

W3C WebRTC WG Meeting

February 27, 2020
8 AM Pacific Time

Chairs: Bernard Aboba
Harald Alvestrand
Jan-Ivar Bruaroey

W3C WG IPR Policy

- This group abides by the W3C Patent Policy <https://www.w3.org/Consortium/Patent-Policy/>
- Only people and companies listed at <https://www.w3.org/2004/01/pp-impl/47318/status> are allowed to make substantive contributions to the WebRTC specs

Welcome!

- Welcome to the interim meeting of the W3C WebRTC WG!
 - During this meeting, we hope to bring the group up to date on KITE test results, and make progress on privacy concerns and WebRTC extensions.

About this Virtual Meeting

Information on the meeting:

- Meeting info:
 - https://www.w3.org/2011/04/webrtc/wiki/February_27_2020
- Link to latest drafts:
 - <https://w3c.github.io/mediacapture-main/>
 - <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-stats/>
 - <https://www.w3.org/TR/mst-content-hint/>
 - <https://w3c.github.io/webrtc-nv-use-cases/>
 - <https://w3c.github.io/webrtc-dscp-exp/>
 - <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.

KITE Test Results (Dr. Alex)

- Quick Follow-up on last meeting decision to update test results. Dom + DrAlex
- Special Simulcast test using open source Medooze SFU with two modes:
 - Lenient (test plan B)
 - Lean and Mean (Spec only)
- Used during IETF Hackathons (+HTA), for W3C updates, and by Apple (youenn?)

KITE Test Results (Dr. Alex) - Browsers

Lots of work. Critical mass only in EU. Difficult to maintain (cosmo effort). Needs update => IETF 107. Madrid better (july).

			chrome 75 (canary)	chrome stable	Safari TP	Safari	firefox
Media Simulcast / ABR		h264 simulcast	yes - but bug pending	only via SDP mangling	yes	yes	no
		vp8 simulcast	yes	only via SDP mangling	yes	yes	yes
W3C Browsers APIs	RTCTransceiver	Have transceivers	yes - with unified plan	yes - unified plan	yes	yes	yes
	Stats API	Compliant Stats	yes - but bug pending	no	no	no	no
		Per layer Stats	no	no	no	no	no
	Simulcast enabling	Standard API + createOffer()	yes	no	no	no	yes - but old setParameter()
legacy SDP mangling		yes	yes	yes	yes	no	
IETF Internet protocols	Signalling (JSEP, SDP O/A)	Standard Unified Plan	yes	yes	yes	opt-in	yes
		Legacy Plan B	opt-in	opt-in	opt-in	yes	no
	Media Transport (RTP) simulcast features	rid	yes - if using addTransceiver	no	no	no	yes
		repairedId (RTX)	yes	no	no	no	no (no RTX at all)
		legacy ssrc in SDP	no - if using addTransceiver	yes	yes	yes	yes
	Bandwidth evaluation and congestion control	transport-wide-cc	yes	yes	yes	yes	no
REMB		yes	yes	yes	yes	yes	
not all standards, but some IETF doc exists		vetted by henrik and harald		vetted by Youenn		vetted by nils	

KITE Test Results (Dr. Alex) - SFUs

		Open Source							
tested at IETF 104		Yes	Yes	Yes	No	No	No	No	No
team member present at IETF 104		Yes	No	Yes	No	No	No	No	No
Point of Contact		sergio	inaki	lorenzo	Jianjun ZHU	Voluntas	emil / boris / saul	?	micael gallego
Name		medooze	mediasoup	janus (VideoRoom plugin)	INTEL	sora shiguredo	jitsi	licode	openvidu / KMS
SDP Plan semantics	Plan B	yes	yes	yes	yes	yes	Yes	yes	yes
	Unified Plan	yes	yes	yes	yes	yes	One way only through conversion	no	no
SDP O/A signaling	direct SDP signalling	yes	ORTC: RTCRtpParameters on the wire, SDP O/A locally	yes	yes	yes	no	yes	yes
	other	JSON on the wire, SDP locally		no	no	no	Jingle / COLIBRI on the wire, SDP locally	no	no
simulcast enabled via	SDP munging	yes	yes	yes	yes	yes	yes	yes	simulcast not supported
	setParameter	yes	yes	yes	yes	no	no	yes	
	addTransceiver	yes	yes	yes	yes	no	no	no	
PC and stream handling	separate publisher and Subscriber PC	no	yes.	yes	yes	no	no	no	yes
	multiple PC	no	no	*master* => multiple.	yes	no	no	it's flexible, depends on scalability: M multistream x N PC	?
	single multi-stream PC	yes	sending (MID and RID), receiving (SSRCs), both with BUNDLE	sending (MID and RID), receiving (SSRCs), both with BUNDLE ("unified-plan" branch)	no	yes	yes		no
video codecs	VP8	yes + simulcast	yes + simulcast	yes + simulcast	yes+simulcast	yes + simulcast	Depends on configuration, but mainly VP8+simulcast	yes + simulcast	yes
	H.264	yes + simulcast	yes + simulcast	yes + simulcast	yes+simulcast	yes + simulcast		yes	yes
	VP9	yes + SVC	yes	yes + SVC	yes	no		yes + SVC	no
IDs	rids supported	yes	yes	yes	yes	yes	no	yes	no
	repairid supported	yes	yes	yes	no	yes	no	no	no
	ssrc-less supported	yes	yes	yes (simulcast only)	no	no	no	no	no
bandwidth congestion control	transport-wide-cc	yes	yes	?	yes	?	?	no	no
	remb	yes	yes	?	yes	?	?	yes	yes
bandwidth limitation on senders		no	simulcast layers dropping	?	yes	?	?	proprietary client API	Proprietary client API or settings
mid rewriting		yes	no	?	?	?	?	no	no

