WebRTC API

WebRTC (Web Real-Time Communication) is a technology which enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary. The set of standards that comprise WebRTC makes it possible to share data and perform teleconferencing peer-to-peer, without requiring that the user installs plug-ins or any other third-party software.

WebRTC consists of several interrelated APIs and protocols which work together to achieve this. The documentation you'll find here will help you understand the fundamentals of WebRTC, how to set up and use both data and media connections, and more.

Interoperability

Because implementations of WebRTC are still evolving, and because each browser has different levels of support for codecs and WebRTC features, you should *strongly* consider making use of the Adapter.js library provided by Google before you begin to write your code.

Adapter.js uses shims and polyfills to smooth over the differences among the WebRTC implementations across the environments supporting it. Adapter.js also handles prefixes and other naming differences to make the entire WebRTC development process easier, with more broadly compatible results. The library is also available as an NPM package.

To learn more about Adapter.js, see Improving compatibility using WebRTC adapter.js.

WebRTC concepts and usage

WebRTC serves multiple purposes; together with the Media Capture and Streams API, they provide powerful multimedia capabilities to the Web, including support for audio and video conferencing, file exchange, screen sharing, identity management, and interfacing with legacy telephone systems including support for sending DTMF (touch-tone dialing) signals. Connections between peers can be made without requiring any special drivers or plug-ins, and can often be made without any intermediary servers.

Connections between two peers are represented by the `RTCPeerConnection` interface. Once a connection has been established and opened using `RTCPeerConnection`, media streams (`MediaStream`s) and/or data channels (`RTCDataChannel`s) can be added to the connection.

Media streams can consist of any number of tracks of media information; tracks, which are represented by objects based on the `MediaStreamTrack` interface, may contain one of a number of types of media data, including audio, video, and text (such as subtitles or even chapter names). Most streams consist of at least one audio track and likely also a video track, and can be used to send and receive both live media or stored media information (such as a streamed movie).

You can also use the connection between two peers to exchange arbitrary binary data using the `RTCDataChannel` interface. This can be used for back-channel information, metadata exchange, game status packets, file transfers, or even as a primary channel for data transfer.

more details and links to relevant guides and tutorials needed

WebRTC reference

Because WebRTC provides interfaces that work together to accomplish a variety of tasks, we have divided up the reference by category. Please see the sidebar for an alphabetical list.

Connection setup and management

These interfaces, dictionaries, and types are used to set up, open, and manage WebRTC connections. Included are interfaces representing peer media connections, data channels, and interfaces used when exchanging information on the capabilities of each peer in order to select the best possible configuration for a two-way media connection.

Interfaces

`RTCPeerConnection`

Represents a WebRTC connection between the local computer and a remote peer. It is used to handle efficient streaming of data between the two peers.

`RTCDataChannel`

Represents a bi-directional data channel between two peers of a connection.

`RTCDataChannelEvent`

Represents events that occur while attaching a `RTCDataChannel` to a `RTCPeerConnection`. The only event sent with this interface is `datachannel`.

`RTCSessionDescription`

Represents the parameters of a session. Each `RTCSessionDescription` consists of a description `type` indicating which part of the offer/answer negotiation process it describes and of the SDP descriptor of the session.

`RTCStatsReport`

Provides information detailing statistics for a connection or for an individual track on the connection; the report can be obtained by calling `RTCPeerConnection.getStats()`. Details about using WebRTC statistics can be found in WebRTC Statistics API.

`RTCIceCandidate`

Represents a candidate Interactive Connectivity Establishment (ICE) server for establishing an `RTCPeerConnection`.

`RTCIceTransport`

Represents information about an ICE transport.

`RTCPeerConnectionIceEvent`

Represents events that occur in relation to ICE candidates with the target, usually an `RTCPeerConnection`. Only one event is of this type: `icecandidate`.

`RTCRtpSender`

Manages the encoding and transmission of data for a `MediaStreamTrack` on an `RTCPeerConnection`.

`RTCRtpReceiver`

Manages the reception and decoding of data for a `MediaStreamTrack` on an `RTCPeerConnection`.

`RTCTrackEvent`

The interface used to represent a `track` event, which indicates that an `RTCRtpReceiver` object was added to the `RTCPeerConnection` object, indicating that a new incoming `MediaStreamTrack` was created and added to the `RTCPeerConnection`.

`RTCSctpTransport`

Provides information which describes a Stream Control Transmission Protocol (**SCTP**) transport and also provides a way to access the underlying Datagram Transport Layer Security (**DTLS**) transport over which SCTP packets for all of an `RTCPeerConnection`'s data channels are sent and received.

Dictionaries

`RTCConfiguration`

Used to provide configuration options for an `RTCPeerConnection`.

`RTCIceServer`

Defines how to connect to a single ICE server (such as a STUN or TURN server).

`RTCRtpContributingSource`

Contains information about a given contributing source (CSRC) including the most recent time a packet that the source contributed was played out.

Types

`RTCSctpTransportState`

Indicates the state of an `RTCSctpTransport` instance.

`RTCSessionDescriptionCallback`

The RTCSessionDescriptionCallback is passed into the `RTCPeerConnection` object when requesting it to create offers or answers.

Identity and security

These APIs are used to manage user identity and security, in order to authenticate the user for a connection.

`RTCIdentityProvider`

Enables a user agent is able to request that an identity assertion be generated or validated.

`RTCIdentityAssertion`

Represents the identity of the remote peer of the current connection. If no peer has yet been set and verified this interface returns `null`. Once set it can't be changed.

