W3C WebRTC WG Meeting

February 21, 2023
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 February 2023 interim meeting of the W3C WebRTC WG, at which we will cover:
  - WebRTC-SVC, WebRTC-Extensions, WebRTC-Stats, Mediacapture-Extensions, IceController, Face Detection, Auto-pause
- Future meetings:
  - March 21
  - April 18
  - May 16
  - June 27

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/February_21_2023
- 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

- **Click**  Raise hand **to get into the speaker queue.**
- **Click**  Lower hand **to get out of the speaker queue.**
- **Please wait for microphone access to be granted before speaking.**
- **If you jump the speaker queue, you will be muted.**
- **Please use headphones when speaking to avoid echo.**
- **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.

Issues for Discussion Today

- 08:10 - 08:30 AM WebRTC-Stats, WebRTC-Extensions
- 08:30 - 08:50 AM WebRTC-SVC, Mediacapture-Extensions
- 08:50 - 09:10 AM Ice Controller API (Sameer Vijakar)
- 09:10 - 09:30 AM Face Detection API (Riju)
- 09:30 - 09:50 AM Auto-pause (Elad)
- 09:50 - 10:00 AM Wrapup and Next Steps (Chairs)

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.

CfC Summary

- Low Latency Streaming Use Cases
  - Concluded on January 16, 2023
  - 6 responses received, 5 in support, 1 no opinion
  - Author(s) to address issues (6 filed)
  - CfC summary
- Face Detection API contained in PR 78
  - Concluded on January 16, 2023
  - 6 responses received, 5 in support, 1 objection
  - 5 issues filed (3 new), discussion in PR 78
  - CfC summary
  - Followup on today's agenda
- WebRTC-NV "One-way media" Use Cases (Section 3.10)
  - Concluded on February 6, 2023
  - 7 responses received, 5 in support, 1 objection, 1 no opinion
  - Author to address issues (10 filed)
  - CfC summary
- Recycling WebRTC-PC at Recommendation
  - Concluded on February 10, 2023
  - 12 responses received, all in favor (WG consensus declared)
  - CfC summary
  - Name changed to "WebRTC: Real-Time Communication in Browsers"

WebRTC-Stats & WebRTC-Extensions

Start Time: 08:10 AM

End Time: 08:30 AM

For Discussion Today

- WebRTC-Stats
  - Issue 2819: totalRoundTripTime stats is MTI, responsesReceived is not? (Fippo)
  - Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik)
- WebRTC-Extensions
  - Issue 141: Header extension: direction woes and rename (Henrik)

Issue 2819: totalRoundTripTime stats is MTI, responsesReceived is not?

- RTCIceCandidatePairStats.totalRoundTripTime is MTI, responsesReceived is not
  - Usually used to calculate average RTT as totalRoundTripTime/responsesReceived
- Both supported in Chrome, Edge, Safari
- WG decision needed
  - Make both MTI?
  - Remove both from list of MTI?

Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik)

`powerEfficient[Encoder/Decoder]` exposes HW capabilities and usage.

- To address privacy concerns, a HW exposure check was added:
  "*Only expose if context capturing state is true*" (= getUserMedia)

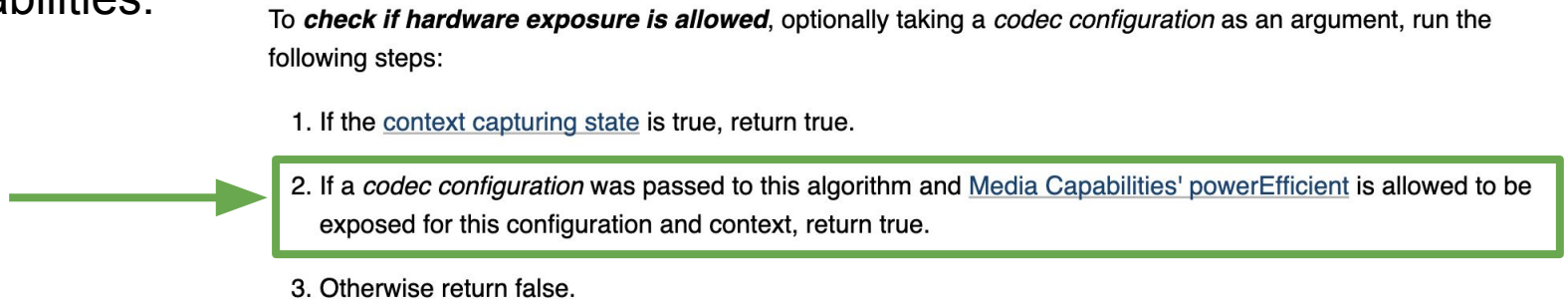
Problem:

- Does not work in the Cloud Gaming use case which does not capture.
- Specs are inconsistent!
  - Media Capabilities already expose `powerEfficient`.
  - The MC privacy considerations section is vague.

Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik)

Proposal (PR 732):

- Delegate the “am I allowed to expose HW information?” question to Media Capabilities.



To ***check if hardware exposure is allowed***, optionally taking a *codec configuration* as an argument, run the following steps:

1. If the context capturing state is true, return true.
2. If a *codec configuration* was passed to this algorithm and Media Capabilities' powerEfficient is allowed to be exposed for this configuration and context, return true.
3. Otherwise return false.

MediaCapabilities being vague is an *existing problem*.

- *In practice, implementations appears to always expose.*
  - Is this a problem? File an issue on MC to clarify.
- Don’t block the webrtc-stats PR…
  - Having getStats be more restrictive than MC does not really help anyone.

Issue 141: Header extension: direction woes and rename (Henrik)

In #130 / previous interim we decided on a “get-modify-set” pattern.

Other open issues:

- `headerExtensionsToOffer` is a confusing name since it also influences answers (#132).
- `FrozenArray` being a mistake or not working (#137, #136, #135).

A “get-modify-set” PR should resolve all issues in one swoop!

Issue 141: Header extension: direction woes and rename (Henrik)

Proposal (same as before, just new names):

```
interface RTCRtpTransceiver {
  sequence<RTCRtpHeaderExtensionCapability> getHeaderExtensionsToNegotiate();

  void setHeaderExtensionsToNegotiate(
      sequence<RTCRtpHeaderExtensionCapability> extensions);

  sequence<RTCRtpHeaderExtensionCapability> getNegotiatedHeaderExtensions();
}
```

Decision: “Ready for PR” on method names?

Alternative names: getHeaderExtensions() + getCurrentHeaderExtensions() to mirror direction and currentDirection?

Issue 141: Header extension: direction woes and rename (Henrik)

Example:

```
const extensions = t.getHeaderExtensionsToNegotiate();
for (let ext of extensions) {
  if (ext.url == 'amazing-extension')
    ext.direction = 'sendrecv';
}
t.setHeaderExtensionsToNegotiate(ext);

… O/A …

t.getNegotiatedHeaderExtenions()
```

Issue 141: Header extension: direction woes and rename (Henrik)

Example:

```
const extensions = t.getHeaderExtensionsToNegotiate();
for (let ext of extensions) {
  if (ext.url == 'amazing-extension')
    ext.direction = 'sendrecv';          // 1a. What if capability is 'recvonly'?
}
t.setHeaderExtensionsToNegotiate(ext);   // 1b. What if t.direction is 'sendonly'?

... O/A ...                              // 2. What if ext not supported by the remote?

t.getNegotiatedHeaderExtenions()
```

Proposal: Don't fail, just downgrade the `direction`! Web compat friendly.

1: getHeaderExtensionsToNegotiate() => 'recvonly'

- Based on capabilities (1a), ***not*** based on transceiver direction (1b).

2: getNegotiatedHeaderExtensions() => 'stopped' or 'inactive'

- Based on transceiver direction and what was negotiated.

Discussion (End Time: 08:30)

-

WebRTC-SVC, Mediacapture-Extensions

Start Time: 08:30 AM

End Time: 08:50 AM

For Discussion Today

- WebRTC-SVC
  - Issue 73/PR 86: S-modes and a single active simulcast layer (Bernard)
- MediaCapture-Extensions
  - Issue 927: Web compat issue with permissions.query() in per-camera/mic permission models (Jan-Ivar)
  - Issue 928: Web compat issue with permissions.query() in non-persistent permission models (Jan-Ivar)

Issue 73/PR 86: S-modes and a single active simulcast layer

- Two types of simulcast are supported in the VP9 and AV1 codecs:
  - Traditional simulcast: multiple encodings, each with its own SSRC and RID.
  - “S” mode simulcast: multiple encodings, single SSRC.
- Issue 73: Should we allow both simulcast types to be configured? If so, when?
  - The specification currently rejects configuration of “S” modes if there is more than a single layer.
    - Restriction arises from a concern that mixing simulcast types complicates SFU and browser implementations without providing any real value.
  - Florent: How does a browser elegantly negotiate codecs and simulcast types?
    - Might be useful to allow “S” mode configuration, as long as only 1 layer was active.

