

# **W3C WebRTC WG Meeting**

April 23, 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 April 2024 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Custom Codecs, Captured Surface Switching, mediacapture-main, background segmentation
- Future meetings:
  - May 21
  - June 18
  - July 16

# About this Virtual Meeting



- Meeting info:
  - [https://www.w3.org/2011/04/webrtc/wiki/April\\_23\\_2024](https://www.w3.org/2011/04/webrtc/wiki/April_23_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 Custom Codecs (Harald)
- 08:30 - 08:50 AM Captured Surface Switching (Tove)
- 08:50 - 09:30 AM Mediacapture-Main & WebRTC-pc (Jan-Ivar)
- 09:30 - 09:50 AM BG Segmentation Mask (Riju)
- 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.



# **Custom Codecs (Harald)**

**Start Time: 08:10 AM**

**End Time: 08:30 AM**

# PR is ready to merge

<https://github.com/w3c/webrtc-encoded-transform/pull/186>

Has been worked on since June 2023 - several iterations on design of the API shape

A lot of credit goes to to Jan-Ivar for the current shape

# Transform allows adding codecs to the SDP negotiation

```
// At sender side.
const sender = pc.addTrack(track);
const {codecs} = RTCRtpSender.getCapabilities();
const vp8 = codecs.find(({mimeType}) => mimeType == "video/vp8");
sender.transform = new RTCRtpScriptTransform(worker, {
  inputCodecs: [vp8],
  outputCodecs: [{mimeType: "video/x-encrypted",
    packetizationMode: "video/sframe"}]
});

// At receiver side.
pc.ontrack = ({receiver}) => {
  const {codecs} = receiver.getParameters();
  const customCodec = codecs.find(({mimeType}) => mimeType == "video/x-encrypted");
  if (customCodec) {
    receiver.transform = new RTCRtpScriptTransform(worker, {
      inputCodecs: [customCodec],
      outputCodecs: [{mimeType: "video/vp8"}]
    });
  }
}
```

# Worker transforms frame and sets MIME type

```
function work() {
  onrtctransform = async ({transformer: {readable, writable}, {inputCodecs, outputCodecs}}) => {
    const [outputCodec] = outputCodecs;
    await readable.pipeThrough(new TransformStream({transform})).pipeTo(writable);
    function transform(frame, controller) {
      // transform chunk
      let metadata = frame.metadata();
      const inputCodec = inputCodecs.find((mimeType) => mimeType == metadata.mediaType);

      if (inputCodec && outputCodec.mimeType == "video/x-encrypted") {
        encryptBody(frame, inputCodec);
        metadata.mediaType = outputCodec.mimeType;
        frame.setMetadata(metadata);
      } else if (inputCodec.mimeType == "video/x-encrypted") {
        decryptBody(frame, outputCodec);
        metadata.mediaType = outputCodec.mimeType;
        frame.setMetadata(metadata);
      }
      controller.enqueue(frame);
    }
  }
}
```

# Open question: Constructor or SetMetadata?

- Frame copy constructor (PR #233) is merged
- SetMetadata (PR #202) is stalled (no action since October)
- Both can change the MIME type

Should we merge SetMetadata, or should we change the example to use the frame copy constructor?

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

-

# **Captured Surface Switching (Tove)**

**Start Time: 08:30 AM**

**End Time: 08:50 AM**

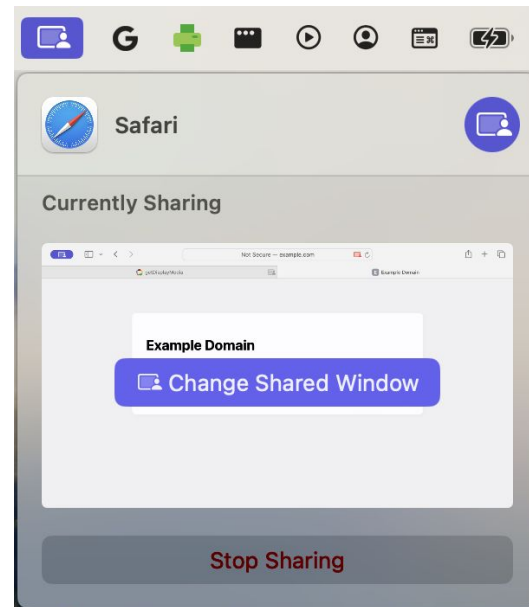
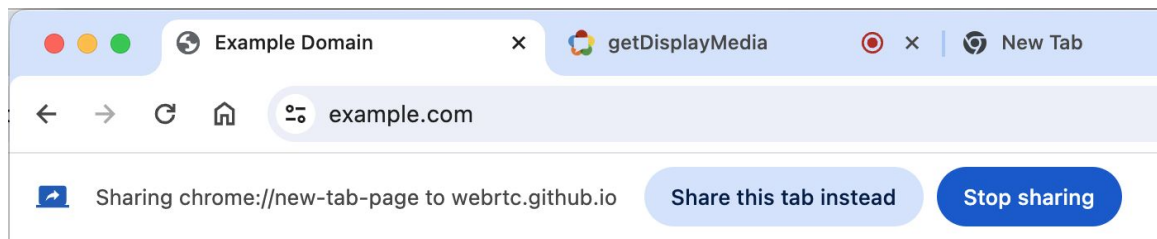
# For Discussion Today

- [Issue 4](#): Auto-pause capture when user switches captured content



# Background

- Same-type surface switching already exists in some user agents:
  - Between tabs in Chrome
  - Between windows/screens in Safari on MacOS
- We now want to provide API-support for switching also between different surfaces.



# Basic Models - Re-Cap

## Injection model:

- Media from the new surface is injected into the existing tracks.

## Switch-track model:

- Each track represents one specific captured surface
- When switching captured surfaces:
  - New tracks are provided for the new surface.
  - Tracks for the old surface are stopped.

# Basic Models - Examples

## Injection model:

```
video.srcObject = await getDisplayMedia(/*opt-in*/, ...));
```

## Switch-track model:

```
controller.onsourceswitch = event => {  
  video.srcObject = event.stream;  
};  
video.srcObject = await getDisplayMedia({controller, /*opt-in*/, ...});
```

**Observation:** Both these API shapes are simple and straightforward.

**Goal:** We should not make things harder than this.

# Model choice

Our preference:

- Switch-track model (with opt-in)
- Existing usage of injection-model remains (without opt-in)

Some concerns:

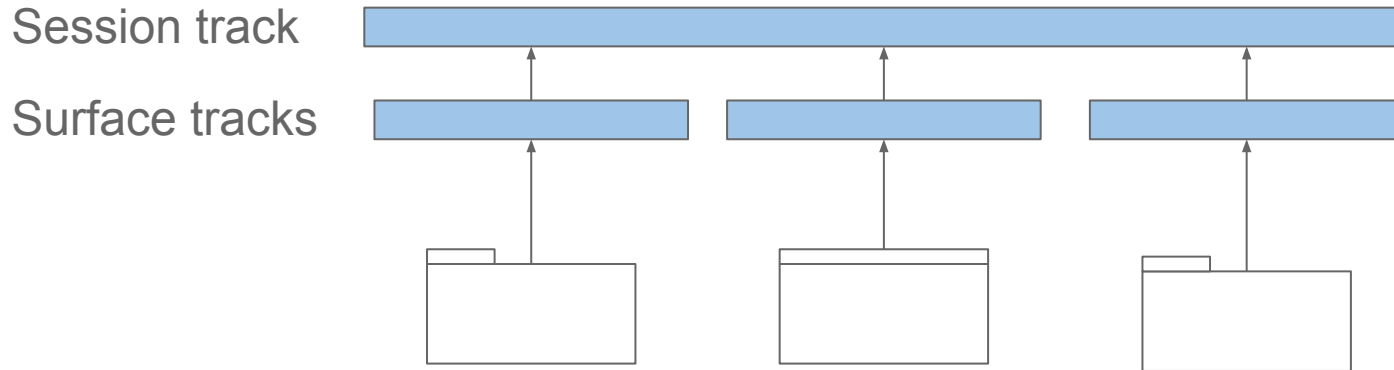
- The injection model is less work for surface-agnostic applications
- Some sinks are interrupted by switching tracks (e.g., MediaRecorder)
- Some apps may need to choose model at event-time

Compromise proposal: Multi-track model (following slides)

# Proposal: Multi-track model

Let's provide two types of tracks in parallel:

- **Surface-tracks** with life-time and and functionality for a single surface.
- **Session-tracks** covering a whole capture-session, switching from one captured surface to the next.



# Multi-track model - Examples

Using session tracks (similar to the injection model):

```
video.srcObject =  
  await getDisplayMedia({surfaceSwitchingTracks: ["session"], ...});
```

Using surface tracks (similar to the switch-track model):

```
controller.onnewsource = event => {  
  video.srcObject = event.stream; // surface tracks  
};  
await getDisplayMedia({controller, surfaceSwitchingTracks: ["surface"], ...});
```

# Multi-track model - Examples (contd)

Using both surface tracks and session tracks:

```
controller.onnewsource = event => {  
  video1.srcObject = event.stream; // surface tracks  
};  
  
const sessionStream = await getDisplayMedia({controller,  
  surfaceSwitchingTracks: ["session", "surface"], ...});  
video2.srcObject = sessionStream;
```

# Multi-track model - Favorable Properties

- An application can request the types of tracks it needs
- Both the injection model and the switch-track models are straightforward to use.
- Retains direct mapping from one surface-track to one surface.
- The injection model can be used for sinks that need it.
- Choice between session-tracks/surface-tracks:
  - at event time.
  - individually per track



# Discussion (End Time: 08:50)

- Can the multi-track model serve as a framework for continued development of this API?
- Concerns that need to be resolved?

# Mediacapture-main & WebRTC-pc (Jan-Ivar)

**Start Time: 08:50 AM**

**End Time: 09:30 AM**

# For Discussion Today

- **Mediacapture-main:**
  - [Issue 972](#): Racy devicechange event design has poor interoperability
- **WebRTC-pc (PR callout only):**
  - [PR 2961](#): Convert RTCIceCandidatePair dictionary to an interface
- **WebRTC-pc:**
  - [Issue 2964](#): setCodecPreferences should trigger negotiationneeded
  - [Issue 2956](#): receiver.getParameters().codecs seems under-specified

# [Issue 972](#) / [PR 996](#): Racy devicechange event design has poor interoperability

The [devicechange](#) event follows [§ 7.7. Use plain Events for state](#), but its "state information in the [target](#) object." is **not** available synchronously, a footgun: 🦶🔫

Previously discussed in <https://www.w3.org/2023/10/17-webrtc-minutes.html>:

| Conclusion: No objection to include the devices in the devicechange event. E.g.:

```
navigator.mediaDevices.onddevicechange = ({devices}) => {  
  // The app compares devices vs. oldDevices to detect changes  
  app.oldDevices = devices;  
}
```

- PR:
3. Let *newExposedDevices* be the result of [creating a list of device info objects](#) with *mediaDevices* and *deviceList*.
  6. Queue a task that [fires an event](#) named [devicechange](#), using the [DeviceChangeEvent](#) constructor with [devices](#) initialized to *newExposedDevices*, at *mediaDevices*.

# PR 996: Add DeviceChangeEvent interface.

## § 9.5 DeviceChangeEvent

The devicechange event uses the DeviceChangeEvent interface.

WebIDL



```
[Exposed=Window]
interface DeviceChangeEvent : Event {
  constructor(DOMString type, DeviceChangeEventInit eventInitDict);
  [SameObject] readonly attribute FrozenArray<MediaDeviceInfo> devices;
};
```

### **constructor()**

Initialize this.devices to the result of creating a frozen array from *eventInitDict.devices*.