Issues for Discussion Today

- Media Capture and Streams
 - [Issue 642](#): Only Firefox turns off device on disabled track. Stronger language needed? (Jan-Ivar)
 - [Issue 655](#): How to avoid wide-lens & telephoto on new phones (Jan-Ivar)
 - [Issue 652](#): In-browser device picker (Jan-Ivar)
- Extensions
 - Insertable Streams (Harald)
 - Capture timestamps (Henrik)
 - RTP Header Extensions (Harald)
 - Content-Hints (Harald)

Issues for Discussion Today

- Media Capture and Streams
 - [Issue 642](#): Turn off device on disabled track. (Jan-Ivar)
 - [Issue 655](#): How to avoid wide-lens & telephoto on new phones (Jan-Ivar)
 - [Issue 652](#): In-content device selection a mistake. Leaks; Complex


[Issue 642](#): Turn off device on disabled track. (Jan-Ivar)

Camera & mic may be temporarily disabled using track.[.enabled](#). Sole use case:



```
micMute.onclick = () => audioTrack.enabled = !micMute.checked;  
endCall.onclick = () => pc.close();  
camMute.onclick = () => videoTrack.enabled = !camMute.checked;
```

This is semantically complete (says it all) & works in all browser permission models.

But some browsers don't turn off the camera HW light... 

Users demand privacy.



[Better privacy on camera mute in Firefox 60](#)

Issue 642: Turn off device on disabled track. (Jan-Ivar)



Sites want this behavior! Here's what they do to work around it (a good example of where a spec not being explicit about *behavior* causes web compat headaches).

```
camMute.onclick = async () => { // Our users demand camera HW light off in other browsers. Let's hack!
  try {
    videoTrack.enabled = !camMute.checked;
    await wait(3000); // because users
    if (camMute.checked == !videoTrack) return;
    if (camMute.checked) {
      videoTrack.stop(); // kill the light!
    } else if (videoTrack && videoTrack.readyState != "live") { // revive the camera!
      selfView.removeTrack(videoTrack);
      videoTrack = null; // avoid racing with ourselves
      videoTrack = (await navigator.mediaDevices.getUserMedia(stashedConstraints)).getVideoTracks()[0];
      await micTransceiver.sender.replaceTrack(videoTrack);
    }
  } catch (e) { /* 🙈 */ }
}
```

- Irony: Does NOT work in all browser permission models (extra needless Firefox prompts)
- Typically hacks camera only, leaving any microphone light lit (e.g. on [Microsoft LifeCam](#))
- Odd corners like adding and renegotiating with ended tracks if someone joins.

Issue 642: Turn off device on disabled track. (Jan-Ivar)

Refresher on how Firefox works ([fiddle](#)):

1. Relinquishes hardware device when all its tracks are disabled.
2. Reacquires hardware device when any of its tracks are re-enabled (Takes ~1 second to reacquire device)
3. Failure to reacquire fires **ended** event on track(s).
4.  *Live* indicator always on for 3 seconds minimum (“*MUST remain observable for a sufficient time*”)
5.  *Accessible* mandated [Privacy Indicator](#) remains in URL bar while muted.
6. Privacy mitigation plan is to have Firefox fire [muted](#) event if re-enabled without focus ([Bug 1598374](#))

Proposal: Enforce web compat around this behavior in all browsers

[Spec](#) today¹: “when a track becomes either muted or disabled, and this brings all tracks connected to the device to be either muted, disabled, or stopped, then the UA ~~MAY~~ **MUST** using the device's deviceId, deviceId, set `[[devicesLiveMap]][[deviceId]]` to false”

1) This is only a close approximation to hardware lights, inasmuch as having physical and logical “privacy indicators” align is POLA.

Issue 642 / PR 662: Relinquish device when all tracks muted/disabled.

When a live track sourced by a device exposed by `getUserMedia()` becomes either muted or disabled, and this brings *all* tracks connected to the device to be either muted, disabled, or stopped, then the UA *MUST* relinquish the device within 3 seconds while allowing time for a reasonably-observant user to become aware of the transition. The UA *MUST* attempt to reacquire the device as soon as any live track sourced by the device becomes both unmuted and enabled again, provided the *document* has focus at that time. If the *document* does not have focus at that time, the UA *MUST* instead queue a task to mute the track, and unmute it later once the document gains focus. If reacquiring the device fails, the UA *MUST* end the track (The UA *MAY* end it earlier should it detect a device problem, like the device being physically removed).

