RTC Helper

By Kundan Singh, http://kundansingh.com

© 2025-2026, All Rights Reserved.

1. Introduction

The *RTC Helper* software facilitates testing and debugging WebRTC-based applications in the browser by web developers. It can also provide the end user control over various multimedia features such as background blur, media recording or bandwidth management, independent of the application provider's web app. When used as a browser extension, it allows the end user to analyze, customize and improve any audio/video communication on third-party websites that use WebRTC. When integrated in the developer's own web application, it allows the developer to quickly experiment with new ideas such as a new virtual background implementation, or adjustment to video encoder parameters for better quality.

At the core, this software can selectively intercept certain WebRTC and related APIs such as `getUserMedia`, `RTCPeerConnection` and `WebSocket`, and can inject some code to alter their behavior in real-time. It can also intercepts any video elements added to the web page to allow dynamically changing the user interface of the video conferencing layout, or to provide other advanced features.

This allows developers (or users) to customize media streams, video elements, screen sharing, data channel, signaling channel, and quality metrics, among others. Some examples of media stream customization are: replacing webcam video track with other generated content, adding watermark or logo on screen sharing, including local webcam feed in shared screen. Video elements customization allows the user to change the layout of the videos, record the displayed videos, or popout the videos in a separate window. There are many more customizations possible, as discussed in this document.

Objectives: Two main goals of this RTC Helper project are: (1) to allow end-user customization of WebRTC experience, without depending on the features provided or supported by the third-party websites, and (2) to allow developers to rapidly create proof-of-concept (PoC) of innovative ideas and emerging features built on top of WebRTC.

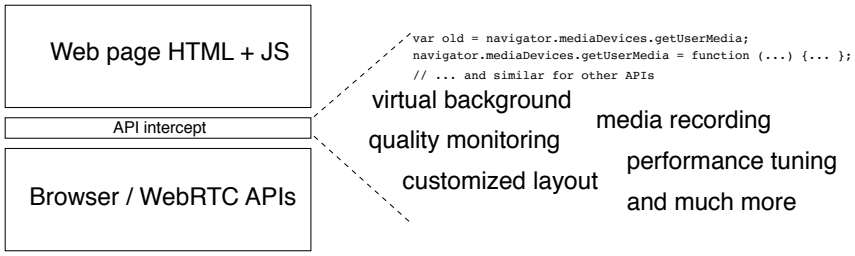
Please see our research paper[1] for this project's motivation, background, related work, and an overview of its software architecture.

1.1 Software Architecture

The software architecture below shows an API intercept layer commonly implemented in web frameworks and tools. Instead of providing a generic but complex intercept process, our software exposes a developer friendly layer for commonly found WebRTC related use cases, divided into more than ten categories or tabs. For example, both the Capture and Media tabs work with the `getUserMedia` API, but the former allows you to customize the video frames captured from the webcam, where as the latter can modify the media constraints, such as to select a specific webcam or disable microphone.

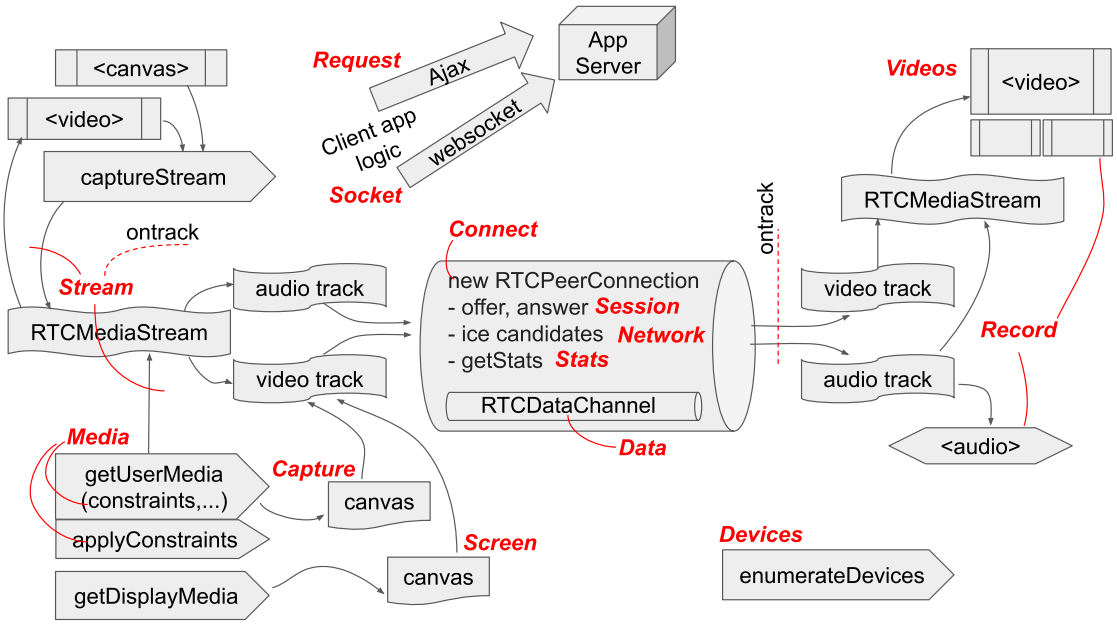
[1]Unlocking WebRTC for End User Driven Innovation, https://doi.org/10.48550/arXiv.2512.23688

RTC Helper © 2025-2026, Kundan Singh



The intercept layer in each category can accept a user or developer supplied function in JavaScript. If provided, the function can alter the default behavior of APIs in that category. We have made available numerous such functions for various examples of customization. But more importantly, you can edit or create your own customization function. And in the future, can share those with other developers, or discover and use functions created by other developers.

As an example, when the web application calls the `getUserMedia` API, this software intercepts it, and prompts the user to customize the behavior. If the user selects to customize, the Capture tab is opened, which allows changing the behavior. If the user declines to customize, the web application proceeds with normal behavior of that API.



The diagram aboves shows the various places in WebRTC and related APIs that are intercepted. The topics mentioned in the diagram in red and italic font represent the various intercept and customization points in this software. For example, the *Connect* category deals with constructing a new `RTCPeerConnection` object, whereas *Network* deals with outgoing and incoming ICE candidates on the peer connection. While *Media* deals with intercept of constraints on `getUserMedia` and `applyConstraints` APIs, the *Capture* and *Screen* categories allow customizing the actual track content by drawing on a canvas.

These topics are further explained later with several examples of the customization functions.

