# WebRTC

* Real-time communication for the web
* Supports video, voice and generic data to be sent between peers.
* Available on all modern browsers as well as on native clients for all major platforms.
* JavaScript APIs, library for Android and iOS
* Open source supported by Apple, Google, Microsoft, Mozilla, etc.

Camera, microphone, screen sharing

## **Application flow**

Accessing the media devices

Opening peer connections

Discovering peers

Start streaming

<https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity>

Signaling

WebRTC can’t create connections without some sort of server in the middle, called the signal challenge or signaling service. (Any sort of channel of communication to exchange information before setting up a connection)

The information we need to exchange is the Offer and Answer which just contains the [**SDP**](https://developer.mozilla.org/en-US/docs/Glossary/SDP) (Session Description Protocol, standard describing a peer-to-peer connection)

Contains the codec, source address and timing information of audio and video.

Never used alone, buy by protocols like RTP and RTSP.

Session descriptions

**Session description:** configuration of an endpoint on a WebRTC connection

Includes information about

* the kind of media being sent
* format
* transfer protocol being used
* the endpoint´s IP address and port
* other information needed to describe a media transfer endpoint

Exchanged and stored using SDP, RFC 2327

This process is performed both when a call is first established, but also anytime the call’s format or other configuration needs to change.

**Offer:** When a user starts a WebRTC call to another user, special description is created

* Information about the caller’s proposed configuration for the call

**Answer:**  What the recipient responds with

* Description of their end of the call

Both devices share with one another the information needed in order to exchange media data.

**Interactive Connectivity Establishment (ICE)**

Protocol which lets two devices use and intermediary to exchange offers and answers even separated by NAT

Each peer, then, keeps two descriptions on hand: the **local description**, describing itself, and the **remote description**, describing the other end of the call.

1. The caller captures local Media via [navigator.mediaDevices.getUserMedia()](https://developer.mozilla.org/en-US/docs/Web/API/Navigator/mediaDevices/getUserMedia" \o "REDIRECT MediaDevices.getUserMedia())
2. The caller creates RTCPeerConnection and called [RTCPeerConnection.addTrack()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/addTrack" \o "The RTCPeerConnection method addTrack() adds a new media track to the set of tracks which will be transmitted to the other peer.) (Since addStream is deprecating)
3. The caller calls [RTCPeerConnection.createOffer()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/createOffer" \o "The createOffer() method of the RTCPeerConnection interface initiates the creation of an SDP offer for the purpose of starting a new WebRTC connection to a remote peer.) to create an offer.
4. The caller calls [RTCPeerConnection.setLocalDescription()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/setLocalDescription" \o "The RTCPeerConnection.setLocalDescription() method changes the local description associated with the connection. This description specifies the properties of the local end of the connection, including the media format.) to set that offer as the *local description* (that is, the description of the local end of the connection).
5. After setLocalDescription(), the caller asks STUN servers to generate the ice candidates
6. The caller uses the signaling server to transmit the offer to the intended receiver of the call.
7. The recipient receives the offer and calls [RTCPeerConnection.setRemoteDescription()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/setRemoteDescription" \o "The RTCPeerConnection.setRemoteDescription() method sets the specified session description as the remote peer's current offer or answer. This description specifies the properties of the remote end of the connection, including the media format.) to record it as the *remote description* (the description of the other end of the connection).
8. The recipient does any setup it needs to do for its end of the call: capture its local media, and attach each media tracks into the peer connection via [RTCPeerConnection.addTrack()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/addTrack" \o "The RTCPeerConnection method addTrack() adds a new media track to the set of tracks which will be transmitted to the other peer.)
9. The recipient then creates an answer by calling [RTCPeerConnection.createAnswer()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/createAnswer" \o "The createAnswer() method on the RTCPeerConnection interface creates an SDP answer to an offer received from a remote peer during the offer/answer negotiation of a WebRTC connection. The answer contains information about any media already attached to the session, codecs and options supported by the browser, and any ICE candidates already gathered. The answer is delivered to the returned Promise, and should then be sent to the source of the offer to continue the negotiation process.).
10. The recipient calls [RTCPeerConnection.setLocalDescription()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/setLocalDescription" \o "The RTCPeerConnection.setLocalDescription() method changes the local description associated with the connection. This description specifies the properties of the local end of the connection, including the media format.), passing in the created answer, to set the answer as its local description. The recipient now knows the configuration of both ends of the connection.
11. The recipient uses the signaling server to send the answer to the caller.
12. The caller receives the answer.
13. The caller calls [RTCPeerConnection.setRemoteDescription()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/setRemoteDescription" \o "The RTCPeerConnection.setRemoteDescription() method sets the specified session description as the remote peer's current offer or answer. This description specifies the properties of the remote end of the connection, including the media format.) to set the answer as the remote description for its end of the call. It now knows the configuration of both peers. Media begins to flow as configured.

Because during renegotiation, an offer might be rejected because it proposes an incompatible format, it's necessary that each endpoint have the ability to propose a new format but not actually switch to it until it's accepted by the other peer. For that reason, WebRTC uses pending and current descriptions.

