

W3C WebRTC WG Meeting

August 27, 2024
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 August 2024 interim meeting of the W3C WebRTC WG, at which we will cover:
 - Captured Surface Control, moving forward with mute, timing model, speaker-selection
- [Future meetings:](#)
 - TPAC, September 23- 27, 2024
 - [October 15](#)
 - [November 19](#)
 - [December 10](#)

TPAC 2024 Schedule

- Venue: Hilton Anaheim, California
 - Time Zone: Pacific Daylight Time (UTC -7)
- Tuesday, September 24, 2024
 - 09:00 - 12:30 WEBRTC WG
 - 16:30 -18:00 WEBRTC WG/SCCG Joint Meeting
- Wednesday, September 25, 2024
 - [Breakout Sessions](#)
- Thursday, September 26, 2024
 - 14:00 - 16:00 WEBRTC WG/MEDIA WG Joint Meeting

About this Virtual Meeting



- Meeting info:
 - https://www.w3.org/2011/04/webrtc/wiki/August_27_2024
- 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 Captured Surface Control (Guido)
- 08:30 - 08:50 AM Moving forward with mute (Guido)
- 08:50 - 09:10 AM Speaker-selection (Jan-Ivar)
- 09:10 - 09:30 AM Timing Model (Bernard)
- 09:30 - 09:50 AM Scale Resolution Down *To* (Henrik)
- TBD/If time permits: RTCRtpParameters.codec (Jan-Ivar)
- 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.

Captured Surface Control (Guido)

Start Time: 08:10 AM

End Time: 08:30 AM

Captured Surface Control

- We recently proposed an API that allows an application to send wheel and zoom events to a captured tab
- Use case: zooming and scrolling the captured content
- Initial comments
 - The use case of the local user zooming and scrolling the captured content is appealing and worth exploring
 - The use case of a remote user zooming and scrolling raises nontrivial privacy concerns and we shouldn't focus on it at this time
 - The API should not be extended to support other user input such as keystrokes or mouse presses
 - The UA might be in a better position than the application to perform these actions

Captured Surface Control - original

```
partial interface CaptureController {  
    // Scrolling  
    Promise<undefined> sendWheel(CapturedWheelAction action);  
  
    // Zooming  
    static sequence<long> getSupportedZoomLevels();  
    long getZoomLevel();  
    Promise<undefined> setZoomLevel(long zoomLevel);  
    attribute EventHandler oncapturedzoomlevelchange;  
};
```

Captured Surface Control - new

```
partial interface CaptureController {  
    // Scrolling  
    Promise<undefined> captureWheel(HTMLElement? element);  
  
    // Zooming  
    static sequence<long> getSupportedZoomLevels();  
    long getZoomLevel();  
    Promise<undefined> setZoomLevel(long zoomLevel);  
    attribute EventHandler oncapturedzoomlevelchange;  
};
```

Captured Surface Control - Scrolling

```
const controller = new CaptureController();  
const stream = await  
    navigator.mediaDevices.getDisplayMedia({ controller });  
const previewTile = document.querySelector('video');  
previewTile.srcObject = stream;  
  
// Start forwarding wheel events  
await controller.captureWheel(previewTile);  
  
...  
  
// Stop forwarding wheel events  
await controller.captureWheel(null);
```

Captured Surface Control - Scrolling

- `captureController.captureWheel(element);`
 - Forwards all the wheel events dispatched on element to the content associated with `captureController`
 - Subject to permission
 - new "captured-surface-control" permission
 - prompts the user if necessary
 - The UA takes care of the forwarding
 - Offset coordinates for the event are scaled from the size of the element to the size of the viewport of the captured surface
 - Delta values are preserved
- Main advantages: easier to use and only supports the local use case