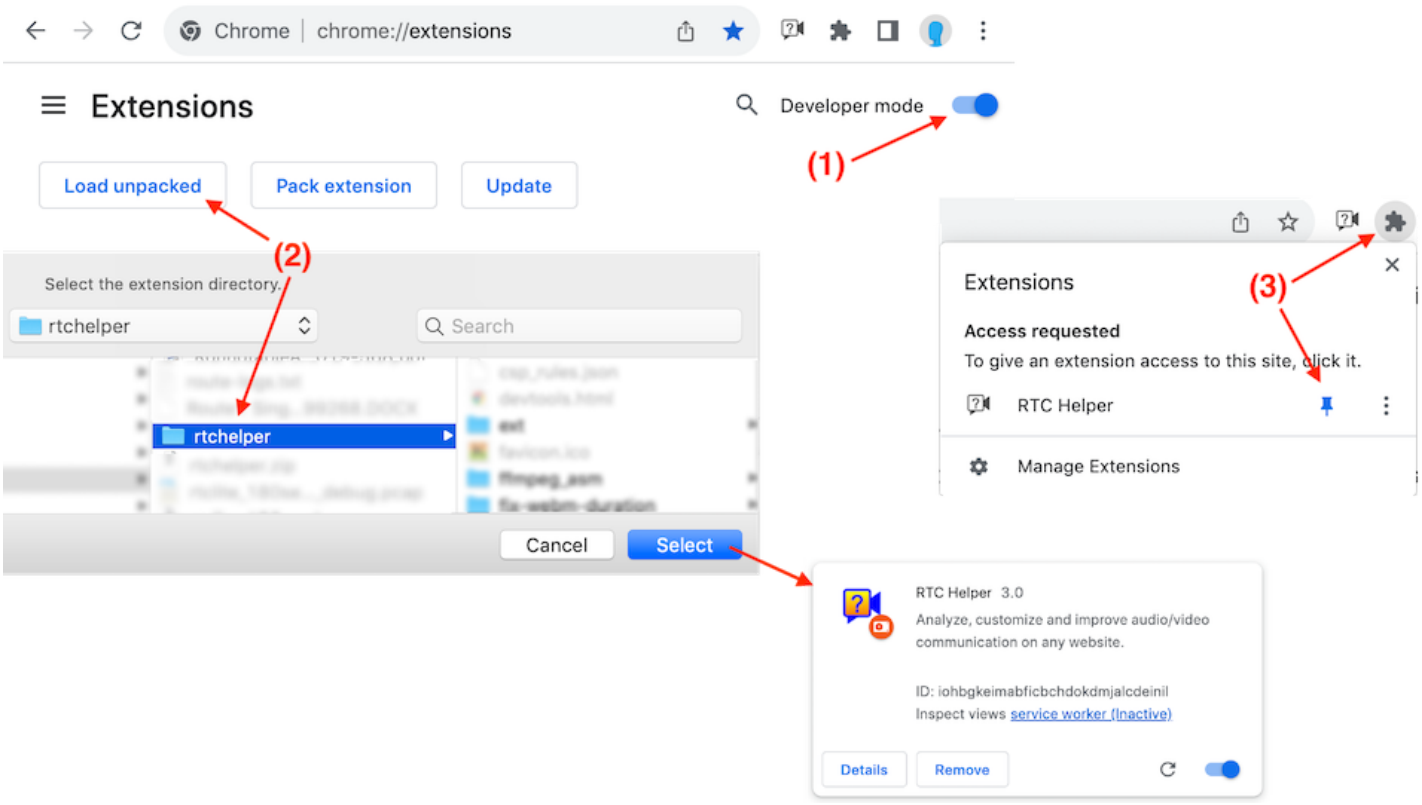
RTC Helper © 2025-2026, Kundan Singh

2. Using the Tool

There are generally three ways to use this software:

1. Any user can install and enable this to test and debug any third-party website using a Google Chrome-compatible browser extension. The browser extension intercepts and applies the features of this software to all the web pages you visit. For privacy reason or to avoid interfering with your regular browsing behavior, it is recommended to use a separate browser persona to enable this extension for web development, testing and debugging effort.
2. As a web developer, you can include this software in your WebRTC enabled website, to make this available to your web app users. You do this using the the <script> tag in your web pages. In this mode all the features in this software are directly available to your web app users, and can be enabled by the end user by clicking on an easter-egg button.
3. As a web developer, you can include some parts of this software via the <script> tag, and enable them programmatically and selectively to your end users, e.g., virtual background or media recording. In this mode, only selected features of this software are enabled for your users, and must be triggered programmatically via your app logic.

Further details on how to accomplish these alternatives are as follows.



1. To install as a browser extension in developer mode, follow the steps in the diagram above - download the rtchelper.zip file, unzip it in a folder, open chrome://extensions, enable developer mode, and click to load the unpacked extension from the unzipped folder. Alternatively, drag the ZIP file from your file explorer or finder window to the browser's extensions page after enabling the developer mode. After that you will notice the new

RTC Helper © 2025-2026, Kundan Singh

extension in the toolbar, and you can pin the extension to be always visible. Once installed, open the included extension.html test webpage to test.

2. To make it available for all users of your website, include the following two scripts in your main HTML file. This creates a hidden button of size 6x6 pixels at the top-right corner of the web page, which when clicked opens the panel for this software.

```
<script type="text/javascript" src="dist/js/inject.js"></script>
<script type="text/javascript" src="dist/js/wrapper.js"></script>
```

Additionally, you can pass certain configurations to the second script to customize the behavior. The following example disables any user driven change in display theme or edit scripts configuration, and makes the default display mode of inline. The actual list of customizable settings is described later or can also be inferred by inspecting the settings panel.

```
src="dist/js/wrapper.js?required=theme:|scripts:&settings=mode:inline"
```

You can also open and analyze the included wrapper.html test webpage that uses this approach.

3. To selectively use some features of this software, include the one script in your main HTML file, and then programmatically invoke the feature. This avoids creating any hidden button or including any user interface panel of this software. Note that the scripts in head and body should be included before any other scripts that may alter the WebRTC API behavior. The following example shows that the webcam capture API, videocapture, is intercepted to replace the captured stream with the supplied app logic, which can add an overlay icon as an example.

```
<head>
    <script type="text/javascript" src="dist/js/inject.js"></script>
    ...
</head>
<body>
    <script type="text/javascript">
        const videocapture = (function(canvas,video) {
            ... // draw on canvas using video and some icon image
        }).toString();

        _rtchelper_content(undefined, undefined, "dist/res");
        _rtchelper_setprocess("videocapture", videocapture);
    </script>
    ...
</body>
```

You can also open and analyze the included script.html test webpage that uses this approach.

RTC Helper © 2025-2026, Kundan Singh

2.1 Settings

Here is the list of customizable settings.

theme - allowed values are light or dark, and defaults to using the underlying browser or system settings.

mode - allowed values are open, inline, open-pinned, open-popup or inline-float, and defaults to open. These values control how the panel is opened - in an external browser tab, in an external tab which is pinned, in an external popup window, as an inline iframe within the web page on the side, or as an inline iframe floating from the side - respectively. For inline or inline-float, optional suffix of :left or :bottom may be present to determine the location of the panel, such as inline-float:left. The location defaults to right.

tabslist - controls which tabs in the panel are shown, and includes comma separated list of one or more from Capture, Screen, Record, Videos, Stream, Devices, Media, Connect, Session, Network, Stats, Data, Socket, Request, Control, Settings, Help. It also controls the order in which these tabs are included in the panel.

tabswrap - controls whether the displayed tabs in the header for navigation are wrapped or scrolled. Allowed values are true (wrap) or false (scroll).

fontsize - controls the fontsize to use as one of small, medium or large. Default is to use small, which appears similar to the browser devtools.

editor - controls the behavior and appearance of the code editor, and includes comma separated list of one or more of these attributes in their positive or negative form: wrap, highlightline, highlightword, indentguide, invisibles, foldcomment. These values control - wrap long lines, highlight selected line, highlight selected word, show indent guides, show invisible characters, fold all block comments - respectively. A negative form is just the value with a no prefix, e.g., nowrap. The default behavior assumes wrap, nohighlightline, nohighlightword, indentguide, noinvisibles, foldcomment.

scripts - allowed values are noedit or named, with default behavior as edit. It controls whether script editing is allowed, and in addition to editing, whether saving the script as named script is allowed. Without named scripts, only one active script is saved per panel. With named scripts, you can experiment with multiple scripts and quickly switch among them.

urls - list of https URLs, one per line, to download the files containing the additional scripts for various tabs. Once loaded these scripts appear in the dropdowns of the panel tabs as applicable. The file must be in JSON format with appropriate objects and attributes to populate various tabs' list of example scripts.

strict - allowed values are always or external, with default as no strict mode. Using the strict mode prevents access to globals and other data in the customizable functions used in various tabs. This is useful if you do not trust the source of those functions or if the function definitions are minified to hide its behavior. You do not need strict mode if these functions are written by you or trusted source, or if you can visually inspect the function definition and find no malicious usage.

