Get started with WebRTC

Sam Dutton
(https://twitter.com/sw12) (https://github.com/samdutton) (https://glitch.com/@samdutton) (https://techhub.social/@samdutton

> WebRTC is a new front in the long war for an open and unencumbered web.
>
> *Brendan Eich, inventor of JavaScript*

Real-time communication without plugins

Imagine a world where your phone, TV, and computer could communicate on a common platform. Imagine it was easy to add video chat an your web app. That's the vision of WebRTC.

Want to try it out? WebRTC is available on desktop and mobile in Google Chrome, Safari, Firefox, and Opera. A good place to start is the sim (https://appr.tc):

1. Open appr.tc (https://appr.tc) in your browser.
2. Click **Join** to join a chat room and let the app use your webcam.
3. Open the URL displayed at the end of the page in a new tab or, better still, on a different computer.

Quick start

Haven't got time to read this article or only want code?

- To get an overview of WebRTC, watch the following Google I/O video or view these slides (https://io13webrtc.appspot.com/):

  Real-time communication with WebRTC: Google I/O 2013

- If you haven't used the `getUserMedia` API, see Capture audio and video in HTML5 (https://www.html5rocks.com/en/tutorials/getusermedia/int getUserMedia (https://www.simpl.info/getusermedia).
- To learn about the `RTCPeerConnection` API, see the following example and 'simpl.info RTCPeerConnection' (https://simpl.info/rtcpeerconn
- To learn how WebRTC uses servers for signaling, and firewall and NAT traversal, see the code and console logs from appr.tc (https://app
- Can't wait and just want to try WebRTC right now? Try some of the more-than 20 demos (https://webrtc.github.io/samples) that exercise t
- Having trouble with your machine and WebRTC? Visit the WebRTC Troubleshooter (https://test.webrtc.org).

Alternatively, jump straight into the WebRTC codelab (https://codelabs.developers.google.com/codelabs/webrtc-web/), a step-by-step guide that e complete video chat app, including a simple signaling server.

A very short history of WebRTC

One of the last major challenges for the web is to enable human communication through voice and video: real-time communication or RTC natural in a web app as entering text in a text input. Without it, you're limited in your ability to innovate and develop new ways for people to

Historically, RTC has been corporate and complex, requiring expensive audio and video technologies to be licensed or developed in house. with existing content, data, and services has been difficult and time-consuming, particularly on the web.

Gmail video chat became popular in 2008 and, in 2011, Google introduced Hangouts, which uses Talk (as did Gmail). Google bought GIPS, a many components required for RTC, such as codecs and echo cancellation techniques. Google open sourced the technologies developed relevant standards bodies at the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) to ensure industry consens the first implementation of WebRTC (https://labs.ericsson.com/developer-community/blog/beyond-html5-peer-peer-conversational-video).

WebRTC implemented open standards for real-time, plugin-free video, audio, and data communication. The need was real:

- Many web services used RTC, but needed downloads, native apps, or plugins. These included Skype, Facebook, and Hangouts.
- Downloading, installing, and updating plugins is complex, error prone, and annoying.
- Plugins are difficult to deploy, debug, troubleshoot, test, and maintain - and may require licensing and integration with complex, exper difficult to persuade people to install plugins in the first place!

The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers, and more technologies.

Where are we now?

WebRTC is used in various apps, such as Google Meet. WebRTC has also been integrated with WebKitGTK+ (https://labs.ericsson.com/developer-community/blog/beyond-html5-conversational-voice-and-video-implemented-webkit-gtk) and Qt native apps.

WebRTC implements these three APIs: - `MediaStream` (also known as `getUserMedia`) - `RTCPeerConnection` - `RTCDataChannel`

The APIs are defined in these two specs:

- WebRTC (https://w3c.github.io/webrtc-pc/)
- `getUserMedia` (https://www.w3.org/TR/mediacapture-streams)

All three APIs are supported on mobile and desktop by Chrome, Safari, Firefox, Edge, and Opera.

`getUserMedia`: For demos and code, see WebRTC samples (https://webrtc.github.io/samples) or try Chris Wilson's amazing examples (https://web use `getUserMedia` as input for web audio.