The **current description** (which is returned by the [RTCPeerConnection.currentLocalDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/currentLocalDescription" \o "The read-only property RTCPeerConnection.currentLocalDescription returns an RTCSessionDescription object describing the local end of the connection as it was most recently successfully negotiated since the last time the  RTCPeerConnection finished negotiating and connecting to a remote peer. Also included is a list of any ICE candidates that may already have been generated by the ICE agent since the offer or answer represented by the description was first instantiated.) and [RTCPeerConnection.currentRemoteDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/currentRemoteDescription" \o "The read-only property RTCPeerConnection.currentRemoteDescription returns an RTCSessionDescription object describing the remote end of the connection as it was most recently successfully negotiated since the last time the RTCPeerConnection finished negotiating and connecting to a remote peer. Also included is a list of any ICE candidates that may already have been generated by the ICE agent since the offer or answer represented by the description was first instantiated.) properties) represents the description currently in actual use by the connection. This is the most recent connection that both sides have fully agreed to use.

The **pending description** (returned by [RTCPeerConnection.pendingLocalDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/pendingLocalDescription" \o "The read-only property RTCPeerConnection.pendingLocalDescription returns an RTCSessionDescription object describing a pending configuration change for the local end of the connection. This does not describe the connection as it currently stands, but as it may exist in the near future. Use RTCPeerConnection.currentLocalDescription or RTCPeerConnection.localDescription to get the current state of the endpoint. For details on the difference, see Pending and current descriptions in WebRTC connectivity.) and [RTCPeerConnection.pendingRemoteDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/pendingRemoteDescription" \o "The read-only property RTCPeerConnection.pendingRemoteDescription returns an RTCSessionDescription object describing a pending configuration change for the remote end of the connection. This does not describe the connection as it currently stands, but as it may exist in the near future. Use RTCPeerConnection.currentRemoteDescription or RTCPeerConnection.remoteDescription to get the current session description for the remote endpoint. For details on the difference, see Pending and current descriptions in WebRTC connectivity.)) indicates a description which is currently under consideration following a call to  setLocalDescription() or setRemoteDescription(), respectively.

When reading the description (returned by [RTCPeerConnection.localDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/localDescription" \o "The read-only property RTCPeerConnection.localDescription returns an RTCSessionDescription describing the session for the local end of the connection. If it has not yet been set, this is null.) and [RTCPeerConnection.remoteDescription](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/remoteDescription" \o "The read-only property RTCPeerConnection.remoteDescription returns a RTCSessionDescription describing the session (which includes configuration and media information) for the remote end of the connection. If this hasn't been set yet, this is null.)), the returned value is the value of pendingLocalDescription/pendingRemoteDescription if there's a pending description (that is, the pending description isn't null); otherwise, the current description (currentLocalDescription/currentRemoteDescription) is returned.

When changing the description by calling setLocalDescription() or setRemoteDescription(), the specified description is set as the pending description, and the WebRTC layer begins to evaluate whether or not it's acceptable. Once the proposed description has been agreed upon, the value of currentLocalDescription or currentRemoteDescription is changed to the pending description, and the pending description is set to null again, indicating that there isn't a pending description.

The pendingLocalDescription contains not just the offer or answer under consideration, but any local ICE candidates which have already been gathered since the offer or answer was created. Similarly, pendingRemoteDescription includes any remote ICE candidates which have been provided by calls to [RTCPeerConnection.addIceCandidate()](https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/addIceCandidate" \o "When a web site or app using RTCPeerConnection receives a new ICE candidate from the remote peer over its signaling channel, it delivers the newly-received candidate to the browser's ICE agent by calling RTCPeerConnection.addIceCandidate().).

ICE Candidates

* Exchanging information about the network connection, details the available methods the peer is able to communicate (directly or through a TURN server).
* Best candidates first, making the way down to the worst candidates
* UDP (more widely supported), TCP used only when UDP is not available

UDP candidate types

Protocol: UDP

**Host:** A host candidate is one for which its [ip](https://developer.mozilla.org/en-US/docs/Web/API/RTCIceCandidate/ip" \o "The RTCIceCandidate interface's read-only ip property is a string providing the address of the device which is the source of the candidate.) address is the actual, direct IP address of the remote peer.

**prflx:** A peer reflexive candidate is one whose IP address comes from a symmetric NAT between the two peers, usually as an additional candidate during trickle ICE (that is, additional candidate exchanges that occur after primary signaling but before the connection verification phase is finished).

**srflex:** A server reflexive candidate is generated by a STUN/TURN server; the connection's initiator requests a candidate from the STUN server, which forwards the request through the remote peer's NAT, which creates and returns a candidate whose IP address is local to the remote peer. The STUN server then replies to the initiator's request with a candidate whose IP address is unrelated to the remote peer.

## **WebRTC APIs**

The WebRTC standard covers, on a high level, two different technologies: media capture devices and peer-to-peer connectivity.

Media capture devices includes video cameras and microphones, but also screen capturing "devices". For cameras and microphones, we use navigator.mediaDevices.getUserMedia() to capture MediaStreams. For screen recording, we use navigator.mediaDevices.getDisplayMedia() instead.

The peer-to-peer connectivity is handled by the RTCPeerConnection interface. This is the central point for establishing and controlling the connection between two peers in WebRTC.

<https://webrtc.org/getting-started/media-devices>