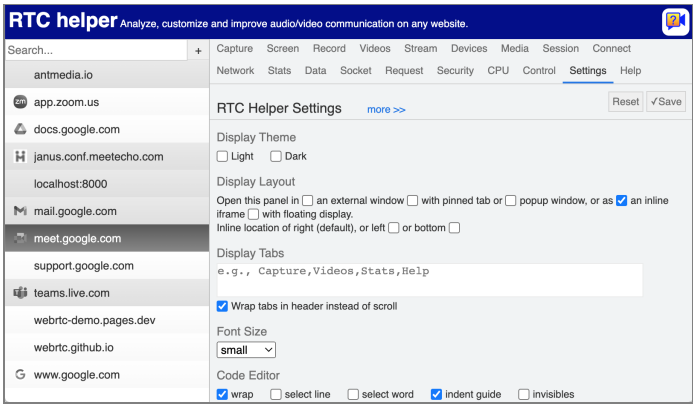
RTC Helper © 2025-2026, Kundan Singh

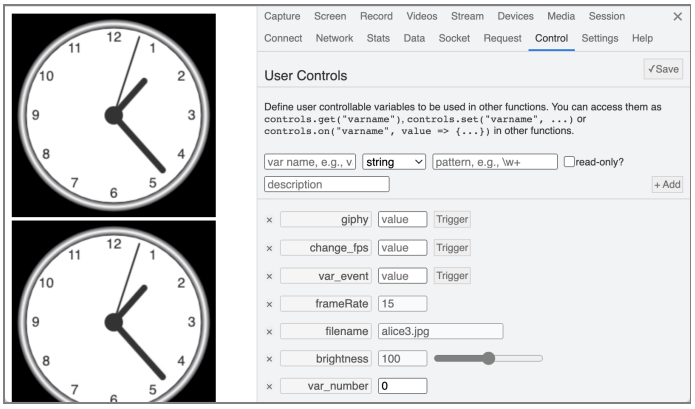
savestats - to store compressed statistics data, the stats tab uses an external service over HTTP POST or websocket. By default it assumes the service running on localhost. You can specify the actual third-party URLs for storage and display of the statistics data. The first line for storage should be a web (http or https) or websocket (ws or wss) URL, and the second line for display should be a web (http or https) URL.

ffmpeg - for converting the recorded video to MP4 format, a specific ffmpeg command is run, such as to transcode video from the default h264 to mpeg4. You can change the command to avoid transcoding, if your player is able to play h264 encoded mp4 format.

2.2 User Interface

Some example user interface of the software is shown below. The first screen is the options panel opened in a new window, showing the website selection on the left, and the customization tabs on the right. The second screen has the options panel opened inline on the side, and shows our included test page with customized webcam capture.





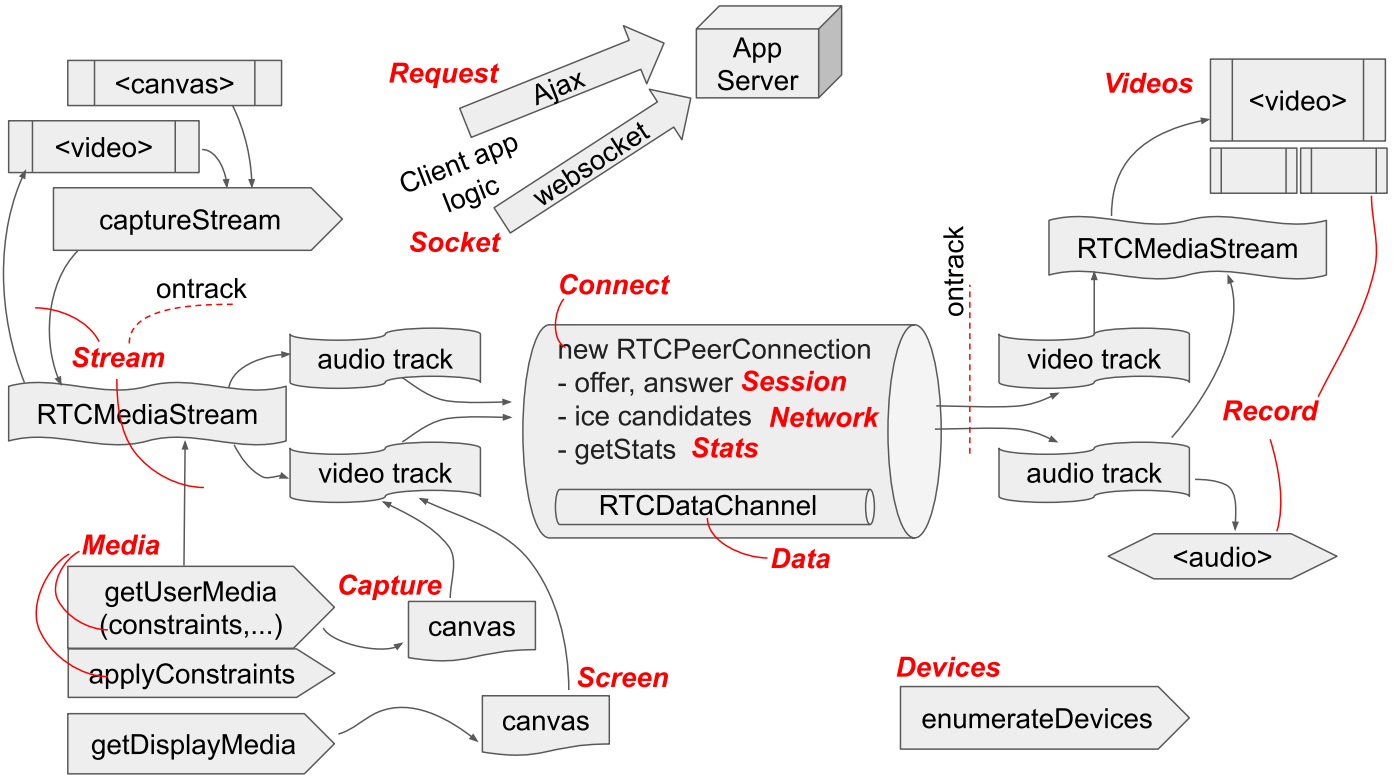
RTC Helper © 2025-2026, Kundan Singh

3. Customization Categories

These high level topics are included:

1. Customize the webcam capture feed
2. Customize the screen capture feed
3. Perform media recording with customized video layout and audio mixing
4. Customize and interact with all the audio and video elements
5. Customize media streams in capture, or receive
6. Customize list of available webcam, microphone and speaker devices
7. Customize user media constraints such as frame rate, width and height
8. Customize session description attributes such as codec and bandwidth
9. Customize connection attributes such as ICE servers and DSCP
10. Customize transport parameters such as ICE candidates
11. Capture quality stats and metrics and store for future analysis
12. Customize data channel behavior
13. Customize WebSocket behavior
14. Customize XMLHttpRequest and fetch behavior
15. Intercept and customize web requests and responses
16. Customize based on CPU performance

The following diagram shows again the various places in the WebRTC and related APIs where these customization categories are applied.



RTC Helper © 2025-2026, Kundan Singh

The various customization categories or tabs are summarized below.