Captured Surface Control - Zooming

```
partial interface CaptureController {  
    // Scrolling  
    Promise<undefined> captureWheel(HTMLElement? element);  
  
    // Zooming  
    static sequence<long> getSupportedZoomLevels();  
    long getZoomLevel();  
    Promise<undefined> setZoomLevel(long zoomLevel);  
    attribute EventHandler oncapturedzoomlevelchange;  
};
```


Captured Surface Control - Zooming

- The Zooming use case is trickier, because it involves UI and applications prefer to build all their UI elements to guarantee a consistent user experience over letting the UA create the UI
- Proposal: Make remote control require transient user activation for all `setZoomLevel()` calls.
 - Mitigates the issue well enough that it's impractical to use it for remote control
 - Easy to understand, specify and implement
 - The requirement is reasonable for applications
 - This does not work for `sendWheel`

Discussion (**End Time: 08:30**)

-

Moving Forward with Mute (Guido)

Start Time: 08:30 AM

End Time: 08:50 AM

Moving forward with Mute

- We have discussed mute recently and made progress in some areas:
 - Synchronizing mute state (including UI) across Application, UA and OS is a good fit for the Media Session API
 - This also includes unmute requests from the application
- There are still areas that need improvement
 - Better interoperability between implementations

Moving forward with Mute

- The spec includes language that is roughly equivalent to:
 - Track muted -> no frames (or black frames)
 - Good fit for the user manually muting the track with hardware switches, or UA/OS controls
 - Chromium uses this definition for audio. IIUC, Safari and Firefox for video and audio.
 - No frames (or black frames?) -> track muted
 - Fits cases when the track temporarily stops sending frames, but there is no underlying failure
 - Indirectly works for manual mutes (as long as no frames)
 - Chromium uses this definition for video.

Better handling of the no-frames case

- Ideally, we would use the first part of the definition only, but:
 - The language has been in the spec for years and there is an implementation (Chromium) that followed it just as long
 - Developers have expressed that they find the direct no-frames signal useful, even if it is not caused by a user mute
- Proposal
 - Leave the spec language as is
 - Add a boolean attribute `isSendingFrames` (name subject to discussion) to `MediaStreamTrackVideoStats`

Better handling of the no-frames case

- Advantages
 - Provide developers the signal they currently get with Chromium's current version of the muted attribute
 - Makes it easier for Chromium to start using the first part of the definition without creating compatibility problems with applications that rely on the no-frames definition
 - Improved interoperability across browsers (making muted more useful)
 - A similar signal can be inferred by an application by looking at the frame counter in `MediaStreamTracksVideoStats` and using timeouts, but having a boolean with the same properties as the old muted attribute makes it more ergonomic, especially for existing applications that already use the existing signal.

What about black frames?

- For video tracks, zero-information content can mean black frames or no frames.
- Applications can (and do) detect black frames by just analyzing them
- On some platforms, there are system APIs that make it possible to detect that the camera is producing black frames due to specific reasons (e.g., due to laptop lid down, or OS setting)
- Proposal:
 - Add two fields to `MediaStreamTrackVideoStats`
 - `systemBlackFrames` - Black frame counter
 - `lastFrameWasSystemBlack` - Makes it easier for applications to detect the condition without using timeouts
 - Mark the track muted when this condition is detected
 - Black frames are zero-information content, no spec change expected since the causes are user action)

What about black frames?

- Advantages
 - Apps can detect black frames more efficiently
 - The mute signal together with the stat can help the application provide more accurate information to the user

Proposed IDL

```
partial interface MediaStreamTracksVideoStats {  
    readonly attribute unsigned boolean isSendingFrames;  
    readonly attribute unsigned long long systemBlackFrames;  
    readonly attribute unsigned boolean lastFrameWasSystemBlack;  
}
```

Discussion (**End Time: 08:50**)

-

Speaker-selection (Jan-Ivar)

Start Time: 08:50 AM

End Time: 09:10 AM

For Discussion Today

mediacapture-output:

- [Issue 142](#) / [PR 143](#): Why prompt for a subset of stored speakers or speakers setSinkId already accepts?
- [Issue 133](#): The first "audiooutput" MediaDeviceInfo returned from enumerateDevices() is not the default device when the default device is not exposed

Issue 142: Why prompt for a subset of stored speakers?