NOTE

The intent is to give users the assurance of privacy that having physical camera (and microphone) hardware lights off brings, by aligning physical and logical “privacy indicators”, at least while the current document is the sole user of a device.

While other applications and documents using the device simultaneously may interfere with this intent at times, they do not interfere with the rules laid forth.

Issue 655: How to avoid wide-lens & telephoto on new phones (jib)

Flagship phones have multiple back cameras now. [How to](#) distinguish wide-lens?

Wide-lens usually means fixed-focus, represented by 0 in [android](#), so might this work?

```
{video: {focusDistance: {exact: 0}}}} // pick wide-lens video camera  
{video: {focusDistance: {min: 0.0001}}}} // avoid wide-lens video cameras
```

But relies on a side-effect: Apparently no rule says all wide lenses have fixed focus.

Also, what if the phone has wide-lens & telephoto? E.g. [Samsung S10](#) specs:

- **Primary** - 12 MegaPixels, f/1.5- f/2.4 Dual Aperture, **26mm Focal Length**, 1/2.55" Sensor Size, 1.4µm Pixel Size, Dual Pixel Phase Detection AutoFocus, Optical Image Stabilization
- **Secondary** - 12 MegaPixels, f/2.4 Aperture, **52mm Focal Length**, 1/3.6" Sensor Size, 1.0µm Pixel Size, AutoFocus, Optical Image Stabilization, 2x Optical Zoom
- **Tertiary** - 16 MegaPixels, f/2.2 Aperture, **12mm Focal Length**, Ultra-Wide

[AFAICT](#) these are [35mm equivalent focal lengths](#), *“a measure that indicates the angle of view”*, because photography. 🙄

Issue 655: How to avoid wide-lens & telephoto on new phones (jib)

Proposal A: A new (“35mm equivalent”) *focalLength* constraint:

```
{video: {focalLength: {min: 0.026}}} // avoid all wide-lenses < 26mm  
{video: {focalLength: {min: 0.052}}} // pick telephoto
```

Proposal B: A new *angleOfView* constraint:

```
{video: {angleOfView: {max: 68}}} // avoid all wide-lenses < 26mm  
{video: {angleOfView: {max: 38}}} // pick telephoto
```

Conversions: $2 \tan^{-1}\left(\frac{0.035}{2 \times 0.026}\right) \times \frac{180}{\pi} = \underline{67.8872}$ $2 \tan^{-1}\left(\frac{0.035}{2 \times 0.052}\right) \times \frac{180}{\pi} = \underline{37.2}$

Bonus Q: Where to specify this? *Mediacapture-main* or *Mediacapture-image*?

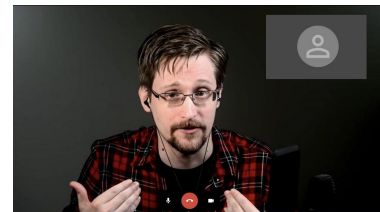
Issue 652: In-browser device picker (Jan-Ivar)

In 2020, exposing all your devices to the web beyond the one you're using, is not POLA. It goes beyond fingerprinting, revealing actual private information users did not intend to share about what they own and have plugged in. This may include devices personalized with names, high-entropy prototype devices, perhaps even embarrassing (adult) devices

It's not **"the minimal information needed to achieve user goals"**.

PING wants privacy-by-default *in-browser device picker*:

1. site asks for category (or categories) of device
2. browser prompts user for one, many or all devices
3. site gains access to only the device + label, of hardware the user selects.



Web developers need this API to work in all browsers before it's usable.

Issue 652: In-browser device picker (Jan-Ivar)

The **TAG** joins **PING** in wanting this. [w3ctag/design-principles#152](https://www.w3ctag.com/design-principles/#152):

Discourage device enumeration, prefer less powerful alternatives

[“The Rule of Least Power”](#) suggests that we should go with the least powerful API which meets its use cases.”

“we ... recommend that, when API designers need to expose devices in some manner, they consider [in order]:

1. An **availability-style API** is best: it allows the site to use the device the user wants to use without exposing device information. It is the least powerful option.
2. A **picker-style API** also respects user preferences. It exposes device information, but only of one device. This minimizes the fingerprinting data exposed.
3. A **filtered device enumeration API** exposes a subset of devices, so is potentially better re: fingerprinting than the next possibilities
4. A **sorted device enumeration API** exposes too much information, but at least it does so in a way that makes the site's job easier
5. A **device enumeration API** is to be resorted to only when the other options are infeasible for some reason (compat, perhaps).”

[Issue 652](#): In-browser device picker (Jan-Ivar)

On availability-style API: Nothing prevents UAs from letting users customize browser default cam/mic & expose a single device per kind. No spec work seems needed for this.