Tab	Summary
Capture	Customize the webcam capture feed. It uses an internal canvas to generate the video returned by any getUserMedia stream, and allows drawing on the canvas for each frame
Screen	Customize the screen capture feed. It uses an internal canvas to generate the video images returned by any getDisplayMedia stream, and allows drawing on the canvas for each frame
Record	Perform media recording with customized video layout and audio mixing. It allows combining all the displayed video elements on the web page into a recorded video, and allows drawing on the internal canvas used for the recorded video stream
Videos	Customize and interact with all the audio and video elements. It allows intercepting and modifying every audio and video element on the webpage
Stream	Customize media streams in capture, or receive. It allows intercepting and modifying any media stream created using getUserMedia, getDisplayMedia, or captureStream, or received on a peer connection in its ontrack event
Devices	Customize list of available webcam, microphone and speaker devices. It allows intercepting and modifying the list of devices returned by any enumerateDevices call, and altering the list devices shown by the application
Media	Customize user media constraints such as frame rate, width and height. It allows intercepting and modifying the constraints supplied to any getUserMedia and applyConstraints calls
Session	Customize session description attributes such as codec and bandwidth. It allows intercepting and modifying various methods related to media session in any RTCPeerConnection object
Connect	Customize connection attributes such as ICE servers and DSCP. It allows intercepting and modifying any call to construct a new RTCPeerConnection
Network	Customize transport parameters such as ICE candidates. It allows intercepting and modifying various method and events related to network connection in any RTCPeerConnection object
Stats	Capture quality stats and metrics and store for future analysis. It allows intercepting and modifying the getStats call in any RTPPeerConnection object
Data	Customize data channel behavior. It allows intercepting and modifying various methods and events on any data channel
Socket	Customize WebSocket behavior. It allows intercepting and modifying various methods and events on any WebSocket object
Request	Customize XMLHttpRequest and fetch behavior. It allows intercepting and modifying various methods and events on the any Ajax request and their responses

RTC Helper © 2025-2026, Kundan Singh

4. Customize the webcam capture feed

It uses an internal canvas to generate the video returned by any getUserMedia stream, and allows drawing on the canvas for each frame. The function defined below is called periodically to draw the video image. The function can access one webcam video or multiple webcam videos. The function may be async, in which case the subsequent calls are not invoked until the previous one returns. These optional named parameters are allowed.

`canvas`	target <canvas> element to draw on to.
`video`	source <video> element from the default webcam.
`videos`	an array of source <video> elements from all the webcams.
`screen`	source <video> element of screen share.
`data`	the state object assigned to the canvas and includes the start time of the stream.
`upload`	asynchronous function for click to select one or more file.
`segment`	asynchronous function to use selfie segmentation using mediapipe on an image or video.
`facedetect`	asynchronous function to use face detection using mediapipe on an image or video.
`respath`	path string to access internal resources such as images.
`cleanup`	if present, then the function is additionally called on cleanup with parameter value of true.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access the shared variables with the controls tab.

Some builtin and preloaded examples are described next.

4.1 Camera - show combined feed from two webcams

Assuming there are more than one webcams, show them side by side, with center zoom. This is similar to horizontal split effect on TV during some interviews.

For single webcam scenario, it just displays that webcam video.

4.2 Camera - show video from the default webcam

Show video from the default webcam

4.3 Color - show changing random color as video feed

Show solid random color continuously changing.

RTC Helper © 2025-2026, Kundan Singh

4.4 Color - show random color as video feed

Show solid random color in place of video image.

4.5 Color - show solid black color as video feed

Show solid black color in place of video image.

4.6 File - show a animated gif overlay on event

Load a third-party gif file (Jim Carrey - Oh Come On! or Bob Downey Jr - frustrated) and use that as overlay. It uses an external libgif-js library to parse the gif file format. This function does not work in strict mode, as it requires adding a third-party library to parse gif files.

4.7 File - show animated gif file as video feed

Load a third-party gif file (Matrix animation) and use that video feed. The animation in the gif file is repeated in the video feed. It uses an external libgif-js library to parse the gif file format. This function does not work in strict mode, as it requires adding a third-party library to parse gif files.

4.8 File - show random image with brightness variable

Use randomly selected pre-configured image as video feed, and set the selected image file name in the controls variable filename. The image location is assumed to be at respath (resource path) and file names are of the type (alice|bob)[123].jog, e.g., bob2.jpg. This is useful for captuing screenshots for demonstrations and tutorials.

4.9 File - show randomly selected image as video

Use randomly selected pre-configured image as video feed. The image location is assumed to be at respath (resource path) and file names are of type (alice|bob)[123].jpg, e.g., bob2.jpg. This is useful for capturing screenshots for demonstrations and tutorials.

4.10 File - upload image files and rotate them as video feed

Allow click to upload one or more image files, and show them as video feed. The image files are rotated every 5 seconds. Only jpeg and png files are allowed. A label to click-to-open is shown until the file upload is completed.

4.11 File - upload video files and share them as video feed

Allow click to upload one or more mp4 video files, and show them as video feed. The files are shown one after another, and the video feed stops when the last selected video file finishes playing. Only mp4 files are allowed. A label to click-to-open is shown until the file upload is completed.

RTC Helper © 2025-2026, Kundan Singh

4.12 Image - show logo and text on top of webcam video

Show a tiny logo and text on webcam video on the bottom right corner. The logo file and text are pre-configured.

4.13 Mediapipe - blur and hide background on webcam video

Using the tensorflow mediapipe library and its body segmentation model, blur and semi-hide the background of webcam video. A blur filter of 8px, is used along with other filters to reduce the clarity and color of the background.

4.14 Mediapipe - blur and hide foreground on webcam video

Using the tensorflow mediapipe library and its body segmentation model, blur and semi-hide the foreground detected face. The background is shown as is. This is useful for anonymity of the person shown in the webcam video. A blur filter of 16px is used for the foreground, along with removing any colors.

See motivation in https://www.scribd.com/document/932752927/Real-Time-Face-blurring and idea in https://github.com/Jaysheel11/Real-time-face-blurring

4.15 Mediapipe - blur background on webcam video

Using the tensorflow mediapipe library and its body segmentation model, blur the background on webcam video. A blur filter of 8px is used.

4.16 Mediapipe - highlight detected faces in webcam video

Using the tensorflow mediapipe library and its face detection model, highlight the detected faces on webcam video. It uses a yellow circle with line width of 4px centered at the detected face.

See https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector

4.17 Mediapipe - remove background on webcam video

Using the tensorflow mediapipe library and its body segmentation model, remove background on webcam video. The background is turned white.

4.18 Mediapipe - show mask of foreground detection on webcam

Using the tensorflow mediapipe library and its body segmentation model, show mask of the foreground. The mask is colored black, and background is colored white.

RTC Helper © 2025-2026, Kundan Singh

4.19 Mediapipe - show randomly changing virtual background on webcam

Using the tensorflow mediapipe library and its body segmentation model, show a virtual background on webcam video. It uses a third-party website to load the background image using the landscape search term. The background images are randomly picked and changed every 10 seconds. The background is displayed with object-fit cover style.

4.20 Mediapipe - show screen share with background removed webcam video

Prompt to select for screen or window share. Use that along with the webcam video to display on top of the screen share video. For the webcam video, it uses body segmentation using the tensorflow mediapipe library, to remove the background, and make it transparent. The webcam video is positioned at the bottom left corner, and is constrained to capture at 320x180. If no screen share is selected, it just displays the webcam video as is.

4.21 Mediapipe - show screen share with face zoomed webcam video

Prompt to select for screen or window share. Use that along with the webcam video in picture-in-picture mode in a smaller circle, instead of a rectangle. For the webcam video, it uses face zoom, using the tensorflow mediapipe library and its face detection model, by doing zoom and pan to the detected face. It uses a filter to avoid changing the view too quickly, and performs face detection only once per second, for improved performance. If no face is detected for some time, then the video is blurred, for privacy.

See https://github.com/cainhill/WebcamCircle, https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0060352#auto-frame and https://stackoverflow.com/questions/71557879/record-video-and-screen-together-and-overlay-with-javascript

4.22 Mediapipe - upload and show a virtual background on webcam

Using the tensorflow mediapipe library and its body segmentation model, show a virtual background on webcam video. It allows uploading one image or video file to be used as the background. Only the jpeg, png and mp4 files are allowed. A video background is looped. A label to click-to-open is shown until the file upload is completed. The background is displayed with object-fit cover style.

4.23 Mediapipe - zoom and pan on detected face in webcam video

Using the tensorflow mediapipe library and its face detection model, implement zoom and pan on detected face similar to Zoom's Auto-Framing Camera, or Apple's Center Stage feature. It uses a filter to avoid changing the view too quickly, and performs face detection only once per second, for improved performance. If no face is detected for some time, then the video is blurred, for privacy.

