

> ***NOTE***
> ***A change in rids values is tolerated in remote offers to receive simulcast as long as at least one rid matches a rid in the encodings that were previously negotiated, or the offer is to no longer receive simulcast. Mismatched or out-of-order rids result in layer removal, and layer expansion is prevented in user agent answers. This specification does not allow remotely initiated RID renegotiation.***

Proposal: **Let's remove it.** Implementations aren't failing, and it's simpler to just drop down to the first layer and answer with unicast to satisfy RFC8853 which has very few situations where an sRD(offer) should error out due to an inconsistency with something previously negotiated (things like removal of m-sections).

Issue 2724 / PR 2794: Don't fail sRD(offer) over simulcast rid mismatch, just answer with unicast. (2/2)

To aid with this, **PR 2794** also adds this line (in green):

7. If *description* is of type "`answer`" or "`pranswer`", and *transceiver*. `[[Sender]]`. `[[SendEncodings]]` .length is greater than 1, then run the following steps:

    1. If *description* indicates that simulcast is not supported or desired, or *description* is missing all of the previously negotiated layers, then remove all dictionaries in *transceiver*.`[[Sender]]`.`[[SendEncodings]]` except the first one and abort these sub steps.

    2. If *description* is missing any of the previously negotiated layers, then remove the dictionaries that correspond to the missing layers from *transceiver*.`[[Sender]]`.`[[SendEncodings]]`.

Chrome & Safari fail to reduce local encodings (crbug 1383336) but answer unicast

Discussion (End Time: 08:50)

-

Face Detection (Riju)

Start Time: 08:50 AM

End Time: 09:10 AM

Face Detection: For Discussion Today

PR: https://github.com/w3c/mediacapture-extensions/pull/78

- TPAC: face detection should fill `VideoFrameMetadata`
  - The PR updates spec and explainer accordingly
  - `DetectedFace` renamed to `HumanFace`
- Other feedback: make API minimal
  - most constraints removed
  - mesh representation removed
  - expressions removed
  - landmarks remain, currently supported by platforms

Face Detection: Metadata

```
partial dictionary VideoFrameMetadata {
  sequence<HumanFace>? humanFaces;
};

dictionary HumanFace {
  long?          id;              // Used for tracking
  float?         probability;     // 0 ... 1
  sequence<Point2D>               contour;
  sequence<HumanFaceLandmark> landmarks;
};
```

```
dictionary HumanFaceLandmark {
  HumanFaceLandmarkType  type;
  sequence<Point2D>      contour;
};

enum HumanFaceLandmarkType {
  "eye",
  "eyeLeft",
  "eyeRight",
  "mouth",
  "nose"
};
```

Note: must follow the requirements in the VideoFrame Metadata Registry. Must be serializable.

Face Detection: Metadata, details

```
dictionary HumanFace {
  long?          id;
```

If not null, the same (arbitrary) integer value in different frames within the same MediaStreamTrack indicates the same face being tracked over the frames.

```
  float?         probability;  // 0 ... 1
```

Approximate `probability ∈ (0,1]` that the detected face is indeed a human face. Could equal to 1 on synthetic images (ground truth).

```
  sequence<Point2D>              contour;
```

Coordinates similar to pointsOfInterest but the points may also lie outside of the frame. Represents locations in a normalized square space. The member can be used to describe also the **center point** (1 point) or a **bounding box** (4 points).

```
enum HumanFaceLandmarkType {
  "eye",
```

Either left or right eye if not known. When possible, more specific type (right or left eye) should be set.

```
  "eyeLeft",
  "eyeRight",
  "mouth",
  "nose"
```

Face Detection: Constraints

```
partial dictionary MediaTrackSupportedConstraints
{
  boolean faceDetectionMode =  true;
  boolean faceDetectionMaxContourPoints =  true;
};

partial dictionary MediaTrackCapabilities  {
  sequence<DOMString> faceDetectionMode;
  ULongRange    faceDetectionMaxContourPoints;
};

partial dictionary MediaTrackConstraintSet  {
  ConstrainDOMString faceDetectionMode;
  ConstrainULong
faceDetectionMaxContourPoints;
};
```

```
partial dictionary MediaTrackSettings  {
  DOMString faceDetectionMode;
  long      faceDetectionMaxContourPoints;
};

enum FaceDetectionMode  {
  "none",
  "contour",
  "landmarks",    // implies also “contour”
};
```

Note: application could also set metadata itself without enabling face detection on `MediaStreamTrack`.

Face Detection: Constraints, details

```
partial dictionary MediaTrackCapabilities {
  sequence<DOMString> faceDetectionMode;
```

Set of available modes, each string is one of the possible values in `enum FaceDetectionMode` If "`landmarks`" is supported on a track, user agent must support also "`contour`". The detection results with "landmarks" is a superset of results with "`contour`", ie. "`landmarks`" = "`contour`" + more.

```
  ULongRange   faceDetectionMaxContourPoints;
```

Maximum number of points in a face or a landmark contour, whichever is larger, that the user agent may return. If user agent can return a bounding box at best, `faceDetectionMaxContourPoints.max`= 4.

```
partial dictionary MediaTrackSettings {
  DOMString faceDetectionMode;
```

The negotiated face detection mode, must match one of the strings in `enum FaceDetectionMode` This controls which of the metadata is filled when a `VideoFrame` is pulled from a `MediaStreamTrack`.

```
  long       faceDetectionMaxContourPoints;
```

Maximum number of points in the contour arrays in `HumanFace` and `HumanFaceLandmark` dictionaries. User agent should try to return this many points for the contour but it is allowed to return less if the case requires, for example if detection algorithm partially fails.

User agent might allow application to set this to zero via constraints, if the application is not interested in feature location information but just its presence.

Face Detection: Example

```
// main.js:
// Open camera with face detection enabled
const stream = await
navigator.mediaDevices.getUserMedia({
  video: {
    faceDetectionMode: 'contour',
    faceDetectionMaxContourPoints: {exact: 4}
  }
});
const [videoTrack] = stream.getVideoTracks();

// Use a video worker
const videoWorker = new Worker('video-worker.js');
videoWorker.postMessage({track: videoTrack},
[videoTrack]);
const {data} = await new Promise(r =>
videoWorker.onmessage);
```

```
// video-worker.js:
self.onmessage = async ({data: {track}}) => {
  const generator = new VideoTrackGenerator();
  parent.postMessage({videoTrack: generator.track},
[generator.track]);
  const {readable} = new
MediaStreamTrackProcessor({track});
  const transformer = new TransformStream({
    async transform(frame, controller) {
      for (const face of
frame.metadata().humanFaces) {
        if ((face.contour || []).length == 4) {
console.log(`Face @ (${face.contour[0].x},
${face.contour[0].y}), (${face.contour[1].x},
${face.contour[1].y}), (${face.contour[2].x},
${face.contour[2].y}), (${face.contour[3].x},
${face.contour[3].y})`);
        }
      }
      controller.enqueue(frame);
    }
  });
  await
readable.pipeThrough(transformer).pipeTo(generator.w
ritable);
};
```

Discussion (End Time: 09:10)

-

Timing Model (Bernard)

Start Time: 09:10 AM

End Time: 09:20 AM

HTMLVideoElement.requestVideoFrameCallback()

§ 2. VideoFrameCallbackMetadata

```
dictionary VideoFrameCallbackMetadata {
  required DOMHighResTimeStamp presentationTime;
  required DOMHighResTimeStamp expectedDisplayTime;

  required unsigned long width;
  required unsigned long height;
  required double mediaTime;

  required unsigned long presentedFrames;
  double processingDuration;

  DOMHighResTimeStamp captureTime;
  DOMHighResTimeStamp receiveTime;
  unsigned long rtpTimestamp;
};
```

Discussion (End Time: 09:20)

-

MessagePort on Capture Handle (Elad)

Start Time: 09:20 AM

End Time: 09:40 AM

Why?

- Capture Handle allows the captured app to expose its identity to the capturing app.
- This can be used to kick-start communication.
- Local communication, like a BroadcastChannel, is superior to that over a shared cloud infrastructure. (Robustness, delay, efficient use of bandwidth.)
- However, local communication is hampered by Storage Partitioning.

What then?

- Expose a MessagePort between capturer and capturee.
- Leverage the existing Capture Handle for such events.

Why events?

- A given tab can be concurrently captured by multiple capturers.
- The captured tab’s top-level document may be navigated.
- The user can use surfaceSwitching to switch which tab is shared.
- The captured app might become ready to receive messages either before or after the capture starts.

Recall also that…

- Abiding design principle - the capturee only becomes alerted to the presence of a capture, if the capturer takes explicit action that reveals this.

Proposed API Shape - Capturee Side

```
dictionary CaptureHandleConfig {
  boolean exposeOrigin = false;
  DOMString handle = "";
  sequence<DOMString> permittedOrigins = [];
  EventHandler newCapturerEventHandler;
};

interface NewCapturerEvent {
  attribute Type type;  // "started" or "stopped"
  attribute MessagePort port;
}
```

Proposed API Shape - Capturer Side

```
dictionary CaptureHandle {
  ... // Pre-existing fields redacted.
  boolean supportsMessagePort;  // Open question - repeat in event?
};

interface CaptureHandleChangeEvent {
  attribute boolean messagePortInvalidated;
}

partial interface CaptureController {
  ... // Pre-existing fields redacted.
  MessagePort getMessagePort();
}
```

Fine Details

- For the fine details, please see the extended proposal here.

Discussion (End Time: 09:40)

-

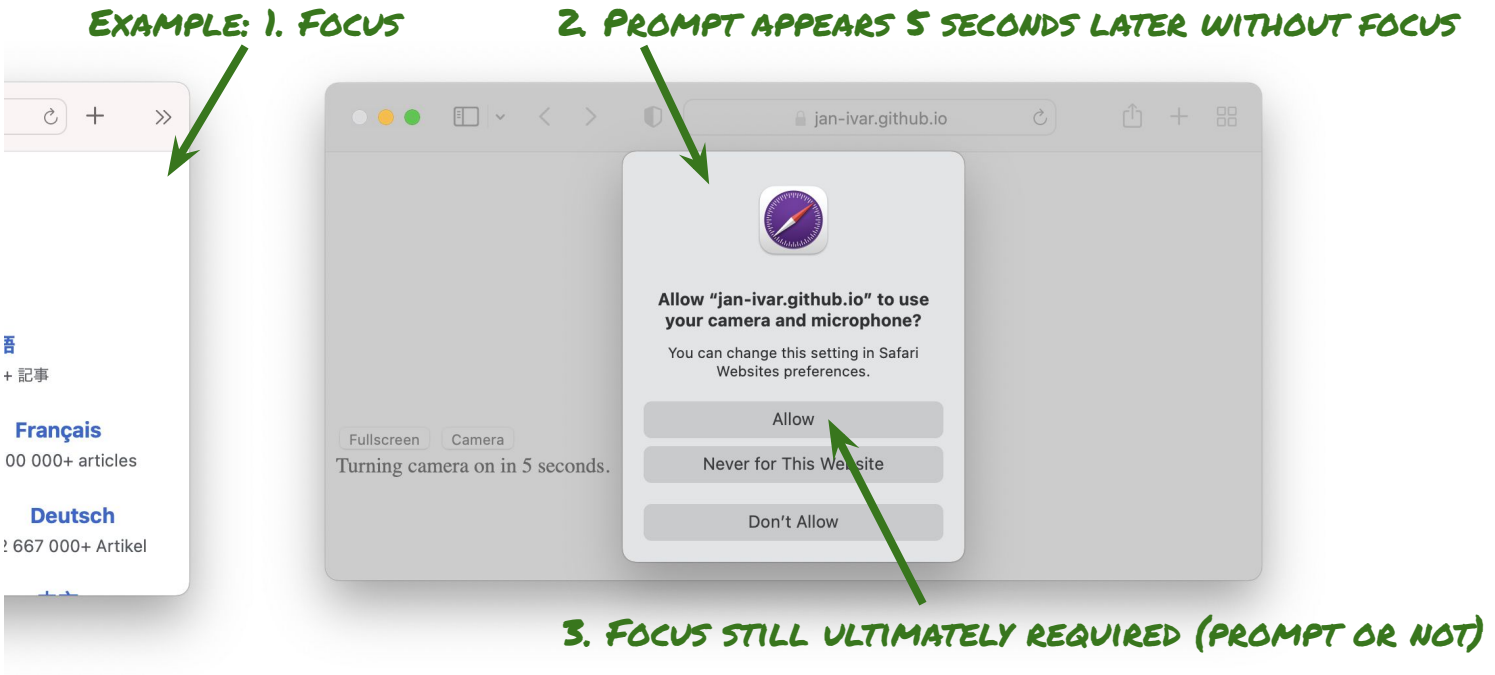
mediacapture-main/pull/912 (Jan-Ivar)

Start Time: 09:40 AM

End Time: 10:00 AM

Issue 752 / PR 912: Allow gUM prompt ahead of focus… (1/5)

Allow Safari prompts in foreground tabs, while maintaining focus requirement on allow:



Maintains anti-spying, and gives users a chance to see a request was made (demo)

Issue 752 / PR 912: Allow gUM prompt ahead of focus… (2/5)

Proposal: Push up-front focus test down, in favor of an up-front is in view check:

When the ***`getUserMedia()`*** method is called, the User Agent *MUST* run the following steps:

8. Let *isInView* be the result of the is in view algorithm.
9. Let *p* be a new promise.
10. Run the following steps in parallel:
    1. While *isInView* is `false`, the User Agent *MUST* wait to proceed to the next step until a task queued to set *isInView* to the result of the is in view algorithm, would set *isInView* to `true`.

To perform an ***is in view*** check, run the following steps:

1. If the relevant global object's associated `Document` is fully active and its visibility state is "`visible`", return `true`. Otherwise, return `false`.

A focus test remains, but is found further down (shown on next slide)

Issue 752 / PR 912: Allow gUM prompt ahead of focus… (3/5)

Focus test is now further down in getUserMedia, after the stall step:

> NOTE
>
> If the user never responds, this algorithm stalls on this step.

2. If the result of the request is "`denied`", jump to the step labeled *Permission Failure* below.
3. Let *hasSystemFocus* be `false`.
4. While *hasSystemFocus* is `false`, the User Agent *MUST* wait to proceed to the next step until a task queued to set *hasSystemFocus* to the result of the has system focus algorithm, would set *hasSystemFocus* to `true`.

To perform a ***has system focus*** check, run the following steps:

1. If the relevant global object's browsing context's top-level browsing context has system focus, return `true`. Otherwise, return `false`.

Changed from a document (iframe) focus check to a system focus check on TLBC (to be replaced by “TLBC has user attention” if whatwg/html#8466 is merged)

Issue 752 / PR 912: Drop enumerateDevices focus requirement (4/5)

Proposal: Drop optional focus requirement on enumerateDevices, just be “visible”.

Rationale: Web compat; anti-fingerprint, not anti-spying; iframes without focus cannot tell when browser window receives focus; apps want to make deterministic checks:

```
if (document.visibilityState == "visible") {
  await navigator.mediaDevices.enumerateDevices(); // never blocks
}
```

Concern: 2+ browser windows open at once accessing (or having accessed) camera or microphone since opening: sites may use polling or devicechange event to time-correlate user across origins if user inserts or removes a USB or Bluetooth device.

Solution: Existing prose puts no time-limit on fuzzing (i.e. can be as long as need be):

> ***NOTE***
>
> ***These events are potentially triggered simultaneously on documents of different origins. User Agents MAY add fuzzing on the timing of events to avoid cross-origin activity correlation.***

Issue 752 / PR 912: Drop enumerateDevices focus requirement (5/5)

The PR's changes to enumerateDevices (in green):

When the `enumerateDevices()` method is called, the User Agent must run the following steps:

1. Let *p* be a new promise.
2. Let *proceed* be the result of device enumeration can proceed.
3. Let *document* be the relevant global object's associated `Document`.
4. Run the following steps in parallel:
    1. While *proceed* is `false`, the User Agent *MUST* wait to proceed to the next step until a task queued to set *proceed* to the result of device enumeration can proceed, would set *proceed* to `true`.

The initial check uses a synchronous result = deterministic.
The convoluted task queue language solves accessing document off main thread.

Discussion (End Time: 10:00)

-

Thank you

Special thanks to:

WG Participants, Editors & Chairs