How a persisted speaker id was exposed in a past session shouldn't matter.

PR 143 updates step 5 of the [selectAudioOutput\(\)](#) algorithm:

5. If *deviceId* is not "" and matches an audio output device id previously exposed by [selectAudioOutput](#) or [enumerateDevices\(\)](#) in an earlier browsing session, the user agent *MAY* decide, based on its previous decision of whether to persist this id or not for this set of origins, to run the following sub steps:
 1. Let *device* be the device identified by *deviceId*, if available.
 2. If *device* is available, resolve *p* with either *deviceId* or a freshly rotated device id for *device*, and abort the in-parallel steps.

Issue 133: First audiooutput in enumerateDevices() is not the default device when the default device is unexposed

[enumerateDevices\(\)](#) says the first speakers are the system default speakers:

4. If *device* is the **system default audio output**, prepend *deviceInfo* to *otherDeviceList*. Otherwise, append *deviceInfo* to *otherDeviceList*.

Websites therefore assume they can do this:

```
const defSpkr = await mediaDevices.enumerateDevices()  
                                .find(d => d.kind == "audio-output");
```

But speakers aren't exposed by default so it might be a different one!

This makes it hard to write apps. E.g. some omit a way to reset to default "".

Issue 133: First audiooutput in enumerateDevices() is not the default device when the default device is unexposed

Proposals: If at least one other audio output is exposed, but the one that is currently the system default is not, prepend an entry for it that looks like this:

A: `{kind: "audiooutput", label: [UA defined], groupId: "", deviceId: ""}`

B: `{kind: "audiooutput", label: [UA defined], groupId: "", deviceId: "default"}`

Pros of A: Works with `setSinkId("")`. No change needed.

Pros of B: avoids `selectAudioOutput({deviceId: ""})` which always prompts

Both solve:

```
const defSpkr = await mediaDevices.enumerateDevices()
                                .find(d => d.kind == "audio-output");
```

Label is up to UA — e.g. might be "" or "System default speakers".

Discussion (**End Time: 09:10**)

-

Timing Model (Bernard)

Start Time: 09:10 AM

End Time: 09:30 AM

For Discussion Today

- RequestVideoFrameCallback Metadata
- Encoded Transform timing metadata
- WebCodecs timing metadata

VideoFrameCallbackMetadata (RVFC)

§ 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;  
};
```

Metadata Definitions

presentationTime, of type [DOMHighResTimeStamp](#)

The time at which the user agent submitted the frame for composition.

expectedDisplayTime, of type [DOMHighResTimeStamp](#)

The time at which the user agent expects the frame to be visible.

width, of type [unsigned long](#)

The width of the video frame, in [media pixels](#).

height, of type [unsigned long](#)

The height of the video frame, in [media pixels](#).

NOTE: [width](#) and [height](#) might differ from [videoWidth](#) and [videoHeight](#) in certain cases (e.g. an anamorphic video might have rectangular pixels). When calling [texImage2D\(\)](#), [width](#) and [height](#) are the dimensions used to copy the video's [media pixels](#) to the texture, while [videoWidth](#) and [videoHeight](#) can be used to determine the aspect ratio to use, when using the texture.

mediaTime, of type [double](#)

The media presentation timestamp (PTS) in seconds of the frame presented (e.g. its timestamp on the [video.currentTime](#) timeline). MAY have a zero value for live-streams or WebRTC applications.

presentedFrames, of type [unsigned long](#)

A count of the number of frames submitted for composition. Allows clients to determine if frames were missed between [VideoFrameRequestCallbacks](#). MUST be monotonically increasing.

Definitions (cont'd)

***processingDuration*, of type double**

The elapsed duration in seconds from submission of the encoded packet with the same presentation timestamp (PTS) as this frame (e.g. same as the mediaTime) to the decoder until the decoded frame was ready for presentation.

In addition to decoding time, may include processing time. E.g., YUV conversion and/or staging into GPU backed memory.

SHOULD be present. In some cases, user-agents might not be able to surface this information since portions of the media pipeline might be owned by the OS.

***captureTime*, of type DOMHighResTimeStamp**

For video frames coming from a local source, this is the time at which the frame was captured by the camera. For video frames coming from remote source, the capture time is based on the RTP timestamp of the frame and estimated using clock synchronization. This is best effort and can use methods like using RTCP SR as specified in RFC 3550 Section 6.4.1, or by other alternative means if use by RTCP SR isn't feasible.

SHOULD be present for WebRTC or getUserMedia applications, and absent otherwise.

Definitions (cont'd)

receiveTime, of type DOMHighResTimeStamp

For video frames coming from a remote source, this is the time the encoded frame was received by the platform, i.e., the time at which the last packet belonging to this frame was received over the network.

SHOULD be present for WebRTC applications that receive data from a remote source, and absent otherwise.

rtpTimestamp, of type unsigned long

The RTP timestamp associated with this video frame.

SHOULD be present for WebRTC applications that receive data from a remote source, and absent otherwise.

RTCEncodedAudioFrameMetadata

4.4. *RTCEncodedAudioFrameMetadata* dictionary

```
dictionary RTCEncodedAudioFrameMetadata {  
  unsigned long synchronizationSource;  
  octet payloadType;  
  sequence<unsigned long> contributingSources;  
  short sequenceNumber;  
  unsigned long rtpTimestamp;  
  DOMString mimeType;  
};
```

rtpTimestamp, of type **unsigned long**

The RTP timestamp identifier is an unsigned integer value per [\[RFC3550\]](#) that reflects the sampling instant of the first octet in the RTP data packet.

RTCEncodedVideoFrameMetadata

4.2. *RTCEncodedVideoFrameMetadata* dictionary

```
dictionary RTCEncodedVideoFrameMetadata {  
  unsigned long long frameId;  
  sequence<unsigned long long> dependencies;  
  unsigned short width;  
  unsigned short height;  
  unsigned long spatialIndex;  
  unsigned long temporalIndex;  
  unsigned long synchronizationSource;  
  octet payloadType;  
  sequence<unsigned long> contributingSources;  
  long long timestamp;    // microseconds  
  unsigned long rtpTimestamp;  
  DOMString mimeType;  
};
```

VideoFrame Timing Metadata

```
[Exposed=(Window,DedicatedWorker), Serializable, Transferable]
interface VideoFrame {
  constructor(CanvasImageSource image, optional VideoFrameInit init = {});
  constructor(AllowSharedBufferSource data, VideoFrameBufferInit init);

  readonly attribute VideoPixelFormat? format;
  readonly attribute unsigned long codedWidth;
  readonly attribute unsigned long codedHeight;
  readonly attribute DOMRectReadOnly? codedRect;
  readonly attribute DOMRectReadOnly? visibleRect;
  readonly attribute unsigned long displayWidth;
  readonly attribute unsigned long displayHeight;
  readonly attribute unsigned long long? duration; // microseconds
  readonly attribute long long timestamp; // microseconds
  readonly attribute VideoColorSpace colorSpace;

  VideoFrameMetadata metadata();
}
```

duration, of type **unsigned long long**, **readonly**, **nullable**

The presentation duration, given in microseconds. The duration is copied from the [EncodedVideoChunk](#) corresponding to this VideoFrame.

The [duration](#) getter steps are to return [\[\[duration\]\]](#).

AudioData Timing MetaData

```
[Exposed=(Window,DedicatedWorker), Serializable, Transferable]
```

```
interface AudioData {
```

```
    constructor(AudioDataInit init);
```

```
    readonly attribute AudioSampleFormat? format;
```

```
    readonly attribute float sampleRate;
```

```
    readonly attribute unsigned long numberOfFrames;
```

```
    readonly attribute unsigned long numberOfChannels;
```

```
    readonly attribute unsigned long long duration; // microseconds
```

```
    readonly attribute long long timestamp; // microseconds
```

duration, of type unsigned long long, **readonly**

The duration, in microseconds, for this AudioData.

The duration getter steps are to:

1. Let *microsecondsPerSecond* be 1,000,000.
2. Let *durationInSeconds* be the result of dividing [[number of frames]] by [[sample rate]].
3. Return the product of *durationInSeconds* and *microsecondsPerSecond*.

Discussion (**End Time: 09:30**)

-

[webrtc-extensions#159](#)

Scale Resolution Down *To* (Henrik)

Start Time: 09:30 AM

End Time: 09:50 AM

#159: Scale Resolution Down To (Henrik)

The `scaleResolutionDownTo` was discussed in the [May 2023 Virtual Interim](#).

Recap:

- It's like `scaleResolutionDownBy` but expressed in absolute terms (“send 360p”) instead of relative terms (“downscale by 2” + frame being 720p).
- Motivation:
 - When disabling top layer(s) we want to do the expensive effects processing on a 360p track instead of a 720p track.
 - Today, changing track resolution on the fly triggers reconfiguration. Adjusting scaling factors is inherently racy! Avoiding race = glitchy.

There was overall support, I promised to follow-up with details... 1y ago :)

#159: Scale Resolution Down To (Henrik)

Proposal:

```
dictionary RTCRect { unsigned long width, height; } // Or DOMRect but ignore x,y?  
dictionary RTCRtpEncodingParameters {  
    RTCRect scaleResolutionDownTo;  
};
```

Keep it simple:

- Never upscale, only downscale (if needed).
- Never change aspect ratio (scale down until both sides fit).
- Orientation agnostic (adjust 1280x720 to 720x1280 internally if needed).

API already exists in C++ as [requested_resolution](#) (% fixing some bugs!).

- Just surface it to JS with a new name.

Discussion (**End Time: 09:50**)

-

RTC RTP Parameters codec (Jan-Ivar)

TBD / if time permits

Issue 2987: RTCRtpParameters.codec matching is probably too strict

TL;DR: Try to align with implementation

The spec needs an RTP stream "selecting codec" algorithm. [codec](#) hand-waves:

"Optional value selecting which codec is used for this encoding's RTP stream. If absent, the user agent can chose to use any negotiated codec."

WPT expects setting .codec based on static getCapabilities, and then negotiating, will cause that codec to be used IF it matches (but how closely?) the remote SDP.

There's no explicit prose around doing this in the spec right now, but it definitely seems to be the intent. The WPT also expects .codec to be automatically unset if the remote SDP does not contain a match (there's definitely nothing in the spec about this).

Issue 2987: RTCRtpParameters.codec matching is probably too strict

Do we allow UA to clear this parameter.codec (singular) after negotiation?

```
{ mimeType: 'video/vp9', clockRate: 90000, sdpFmtpLine: 'max-fs=12288;max-fr=60' }
```

...because the closest match in the negotiated parameters.codecs (plural) is this?

```
{ mimeType: 'video/vp9', clockRate: 90000, sdpFmtpLine: 'max-fs=12288;max-fr=30', payloadType: 120 }
```

If so, then UAs are free to chose any negotiated codec anyway, and might take codec as a hint, and at least send VP9. But having it work in some browsers and not others would surprise → poor interop. Should we standardize something here?

Instead of clearing, why not let UA update the codec parameter to the 2nd codec?

Wrapup and Next Steps

Start Time: 09:50 AM

End Time: 10:00 AM

Next Steps

- Content goes here

Thank you

Special thanks to:

WG Participants, Editors & Chairs