Issue 73: Florent's Example

```
pc.addTransceiver(track, {
      direction: 'sendonly',
      sendEncodings: [
        {rid: 'q', scaleResolutionDownBy: 1.0, scalabilityMode: 'S3T3', active: true}
        {rid: 'h', active: false},
        {rid: 'f', active: false},
      ]
});
// Later, we figure out VP8 was negotiated and scalabilityMode automatically downgraded,
// getParameters() could return something like:
[
        {rid: 'q', scaleResolutionDownBy: 1.0, scalabilityMode: 'L1T3', active: true}
        {rid: 'h', active: false},
        {rid: 'f', active: false},
]
// So we update our configuration with setParameters():
[
        {rid: 'q', scaleResolutionDownBy: 4.0, scalabilityMode: 'L1T3', active: true}
        {rid: 'h', scaleResolutionDownBy: 2.0, scalabilityMode: 'L1T3', active: true},
        {rid: 'f', scaleResolutionDownBy: 1.0, scalabilityMode: 'L1T3', active: true},
]
```

PR 86: S-modes and a single active simulcast layer

- Changes to Section 4.2.1 addTransceiver()
  - To validate `scalabilityMode`, add the following validation steps after step 3 (within step 8):

2. If ~~the number of~~ `RTCRtpEncodingParameters` stored in *sendEncodings* ~~is~~ contains more than ~~1,~~ 1 encoding with an `active` member with a value of `true` and *sendEncodings* contains any encoding whose `scalabilityMode` value represents an "S mode" and whose `active` member has a value of `true`, throw an `OperationError`.

- Changes to Section 4.2.2 setParameters()
  - Add the following to the conditions under which the operation causes a promise rejected

3. If `RTCRtpEncodingParameters` stored in ~~N~~ *encodings* ~~is greater~~ contains more than ~~1,~~ 1 encoding with an `active` member with a value of `true` and *encodings* contains ~~an~~ any encoding whose `scalabilityMode` value represents an "S ~~mode".~~ mode" and whose `active` member has a value of `true`.

Issue 73: S-modes and a single active simulcast layer (discussion)

- Q: In the example, if AV1 is negotiated, attempts to set `active: true` on RIDs h and f will fail.
  - A: That's ok if we prohibit the browser from sending both "S" and traditional simulcast at the same time.
- Q: Will the SFU be confused to receive S3T3 and RID q?
  - Background: "S" modes are not represented in SDP, encodings not distinguished via RIDs.
  - If the browser sends "S" modes to an SFU that can parse AV1 payloads, it won't be confused because it won't rely on RIDs for forwarding "S" mode simulcast.
- Harald: Do we have enough justification for preventing people from being stupid on this point (enabling multiple S-mode layers)?
  - What if an application finds a use for multiple S-mode layers?
    - Example: Sending 6 encodings (3 layers of S2T2)
  - Browser can send an "S" mode to a traditional SFU anyway
    - SFU may not be able to forward without RIDs or ability to parse the AV1 payload

Issue 927: Web compat issue with permissions.query() in per-camera/mic permission models (Jan-Ivar)

Permissions integration says: ***"If the descriptor does not have a deviceId, its semantic is that it queries for access to all devices of that class."*** — No-one's implemented deviceId yet, so sites today use e.g.:

```
const perm = await navigator.permissions.query({name: "camera"});
if (perm.state != "granted") {
  nagTheUserAboutEnablingPermission();
}
```