`RTCIdentityProviderRegistrar`

Registers an identity provider (idP).

`RTCIdentityEvent`

Represents an identity assertion generated by an identity provider (idP). This is usually for an `RTCPeerConnection`. The only event sent with this type is `identityresult`.

`RTCIdentityErrorEvent`

Represents an error associated with the identity provider (idP). This is usually for an `RTCPeerConnection`. Two events are sent with this type: `idpassertionerror` and `idpvalidationerror`.

`RTCCertificate`

Represents a certificate that an `RTCPeerConnection` uses to authenticate.

Telephony

These interfaces are related to interactivity with Public-Switched Telephone Networks (PTSNs).

`RTCDTMFSender`

Manages the encoding and transmission of Dual-Tone Multi-Frequency (DTMF) signaling for an `RTCPeerConnection`.

`RTCDTMFToneChangeEvent`

Used by the `tonechange` event to indicate that a DTMF tone has either begun or ended. This event does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated).

Guides

Introduction to WebRTC protocols

This article introduces the protocols on top of which the WebRTC API is built.

WebRTC connectivity

A guide to how WebRTC connections work and how the various protocols and interfaces can be used together to build powerful communication apps.

Lifetime of a WebRTC session

WebRTC lets you build peer-to-peer communication of arbitrary data, audio, or video—or any combination thereof—into a browser application. In this article, we'll look at the lifetime of a WebRTC session, from establishing the connection all the way through closing the connection when it's no longer needed.

Establishing a connection: The perfect negotiation pattern

Perfect negotiation is a design pattern which is recommended for your signaling process to follow, which provides transparency in negotiation while allowing both sides to be either the offerer or the answerer, without significant coding needed to differentiate the two.

Signaling and two-way video calling

A tutorial and example which turns a WebSocket-based chat system created for a previous example and adds support for opening video calls among participants. The chat server's WebSocket connection is used for WebRTC signaling.

Codecs used by WebRTC

A guide to the codecs which WebRTC requires browsers to support as well as the optional ones supported by various popular browsers. Included is a guide to help you choose the best codecs for your needs.

Using WebRTC data channels

This guide covers how you can use a peer connection and an associated `RTCDataChannel` to exchange arbitrary data between two peers.

Using DTMF with WebRTC

WebRTC's support for interacting with gateways that link to old-school telephone systems includes support for sending DTMF tones using the `RTCDTMFSender` interface. This guide shows how to do so.

Tutorials

Improving compatibility using WebRTC adapter.js

The WebRTC organization provides on GitHub the WebRTC adapter to work around compatibility issues in different browsers' WebRTC implementations. The adapter is a

JavaScript shim which lets your code to be written to the specification so that it will "just work" in all browsers with WebRTC support.

Taking still photos with WebRTC
This article shows how to use WebRTC to access the camera on a computer or mobile phone with WebRTC support and take a photo with it.

A simple RTCDataChannel sample
The `RTCDataChannel` interface is a feature which lets you open a channel between two peers over which you may send and receive arbitrary data. The API is intentionally similar to the WebSocket API, so that the same programming model can be used for each.

Resources

Protocols

WebRTC-proper protocols

- Application Layer Protocol Negotiation for Web Real-Time Communications
- WebRTC Audio Codec and Processing Requirements
- RTCWeb Data Channels
- RTCWeb Data Channel Protocol
- Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
- WebRTC Security Architecture
- Transports for RTCWEB

Related supporting protocols

- Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocol
- Session Traversal Utilities for NAT (STUN)
- URI Scheme for the Session Traversal Utilities for NAT (STUN) Protocol
- Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers
- An Offer/Answer Model with Session Description Protocol (SDP)

- Session Traversal Utilities for NAT (STUN) Extension for Third Party Authorization

WebRTC statistics

- WebRTC Statistics API

Specifications

Specification	Status	Comment
WebRTC 1.0: Real-time Communication Between Browsers	CR Candidate Recommendation	The initial definition of the API of WebRTC.
Media Capture and Streams	CR Candidate Recommendation	The initial definition of the object conveying the stream of media content.
Media Capture from DOM Elements	WD Working Draft	The initial definition on how to obtain stream of content from DOM Elements

In additions to these specifications defining the API needed to use WebRTC, there are several protocols, listed under resources.

See also

- `MediaDevices`
- `MediaStreamEvent`
- `MediaStreamConstraints`
- `MediaStreamTrack`
- `MessageEvent`
- `MediaStream`
- Media Capture and Streams API
- Firefox multistream and renegotiation for Jitsi Videobridge
- Peering Through the WebRTC Fog with SocketPeer

- Inside the Party Bus: Building a Web App with Multiple Live Video Streams + Interactive Graphics
- Web media technologies

Last modified: Jul 13, 2020, by MDN contributors

Related Topics

WebRTC API

▼ WebRTC Guides

WebRTC Architecture

WebRTC Basics

WebRTC Protocols

Dealing with connectivity

Overview of WebRTC interfaces

Lifetime of a WebRTC Session

Using data channels

▼ WebRTC Tutorials

Interoperability with adapter.js

Taking still photos from the camera

A simple data channel example

▼ Interfaces

`RTCPeerConnection`

`RTCSessionDescription`

`RTCIceCandidate`

`RTCPeerConnectionIceEvent`

`MessageEvent`

`MediaStream`

`RTCStatsReport`

`RTCIdentityEvent`