See https://mediapipe.readthedocs.io/en/latest/solutions/autoflip.html, https://support.apple.com/en-us/111102 or https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0060352#auto-frame

RTC Helper © 2025-2026, Kundan Singh

4.24 No operation - do not generate any frames

Do not do anything, and do not generate any frames. This causes the video to not load the first frame.

4.25 Remote - receive remote video stream and use that as webcam

Receive video stream from a separate publisher, and deliver that as the local webcam track. It uses the external software described in https://blog.kundansingh.com/2019/12/named-stream-abstraction-for-webrtc.html for named stream, to subscribe to a randomly picked stream name, and shows a QR code for the publisher to start publishing to that stream using external device such as a mobile phone or separate laptop. The subscribed stream is then played as local webcam stream. For testing this, copy the streams.html file from that project in the test directory of this project, and run the streams.py server on default port 8080, assuming this server is on default port 8000, to match the ports and URLs used in this function.

4.26 Screen - show screen or app shared as video

Prompt to select for screen or window share, and use that as the video feed.

4.27 Screen - show screen share with picture-in-picture of webcam

Prompt to select for screen or window share. Use that along with the webcam video in picture- in-picture mode. If screen selection is not done, then only the webcam video is shown in full size.

4.28 Timer - show analog clock with time zone offset variable

Show an analog clock with current time, and timezone offset applied from a controls variable tzoffset. It also sets the controls variable started_on with the date/time when this function is called the first time for a canvas. It uses code from https://www.w3schools.com/graphics/canvas_clock.asp

4.29 Timer - show sub-second digital timer

Show a digital timer counting up in fractional seconds. Black background with white text in the center is used to display the timer.

4.30 Timer - show the current time in an analog clock

Show an analog clock with the current time. It uses code from
https://www.w3schools.com/graphics/canvas_clock.asp

RTC Helper © 2025-2026, Kundan Singh

5. Customize the screen capture feed

It uses an internal canvas to generate the video images returned by any getDisplayMedia stream, and allows drawing on the canvas for each frame. This is similar to the webcam capture feature. The function defined below is called periodically to draw the video image. The function can access one screen or app video, one webcam video or multiple webcam videos, or multiple screen or app videos. For single screen or app video use the screen parameter described below. For multiple screen or app videos, use screen1, screen2, and similar parameters. The function may be async, in which case subsequent calls are not invoked until the previous one returns. These optional named parameters are allowed.

`canvas`	target <canvas> element to draw on to.
`video`	source <video> element from the default webcam.
`videos`	an array of source <video> elements from all the webcams.
`screen`	source <video> element of screen or app share, or use screen1, screen2, and so on for multiple sources.
`respath`	path string to access internal resources such as images.
`data`	the state object assigned to the canvas and includes the start time of the stream.
`cleanup`	if present, then the function is additionally called on cleanup with parameter value of true.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

5.1 Screen - log if there are significant change

Log if the shared screen or window has changed, e.g., indicating a slide change. This is done by capturing pixels every second, and comparing the diff.

5.2 Screen - pixelate screen share for privacy of text

Pixelate captured screen using an intermediate canvas that avoids smoothing. Default pixelation size is 4. increase this to make more blocking pixelation.

5.3 Screen - share two apps in the same stream

Capture two apps to share, and create a combined video containing both the apps video. The videos may be arranged horizontally or vertically depending on the sizes of those app windows.

RTC Helper © 2025-2026, Kundan Singh

5.4 Screen - show logo and text at the bottom right corner of screen video

Show a tiny logo and text on screen share video on the bottom right corner. The logo file and text are pre-configured.

5.5 Screen - show overlay text on top of screen video

Show a text overlay using controls variable.

5.6 Screen - show screen share with picture-in-picture of webcam

Use the webcam video in picture-in-picture mode at bottom left corner of the screen share video. The webcam video is in a circle with diameter 1/4th the smaller of height or width of the screen video.

5.7 Screen - show text overlay with auto-generated caption from mic

Use browser's SpeechRecognition feature to show caption in real-time using microphone captured audio, on top of the screen share video. Since only one speech recognition object should be active, it uses the cleanup parameter to cleanup the object when canvas stream is cleaned up. Note that the quality of the built-in speech recognition is not the best.

5.8 Screen - subset rectangle selection for screen share

Prompt to select for screen or window share, and then allow drag-select a subset rectangle from the captured video to actually share. The position and size of the subset rectangle remains fixed, so apps or windows can be moved in or out of that area.

RTC Helper © 2025-2026, Kundan Singh

6. Perform media recording with customized video layout and audio mixing

It allows combining all the displayed video elements on the web page into a recorded video, and allows drawing on the internal canvas used for the recorded video stream. The function defined below is called periodically to draw the combined recorded video image. Audios from all the audio and video elements are implicitly combined in the recording for all audio sources of media streams. The function may be async, in which case the subsequent calls are not invoked until the previous one returns. These optional named parameters are allowed.

`canvas`	target <canvas> element to draw on to.
`videos`	an array of all the <video> elements on the page.
`states`	an array of strings representing video element state in the same order, e.g., one of playing, empied, suspend, ended, or error.
`data`	the configuration object used as follows. On first invocation, the frameRate and mimeType attributes of data may be set in the function. Defaults are 15 (fps) and 'video/webm; codecs="h264,opus"'. Use mimeType of 'video/webm; codecs="h264,pcm"' to enable download in MP4, which currently works only on Chrome.
`usermedia`	if present then includes audio from the user media calls, e.g., if microphone audio is not attached to any audio or video element.
`mediaconnect`	if present then intercepts audio from peer connections instead of audio and video elements.
`noaudio`	if present then disables audio recording.
`getstyle`	function to call in lieu of window.getComputedStyle on a video object in strict mode.
`cleanup`	if present, then the function is additionally called on cleanup with parameter value of true.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

6.1 Record all playing videos in a smarter layout

Include only the playing videos in a layout that attempts to reduce the empty space. This layout idea is borrowed from an earlier videocity project. https://github.com/theintencity/videocity

The recording layout is scaled to fit in 640x480 size. Each video is assumed to have object-fit of cover.

6.2 Record all playing videos side-by-side

Combine all playing videos side-by-side zooming at the center in a wide video.

RTC Helper © 2025-2026, Kundan Singh

6.3 Record all videos even if empty in an NxN layout

Include all the videos in an NxN layout. Even if the video is not showing, it is included in the layout. The layout size is determined by the next NxN that can fit the number of videos. For example, if there are 10 videos, then N is 4, in a 4x4=16 layout. Other slots remain empty.

The recording layout is scaled to fit in 640x480 size. The videos are scaled to fit in 4:3 aspect ratio, without maintaining their original aspect ratio.

6.4 Record all videos in their original layout, enabling mp4 download

Include all the videos in their original relative position on the web page. The layout of the videos in recording is calculated based on the relative positions and sizes of videos as they appear on the page. When their positions and sizes change, the recording layout is also updated. The recording layout is scaled to fit in 640x480 size.

Each video element's object-fit style of cover or contain is honored in the layout. A background color of gray and dark gray is used for empty space and non-playing videos.

Additionally, the content type is configured to allow conversion to and download in mp4 format.

6.5 Record one video at a time, but allow selecting

Open a popup window to allow selecting the video element to record. The window is closed automatically when recording is stopped. It uses cleanup parameter to close the popup window. The window appears on right side of the screen. It also puts a text label overlay on the video to indicate which video is selected, or a text of "Not Available" if the video content is not available.

RTC Helper © 2025-2026, Kundan Singh

7. Customize and interact with all the audio and video elements

It allows intercepting and modifying every audio and video element on the webpage. The function defined below is called for every audio and video element on the web page, on creation, destroy, state change and click. It can intercept changes to the media elements. The function may be async. These optional named parameters are allowed.