On picker-style API: How far do we go?

Goal 1: Get rid of in-content device selection & `label` in `enumerateDevices` [#640](#)

Goal 2: Somehow prevent browsers from granting all devices of a kind by default.

Let's focus on goal 1. We may never reach consensus on goal 2.

What's a minimal approach? What are the minimal API changes needed?

[Issue 652](#): In-browser device picker — minimal approach (Jan-Ivar)

Use cases. New visitor: (browsers that do per-device grants will need a picker)

```
await navigator.mediaDevices.getUserMedia({video: true});
await navigator.mediaDevices.getUserMedia({video: {facingMode: "environment"}});
await navigator.mediaDevices.getUserMedia({video: {width: {min: 1280}}});
```

Repeat visitor: (picker undesired, unless previous device removed *and* multiple choices)

```
await navigator.mediaDevices.getUserMedia({video: {deviceId}});
await navigator.mediaDevices.getUserMedia({video: {deviceId: {exact: deviceId}}});
await navigator.mediaDevices.getUserMedia({video: true});
```

Changing (or adding a 2nd) camera view in a world without labels: (picker is *required*)

```
await navigator.mediaDevices.getUserMedia({video: {existingDeviceId}});
await navigator.mediaDevices.getUserMedia({video: {existingDeviceId: [...ids]}});
```

Lazy getter, or don't know about `stream.clone()`: (picker would be web compat bug) ❌

```
await navigator.mediaDevices.getUserMedia(sameConstraints);
```

[Issue 652](#): In-browser device picker — minimal approach (Jan-Ivar)

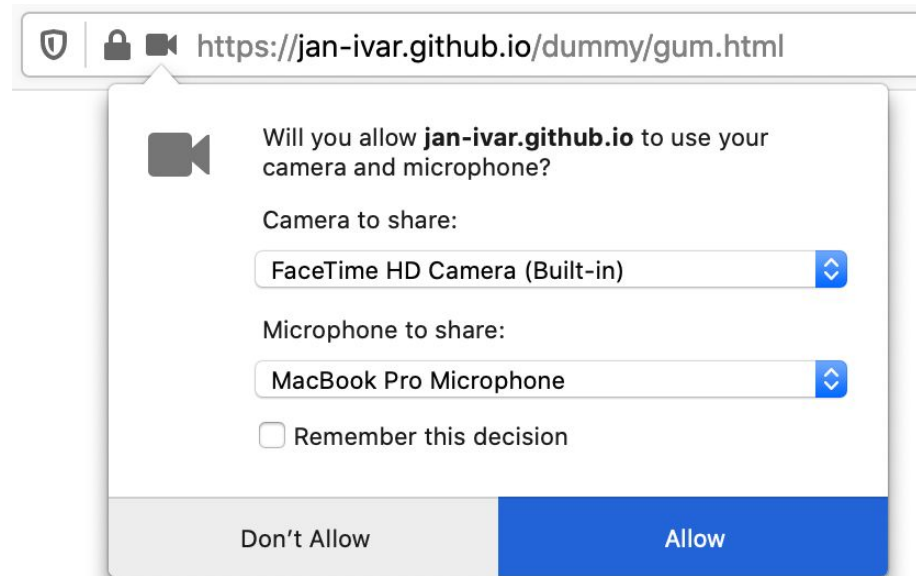
New visitor/repeat visitor: Firefox already has an in-browser device picker for this.

```
await navigator.mediaDevices.getUserMedia({video: true, audio: true});
```

Lists {min ... max}, w/ideal chosen by default.

But it only appears if permission is absent.

Strong emphasis on permission and defaults.



No known obstacles to other browsers adding something comparable if they want to.

Issue 652: In-browser device picker — minimal approach (Jan-Ivar)

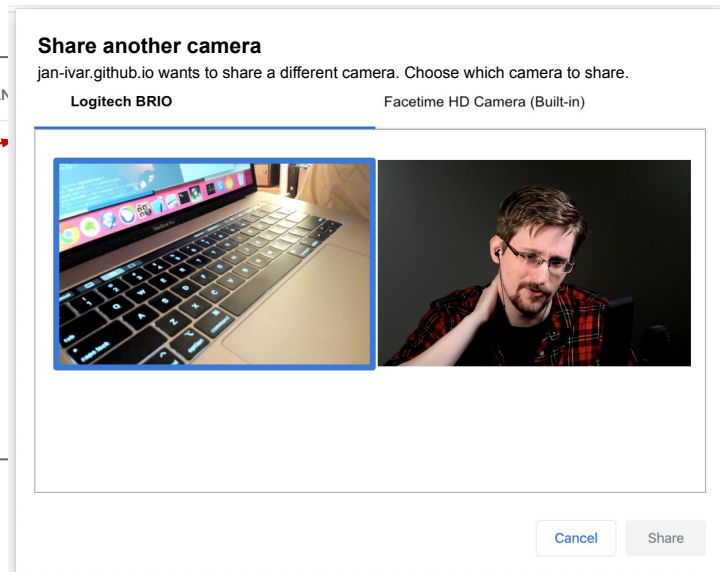
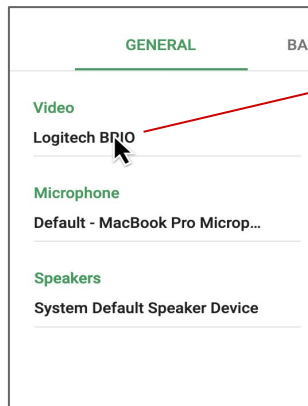
Changing camera: We need an in-browser picker shown regardless of permission:

```
button.onclick = async () => {
  if (numberOfVideoInputDevices < 2) return;
  const constraints = {
    video: {existingDeviceId: cameraTrack.getSettings().deviceId}
  };
  cameraTrack = (await navigator.mediaDevices.getUserMedia(constraints)).getVideoTracks()[0];
  button.innerText = cameraTrack.label;
}
```