`RTCPeerConnection`: For a simple demo and a fully functional video-chat app, see WebRTC samples Peer connection (https://webrtc.github.io/samples/src/content/peerconnection/pc1/) and appr.tc (https://appr.tc), respectively. This app uses adapter.js (https://github.com shim maintained by Google with help from the WebRTC community (https://github.com/webrtc/adapter/graphs/contributors), to abstract away bro changes.

`RTCDataChannel`: To see this in action, see WebRTC samples (https://webrtc.github.io/samples/) to check out one of the data-channel demos.

The WebRTC codelab (https://codelabs.developers.google.com/codelabs/webrtc-web/#0) shows how to use all three APIs to build a simple app for

Your first WebRTC

WebRTC apps need to do several things:

- Get streaming audio, video, or other data.
- Get network information, such as IP addresses and ports, and exchange it with other WebRTC clients (known as **peers**) to enable con (https://en.wikipedia.org/wiki/NAT_traversal) and firewalls.
- Coordinate signaling communication to report errors and initiate or close sessions.
- Exchange information about media and client capability, such as resolution and codecs.
- Communicate streaming audio, video, or data.

To acquire and communicate streaming data, WebRTC implements the following APIs:

- `MediaStream` (https://dvcs.w3.org/hg/audio/raw-file/tip/streams/StreamProcessing.html) gets access to data streams, such as from the user's c
- `RTCPeerConnection` (https://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-interface) enables audio or video calling with facilit bandwidth management.
- `RTCDataChannel` (https://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcdatachannel) enables peer-to-peer communication of generic data.

(There is detailed discussion of the network and signaling aspects of WebRTC later.)

`MediaStream` API (also known as `getUserMedia` API)

The `MediaStream` API (https://dev.w3.org/2011/webrtc/editor/getusermedia.html) represents synchronized streams of media. For example, a stream microphone input has synchronized video and audio tracks. (Don't confuse `MediaStreamTrack` with the `<track>` element, which is somethi (https://www.html5rocks.com/en/tutorials/track/basics/).)

Probably the easiest way to understand the `MediaStream` API is to look at it in the wild:

1. In your browser, navigate to WebRTC samples `getUserMedia` (https://webrtc.github.io/samples/src/content/getusermedia/gum/).
2. Open the console.
3. Inspect the `stream` variable, which is in global scope.

Each `MediaStream` has an input, which might be a `MediaStream` generated by `getUserMedia()`, and an output, which might be passed to a `RTCPeerConnection`.

The `getUserMedia()` method takes a `MediaStreamConstraints` object parameter and returns a `Promise` that resolves to a `MediaStream` o

Each `MediaStream` has a `label`, such as `'Xk7EuLhsuHKbnjLWkW4yYGNJJ8ONsgwHBvLQ'`. An array of `MediaStreamTrack`s is returned by the `g` `getVideoTracks()` methods.

For the `getUserMedia` (https://webrtc.github.io/samples/src/content/getusermedia/gum/) example, `stream.getAudioTracks()` returns an empty ar and, assuming a working webcam is connected, `stream.getVideoTracks()` returns an array of one `MediaStreamTrack` representing the st `MediaStreamTrack` has a kind (`'video'` or `'audio'`), a `label` (something like `'FaceTime HD Camera (Built-in)'`), and represents one or or video. In this case, there is only one video track and no audio, but it is easy to imagine use cases where there are more, such as a chat ap front camera, rear camera, microphone, and an app sharing its screen.

A `MediaStream` can be attached to a video element by setting the `srcObject` attribute (https://developer.mozilla.org/docs/Web/API/HTMLMediaEle was done by setting the `src` attribute to an object URL created with `URL.createObjectURL()`, but this has been deprecated (https://developer.mozilla.org/docs/Web/API/URL/createObjectURL).

Note: The `MediaStreamTrack` is actively using the camera, which takes resources, and keeps the camera open and camera light on. When you are no longer us `track.stop()` so that the camera can be closed.

`getUserMedia` can also be used as an input node for the Web Audio API (https://developer.chrome.com/blog/live-web-audio-input-enabled):

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.