`video`	the source <audio> or <video> element.
`open`	function to call in lieu of window.open with blob URL in a blank tab in strict mode.
`data`	the state object assigned to the element and includes an state attribute with string value such as found, ignored, added, removed, playing, suspend, emptied, or ended.
`getstyle`	function to call in lieu of window.getComputedStyle on a video object in strict mode.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

7.1 Add a button to pause or play all videos

Show a button to pause and play all videos on the page.

7.2 Assign a random image as poster

Set a random image as poster on each video.

7.3 Click on a video to open it in a separate window

Allow click on any video element to open it in a separate window. The original video element on the web page is paused while it is playing in the separate window. When the separate window is closed, the one on the web page starts playing again.

The video element's srcObject property is used to clone the video stream to the separate window.

7.4 Click to copy and open all playing videos in a separate window

Allow click on any video element to open all the playing videos in a separate window. The original video elements on the web page are paused while they are playing in the separate window. When the separate window is closed, all the video elements on the web page are restored.

RTC Helper © 2025-2026, Kundan Singh

The video element's srcObject property is used to clone the video stream to the separate window. If the video element is not playing, it is not included on the separate window.

7.5 Click to expand video to full window size

Click on a video element to make it occupy full window size.

7.6 Click to move and open all videos in a separate window

Allow click on any video element to open all the videos in a separate window. The original video elements on the web page are hidden and paused while they are playing in the separate window. When the separate window is closed, all the video elements on the web page are restored.

The video element's srcObject property is used to clone the video stream to the separate window. Even if a video element is not playing, it is included in the separate window.

7.7 Click to show picture-in-picture popout with all the videos

Allow click on any video element to open all the playing videos in a picture-in-picture popout. It uses the built-in picture-in-picture feature, and internally creates a canvas backed media stream to attach to the picture-in-picture popout.

The layout of the videos are preserved to their relative positions and sizes on the web page. If the positions and sizes of the video changes, so does the layout of the picture-in-picture popout content.

Each video element's object-fit style of cover or contain is honored in the layout. A background color of gray and dark gray is used for empty space and non-playing videos.

7.8 Disable controls on all media elements and set a background

Disable built-in controls on all the audio and video elements, and set a gray background color.

7.9 Disable fullscreen, download, and picture-in-picture in controls

Disable built-in buttons for fullscreen, download and picture-in-picture.

7.10 Make video display in a circle shape

make all videos in a circle shape with fixed size and black background, using CSS.

RTC Helper © 2025-2026, Kundan Singh

7.11 Make video display transparent, if not on mouse hover

fade away video display, unless mouse hover, using opacity, and transition styles.

RTC Helper © 2025-2026, Kundan Singh

8. Customize media streams in capture, or receive

It allows intercepting and modifying any media stream created using getUserMedia, getDisplayMedia, or captureStream, or received on a peer connection in its ontrack event. The function defined below is called for each such instance. It can intercept changes to the media stream and included media tracks. Because of the underlying implementation, the function may be async only for getUserMedia or getDisplayMedia use cases, but not for others. These optional named parameters are allowed.

`stream`	the media stream object containing zero or more media tracks, or a track object if a streamless track is received.
`context`	a string representing the intercepted function such as getUserMedia, getDisplayMedia, captureStream, or ontrack.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

8.1 Disable any captured stream on media or canvas element

It throws an error when captureStream is called on any media or canvas element.

8.2 Disable/stop audio track in received media stream

Stop the audio track in received ontrack event. This will cause no audio on received stream.

8.3 Receive remote video stream and use that as webcam and mic

Replace captured webcam/microphone stream with audio/video from third-party video feed. It uses the external software described in https://blog.kundansingh.com/2019/12/named-stream-abstraction-for-webrtc.html for named stream, to subscribe to a fixed stream name, which can be published by external device such as mobile phone or separate laptop. For testing this, copy the streams.html file from that project in the test directory of this project, and run the streams.py server on default port 8080, assuming this server is on default port 8000, to match the ports and URLs used in this function.

8.4 Remove audioinput track from any captured stream

Remove any microphone/audioinput tracks from getUserMedia or getDisplayMedia streams.

RTC Helper © 2025-2026, Kundan Singh

8.5 Replace captured webcam/mic stream with a video file

Replace captured webcam/microphone stream with audio/video from an external video file. It loads the video file in a video element from a URL, and uses the captureStream to return the stream of that video element. To ensure that the audio is not muted, the video element's muted property is false, but the volume is set to very low, to avoid local audio playback. The video element is displayed in the bottom right corner, and played in a loop. To avoid displaying the video element, you can use visibility: hidden style. If the video element is not present in the foreground, then the browser may not play the video. The controls on the video element can be used to seek forward or reverse in the timeline.

8.6 Replace captured webcam/mic stream, but hide video file

Replace captured webcam/microphone stream with audio/video from an external video file. This is similar to the previous example, but does not loop, and avoids displaying the video element.

8.7 Replace received media stream with an external video

Replace a received stream with a custom stream from the externally loaded video source. It uses the video element to load an external video file. Then it uses captureStream to get the audio/video stream of that video. And finally it return this new stream, instead of the original for the ontrack event. This causes the received stream to change to the external video content. Double click on the external video to close it, and switch the received stream back to the original.

8.8 Show a text overlay when video stream is captured

It uses an internal canvas to generate a video feed including the original video in the stream, and an overlay text on top.

8.9 Show a warning text when video/canvas stream is captured

It uses an internal canvas to draw the text, and returns the captureStream of that canvas instead. Invoking captureStream in this customization does not recursively call this function. Since it checks for source node name, apps can bypass this by using a derived object from those built-in classes. To enforce, ignore the node name check.

RTC Helper © 2025-2026, Kundan Singh

9. Customize list of available webcam, microphone and speaker devices

It allows intercepting and modifying the list of devices returned by any enumerateDevices call, and altering the list devices shown by the application. The function defined below is called to intercept the enumerateDevices calls, and to alter the list of devices returned. The function must return the new list of devices. The function may be async. These optional named parameters are allowed.

`devices`	list returned by an internal call to enumerateDevices.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

9.1 Add dummy devices in each category

Add a dummy device in each catogory, e.g., dummy camera, microphone and speaker.

9.2 Hide all microphone audioinput devices

Do not include any microphone devices, in device enumeration.

9.3 Hide all webcam videoinput devices

Do not include any webcam devices, in device enumeration.

9.4 Select only one default per category

Allow only one device per category, e.g., one microphone, one speaker and one webcam, in the device list.

RTC Helper © 2025-2026, Kundan Singh

10. Customize user media constraints such as frame rate, width and height

It allows intercepting and modifying the constraints supplied to any getUserMedia and applyConstraints calls. The function defined below is called to intercept the getUserMedia and applyConstraints calls, and to alter the supplied constraints. The function can modify the supplied constraints object or return a new constraints object. The function may be async. These optional named parameters are allowed.

`constraints`	the object that was supplied to the intercepted function as constraints.
`context`	a string representing the intercepted function such as getUserMedia or applyConstraints.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

10.1 Remove audio constraints to avoid microphone capture

Remove microphone capture from the constraints, to avoid sound capture.

10.2 Select a specific webcam to capture

Set the minimum frame rate of 15fps, if the video constraints is set, and has frame rate configured. This won't work in strict mode since it accesses the global navigator object.

10.3 Set minimum capture frame rate of 15

Set the minimum frame rate of 15fps, if the video constraints is set, and has frame rate configured.

RTC Helper © 2025-2026, Kundan Singh

11. Customize session description attributes such as codec and bandwidth

It allows intercepting and modifying various methods related to media session in any RTCPeerConnection object. The function defined below is called to intercept the session description objects in the createOffer, createAnswer, setLocalDescription and setRemoteDescription methods of any RTCPeerConnection object. The function may be async, in which case the intercepted function awaits its completion. The function may return the new or modified session object, or may alter the supplied session object. These optional named parameters are allowed.