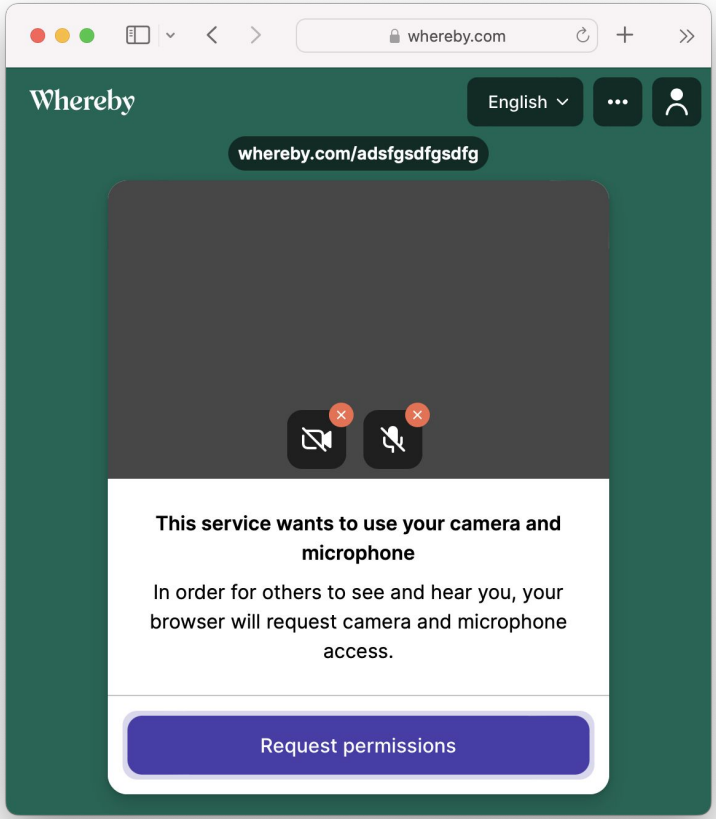
But site is trying to ask ***"can I call getUserMedia unprompted?"***, not ***"do I have access to ALL cameras?"*** Firefox cannot answer yes to the latter even ***after*** site has camera, because permissions is per-device, not ALL.

Proposal: It's more web compatible to say:

- ***"If the descriptor does not have a deviceId, its semantic is that it queries for access to any device of that class."***

Remains compatible with all-device-permission browsers. Sites can use deviceId to inspect each camera.

Issue 928: Web compat issue with permissions.query() in non-persistent permission models (Jan-Ivar)



Shown only once ever in Chrome

Shown *every meeting* in Safari & Firefox

(Two extra clicks instead of one)

Issue 928: Web compat issue with permissions.query() in non-persistent permission models (Jan-Ivar)

That users in Safari and Firefox granted camera/mic last time they used a site, and the time before that, counts for nothing in the permissions spec. See #414 **Add another permission state "granted last time"?**

As a result, many video conferencing sites offer a smoother user experience to returning Chrome users than to returning users in other browsers, because they ignore past ***non-persisted*** permissions entirely.

Sets sites up to expect `"granted"` persisted permission, treating anything less as a user retraining problem.

Causes diverging site-UX for returning users in browsers that persist permission from those that don't.

Proposal (not waiting on long-term fix #414): More web compatible to add:

- ***"If the descriptor does not have a deviceId, and permission state is `"prompt"`, User Agents that offer non-persisted permissions MAY return `"granted"` if the user granted device access to this origin the last time this origin requested it."***

Mimics persisted permissions. Users shouldn't have to grant *persisted* permission to get sites off their back. Sites can use **deviceId** (which they'll have on return visits) to still get `"prompt"` (e.g. TeleVisit sites)

Issue 2820/PR 2829: setParameters/insertDtmf/replaceTrack should reject on [[Stopping]] as well as [[Stopped]]?

Align spec with Chrome, Edge & WPT, which predate introduction of **[[Stopping]]**, preserving the following behavior:

```
tc1.stop();
await tc1.sender.setParameters(...); // InvalidStateError

tc2.stop();
await tc2.sender.track.replaceTrack(...); // InvalidStateError

tc3.stop();
await tc3.sender.dtmf.insertDtmf(...); // InvalidStateError
```

Matches:

```
tc4.stop();
tc4.direction = "sendrecv"; // InvalidStateError
```

Issue 2827/PR 2828: Hard to tell if there are state gaps in connectionState algorithm

Editorial FYI:

Help show that

connectionState = iceConnectionState + DTLS state

RTCPeerConnectionState Enumeration description

Enum value	Description
`closed`	[[IceConnectionState]] is "closed".
`failed`	The previous state doesn't apply, and either [[IceConnectionState]] is "failed" or any RTCDtlsTransports are in the "failed" state.
`disconnected`	None of the previous states apply, and [[IceConnectionState]] is "disconnected".
`new`	None of the previous states apply, and either [[IceConnectionState]] is "new", and all RTCDtlsTransports are in the "new" or "closed" state, or there are no transports.
`connected`	None of the previous states apply, [[IceConnectionState]] is "connected", and all RTCDtlsTransports are in the "connected" or "closed" state.
`connecting`	None of the previous states apply.