### **devices** of type **FrozenArray**<**MediaDeviceInfo**>, readonly

The devices attribute returns an array of MediaDeviceInfo objects representing the current result from enumerateDevices().

# Issue 2964: setCodecPreferences should trigger negotiationneeded



jan-ivar commented 4 hours ago

Member ...

`setCodecPreferences` "overrides the default receive codec preferences used by the [user agent](#)", but negotiation is needed before changes affect transmission. Example:

- <https://jsfiddle.net/jib1/8mosy476> changes codecs mid-stream with `setCodecPreferences`<sup>1</sup>



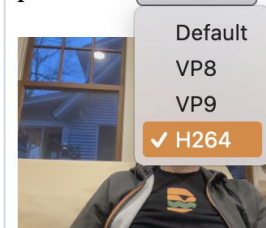
Oddly, `setCodecPreferences` doesn't trigger `negotiationneeded`, so the fiddle has to call its own `pc1.onnegotiationneeded()` explicitly as a workaround. Unfortunately, this hack risks introducing intermittent timing issues with other negotiation needs.

**Proposal:** make `setCodecPreferences` trigger `negotiationneeded` as needed.

The "as needed" part might require some judgement: it's sometimes desirable to call `setCodecPreferences` in "have-remote-offer" just in time to affect an answer. Since answers can carry just as much weight as offers (arguably more), it might be undesirable to trigger what would be a reverse-role negotiation in this case (once it returns to "stable").

We'd need to work out these details in the [check if negotiation is needed](#) algorithm. Language similar to how we handle direction there might work (i.e. consider a match in the local description satisfactory regardless of whether it's an offer or an answer).

pc1 codec: H264 ▾



checking  
connected  
Switching codec to video/H264  
pc1 is sending video/H264  
pc2 is sending video/H264

1. Works in Chrome/Edge and Safari right now. Landing soon in Firefox 127.

# Issue 2956: `receiver.getParameters().codecs` seems under-specified



jan-ivar commented 3 days ago

Member



[receiver.getParameters](#) has this non-normative note:

- `codecs` is set to the value of the "enabled" codecs from the `[[ReceiveCodecs]]` internal slot.

## **NOTE**

*Both the local and remote description may affect this list of codecs. For example, if three codecs are offered, the receiver will be prepared to receive each of them and will return them all from `getParameters`. But if the remote endpoint only answers with two, the absent codec will no longer be returned by `getParameters` as the receiver no longer needs to be prepared to receive it.*

It suggests that post SRD(answer), `receiver.getParameters().codecs` is updated to only reflect what's been negotiated.

But I can't find any normative steps to support this note:

- `[[ReceiveCodecs]]` is [initialized](#) to "the [list of implemented receive codecs](#) for kind"
- `[[ReceiveCodecs]]` is only ever assigned to `[[LastStableStateReceiveCodecs]]` (on rollback)
- `[[ReceiveCodecs]]` does not appear to ever be modified
- The "enabled" flag is only ever set to true

Is the note wrong or are we missing some normative steps? What was the intent?

# Issue 2956: receiver.getParameters().codecs seems under-specified

- `codecs` is set to the value of the "enabled" codecs from the `[[ReceiveCodecs]]` internal slot.

## NOTE

*Both the local and remote description may affect this list of codecs. For example, if three codecs are offered, the receiver will be prepared to receive each of them and will return them all from `getParameters`. But if the remote endpoint only answers with two, the absent codec will no longer be returned by `getParameters` as the receiver no longer needs to be prepared to receive it.*

IOW (assuming 10 enabled codecs total):

- ahead of sLD(offer): 0 or 10 codecs?<sup>1</sup>
- after sLD(offer): 3 codecs
- after sRD(answer): 2 codecs

With roles reversed:

- ahead of sRD(offer): 0 or 10 codecs
- after sRD(offer): 3 codecs
- after sLD(answer): 2 codecs

1) <https://jsfiddle.net/jjb1/9kz2bf85/> seems to suggest 0 in implementations



# Issue 2956: receiver.getParameters().codecs seems under-specified



jan-ivar commented 4 minutes ago

Member

Author



This turns out to be regression from [#2935](#). Prior to that it said this in [sLD](#):

6. Set *transceiver*.[\[\[Receiver\]\]](#).[\[\[ReceiveCodecs\]\]](#) to the codecs that *description* negotiates for receiving and which the user agent is currently prepared to receive.

## **NOTE**

***If the direction is "[sendonly](#)" or "[inactive](#)", the receiver is not prepared to receive anything, and the list will be empty.***

...and this in [create an RTCRtpTransceiver](#):

12. Let *receiver* have a [\[\[ReceiveCodecs\]\]](#) internal slot, representing a list of [RTCRtpCodecParameters](#) dictionaries, and initialized to an empty list.

I.e. it starts out empty and is then populated based on negotiation (which may be non-empty or empty).

# Issue 2956: receiver.getParameters().codecs under-specified

But with [#2935](#) this got replaced with this in [sLD](#):

6. For each of the codecs that *description* negotiates for receiving, execute the following steps:

1. Locate the matching codec description in *transceiver*. [\[\[Receiver\]\]](#). [\[\[ReceiveCodecs\]\]](#).
2. If the matching codec description is not found, abort these steps.
3. Set the "enabled" flag in the matching codec description to "true".

## **NOTE**

***If the direction is "["sendonly"](#)" or "["inactive"](#)", the receiver is not prepared to receive anything, and the list will be empty.***

...and this in [create an RTCRtpTransceiver](#):

12. Let *receiver* have a [\[\[ReceiveCodecs\]\]](#) internal slot, representing a list of [tuples](#), each containing a [RTCRtpCodecParameters](#) dictionaries, and initialized to an list containing all the codecs in the [list of implemented receive codecs](#) for *kind*, and with the "enabled" flag set in an implementation defined manner.

i.e. it starts out being entirely implementation-defined whether it is empty or includes the full set or something in between, and then "enabled" is only ever set to true, never false.

This means it cannot possibly represent the (subset of) codecs that were negotiated, which seems broken.

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

-

# **Background Segmentation Mask (Riju)**

**Start Time: 09:30 AM**

**End Time: 09:50 AM**

# Proposal : BG Mask

- BG Blur available behind flag in Chrome/Edge
- BG Blur [landed](#) in Safari / WebKit 🙌 @youenn

## Non-goals

---

One adjacent goal was to combine Background Replacement with Background Blur as part of an overall Background Concealment API. Presently there's no platform APIs to support Background Replacement (green screen, animated gif, image, video). Combining both might be too premature. Instead, we would like to keep Background Replacement as a separate feature proposal for a later date.

*Agreed, [Background Replacement is now stated as Future Work](#). Some of the new features on Windows have [requirements](#) which might not be available to a lot of users right away, so we work on a minimal set which works on a broader set of clients and add newer features later.*

PoC on Windows (NPU), should work on M series Macs (NPU) [macOS 12 and later](#)  
CrOS today on GPU backend.

# Demo

The screenshot shows a web browser window displaying a configuration page for a video player. The page has a dark theme and includes several sections:

- Video settings:** Includes options for "Different aspect ratio" (Yes), "background/mask constraint" (Yes/Yes), "frame rate constraint" (Yes/Yes), "height constraint" (Yes/Yes), and "width constraint" (Yes/Yes). Below these are input fields for "Video device" (set to "HP 2000 Vision Web Camera v..."), "Video width" (200), "Video height" (200), and "Video frame rate".
- Background options:** Includes a "Foreground overlay color" selector (purple), a "Background color" selector (green), and a "Background image" dropdown (set to "Custom file"). There are also checkboxes for "Fullscreen context" (Main/Window) and a "Create a link to this page" section with "Add" and "Remove link" buttons.
- Deferred frames:** A section titled "Deferred frames: 0 of 14" with a "Show" dropdown. Below it is a list of checkboxes for various frame types, all of which are checked:
  - Mask frames
  - Normal frames
  - Foreground frames
  - Background frames
  - Background color frames
  - Foreground with background color frames
  - Background image frames
  - Foreground with background image frames
  - Background with foreground overlay color mask frames
  - Background with foreground overlay color frames

On the right side of the screen, a Windows Performance Monitor window is open, showing system performance metrics for CPU, Memory, Disk (C:), WiFi, GPU 0, and NPU 0. The system tray at the bottom shows the date and time as 14:47 on 22/04/2023.

# API

```
partial dictionary MediaTrackSupportedConstraints {  
  boolean backgroundMask = true;  
};
```

```
partial dictionary MediaTrackCapabilities {  
  sequence<boolean> backgroundMask;  
};
```

```
partial dictionary MediaTrackConstraintSet {  
  ConstrainBoolean backgroundMask;  
};
```

```
partial dictionary MediaTrackSettings {  
  boolean backgroundMask;  
};
```

```
partial dictionary VideoFrameCallbackMetadata  
  boolean backgroundMask;  
};
```

```
partial dictionary VideoFrameMetadata {  
  boolean backgroundMask;  
};
```

<https://github.com/riju/backgroundBlur/blob/main/explainer.md#background-segmentation-mask-example-green-background>

```
await videoProcessor.readable  
  .pipeThrough(new TransformStream({  
    start(controller) {  
      this.backgroundCanvas = new OffscreenCanvas(this.width, this.height);  
      this.canvas = new OffscreenCanvas(this.width, this.height);  
      this.maskFrame = null;  
    },  
    transform(videoFrame, controller) {  
      // If the video frame is a mask frame, store it for a later use.  
      if (videoFrame.metadata && videoFrame.metadata().backgroundMask) {  
        if (this.maskFrame)  
          this.maskFrame.close();  
        this.maskFrame = videoFrame;  
        return;  
      }  
  
      if (this.maskFrame) {  
        const backgroundContext = this.backgroundCanvas.getContext('2d');  
        const context = this.canvas.getContext('2d');  
  
        // Draw a green background and a black foreground:  
        // Invert and draw the mask frame.  
        backgroundContext.globalCompositeOperation = 'copy';  
        backgroundContext.fillStyle = 'white';  
        backgroundContext.fillRect(0, 0, this.width, this.height);  
        backgroundContext.globalCompositeOperation = 'difference';  
        backgroundContext.drawImage(this.maskFrame, 0, 0);  
        // Draw the background color.  
        backgroundContext.globalCompositeOperation = 'multiply';  
        backgroundContext.fillStyle = 'lime';  
        backgroundContext.fillRect(0, 0, this.width, this.height);  
  
        // Draw the foreground and a black background:  
        // Draw the mask frame.  
        context.globalCompositeOperation = 'copy';  
        context.drawImage(this.maskFrame, 0, 0);  
        // Draw the foreground from the video frame.  
        context.globalCompositeOperation = 'multiply';  
        context.drawImage(videoFrame, 0, 0);  
  
        // Combine the foreground and the green background.  
        context.globalCompositeOperation = 'lighter';  
        context.drawImage(this.backgroundCanvas, 0, 0);  
  
        this.maskFrame.close();  
        this.maskFrame = null;  
      } else {  
        // Draw green.  
        const context = this.canvas.getContext('2d');  
        context.globalCompositeOperation = 'copy';  
        context.fillStyle = 'lime';  
        context.fillRect(0, 0, this.width, this.height);  
      }  
    },  
    // Create and enqueue a new video frame.  
    const {timestamp} = videoFrame;  
    videoFrame.close();  
    controller.enqueue(new VideoFrame(this.canvas, {timestamp}));  
  }},
```

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

-



# **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