`id`	a unique string representing the underlying peer connection.
`connection`	the underlying RTCPeerConnection instance used internally.
`session`	the offer or answer object of type RTCSessionDescription associated with the intercepted function.
`context`	a string representing the intercepted function such as createOfferOnSuccess, createAnswerOnSuccess, setLocalDescription or setRemoteDescription.
`data`	an state object assigned to the peer connection.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

11.1 Disable video in session description

Disable any video media type in the session description (SDP)

11.2 Include only H264 in offer session

Modify the session description (SDP) to only allow H264 video codec. The related RTX formats are also adjusted based on whether the associated codec was included or not in the session. This is done only for the createOffer response, typically for the publishing webcam stream.

11.3 Limit receiver bandwidth to 100kbps

Modify the session description to limit receiving total bandwidth to 100kb/s. This is done only for the remote description, and thus applies only to the receiver side streams.

RTC Helper © 2025-2026, Kundan Singh

11.4 Limit sender video bandwidth to 75kbps

Modify the session senders to limit the maximum video bitrate to 75kb/s. This is done only for the local description, and thus applies only to the video publishing stream.

11.5 Limit sender video framerate to 5 fps

Modify the session senders to limit the maximum frame rate to 5fps. This is done only for the local description, and thus applies only to the video publishing stream.

11.6 Prefer G711 and remove G722 and ISAC in audio session

Modify the session description (SDP) to prefer G711 (pcma and pcmu), and remove G722 and ISAC codecs. This is done only for the createOffer and createAnswer responses, and hence, applies to both publishing and playing audio stream.

11.7 Print the session description

Print the session description (SDP) data

RTC Helper © 2025-2026, Kundan Singh

12. Customize connection attributes such as ICE servers and DSCP

It allows intercepting and modifying any call to construct a new RTCPeerConnection. The function defined below is called to intercept the constructor of RTCPeerConnection, and to alter the supplied configuration and constraints. The function can modify the supplied configuration or return the new configuration. The function can also modify the supplied constraints. The function must not be async. These optional named parameters are allowed.

`id`	a unique string representing the peer connection.
`config`	the object supplied to the intercepted constructor function as configuration.
`configuration`	same as config above.
`constraints`	the object supplied to the intercepted constructor function as optional constraints.
`data`	an state object assigned to the peer connection.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

12.1 Enable proprietary constraints for DSCP

Enable Google's proprietary constraints of using DSCP packet marking. For more information see http://www.rtcbits.com/2017/01/using-dscp-for-webrtc-packet-marking.html

12.2 Force a specific media relay, and use relay policy

Always use a specific TURN server, and remove all others, if supplied by the application. Also force use the TURN server.

12.3 Inject a specific reflexive and relay server

Always use a specific STUN and TURN server. This is in addition to any application supplied configuration.

RTC Helper © 2025-2026, Kundan Singh

13. Customize transport parameters such as ICE candidates

It allows intercepting and modifying various method and events related to network connection in any RTCPeerConnection object. The function defined below is called to intercept the ICECandidate objects in the icecandidate event and the addIceCandidate method of any RTCPeerConnection object. The function may modify the supplied candidate or return the new candidate. If the return value is null, then the supplied candidate is not used and is filtered out. The function must not be async. These optional named parameters are allowed.

`id`	a unique string representing the underlying peer connection.
`connection`	the underlying RTCPeerConnection instance used internally.
`candidate`	the object of type RTCCandidate associated with the intercepted function or event.
`context`	a string representing the intercepted function or event such as icecandidate or addIceCandidate.

Following optional parameters are common across most tabs.

`data`	an state object assigned to the peer connection.
`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

13.1 Filter out candidates with IPv6 addresses

Remove candidates with IP version 6 addresses, if any.

13.2 Filter out host candidates with private IP addresses

Remove host candidates with private IP addresses, if any.

13.3 Include only a specific relay candidate and remove others

Allow a specific TURN server candidate, and ignore all others. This could force the media path to always go through that TURN server, but must be done together with the connection function to inject or limit the use of TURN servers.

RTC Helper © 2025-2026, Kundan Singh

14. Capture quality stats and metrics and store for future analysis

It allows intercepting and modifying the getStats call in any RTPPeerConnection object. The function defined below is called periodically to capture and analyze the statistics data returned by getStats on any RTCPeerConnection object. In addition, it also intercepts any other other methods and events, for logging or other purpose. The function may be async. These optional named parameters are allowed.

`id`	a unique string representing the underlying peer connection.
`connection`	the underlying RTCPeerConnection instance used internally.
`type`	a string indicating either method or event.
`name`	a string with method or event name such as setLocalDescription or icecandidate.
`args`	a list of arguments to the intercepted method or event.
`data`	an state object assigned to the peer connection.
`videos`	an array of all the <video> elements on the page.
`parsequery`	a convenience function that can be used to parse SQL-like query on getStats result.
`query`	a convenience function that can be used to perform SQL-like query on getStats result.
`plot`	a function that plots graph in local display panel, e.g., using metrics derived from getStats result.
`compress`	a function that can be used to compress the getStats result.
`send`	a function to send the compressed stats to the default or configured monitoring service.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

14.1 Capture video frame rate from outbound-rtp and set a variable

Get the current frame rate used in the outbound-rtp video reports, and set the controls variable frameRate with that value in fps. The value is updated whenever a change is detected during the stats collection interval.

14.2 Change sender frame rate from controls tab event trigger

Intercept change in controls variable change_fps, and use that to trigger a change in the session senders frame rate. This only applies to the video publishing stream. The function ensures proper cleanup of the change handler.

RTC Helper © 2025-2026, Kundan Singh

14.3 Display bitrate, delay, jitter, packets lost and available bitrate locally

Display certain quality metrics locally for audio (a) and video (v) in send (s) and receive (r) direction.

14.4 Display total bitrate inbound and outbound locally

Display calculated total bitrate locally

14.5 Log all connection methods and events

Log all connection methods and events

14.6 Save connection statistics with customization

Save customized statistics to the default storage server. It includes only candidate-pair, local/remote- candidate, outbound/inbount-rtp from the statistics, and adds extra metric about difference between estimated and actual bitrate in send and receive. A positive value indicates the client is within the estimated bitrate, and a negative value or close to 0 indicates that the client is attempting to send or receive more than estimated bitrate, and could cause a problem.

14.7 Save connection statistics with different interval

Save connection statistics with 1 second interval to the default storage server.

14.8 Save events and connection statistics to default server

Save connection events and statistics to the default storage server.

Internally, compress uses delta compression inspired by https://github.com/fippo/rtcstats/blob/master/rtcstats.js

RTC Helper © 2025-2026, Kundan Singh

15. Customize data channel behavior

It allows intercepting and modifying various methods and events on any data channel. The function defined below is called for various data channel related method and events to intercept and alter the behavior. The function must not be async. These optional named parameters are allowed.

`id`	a unique string representing the underlying peer connection.
`connection`	the underlying RTCPeerConnection instance used internally.
`type`	a string indicating method, event, RTCDataChannel.method or RTCDataChannel.event.
`context`	a string representing the intercepted function or event.
`channel`	the underlying RTCDataChannel object.
`args`	a list of arguments to the intercepted method or event.
`data`	an state object assigned to the peer connection.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

15.1 Add prefix to text data sent and received

Prefix sent text with "> " and received text with "< ". Ignore if it is not text string.

15.2 Add sha1-hmac integrity protection in text data

include hmac-sha1 in sent data, and verify on receive.

See https://gist.github.com/noonat/4014105

15.3 Disable any data channel

Disallow use of any data channel by throwing an error on outbound data channel, and ignoring inbound data channel event.

15.4 Log all messages sent and received on data channel

Log all the messages sent and received on any data channel.

RTC Helper © 2025-2026, Kundan Singh

16. Customize WebSocket behavior

It allows intercepting and modifying various methods and events on any WebSocket object. The function defined below is called for various WebSocket related method and events to intercept and alter the behavior. These optional named parameters are allowed.

`socket`	the underlying WebSocket instance used internally.
`type`	a string indicating method or event.
`context`	a string representing the intercepted function or event, with special values of preconstruct and construct.
`args`	a list of arguments to the intercepted method or event.
`data`	an state object assigned to the WebSocket.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

16.1 Change localhost to 127.0.0.1 in URL

Change the web socket target URL host. This example changes localhost to 127.0.0.1. Similar technique can be used for other types of changes, e.g., to add an authentication parameter, or to force use of secure websocket URL (wss).

16.2 Delay connection by randomized 2-10 seconds

Delay web socket connection attempt by a random time between 2 and 10 seconds. This is useful for emulating high connection setup latency in testing.

16.3 Delay send and receive by 200ms

Delay data send and receive on websocket by 200 milliseconds. Similar function can be used for emulating high latency network conditions. In case of random and variable delay, it should ensure in-order delivery of underlying send and receive method and event.

16.4 Disable any web socket

Disallow use of any web socket by throwing an error during web socket construction.

RTC Helper © 2025-2026, Kundan Singh

16.5 Encrypt and decrypt all text messages sent and received

use simple RC4 cipher to encrypt sent message and decrypt received message for basic obfuscation against reverse engineering using browser devtools.

See https://gist.github.com/salipro4ever/e234addf92eb80f1858f

16.6 Force close web socket after 30 seconds

Impose a 30s disconnection event on web socket. This is useful for emulating strict firewall rules of certain enterprises. The function ensures cleanup of timer handler.

16.7 Log all messages sent and received on web socket

Log all the messages sent and received on any web socket.

RTC Helper © 2025-2026, Kundan Singh

17. Customize XMLHttpRequest and fetch behavior

It allows intercepting and modifying various methods and events on the any Ajax request and their responses. The function defined below is called for various Ajax related method and events to intercept and alter the behavior. In the intercepting function for request or response, removing the arguments from the args list suppresses the request or response, modifying the args list item can affect the next behavior, returning from the function can return a success response, and throwing an error can return an error response. Additionally, calling the resolve or reject function with the right parameter, causes the related request or response to skip further processing and return the success or error response, respectively. These optional named parameters are allowed.

`context`	a string representing the intercepted function or event; and is one of fetch, fetchresponse or fetcherror when intercepting fetch, or one of open, send or readystatechange when intercepting XMLHttpRequest.
`args`	a list of arguments to the intercepted method or event.
`xhr`	optional underlying XMLHTTPRequest instance used internally. If this is undefined, then it is a fetch request.
`resolve`	function to call to immediately return a success response object.
`reject`	function to call to immediately return an error response object.
`data`	an state object assigned to the request and reused in response.

Following optional parameters are common across most tabs.

`global`	an object representing global data useful in strict mode.
`console`	an object in lieu of window.console for logging in strict mode.
`location`	an object in lieu of window.location for accessing web page URL in strict mode.
`controls`	an object to access shared variables with the controls tab.

Some builtin and preloaded examples are described next.

17.1 Disable any web request

Disallow any Ajax request. Both fetch and XMLHttpRequest are blocked by throwing an error.

17.2 Do not send any cookies and ignore in response

Set the credentials of omit in all the fetch requests.

17.3 Do not use browser cache for web requests

Set the cache of no-cache in all the fetch requests.

RTC Helper © 2025-2026, Kundan Singh

17.4 Fake error on any web request if not completed in 2s

Return a fake timeout error on Ajax requests that are not completed in 2 seconds. Both fetch and XMLHttpRequest are considered.

17.5 Fake success response if error encountered

Return a fake success response, if an error is received in Ajax. Both fetch and XMLHttpRequest are considered. Note that the response formats are different for the two cases.

17.6 Force same-origin cors policy in all requests

Set the mode of same-origin in all fetch requests.

17.7 Log JSON body from response received

Log the response body assuming JSON. Both fetch and XMLHttpRequest are considered.

17.8 Log original, but return fake response

Return a fake response, but log the original for the Ajax request. Both fetch and XMLHttpRequest are considered. Note that the response formats are different for the two cases.

17.9 Log web request that take more than 2 seconds

Log web requests that take more than 2 seconds to get a response or error. Both fetch and XMLHttpRequest are considered.

17.10 Log with time all web requests sent and responses received

Log time for all Ajax requests and responses. Both fetch and XMLHttpRequest and their responses, including errors, are logged.

RTC Helper © 2025-2026, Kundan Singh

18. Intercept and customize web requests and responses

It allows intercepting and modifying any web request or response headers by the browser; but this customization is only available in the browser extension using the older manifest version 2, and it does not work with version 3. In the version 3 browser extension, any x-frame-origin and content-security-policy are removed from the web response headers. In version 2, the function defined below is called for every web request, to intercept and alter the behavior. Common customizations include the ability to alter or relax the content-security-policy, e.g., to load certain JavaScript libraries used in the functions defined in other tabs. The function takes one argument, details, which contains all the web request and response details, and is invoked on headers received. The function must not be async. Note that this function is not related to real-time communication feature, but just to the regular web requests.

Some builtin and preloaded examples are described next.

18.1 Add jsdelivr CDN in content security policy

Include https://cdn.jsdelivr.net as script-src in content-security-policy and other related headers in response to any web request. This is useful if the tool is used as a browser extension, and the customization functions in other tabs, require use of third-party libraries hosted on jsdelivr.

RTC Helper © 2025-2026, Kundan Singh

19. Customize based on CPU performance

It allows customization based on CPU utilization; but this customization is only available in the browser extension, not when used as a wrapper or script in the web application. If enabled, it periodically captures the CPU usage metrics, and allows customization in the application based on that. Note that this function is not related to real-time communication feature, but just to the underlying machine performance.

Some builtin and preloaded examples are described next.

19.1 Set a controls variable with average and maximum CPU percent

Calculate the average CPU percent, and set controls variable cpuusage with that value. Also include maximum on any core in the variable value.

RTC Helper © 2025-2026, Kundan Singh

20. How do I extend this to include that?

On the settings tab, you can specify a list of https URLs, which are used by this software to download external customization functions. These external functions are then added to the list of pre-loaded functions in various tabs as applicable. This allows you to extend the existing list of functions available to the user. This can be done both by the end user when this software is used as a browser extension, or by the web developer when this software is integrated in the website.

Each external URL should allow downloading a text file with content in JSON format, that specify one or more customization function in one or more categories. An example is shown below, that includes a function in the *capture* category, and two functions in the *connect* category. The text for the function definition is snipped for brevity, but should follow the signature, conventions and examples shown earlier in various customization categories. Note that the function definition is included as a string, can have escaped new-line or other characters, and should be parseable by a JSON parser.

```
{
    "videocapture": {
        "Testing - my new background blur":
            "function(canvas,video,data,segment) {\n ... \n}"
    },
    "mediaconnect": {
        "Test1 - remove any media relay":
            "function(config) {\n ... \n}",
        "Test2 - remove any optional constraints":
            "function(config, constraints) {\n ... \n}",
    },
}
```

Alternatively, instead of using a strict JSON format, it can be a text file with some function definitions and preceding comment containing the metadata as shown below.

```
//@tab Capture: Testing - my new background blur
function (canvas, video, data, segment) {
    ...
}

//@tab Connect: Test1 - remove any media relay
function (config) {
    ...
}

//@tab Connect: Test2 - remove any optional constraints
function (config, constraints) {
    ...
}
```

RTC Helper © 2025-2026, Kundan Singh

This format is easier to read, but certain convention must be followed strictly for correct parsing. Any line must start with either an empty space or tab character, or one of this strings: `//@tab`, or `function`, or `async function`, or closing curly brace indicating end of function, i.e., `}`, without any other character on that line.

RTC Helper © 2025-2026, Kundan Singh