A new `existingDeviceId` constraint tells UA this is a request for a device other than the (non-ideal) specified, and it always prompts the user.

(It's up to UX whether to default to or exclude the specified device as a choice.)

We dunno if JS intends to replace or supplement it)



Issue 652: In-browser device picker (Jan-Ivar)

Pros (of minimal API):

- Should suffice to let us experiment with picker UX specifically to replace current in-content device selection only. We retain the option of going further later.
- Knowing which video track is potentially being replaced may be useful for UX, e.g. on devices with limits on the number of simultaneous devices that can be open.
- Might let us throw (`OverconstrainedError`) if there's no second device available.
- Signals our intent to solve post-gUM selection (UA could make it only work then)
- Least disruption of existing models.

Cons:

- The constraint is still barely more than a glorified boolean
- (Unless we prevent it, e.g. by demanding the `existingDeviceId` track be active), JS may exploit API to launch picker upfront anyway, so pick a better boolean already.

Issue 652: In-browser device picker (Jan-Ivar)

Better booleans: `chooseUserMedia` unless we change the semantics of `getUserMedia`.

Option A: Morph already-expressive gUM w/transition plan inspired by `{sdpSemantics}`:

```
await navigator.mediaDevices.getUserMedia({video: true, semantics: "browser-chooses"});  
await navigator.mediaDevices.getUserMedia({video: true, semantics: "user-chooses"});
```

Step 1: Implement picker in all browsers

Step 2: Announce `{semantics}` and default to `"browser-chooses"` in all browsers

Step 3: Sites prepare to avoid redundant pickers (preferably using `exact`, `clone()`)

Step 4: Flip default to `"user-chooses"` in all browsers

Step 5: (Optional) Deprecate `{semantics}` in all browsers (mention caveat)

Option B: Concede status quo with above syntax. **Option C:** Concede status quo with:

```
await navigator.mediaDevices.getUserMedia({video: true}); // browser chooses  
await navigator.mediaDevices.chooseUserMedia({video: true}); // user chooses
```

Issue 652: In-browser device picker (Jan-Ivar)

```
WebIDL   
  
partial interface MediaDevices {  
  MediaTrackSupportedConstraints getSupportedConstraints();  
  Promise<MediaStream> getUserMedia(optional MediaStreamConstraints constraints = {});  
  Promise<MediaStream> chooseUserMedia(optional MediaStreamConstraints constraints =  
  {});  
};
```

chooseUserMedia

Works the same as **getUserMedia**, except instead of [requesting permission to use](#) a device, this method will [prompt the user to choose](#), unless constraints widdle choices down to one device per kind.

When the **chooseUserMedia()** method is called, the User Agent *MUST* return the result of [getting user media](#) with a value of **true**.

To **get user media** given a boolean *choose*, run the following steps:

1. Let *constraints* be the method's first argument.

[Issue 640](#): Only reveal labels of devices user has given permis.. (Youenn)

If difficult to enforce per-device exposure rule, enforce per-device-type exposure

- Expose all microphones if one microphone is granted
- Expose all cameras if one camera is granted
- Do not expose speakers once output speaker picker API is available

Still possible to use `groupId` to get microphone corresponding to camera

Issue 639: Enforcing user gesture for getUserMedia (Youenn)

- Problem: getUserMedia should only be callable on user gesture
 - Most modern APIs add such restrictions
 - This is not web compatible
- Can we start shipping such restrictions in getUserMedia?
- Potential ideas
 - Require a user gesture past initial page load
 - Require user gesture once a previous call to getUserMedia was denied for the given page
 - Implemented in Safari

Extensions for Discussion Today

- Insertable Streams (Harald)
- Capture timestamp (Henrik)
- RTP Header Extensions (Harald)
- Content-Hints (Harald)

Insertable Streams (Harald)

- Purpose: Javascript-defined processing of (un)encoded video/audio data at sender & receiver
- Use case 1: Web client interoperability with deployed multiuser conferencing with end-to-end encryption
 - This use case requires encoded data access
 - Waiting for a group conferencing standard is not an option
- Proof-of-concept implementation done
- Explainer here:
 - <https://github.com/alvestrand/webrtc-media-streams/blob/master/explainer.md>
 - Considerably changed from previous iterations - not all docs updated