NOTE

In the "connecting" state, one or more RTCIceTransports are in the "new" or "checking" state, or one or more RTCDtlsTransports are in the "new" or "connecting" state.

…and that there are no gaps (see issue for proof — thanks @pthatcher!)

Discussion (End Time: 08:50)

-

Ice Controller API (Sameer Vijakar)

Start Time: 08:50 AM

End Time: 09:10 AM

What?

Allow applications to have greater visibility and control over the choice of connection used for transport.

Draft: https://sam-vi.github.io/webrtc-icecontroller

GitHub: https://github.com/sam-vi/webrtc-icecontroller

Why?

- Improve reliability by letting the transport be managed actively
- The application is better positioned than the browser to evaluate trade-offs suitable to its own use cases
  - Network interface
  - IP version
  - Transport protocol
  - Relay or not
- Allow applications to innovate and experiment independent of the standards

What apps can do today wrt ICE

- Limited control over local candidates through `iceServers` and `iceTransportPolicy`
  - Local candidate attributes, eg. priority, cannot be affected
- Once added with `addIceCandidate`, remote candidates may be changed with an ICE restart
- ICE restart can be initiated, possibly after changing the configuration
- Active candidate statistics can be gathered with `getStats`

Proposed API - webrtc-icecontroller

- Allow applications to
  - Observe the lifecycle of ICE candidate pairs
    - all candidate pairs, not just the active pair
  - Observe and affect certain actions of the ICE agent
    - ping / select / prune candidate pairs
  - Instruct the ICE agent to perform certain actions
    - ping / select / prune specific candidate pairs

Proposed API - webrtc-icecontroller

WebIDL

```
[Exposed=Window]
interface RTCIceController : EventTarget {
  constructor();

  readonly attribute RTCIceRole role;
  RTCIceCandidatePair? getSelectedCandidatePair();

  attribute EventHandler oncandidatepairadded;
  attribute EventHandler oncandidatepairupdated;
  attribute EventHandler oncandidatepairswitched;
  attribute EventHandler oncandidatepairdestroyed;

  attribute EventHandler onicepingproposed;
  attribute EventHandler oniceswitchproposed;
  attribute EventHandler onicepruneproposed;

  Promise<undefined> sendIcePing(RTCIceCandidatePair candidatePair);
  Promise<undefined> switchToCandidatePair(RTCIceCandidatePair candidatePair);
  Promise<undefined> pruneCandidatePairs(sequence<RTCIceCandidatePair> candidatePairs);
};
```

WebIDL

```
partial dictionary RTCConfiguration {
  RTCIceController iceController;
};
```

WebIDL — cancelable

```
[Exposed=Window]
interface RTCIcePingProposalEvent : Event {
  constructor(DOMString type, RTCIcePingProposalEventInit eventInitDict);
  readonly attribute RTCIceCandidatePair candidatePair;
  readonly attribute DOMHighResTimeStamp? lastPingSentTimestamp;
  readonly attribute DOMHighResTimeStamp? earliestNextPingTimestamp;
};
```

WebIDL — conditionally cancelable

```
[Exposed=Window]
interface RTCIceSwitchProposalEvent : Event {
  constructor(DOMString type, RTCIceSwitchProposalEventInit eventInitDict);
  readonly attribute RTCIceCandidatePair candidatePair;
  readonly attribute RTCIceSwitchReason reason;
  readonly attribute RTCIceRecheck? recheck;
};
```

WebIDL — cancelable

```
[Exposed=Window]
interface RTCIcePruneProposalEvent : Event {
  constructor(DOMString type, RTCIcePruneProposalEventInit eventInitDict);
  sequence<RTCIceCandidatePair> getCandidatePairs();
};
```