Insertable Streams - WebIDL API

```
partial dictionary RTCConfiguration {  
    boolean forceEncodedVideoInsertableStreams =  
    false;  
    boolean forceEncodedAudioInsertableStreams =  
    false;  
};  
  
partial interface RTCRtpSender {  
    RTCInsertableStreams createEncodedVideoStreams();  
    RTCInsertableStreams createEncodedAudioStreams();  
};  
  
partial interface RTCRtpReceiver {  
    RTCInsertableStreams createEncodedVideoStreams();  
    RTCInsertableStreams createEncodedAudioStreams();  
};
```

```
dictionary RTCInsertableStreams {  
    ReadableStream readable;  
    WritableStream writable;  
};  
  
enum RTCEncodedVideoFrameType {  
    "Empty", "key", "delta",  
};  
  
interface RTCEncodedVideoFrame {  
    readonly attribute RTCEncodedVideoFrameType type;  
    readonly attribute unsigned long long timestamp;  
    attribute ArrayBuffer data;  
    readonly attribute ArrayBuffer additionalData;  
};
```

Insertable Streams - example use

```
let senderTransform = new TransformStream({
  async transform(chunk, controller) {
    let view = new DataView(chunk.data);
    let newData = new ArrayBuffer(chunk.data.byteLength + 4);
    let newView = new DataView(newData);
    // Invert and pad the bits in the frame
    for (let i = 0; i < chunk.data.byteLength; ++i)
      newView.setInt8(i, ~view.getInt8(i));
    // Set the padding bytes to zero.
    newView.setInt8(chunk.data.byteLength + i, 0);
    chunk.data = newData;
    controller.enqueue(chunk);
  },
});
```

```
let senderStreams =
videoSender.getEncodedVideoStreams();

// After ICE and offer/answer exchange.

senderStreams.readable
  .pipeThrough(senderTransform)
  .pipeTo(senderStreams.writable);
```

Insertable Streams - Relation to other efforts

WebCodec

- Aims to reuse “VideoFrame” and “AudioFrame” types
- Experience will be fed back to that effort

TransferableStreams

- Allows processing in WebWorkers
- Uses unmodified proposal
- Origin trial will also enable TS

Insertable Streams - Change from previous API proposal

- Previous API was “factory” based
 - This presented some challenges in Chrome implementation
- Current API allows easy shimming of a “factory” based API on top of it - all in JS

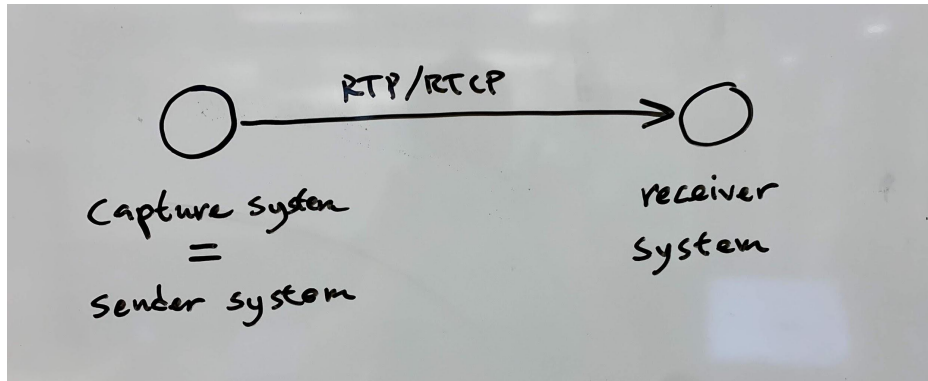
Insertable Streams - Next Steps

- Experiment with API in a real app under Chrome's "origin trial" mechanism
- Synchronize with WebCodecs as appropriate
- Propose (revised) API for standardization

Capture Timestamp (Henrik)

Problem 1: How to calculate the end-to-end delay?

- e2eDelay = time passed between capturing at one endpoint and playout at another endpoint.



`RTCRTpContributingSource.timestamp`
tells you time of playout.

Just need time of capture...

Capture Timestamp (Henrik)

Problem 1: How to calculate the end-to-end delay?

- $e2eDelay$ = time passed between capturing at one endpoint and playout at another endpoint.

RTP header extension **abs-capture-time** gives you...

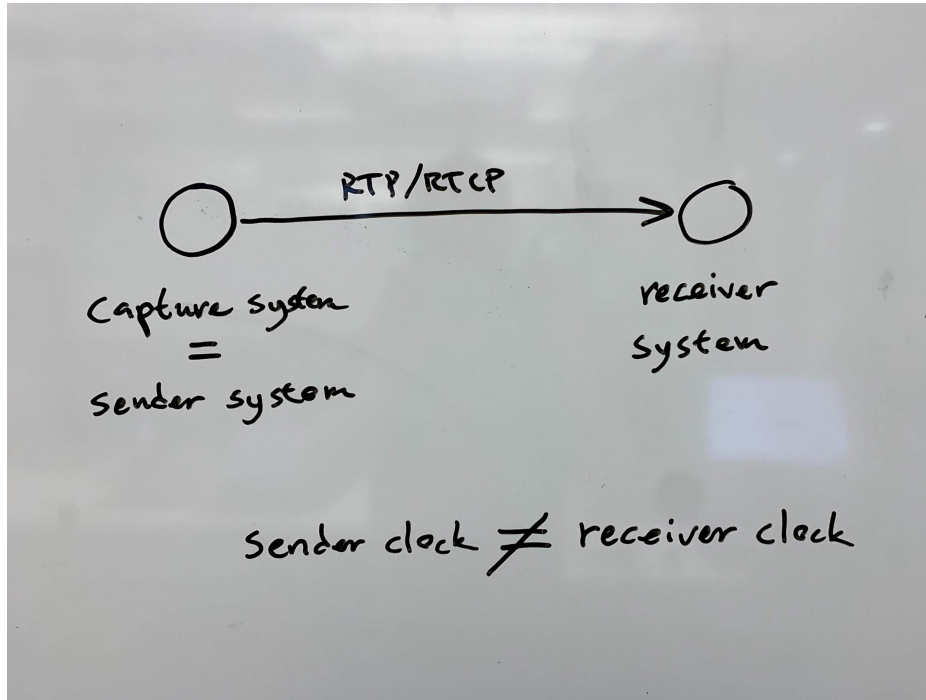
- `captureTimestamp`
- `estimatedClockOffset` (*more on this later*)

Problem 2: How to measure A/V sync?

- Solution: Expose `captureTimestamp`.

Capture Timestamp (Henrik)

Example!

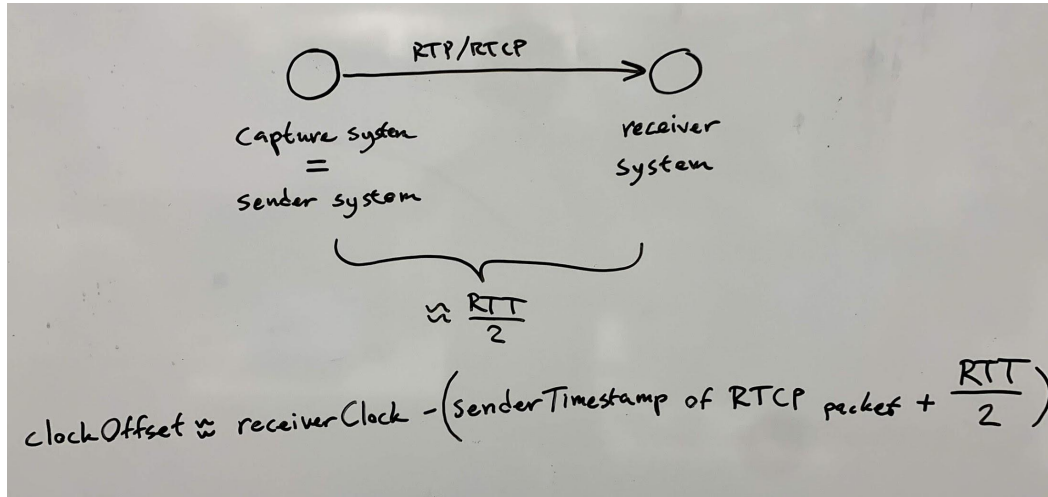


Problem:

- captureTimestamp is in capturer's (sender's) clock.
- We only know of the receiver's clock.

Capture Timestamp (Henrik)

Example!



With clockOffset you can calculate captureTimestamp in receiver's clock, ergo you can calculate e2eDelay.

Capture Timestamp (Henrik)

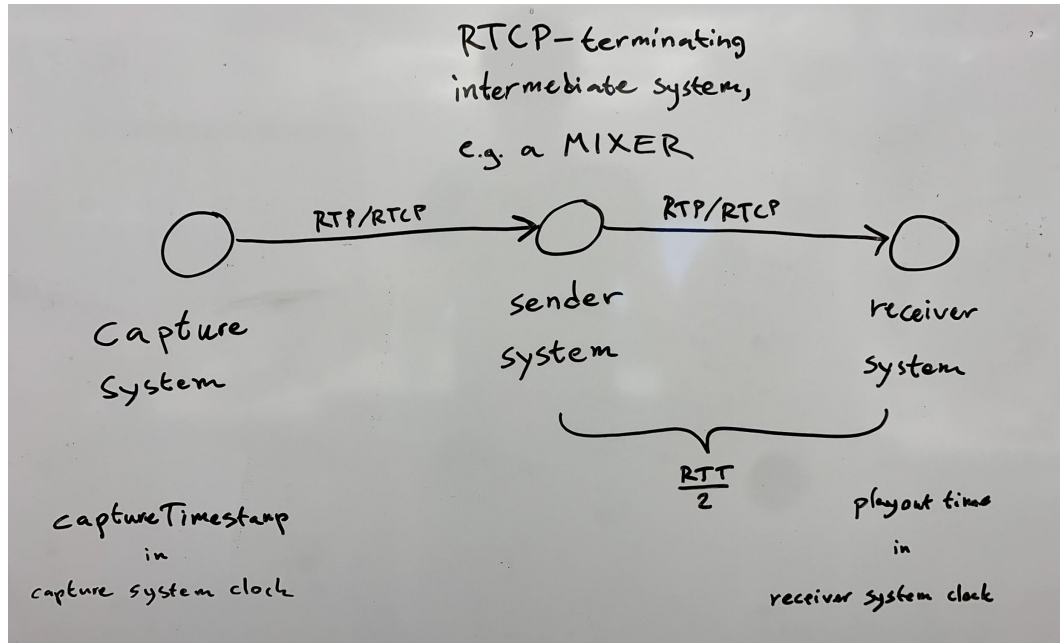
Example!