Sample Use

```
const iceController = new RTCIceController();

iceController.addEventListener('icepingproposed', event
=> {
   if (!shouldPingPair(event.candidatePair)) {
       event.preventDefault();
   }
});
iceController.addEventListener('iceswitchproposed',
event => {
   // active connection is managed by the app
   if (event.cancelable) {
       event.preventDefault();
   }
});
iceController.addEventListener('icepruneproposed',
event => {
   // don't prune candidate pairs unless app requests
   event.preventDefault();
});
```

```
function doPing() {
    iceController.sendIcePing(selectPairToPing());
    setTimeout(() => doPing(), 5.0 * 1000);
}
iceController.addEventListener('candidatepairadded', () => {
   doPing();
}, { once: true });

function doSelect() {
   iceController.switchToCandidatePair(selectBestPair());
   iceController.pruneCandidatePairs(selectPairsToPrune());
   setTimeout(() => doSelect(), 10.0 * 1000);
}
iceController.addEventListener('candidatepairupdated', () => {
   doSelect();
}, { once: true });

const configuration = { iceController: iceController };
const pc = new RTCPeerConnection(configuration);
```

Sample Use

```
const iceController = new RTCIceController();

iceController.addEventListener('iceswitchproposed',
   event => {
       if (!event.cancelable) return;
       const proposed = event.candidatePair;
       if (proposed.address.startsWith("10.") ||
           proposed.protocol === "tcp" ||
           (proposed.address.includes(":") &&
               proposed.type === "host")) {
           event.preventDefault();
       }
   });
```

```
var recheck_interval_ms = 5.0 * 1000;
var recheck_timer = 0;

function doPing() {
   iceController.sendIcePing(selectPairToPing());
   recheck_timer = setTimeout(() => doPing(), recheck_interval);
}

iceController.addEventListener('candidatepairupdated', (event) => {
   if (event.candidatePair ===
iceController.getSelectedCandidatePair()) {
       const rttSamples = event.report.getRoundTripTimeSamples();
       recheck_interval_ms = deriveRecheckIntervalFromRtt(rttSamples);
   }
   if (!recheck_timer) {
       doPing();
   }
});
```

Caveats

- Gathering of local ICE candidates is not affected
  - only controls are `iceServers` and `iceTransportPolicy`, as before
- STUN pings are still constructed by the ICE agent
  - specific attributes cannot be set by the application
- To start with a simpler API, only allow RTCIceController when
  - bundlePolicy = max-bundle
  - iceCandidatePoolSize = 0

Discussion (End Time: 09:10)

-

Face Detection API (Riju)

Start Time: 09:10 AM

End Time: 09:30 AM

Face Detection CfC : 5 Support , 1 Objection

Scope of applicability

https://github.com/w3c/mediacapture-extensions/issues/84

Response:

- Face detection API has been available for Android phone vendors since API level 14 (Android 4.0, 2011) and in 2015, 54% or 1.3 billion devices shipped.
- On ChromeOS, the Android Camera2 API, which supports face detection, has been available on all Pixelbooks from Google (from 2017 onwards)
- iOS devices have cameras supporting this for quite some time. Coverage on this particular platform should be pretty good today - Youennf
- On Windows 10 and above, clients with driver support have supported a face detection API.

Face Detection CfC

Generality

https://github.com/w3c/mediacapture-extensions/issues/79

Response: we propose to extend the metadata and to generalize it as follows:

```
partial dictionary VideoFrameMetadata {
  sequence<Segment> segment;
};

dictionary Segment {
  DOMString          type;         // One of enum SegmentType
  long               id;
  long               partOf;       // References the parent
segment id
  float              probability;  // or confidence
  Point2D?           centerPoint;
  DOMRectReadOnly?   boundingBox;
//  sequence<Point2D>? contour;    // Possible future extension
};
```

```
enum SegmentType {
  "human-face",
  "left-eye",
  "right-eye",
  "mouth",
 // To be extended
later with other
types of segments
  };
```

Face Detection CfC

Variance of results

https://github.com/w3c/mediacapture-extensions/issues/85

Response:

- UAs are responsible for enabling FD only where the implementation is good enough.
- It is a common practice in industry to define output syntax instead of specifying underlying algorithm. This enables innovation and improvement over time without app changes.
- In multiple other cases, a web API is provided even if implementations vary:
  - Echo cancellation for instance can be done by the OS, the UA or the web application.
  - HW encoders (WebCodecs) also do not have uniform results across the spectrum.
- Performance: In many cases the FD using the proposed API is free or near-free in computation (camera algorithms often run internally FD whether user wants the results or not) and many users might opt to using the API even if the results might vary to some degree. This is available to native developers already.
- Even if an app would decide to deploy its own ML model, it could still make use of the metadata definitions from this proposal.

Video Conference Features

1. Face Framing
   https://github.com/w3c/mediacapture-extensions/pull/55

2. Lighting Correction
   https://github.com/w3c/mediacapture-extensions/pull/53

3. Eye Gaze Correction
   https://github.com/w3c/mediacapture-extensions/pull/56

Almost a year, next steps ? **blockers** for landing the PR ?
Customers for Origin Trial (1,2) - Google (CrOS) + Whereby

onConfigurationChange()

https://github.com/w3c/mediacapture-extensions/pull/83

1. The configuration of a MediaStreamTrack maybe changed dynamically outside the control of web applications. One example is when a user decides to switch on background blue through the operating system. Web applications might want to know the configuration of a particular MediaStreamTrack has changed.

2. Should the configurationchange event be a ConfigurationChangeEvent with a list of changes ?
   Con: Web Platform Design Principles 7.7 : Use plain Events for state
   Pro : Developer ergonomics, has recent precedence (HTML popover beforetoggle & toggle)

3. Should all capability changes trigger the event ? YES

4. What setting changes should trigger the event ?
   Exclude e.g. estimated settings in order to not trigger on every frame

Discussion (End Time: 09:30)

-

Auto-pause (Elad)

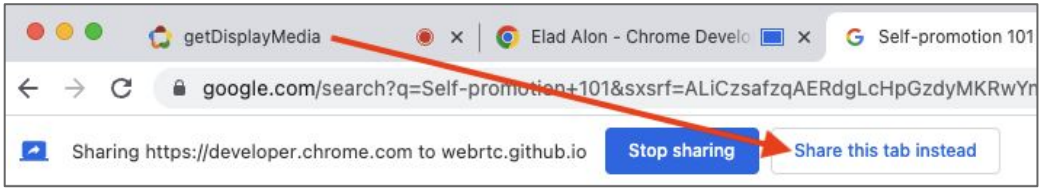
Start Time: 09:30 AM

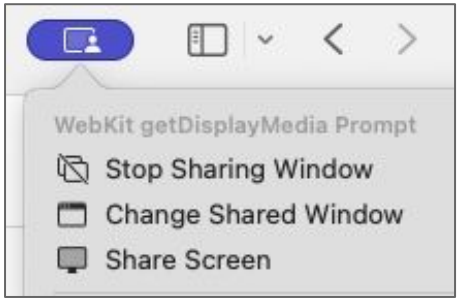
End Time: 09:50 AM

Reminder - State of the Art

A captured surface could change at any moment.

- User navigates a captured tab's top-level document
- Captured app navigates its top-level document (potentially cross-site)
- Dynamic surface switching (deployed by Safari and Chromium)





Reminder - Problem (Issue #255)

- Apps might wish to adapt themselves to the shared content.
  - Re-prompt the user
  - **Adjust cropping parameters.**
    - New tab - new potential CropTargets.
  - Adjust constraints (resolution, frame-rate)
  - Adjust encoding parameters
  - Pipe frames to a new file
- Actions by the app could lag behind production of new frames; possibly even their remote transmission.

Reminder - Discussion last month

A proposal was made to expose this on the source.

- Disadvantages:
  - Counter-productive for cloned tracks.
  - No source object atm.
- Advantages:
  - ?

(CaptureController? Same thing.)

Proposal (adjusted from last time)

```
enum PauseReason {
  “top-level-navigation”,
  "surface-switch",
  “config-change”
};

interface PauseEvent : Event {
  readonly attribute PauseReason reason;
};
```

```
interface MediaStreamTrack : EventTarget {
  readonly attribute boolean paused;
  Attribute EventHandler handler;
  Promise<undefined> SetAutoPause();
  undefined unpause();
};
```

Proposal clarified

- To avoid associating a side-effect with the act of setting an event handler, a dedicated method is used to set the auto-pause behavior.
- The Promise returned by SetAutoPause() is resolved once the app can start relying on the auto-pause behavior.
- Reusing mute still a possibility.

Discussion (End Time: 09:50)

-

Wrapup and Next Steps

Start Time: 09:50 AM

End Time: 10:00 AM

Title Goes Here

- Content goes here

Thank you

Special thanks to:

WG Participants, Editors & Chairs