- clockOffset is available in getStats() today.
- *We just need to expose captureTimestamp!*

$$= \text{RTCRemoteOutboundRtpStreamStats.timestamp} - \left(\text{RTCRemoteOutboundRtpStreamStats.remoteTimestamp} + \frac{\text{RTCRemoteInboundRtpStreamStats.roundTripTime}}{2} \right)$$

Capture Timestamp (Henrik)

But this doesn't work if there's a middlebox! :(



clockOffset transforms sender's clock to receiver's clock.

But captureTimestamp is in capturer's clock.

Capture Timestamp (Henrik)

abs-capture-time's `estimatedClockOffset` gives us the sender's estimate of the `clockOffset` between the capturer's clock and the sender's clock. Thus:

$$\begin{aligned} &\text{sender's captureTimestamp} \\ &= \\ &\text{capturer's captureTimestamp} + \text{estimatedClockOffset} \end{aligned}$$

This works regardless of how many “hops” between the capturer and the receiver, by adding to `estimatedClockOffset` each “hop”.

Capture Timestamp (Henrik)

For more info, see [webrtc-extensions PR#33](#) which added:

WebIDL



```
partial dictionary RTCRtpContributingSource {  
    DOMHighResTimeStamp captureTimestamp;  
    DOMHighResTimeStamp senderCaptureTimeOffset;  
};
```

Based on **abs-capture-time** RTP header extension:

```
  0           1           2           3  
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1  
  +-----+-----+-----+-----+  
  | ID   | len=15|   absolute capture timestamp (bit 0-23)   |  
  +-----+-----+-----+-----+  
  |           absolute capture timestamp (bit 24-55)           |  
  +-----+-----+-----+-----+  
  | ... (56-63) |   estimated capture clock offset (bit 0-23) |  
  +-----+-----+-----+-----+  
  |           estimated capture clock offset (bit 24-55)           |  
  +-----+-----+-----+-----+  
  | ... (56-63) |  
  +-----+-----+-----+-----+
```

RTP Header Extensions (Harald)

This section:

```
a=extmap:14 urn:ietf:params:rtp-hdext:toffset
a=extmap:2 http://www.webrtc.org/experiments/rtp-hdext/abs-send-time
a=extmap:13 urn:3gpp:video-orientation
a=extmap:3 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
a=extmap:12 http://www.webrtc.org/experiments/rtp-hdext/playout-delay
a=extmap:11 http://www.webrtc.org/experiments/rtp-hdext/video-content-type
a=extmap:7 http://www.webrtc.org/experiments/rtp-hdext/video-timing
a=extmap:8 http://tools.ietf.org/html/draft-ietf-avtext-framemarking-07
a=extmap:9 http://www.webrtc.org/experiments/rtp-hdext/color-space
a=extmap:4 urn:ietf:params:rtp-hdext:sdes:mid
a=extmap:5 urn:ietf:params:rtp-hdext:sdes:rtp-stream-id
a=extmap:6 urn:ietf:params:rtp-hdext:sdes:repaired-rtp-stream-id
```

The need:

- Controlling what RTP header extensions get negotiated in SDP
 - SDP munging is not a long term viable method
- Controlling what RTP header extensions get sent in RTP

RTP header extensions (Harald)

- Done since last 2 VIs
 - [Public design document](#) published
 - PR on webrtc-extensions written
 - [Intent to Prototype](#) (Chromium/Blink) sent out
 - Proposal merged ([#25](#)) to webrtc-extensions
- Next steps:
 - Implement as Chrome experimental feature
 - Release as origin trial
 - Learn

Content-Hints (Harald)

Status update!

- Spec now includes RTCDegradationPreference
- Spec is more specific on actions resulting from Content-Hint settings
- Spec says how Content-Hint and constraints interact
- It has a [little bit of usage](#) (0.013%)
- CfC for publishing a new WD sent Feb 20
- Favorable (but somewhat low) response
- Is it time to ask for CR publication?

For extra credit



Name that bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird