

1. **WebRTC 1.0: Real-time Communication Between Browsers**
2. **W3C Editor's Draft 30 May 2012**

This version:

[http://dev.w3.org/2011/webrtc/editor/webrtc-20120530.html](http://dev.w3.org/2011/webrtc/editor/webrtc.html-20120530)

Latest published version:

<http://www.w3.org/TR/webrtc/>

Latest editor's draft:

<http://dev.w3.org/2011/webrtc/editor/webrtc.html>

Previous version:

<http://dev.w3.org/2011/webrtc/editor/webrtc-20120528.html>

Editors:

Adam Bergkvist, Ericsson

Daniel C. Burnett, Voxeo

Cullen Jennings, Cisco

Anant Narayanan, Mozilla

Initial Author of this Specification was Ian Hickson, Google Inc., with the following copyright statement:
© Copyright 2004-2011 Apple Computer, Inc., Mozilla Foundation, and Opera Software ASA. You are granted a license to use, reproduce and create derivative works of this document.

All subsequent changes since 26 July 2011 done by the W3C WebRTC Working Group are under the following [Copyright](http://www.w3.org/Consortium/Legal/ipr-notice#Copyright):
© 2011-2012 [W3C](http://www.w3.org/)® ([MIT](http://www.csail.mit.edu/), [ERCIM](http://www.ercim.eu/), [Keio](http://www.keio.ac.jp/)), All Rights Reserved. [Document use](http://www.w3.org/Consortium/Legal/copyright-documents) rules apply.

For the entire publication on the W3C site the [liability](http://www.w3.org/Consortium/Legal/ipr-notice#Legal_Disclaimer) and [trademark](http://www.w3.org/Consortium/Legal/ipr-notice#W3C_Trademarks) rules apply.

1. **Abstract**

This document defines a set of APIs to represent streaming media, including audio and video, in JavaScript, to allow media to be sent over the network to another browser or device implementing the appropriate set of real-time protocols, and media received from another browser or device to be processed and displayed locally. This specification is being developed in conjunction with a protocol specification developed by the IETF RTCWEB group and an API specification to get access to local media devices developed by the Media Capture Task Force.

1. **Status of This Document**

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the* [*W3C technical reports index*](http://www.w3.org/TR/) *at http://www.w3.org/TR/.*

This document is not complete. It is subject to major changes and, while early experimentation is encouraged, it is therefore not intended for implementation. The API is based on preliminary work done in the WHATWG. The Web Real-Time Communications Working Group expects this specification to evolve significantly based on:

* The outcome of ongoing exchanges in the companion RTCWEB group at IETF to define the set of protocols that, together with this document, will enable real-time communications in Web browsers.
* Privacy issues that arise when exposing local capabilities and local streams.
* Technical discussions within the group, on the data channel in particular.
* Experience gained through early experimentations.
* Feedback received from other groups and individuals.

This document was published by the [Web Real-Time Communications Working Group](http://www.w3.org/2011/04/webrtc/) as an Editor's Draft. If you wish to make comments regarding this document, please send them to public-webrtc@w3.org (subscribe, [archives](http://lists.w3.org/Archives/Public/public-webrtc/)). All feedback is welcome.

Publication as an Editor's Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](http://www.w3.org/Consortium/Patent-Policy-20040205/). W3C maintains a [public list of any patent disclosures](http://www.w3.org/2004/01/pp-impl/47318/status) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim(s)](http://www.w3.org/Consortium/Patent-Policy-20040205/#def-essential) must disclose the information in accordance with [section 6 of the W3C Patent Policy](http://www.w3.org/Consortium/Patent-Policy-20040205/#sec-Disclosure).

1. **Table of Contents**
* [1. Conformance](http://dev.w3.org/2011/webrtc/editor/webrtc.html#conformance)
* [2. Introduction](http://dev.w3.org/2011/webrtc/editor/webrtc.html#intro)
* [3. Network Stream API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#network-stream-api)
	+ [3.1 Introduction](http://dev.w3.org/2011/webrtc/editor/webrtc.html#introduction)
	+ [3.2 Interface definitions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#interface-definitions)
		- [3.2.1 MediaStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastream)
			* [3.2.1.1 label](http://dev.w3.org/2011/webrtc/editor/webrtc.html#label)
			* [3.2.1.2 Events on MediaStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#events-on-mediastream)
		- [3.2.2 MediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamtrack)
	+ [3.3 AudioMediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#audiomediastreamtrack)
		- [3.3.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes)
		- [3.3.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods)
* [4. Peer-to-peer connections](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-connections)
	+ [4.1 PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection)
		- [4.1.1 SdpType](http://dev.w3.org/2011/webrtc/editor/webrtc.html#sdptype)
		- [4.1.2 SessionDescription Class](http://dev.w3.org/2011/webrtc/editor/webrtc.html#sessiondescription-class)
			* [4.1.2.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-1)
			* [4.1.2.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-1)
		- [4.1.3 SessionDescriptionCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#sessiondescriptioncallback)
		- [4.1.4 PeerConnectionErrorCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnectionerrorcallback)
		- [4.1.5 PeerState Enum](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerstate-enum)
		- [4.1.6 IceState Enum](http://dev.w3.org/2011/webrtc/editor/webrtc.html#icestate-enum)
		- [4.1.7 IceCandidate Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#icecandidate-type)
			* [4.1.7.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-2)
			* [4.1.7.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-2)
		- [4.1.8 IceCandidateCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#icecandidatecallback)
		- [4.1.9 IceServers Type - Option 1](http://dev.w3.org/2011/webrtc/editor/webrtc.html#iceservers-type---option-1)
			* [4.1.9.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-3)
		- [4.1.10 IceServers Type - Option 2](http://dev.w3.org/2011/webrtc/editor/webrtc.html#iceservers-type---option-2)
			* [4.1.10.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-4)
		- [4.1.11 PeerConnection Interface](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-interface)
			* [4.1.11.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-5)
			* [4.1.11.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-3)
* [5. IANA Registrations](http://dev.w3.org/2011/webrtc/editor/webrtc.html#iana-registrations)
	+ [5.1 Constraints](http://dev.w3.org/2011/webrtc/editor/webrtc.html#constraints)
* [6. Simple Example](http://dev.w3.org/2011/webrtc/editor/webrtc.html#simple-example)
* [7. Advanced Example](http://dev.w3.org/2011/webrtc/editor/webrtc.html#advanced-example)
* [8. Peer-to-peer Data API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-data-api)
	+ [8.1 DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#datachannel)
		- [8.1.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-6)
		- [8.1.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-4)
		- [8.1.3 Constants](http://dev.w3.org/2011/webrtc/editor/webrtc.html#constants)
		- [8.1.4 Dictionary DataChannelInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-datachannelinit-members)
	+ [8.2 Examples](http://dev.w3.org/2011/webrtc/editor/webrtc.html#examples)
* [9. Garbage collection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#garbage-collection)
* [10. Event definitions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-definitions)
	+ [10.1 PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnectionevent)
		- [10.1.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-7)
		- [10.1.2 Dictionary PeerConnectionEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-peerconnectioneventinit-members)
	+ [10.2 PeerConnectionIceEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnectioniceevent)
		- [10.2.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-8)
		- [10.2.2 Dictionary PeerConnectionEventIceInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-peerconnectioneventiceinit-members)
	+ [10.3 MediaStreamTrackEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamtrackevent)
		- [10.3.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-9)
		- [10.3.2 Dictionary MediaStreamTrackEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-mediastreamtrackeventinit-members)
	+ [10.4 MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamevent)
		- [10.4.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-10)
		- [10.4.2 Dictionary MediaStreamEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-mediastreameventinit-members)
	+ [10.5 DataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#datachannelevent)
		- [10.5.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-11)
		- [10.5.2 Dictionary DataChannelEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-datachanneleventinit-members)
* [11. Event summary](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-summary)
* [12. Change Log](http://dev.w3.org/2011/webrtc/editor/webrtc.html#change-log)
* [A. Acknowledgements](http://dev.w3.org/2011/webrtc/editor/webrtc.html#acknowledgements)
* [B. References](http://dev.w3.org/2011/webrtc/editor/webrtc.html#references)
	+ [B.1 Normative references](http://dev.w3.org/2011/webrtc/editor/webrtc.html#normative-references)
	+ [B.2 Informative references](http://dev.w3.org/2011/webrtc/editor/webrtc.html#informative-references)
1. **1. Conformance**

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *must*, *must not*, *required*, *should*, *should not*, *recommended*, *may*, and *optional* in this specification are to be interpreted as described in [[*RFC2119*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-RFC2119)].

Implementations that use ECMAScript to implement the APIs defined in this specification must implement them in a manner consistent with the ECMAScript Bindings defined in the Web IDL specification [[*WEBIDL*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-WEBIDL)], as this specification uses that specification and terminology.

1. **2. Introduction**

*This section is non-normative.*

There are a number of facets to video-conferencing in HTML covered by this specification:

* Representing a multimedia stream (video, audio, or both) from local devices (video cameras, microphones, Web cams) or from prerecorded files provided by the user.
* Connecting to remote peers using NAT-traversal technologies such as ICE, STUN, and TURN.
* Sending the locally-produced streams to remote peers and receiving streams from remote peers.
* Sending arbitrary data directly to remote peers.

This document defines the APIs used for these features. This specification is being developed in conjunction with a protocol specification developed by the [IETF RTCWEB group](http://datatracker.ietf.org/wg/rtcweb/) and an API specification to get access to local media devices developed by the [Media Capture Task Force](http://www.w3.org/2011/04/webrtc/).

1. **3. Network Stream API**
2. **3.1 Introduction**

The MediaStream interface, as defined in the [*[GETUSERMEDIA](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22bib-GETUSERMEDIA)*] specification, typically represents a stream of data of audio and/or video. A MediaStream may be extended to represent a stream that either comes from or is sent to a remote node (and not just the local camera, for instance). The extensions required to enable this capability on the MediaStream object will be described in this document.

A MediaStream as defined in [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] may contain zero or more MediaStreamTrack objects. A MediaStreamTrack sent to another peer will appear as one and only one MediaStreamTrack to the recipient.

Channels are the smallest unit considered in the MediaStream specification. Channels are intended to be encoded together for transmission as, for instance, an RTP payload type. All of the channels that a codec needs to encode jointly *must* be in the same MediaStreamTrack and the codecs *should* be able to encode, or discard, all the channels in the track.

The concepts of an input and output to a given MediaStream apply in the case of MediaStream objects transmitted over the network as well. A MediaStream created by a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object (later described in this document) will take as input the data received from a remote peer. Similarly, a MediaStream from a local source, for instance a camera via [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] will have an output that represents what is transmitted to a remote peer if the object is used with a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object.

The concept of duplicating MediaStream objects as described in [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] is also applicable here. This feature can be used, for instance, in a video-conferencing scenario to display the local video from the user’s camera and microphone in a local monitor, while only transmitting the audio to the remote peer (e.g. in response to the user using a "video mute" feature). Combining tracks from different MediaStream objects into a new MediaStream is useful in certain cases.

1. **3.2 Interface definitions**

In this section, we only specify aspects of the the following objects that are relevant when used along with a PeerConnection. Please refer to the original definitions of the objects in the [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] document for general information on using MediaStream and MediaStreamTrack both in and outside the context of PeerConnection.

**3.2.1 MediaStream**

**3.2.1.1 label**

The label attribute specified in MediaStream returns a label that is unique to this stream, so that streams can be recognized after they are sent through the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection) API.

When a [MediaStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastream) is created to represent a stream obtained from a remote peer, the label attribute is initialized from information provided by the remote source.

The label of a MediaStream object is unique to the source of the stream, but that does not mean it is not possible to end up with duplicates. For example, a locally generated stream could be sent from one user to a remote peer using [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) , and then sent back to the original user in the same manner, in which case the original user will have multiple streams with the same label (the locally-generated one and the one received from the remote peer).

**3.2.1.2 Events on MediaStream**

A new media component may be associated with an existing MediaStream . This happens, e.g., on the A-side when the B-side adds a new MediaStreamTrack object to one of the track lists of a MediaStream that is being sent over a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) . If this happens for the reason exemplified, or for any other reason than the add() [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] method being invoked locally on a MediaStreamTrackList or tracks are being added as the stream is created (i.e. the stream is initialized with tracks), the user agent *must* run the following steps:

1. Create a MediaStreamTrack object *track* to represent the new media component.
2. If *track’s* [kind](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtrack-kind) attribute equals "audio", add it to the MediaStream object’s [audioTracks](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastream-audiotracks) MediaStreamTrackList object. [[OPEN ISSUE: Is there a way to generalize this so that if we add a "smell" track this continues to work.]]
3. If *track’s* [kind](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtrack-kind) attribute equals "video", add it to the MediaStream object’s [videoTracks](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastream-videotracks) MediaStreamTrackList object.
4. [Fire a track event](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-fire-a-track-event-2) named [addtrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastreamtracklist-addtrack) with the newly created *track* at the MediaStreamTrackList object.

An existing media component may also be disassociated from a MediaStream . If this happens for any other reason than the remove() [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] method being invoked locally on a MediaStreamTrackList or the stream is being destroyed, the user agent *must* run the following steps:

1. Let *track* be the MediaStreamTrack object representing the media component about to be removed.
2. Remove *track* from the MediaStreamTrackList object.
3. [Fire a track event](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-fire-a-track-event-2) named [removetrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastreamtracklist-removetrack) with *track* at the MediaStreamTrackList object.

The event source for the onended event in the networked case is the PeerConnection object.

**3.2.2 MediaStreamTrack**

A MediaStreamTrack object’s reference to its MediaStream in the non-local media source case (an RTP source, as is the case for a MediaStream received over a PeerConnection) is always strong.

When a track belongs to a MediaStream that comes from a remote peer and the remote peer has permanently stopped sending data the ended event *must* be fired on the track, as specified in [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)]. [[ OPEN ISSUE: How do you know when it has stopped? This seems like an SDP question, not a media-levelquestion.]]

A track in a MediaStream , received with a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) , *must* have its readyState attribute [[*GETUSERMEDIA*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-GETUSERMEDIA)] set to [MUTED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-MediaStreamTrack-MUTED) (1) until media data arrives.

In addition, a MediaStreamTrack has its readyState set to MUTED on the B-side if the A-side disables the corresponding MediaStreamTrack in the MediaStream that is being sent. When the addstream event triggers on a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) , all MediaStreamTrack objects in the resulting MediaStream are muted until media data can be read from the RTP source. [[ OPEN ISSUE: How do you know when it has been disabled? This seems like an SDP question, not a media-levelquestion.]]

1. **3.3 AudioMediaStreamTrack**

The [AudioMediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-AudioMediaStreamTrack) is a specialization of of a normal MediaStreamTrack that only carries audio and is extended to have the capability to send and/or receive DTMF codes.

interface AudioMediaStreamTrack : MediaStreamTrack {

 readonly attribute boolean [canInsertDTMF](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-AudioMediaStreamTrack-canInsertDTMF);

 void [insertDTMF](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-AudioMediaStreamTrack-insertDTMF-void-DOMString-tones-long-duration) (DOMString tones, optional long duration);

};

**3.3.1 Attributes**

canInsertDTMF of type boolean, readonly

The *canInsertDTMF* attribute *must* indicate if the [AudioMediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-AudioMediaStreamTrack) is capable of sending DTMF.

**3.3.2 Methods**

insertDTMF

When a [AudioMediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-AudioMediaStreamTrack) object’s *insertDTMF()* method is invoked, the user agent *must* queue a task that that sends the DTMF tones.

The tone parameters is treated as a series of characters. The characters 0 to 9, A to D, #, and \* generated the associated DTMF tones. The characters a to d are equivalent to A to D. The character , indicates a an delay of 2 seconds before processing the next character in the tones parameter. Unrecognized characters are ignored.

The duration parameters indicates the duration in ms to play the each DTMF passed in the tones parameters. The duration can not be more than 6000 or less than 70. The default duration is 100 ms for each tone. The gap between tones *must* be at least 50 ms but should be as short as possible. [[OPEN ISSUE: How are invalid values handled?]]

If insertDTMF is called on the same object while an existing task for this object is generate DTMF is still running, the previous task is canceled. Calling insertDTMF with an empty tones parameter can be used to cancel any tones currently being send.

Editor Note: We need to add a callback that is set on the object that is called after the tones are sent. This is needed to allow the application to know when it can send new tones without canceling the tones that are currently being sent.

Editor Note: It seems we would want a callback or event for incoming tones. The proposal sent to the list had them played as audio to the speaker but I don’t see how that is useful.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| tones | DOMString | ✘ | ✘ |  |
| duration | long | ✘ | ✔ |  |

*Return type:* void

1. **4. Peer-to-peer connections**

A [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) allows two users to communicate directly, browser to browser. Communications are coordinated via a signaling channel which is provided by unspecified means, but generally by a script in the page via the server, e.g. using XMLHttpRequest.

Calling new [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection)(*configuration* ) creates a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object.

The *configuration* has the information to find and access the [[*STUN*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-STUN)] and [[*TURN*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-TURN)] servers. There may be multiple servers of each type and any TURN server also acts as a STUN server.

A PeerConnection object has an associated ICE Agent, PeerConnection state, and ICE State. These are initialized when the object is created.

When the *PeerConnection()* constructor is invoked, the user agent *must* run the following steps. This algorithm has a synchronous section (which is triggered as part of the event loop algorithm).

1. Create an ICE Agent and let *connection*’s [PeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-ice-agent) be that ICE Agent and provide it the STUN and TURN servers from the configuration array. The [[*ICE*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)] will proceed with gathering as soon as the IceTransports constraint is not set to "none". At this point the ICE Agent does not know how many ICE components it needs (and hence the number of candidates to gather) but it can make a reasonable assumption and as the PeerConnection object gets more information, it can adjust the number of components.
2. Set *connection*’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) to ["new"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-NEW) .
3. Set *connection*’s [PeerConnection ice state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) to ["new"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-NEW) .
4. Let *connection*’s [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams) attribute be an empty read-only MediaStream array.
5. Let *connection*’s [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams) attribute be an empty read-only MediaStream array.
6. Return *connection*, but continue these steps asynchronously.
7. Await a stable state. The synchronous section consists of the remaining steps of this algorithm.

During the lifetime of the PeerConnection object, the following procedures are followed:

1. If the ice state is "new" and the IceTransports constraint is not set to "none", it *must* queue a task to start gathering ICE address and set the ice state to "gathering".
2. If the ICE Agent has found one or more candidate pairs for any MediaTrack that forms a valid connection, the ICE state is changed to "connected".
3. When the ICE Agent finishes checking all candidate pairs, if at least one connection has been found for some MediaTrack, the iceState is changed to "completed" and if no connection has been found for any MediaTrack, the iceState is changed to "failed". [[OPEN ISSUE: Note that this means that if I was able to negotiate audio but not video via ICE, then iceState == "completed". Is this really what is desired?]]
4. If the iceState is "connected" or "completed" and both the local and remote session descriptions are set, the peerState is set to "active".
5. If the iceState is "failed", a task is queued to calls the close method. Open Issue: CJ - this seems wrong to me.

User agents negotiate the codec resolution, bitrate, and other media parameters. User agents are encouraged to initially negotiate for the maximum resolution of a video stream. For streams that are then rendered (using a video element), user agents are encouraged to renegotiate for a resolution that matches the rendered display size.

Starting with the native resolution means that if the Web application notifies its peer of the native resolution as it starts sending data, and the peer prepares its video element accordingly, there will be no need for a renegotiation once the stream is flowing.

The word "components" in this context refers to an RTP media flow and does not have anything to do with how [[*ICE*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)] uses the term "component".

When a user agent has reached the point where a MediaStream can be created to represent incoming components, the user agent *must* run the following steps:

1. Let *connection* be the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) expecting this media.
2. Create a MediaStream object to represent the media stream. [[OPEN ISSUE: What if one already exists?]]
3. Run the following steps for each component in the media stream.
	1. Create a MediaStreamTrack object *track* to represent the component. [[EDITORIAL: Can we just reference 3.2.1.2 here?]]
	2. If *track's* [kind](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtrack-kind) attribute equals "audio", add it to the MediaStream object's [audioTracks](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastream-audiotracks) MediaStreamTrackList object.
	3. If *track's* [kind](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtrack-kind) attribute equals "video", add it to the MediaStream object's [videoTracks](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastream-videotracks) MediaStreamTrackList object.

The creation of new incoming MediaStreams may be triggered either by SDP negotiation or by the receipt of media on a given flow.

The internal order in the MediaStreamTrackList objects on the receiving side should reflect the order on the sending side. One way to enforce this is to specify the order in the SDP.

1. Queue a task to run the following substeps:
	1. If the *connection*’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), abort these steps.
	2. Add the newly created MediaStream object to the end of *connection*’s [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams) array.
	3. [Fire a stream event](http://dev.w3.org/2011/webrtc/editor/webrtc.html#fire-a-stream-event) named [addstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-addstream) with the newly created MediaStream object at the *connection* object.

When a user agent has negotiated media for a component that belongs to a media stream that is already represented by an existing MediaStream object, the user agent *must* associate the component with that MediaStream object.

When a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection) finds that a stream from the remote peer has been removed , the user agent *must* follow these steps:

1. Let *connection* be the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) associated with the stream being removed.
2. Let *stream* be the MediaStream object that represents the media stream being removed, if any. If there isn't one, then abort these steps.
3. By definition, *stream* is now finished.

A task is thus queued to update *stream* and fire an event.

1. Queue a task to run the following substeps:
	1. If the *connection*’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), abort these steps.
	2. Remove *stream* from *connection*’s [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams) array.
	3. [Fire a stream event](http://dev.w3.org/2011/webrtc/editor/webrtc.html#fire-a-stream-event) named [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream) with *stream* at the *connection* object.

The task source for the tasks listed in this section is the networking task source.

If something in the browser changes that causes the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object to need to initiate a new session descipriton negotiation, an [renegotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-renegotiation) event is fired at the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object.

In particular, if a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object is consuming a MediaStream and a track is added to one of the stream's MediaStreamTrackList objects, by, e.g., the [add()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtracklist-add) method being invoked, the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object *must* fire the "renegotiationneeded" event. Removal of media components must also trigger "renegotianneeded".

To prevent network sniffing from allowing a fourth party to establish a connection to a peer using the information sent out-of-band to the other peer and thus spoofing the client, the configuration information *should* always be transmitted using an encrypted connection.

1. **4.1 PeerConnection**

The general operation of the PeerConnection is described in [[*RTCWEB-JSEP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-RTCWEB-JSEP)].

**4.1.1 SdpType**

The SdpType enums serve as arguments to setLocalDescription and setRemoteDescription. They provide information as to how the SDP should be handled.

 enum SdpType { "offer", "pranswer", "answer" }

["offer"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SdpType-offer)

An SdpType of "offer" indicates that a description should be treated as an [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] offer.

["pranswer"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SdpType-pranswer)

An SdpType of "pranswer" indicates that a description should be treated as an [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] answer, but not a final answer. A description used as a SDP "pranswer" may be applied as a response to a SDP offer, or an update to a previously sent SDP "pranswer".

["answer"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SdpType-answer)

An SdpType of "answer" indicates that a description should be treated as an [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] final answer, and the offer-answer exchange should be considered complete. A description used as a SDP answer may be applied as a response to a SDP offer, or an update to a previously send SDP "pranswer".

**4.1.2 SessionDescription Class**

The *SessionDescription()* constructor takes one argument, *description*, whose content is used to construct the new [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) object. This class is a future extensible carrier for for the data contained in it and does not perform any substantive processing.

[Constructor (DOMString description)]

interface SessionDescription {

 attribute SdpType [type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SessionDescription-type);

 attribute DOMString [sdp](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SessionDescription-sdp);

 stringifier [DOMString](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-SessionDescription-DOMString-stringifier) ();

};

**4.1.2.1 Attributes**

sdp of type DOMString

The string representation of the SDP [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)]

type of type SdpType

What type of SDP this SessionDescription represents.

**4.1.2.2 Methods**

DOMString

Objects that implement the [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) interface must stringify as [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)].

*No parameters.*

*Return type:* stringifier

**4.1.3 SessionDescriptionCallback**

 callback SessionDescriptionCallback = void (SessionDescription

 sdp)

SessionDescription sdp

The object containing the SDP [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)].

**4.1.4 PeerConnectionErrorCallback**

 callback PeerConnectionErrorCallback = void (DOMString errorInformation)

DOMString errorInformation

Information about what went wrong. Open Issue: How does this work? Is it human readable? I18N? ENUM?

TODO: Open Issue: should this be defined as event like NavigatorUserMediaErrorCallback in getusermedia

**4.1.5 PeerState Enum**

enum PeerState { "new" "opening", "active", "closing", "closed"

 }

["new"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-NEW)

The object was just created, and no networking has yet occurred.

["opening"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-OPENING)

The user agent is attempting to establish an connection with the ICE Agent and waiting for local and remote SDP to be set. (Open Issue: do we need more states between "opening" and "active")

["active"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ACTIVE)

The ICE Agent has found a connection both the local and remote SDP have been set. It is possible for media to flow.

["closing"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object is terminating all media and is in the process of closing the connection.

["closed"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The connection is closed.

**4.1.6 IceState Enum**

 enum IceState { "new" "gathering", "waiting", "checking",

 "connected", "completed","failed", "closed" }

["new"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-NEW)

The PeerConnection object was just created, and no networking has yet occurred.

["gathering"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ICE_GATHERING)

The ICE Agent is attempting to gather addresses.

["waiting"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ICE_WAITING)

The ICE Agent is not gathering any addresses and is waiting for candidates from the other side before it can start checking.

["checking"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ACTIVE)

The ICE Agent is checking candidate pairs but has not yet found a connection. In addition to checking, it may also still be gathering.

["connected"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The ICE Agent has found a connection but is still checking other candidate pairs to see if there is a better connection. It may also still be gathering.

["completed"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The ICE Agent has finished gathering and checking and found a connection.

["failed"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The ICE Agent is finished checking all candidate pairs and failed to find a connection.

["closed"](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED)

The ICE Agent has shut down and is no longer responding to STUN requests.

**4.1.7 IceCandidate Type**

The *IceCandidate()* constructor takes one argument, *candidate*, whose content is used to construct the new [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-icecandidate) object. This class is a future extensible carrier for for the data contained in it and does not perform any substantive processing.

[Constructor (DOMString candidate)]

interface IceCandidate {

 attribute DOMString [candidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-IceCandidate-candidate);

 stringifier [DOMString](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-IceCandidate-DOMString-stringifier) ();

};

**4.1.7.1 Attributes**

candidate of type DOMString

This carries the candidate-attribute as defined in section 15.1 of [[*ICE*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)]. ( TODO - need to add more information to allow this to match to correct m line - Open Issue: How to correlate. Need to wait for the mapping from media tracks to SDP to be resolved in IETF before tackling this problem).

**4.1.7.2 Methods**

DOMString

Objects that implement the [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-icecandidate) interface must stringify as the candidate-attribute as defined in section 15.1 of [[*ICE*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)].

*No parameters.*

*Return type:* stringifier

**4.1.8 IceCandidateCallback**

 callback IceCandidateCallback = void (IceCandidate candidate)

IceCandidate candidate

The new ICE candidate.

**4.1.9 IceServers Type - Option 1**

Open Issue: choose option 1 or option 2 for IceServers Type.

interface IceServers {

 attribute DOMString [servers[][]](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-IceServers-servers);

};

**4.1.9.1 Attributes**

servers[][] of type DOMString

The IceServers type is an array of pairs where each pair is defined as an array. Each pair provides the information to reach and use one STUN or TURN server. The first element in each pair is a stun or turn URIs as defined in [[*STUN-URI*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-STUN-URI)] and [[*TURN-URI*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-TURN-URI)]. If the first element of the pair is TURN URI, then the second element of the pair is the credential to use with that TURN server.

In network topologies with multiple layers of NATs, it is desirable to have a STUN servers between every layer of NATs in addition to the TURN servers to minimize the number peer to peer network latency.

An example configuration object is:

{ servers:[ ["stun:stun.example.net"] , ["turn:user@turn.example.org","myPassword"] ]}

**4.1.10 IceServers Type - Option 2**

Open Issue: choose option 1 or option 2 for IceServers Type.

interface IceServers {

 attribute DOMString [servers[]](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-IceServers-servers);

};

**4.1.10.1 Attributes**

servers[] of type DOMString

The IceServers type is an array of strings where each string provides the URL and credentials for a server. Each string is either a the URL to reach a STUN server ad defined in [[*STUN-URI*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-STUN-URI)] or is the URL of a TURN server as defined in [[*TURN-URI*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-TURN-URI)] followed by a single space and then the rest of the string is the credential used to access that server. Note the credential may contains spaces.

In network topologies with multiple layers of NATs, it is desirable to have a STUN servers between every layer of NATs in addition to the TURN servers to minimize the number peer to peer network latency.

An example configuration object is:

{ servers:[ "stun:stun.example.net" , "turn:user@turn.example.org myPassword" ]}

**4.1.11 PeerConnection Interface**

Open Issue: should we collapse some of these functions a single "processRemoteSignal" method?

[Constructor (IceServers configuration, optional MediaConstraints constraints)]

interface PeerConnection {

 void [createOffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-createOffer-void-SessionDescriptionCallback-successCallback-PeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints) (SessionDescriptionCallback successCallback, optional PeerConnectionErrorCallback failureCallback, optional MediaConstraints constraints);

 void [createAnswer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-createAnswer-void-SessionDescription-offer-SessionDescriptionCallback-successCallback-PeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints-Boolean-createProvisionalAnswer-false) ([SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22idl-def-SessionDescription) offer, SessionDescriptionCallback successCallback, optional PeerConnectionErrorCallback failureCallback, optional MediaConstraints constraints, optional Boolean createProvisionalAnswer=false);

 void [setLocalDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-setLocalDescription-void-SdpType-action-SessionDescription-description) (SdpType action, [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) description);

 readonly attribute [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) [localDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localDescription);

 void [setRemoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-setRemoteDescription-void-SdpType-action-SessionDescription-description) (SdpType action, [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) description);

 readonly attribute [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) [remoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteDescription);

 readonly attribute PeerState [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-readyState);

 void [updateIce](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-updateIce-void-IceServers-configuration-MediaConstraints-constraints-Boolean-restart-false) (optional [IceServers](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceServers) configuration, optional MediaConstraints constraints, optional Boolean restart=false);

 void [addIceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-addIceCandidate-void-IceCandidate-candidate) ([IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22idl-def-IceCandidate) candidate);

 readonly attribute IceState [iceState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-iceState);

 readonly attribute MediaStream[] [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams);

 readonly attribute MediaStream[] [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams);

 [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) [createDataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-createDataChannel-DataChannel-DOMString-label-DataChannelInit-dataChannelDict) ([TreatNullAs=EmptyString] DOMString? label, optional [DataChannelInit](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelInit)? dataChannelDict);

 attribute Function? [ondatachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ondatachannel);

 void [addStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-addStream-void-MediaStream-stream-MediaConstraints-constraints) (MediaStream stream, optional MediaConstraints constraints);

 void [removeStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-removeStream-void-MediaStream-stream) (MediaStream stream);

 void [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-close-void) ();

 attribute Function? [onrenegotationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onrenegotationneeded);

 attribute Function? [onicecandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onicecandidate);

 attribute Function? [onconnecting](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onconnecting);

 attribute Function? [onopen](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onopen);

 attribute Function? [onstatechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onstatechange);

 attribute Function? [onaddstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onaddstream);

 attribute Function? [onremovestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onremovestream);

 attribute Function? [onicechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-onicechange);

};

**4.1.11.1 Attributes**

iceState of type IceState, readonly

The *iceState* attribute *must* return the state of the [PeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-ice-agent) ICE state.

localDescription of type [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription), readonly

The localDescription method returns a copy of the SessionDescription that was most recently passed to setLocalDescription, plus any local candidates that have been generated by the ICE Agent since then.

A null object will be returned if the local description has not yet been set.

localStreams of type array of MediaStream, readonly

Returns a live array containing the local streams (those that were added with [addStream()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-addstream) ).

onaddstream of type Function, nullable

This event handler, of event handler event type [addstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-addstream) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface. Open Issue: It seems like this even handler needs to be fired when the first of two things happens - the remote side sends signaling indicating a media will be sent, or the side that sent an offer start receiving media in reply to that offer.

onconnecting of type Function, nullable

This event handler, of event handler event type [connecting](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-connecting) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface.

ondatachannel of type Function, nullable

This event handler, of type [datachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-peerconnection-datachannel) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface.

onicecandidate of type Function, nullable

This event handler, of event handler event type [onicecandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-onicecandidate) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface. It is called any time there is a new ICE candiate can be added to the a previos offer or answer.

onicechange of type Function, nullable

This event handler, of event handler event type [icechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-icechange) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface. It is called any time the iceState changes.

onopen of type Function, nullable

This event handler, of event handler event type [open](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-open) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface.

onremovestream of type Function, nullable

This event handler, of event handler event type [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface.

onrenegotationneeded of type Function, nullable

This event handler, of event handler event type [renegotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-renegotiation-needed) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface. Open Issue: Need to sort out which things should be Function and which should be a Callback.

onstatechange of type Function, nullable

This event handler, of event handler event type [statechange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-open) , *must* be supported by all objects implementing the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) interface. It is called any time the readyState changes.

readyState of type PeerState, readonly

The *readyState* attribute *must* return the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state).

remoteDescription of type [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription), readonly

The remoteDescription method returns a copy of the current remote the SessionDescription that was most recently passed to setRemoteDescription, plus any remote candidates that have been supplied via addIceCandidate since then.

A null object will be returned if the remote description has not yet been set.

remoteStreams of type array of MediaStream, readonly

Returns a live array containing the streams that the remote streams. (those that were added by the remote side).

This array is updated when [addstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-addstream) and [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream) events are fired.

**4.1.11.2 Methods**

addIceCandidate

The addIceCandidate method provides a remote candidate to the ICE Agent, which will be added to the remote description. Connectivity checks will be sent to the new candidates as long as the "IceTransports" constraint is not set to "none". This call will result in a change to the state of the ICE Agent, and may result in a change to media state if it results in different connectivity being established.

A TBD exception will be thrown if candidate parameter is malformed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| candidate | [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceCandidate) | ✘ | ✘ |  |

*Return type:* void

addStream

Adds a new stream to the PeerConnection.

When the *addStream()* method is invoked, the user agent *must* run the following steps:

1. If the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), throw an INVALID\_STATE\_ERR exception.
2. If *stream* is already in the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams) object, then abort these steps.
3. Add *stream* to the end of the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams) object.
4. Parse the *constraints* provided by the application and apply them to the MediaStream, if possible. NOTE - need to deal with throwing an exception here.
5. Fire a renegotiationneeded event. [[OPEN ISSUE: Should this fire if the PeerConnection is in "new"?]]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| stream | MediaStream | ✘ | ✘ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |

*Return type:* void

close

When the *close()* method is invoked, the user agent *must* run the following steps:

1. If the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), throw an INVALID\_STATE\_ERR exception.
2. Destroy the [PeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-ice-agent), abruptly ending any active ICE processing and any active streaming, and releasing any relevant resources (e.g. TURN permissions).
3. Set the object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) to [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3).

*No parameters.*

*Return type:* void

createAnswer

The createAnswer method generates a [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] answer with the supported configuration for the session that is compatible with the parameters supplied in offer. Like createOffer, the returned blob contains descriptions of the local MediaStreams attached to this PeerConnection, the codec/RTP/RTCP options negotiated for this session, and any candidates that have been gathered by the ICE Agent. The constraints parameter may be supplied to provide additional control over the generated answer.

As an answer, the generated SDP will contain a specific configuration that, along with the offer, specifies how the media plane should be established. The generation of the SDP must follow the appropriate process for generating an answer or provisional answer.

Session descriptions generated by createAnswer must be immediately usable by setLocalDescription without generating an error if setLocalDescription is called from the successCallback function. Like createOffer, the returned description should reflect the current state of the system. The session descriptions *must* remain usable by setLocalDescription without causing an error until at least the end of the successCallback function. Calling this method is is needed to get the ICE user name fragment and password. Provisional offers, as described in [[*RTCWEB-JSEP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-RTCWEB-JSEP)], are created if and only if the createProvisionalOffer flag is true.

The failureCallback will be called if the system can not generate an appropriate answer given the offer.

A TBD exception is thrown if the constraints parameter is malformed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| offer | [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) | ✘ | ✘ |  |
| successCallback | SessionDescriptionCallback | ✘ | ✘ |  |
| failureCallback | PeerConnectionErrorCallback | ✘ | ✔ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |
| createProvisionalAnswer=false | Boolean | ✘ | ✔ |  |

*Return type:* void

createDataChannel

Creates a new [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object with the given label. The [DataChannelInit](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelInit) dictionary can be used to configure properties of underlying channel such as data reliability. A corresponding [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object is dispatched at the other peer if the channel setup was successful.

When the *createDataChannel()* method is invoked, the user agent *must* run the following steps.

1. If the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), throw an INVALID\_STATE\_ERR exception.
2. Let *channel* be a newly created [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object.
3. Initialize *channel*’s [label](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-label) attribute to the value of the first argument.
4. Initialize *channel*’s [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-reliable) attribute to true.
5. If the second argument is present and it contains a [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannelInit-reliable) dictionary member, then set *channel*’s [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-reliable) attribute to the dictionary member value.
6. Return *channel* and continue these steps in the background.
7. Create *channel*’s associated [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| label | DOMString | ✔ | ✘ |  |
| dataChannelDict | [DataChannelInit](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelInit) | ✔ | ✔ |  |

*Return type:* [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel)

createOffer

The createOffer method generates a blob of SDP that contains a RFC offer with the supported configurations for the session, including descriptions of the local MediaStreams attached to this PeerConnection, the codec/RTP/RTCP options supported by this implementation, and any candidates that have been gathered by the ICE Agent. The constraints parameter may be supplied to provide additional control over the offer generated.

As an offer, the generated SDP will contain the full set of capabilities supported by the session (as opposed to an answer, which will include only a specific negotiated subset to use); for each SDP line, the generation of the SDP must follow the appropriate process for generating an offer. In the event createOffer is called after the session is established, createOffer will generate an offer that is compatible with the current session, incorporating any changes that have been made to the session since the last complete offer-answer exchange, such as addition or removal of streams. If no changes have been made, the offer will be include the capabilities of the current local description as well as any additional capabilities that could be negotiated in an updated offer.

Session descriptions generated by createOffer *must* be immediately usable by setLocalDescription without causing an error as long as setLocalDiscription is called within the successCallback function. If a system has limited resources (e.g. a finite number of decoders), createOffer needs to return an offer that reflects the current state of the system, so that setLocalDescription will succeed when it attempts to acquire those resources. The session descriptions *must* remain usable by setLocalDescription without causing an error until at least end of the successCallback function. Calling this method is needed to get the ICE user name fragment and password.

The failureCallback will be called if the system can not generate an appropriate offer given the state of the PeerConnection.

A TBD exception is thrown if the constraints parameter is malformed. [[ OPEN ISSUE: How are errors reported? ]]

To Do: Discuss privacy aspects of this from a finger printing point of view - it's probably around as bad as access to a canvas :-)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| successCallback | SessionDescriptionCallback | ✘ | ✘ |  |
| failureCallback | PeerConnectionErrorCallback | ✘ | ✔ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |

*Return type:* void

removeStream

Removes the given stream from the localStream array in the PeerConnection and fires 'renegotiationneeded.

When the other peer stops sending a stream in this manner, a [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream) event is fired at the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object.

When the *removeStream()* method is invoked, the user agent *must* run the following steps:

1. If the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), throw an INVALID\_STATE\_ERR exception.
2. If *stream* is not in the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams) object, then abort these steps. TODO: Do we need an exception here?
3. Remove *stream* from the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-localStreams) object.
4. Fire a renegotiationneeded event.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| stream | MediaStream | ✘ | ✘ |  |

*Return type:* void

setLocalDescription

The setLocalDescription method instructs the PeerConnection to apply the supplied [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] description as the local offer or answer. The type parameter indicates whether the description should be processed as an offer, provisional answer, or final answer. [[OPEN ISSUE: The type appears as both the "action" argument and in the struct. That is redundant and just sort of crazy. We need to pick one.]]

This API changes the local media state. In order to successfully handle scenarios where the application wants to offer to change from one media format to a different, incompatible format, the PeerConnection must be able to simultaneously support use of both the old and new local descriptions (e.g. support codecs that exist in both descriptions) until a final answer is received, at which point the PeerConnection can fully adopt the new local description, or roll back to the old description if the remote side denied the change.

Open issues: how to indicate to roll back?

To Do: specify what parts of the SDP can be changed between the createOffer and setLocalDescription

Changes to the state of media transmission will occur when a final answer is successfully applied.

A TBD exception is thrown if sdp is invalid. A TBD exception is thrown if there are insufficient local resources to apply the sdp.

Open Issues: for setLocal and setRemote, discuss how to return erro codes and if they need to be asynchronous.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| action | SdpType | ✘ | ✘ |  |
| description | [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) | ✘ | ✘ |  |

*Return type:* void

setRemoteDescription

The setRemoteDescription method instructs the PeerConnection to apply the supplied [[*SDP*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-SDP)] in the description. As in setLocalDescription, the action parameter indicates how the blob should be processed. This API changes the local media state.

Changes to the state of media transmission will occur when a final answer is successfully applied.

A TBD exception is thrown if the sdp parameter is invalid. A TBD exception is thrown if there are insufficient local resources to apply the SDP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| action | SdpType | ✘ | ✘ |  |
| description | [SessionDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-SessionDescription) | ✘ | ✘ |  |

*Return type:* void

updateIce

The updateIce method restarts or updates the ICE Agent process of gathering local candidates and pinging remote candidates. If there is a mandatory constraint called "IceTransports" it will control which how the ICE engine can act. This can be used to limit the use to TURN candidates by a callee to avoid leaking location information prior to the call being accepted.

This call may result in a change to the state of the ICE Agent, and may result in a change to media state if it results in connectivity being established.

If the restart parameter is set to true, the ICE state machine discards all candidates it has gathered, allocates new ports for the host candidates, and restarts ICE as if there had been no previos ICE session. Applications can use this to reset all ICE negotiation when something has gone terribly wrong.

A TBD exception will be thrown if constraints parameter is malformed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| configuration | [IceServers](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceServers) | ✘ | ✔ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |
| restart=false | Boolean | ✘ | ✔ |  |

*Return type:* void

[PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) implements EventTarget;

1. **5. IANA Registrations**

IANA is requested to register the constraints defined in [Constraints Section](http://dev.w3.org/2011/webrtc/editor/webrtc.html#sec-constraints) as specified in [[*RTCWEB-CONSTRAINTS*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-RTCWEB-CONSTRAINTS)].

1. **5.1 Constraints**

TOOD: Need to change the naming and declaration of these constraints to match the constraints draft once that is a bit further along. The names here now are likely not quite right but they serve as a place holder. [[OPEN ISSUE: there are multiple ways to add constraints. How are multiple values reconciled?]]

The following new constraints are defined that can be used with a PeerConnection object:

OfferToReceiveVideo

This is a enum type constraint that can take the values "true" and "false". The default is a non mandatory "true" for a PeerConnection object that has a video stream at the point in time when the constraints are being evaluated and is non mandatory "false" otherwise.

In some cases, a PeerConnection may wish to receive video but it is not going to send any video. The PeerConnection needs to know if it should signal to the remote side if it wishes to receive video or not. This constraints allows an application to indicate its preferences for receiving video when creating an offer.

OfferToReceiveAudio

This is a enum type constraint that can take the values "true" and "false". The default is a non mandatory "true".

In some cases, a PeerConnection may wish to receive audio but it is not going to send any audio. The PeerConnection needs to know if it should signal to the remote side if it wishes to receive audio. This constraints allows an application to indicate its preferences for receiving audio when creating an offer.

VoiceActivityDetection

This is a enum type constraint that can take the values "true" and "false". The default is a non mandatory "true".

Many codecs and system are capable of detecting "silence" and changing there behavior in this case by doing things such as not transmitting any media. In many cases, such as when dealing with sounds other than spoken voice or emergency calling, it is desirable to be able to turn off this behavior. This constraints allows the application to provide information about if it wishes this type of processing enable or disabled.

IceTransports

This is a enum type constraint that can take the values "none", "relay", and "all". The default is a non mandatory "all".

This constraints indicates which candidates the ICE engine is restricted use. The value "none" means the ICE engine *must* not send or receive any packets at this point. The value "relay" indicates the ICE engine *must* only using media relay candidates such as candidates passing through a TURN server. This can be used to reduce leakage of IP addresses in certain use cases. The value of "all" indicates all values can be used.

TODO items - need to register with IANA.

1. **6. Simple Example**

When two peers decide they are going to set up a connection to each other, they both go through these steps. The STUN/TURN server configuration describes a server they can use to get things like their public IP address or to set up NAT traversal. They also have to send data for the signaling channel to each other using the same out-of-band mechanism they used to establish that they were going to communicate in the first place.

NOTE: TODO - This code does not match the API yet and might have a few other problems.

var signalingChannel = createSignalingChannel();

var pc;

// set up the call, get access to local media, and establish connectivity

function start(isCaller) {

 // Create a PeerConnection and hook up the IceCallback

 pc = new PeerConnection("", function (candidate) {

 signalingChannel.send(JSON.stringify({ "type": "candidate", "sdp": candidate }));

 });

 // get the local stream and show it in the local video element

 navigator.getUserMedia({"audio": true, "video": true}, function (stream) {

 selfView.src = URL.createObjectURL(stream);

 pc.addStream(stream);

 var type;

 if (isCaller) {

 pc.createOffer(gotDescription);

 type = "offer";

 } else {

 pc.createAnswer(pc.remoteDescription, gotDescription);

 type = "answer";

 }

 function gotDescription(desc) {

 pc.setLocalDescription(type, desc);

 signalingChannel.send(JSON.stringify({ "type": type, "sdp": desc }));

 }

 });

 // once remote stream arrives, show it in the remote video element

 pc.onaddstream = function (evt) {

 remoteView.src = URL.createObjectURL(evt.stream);

 };

}

signalingChannel.onmessage = function (evt) {

 var msg = JSON.parse(evt.data);

 var sdp = SessionDescription(msg.sdp)

 switch (msg.type) {

 case "offer":

 // create the PeerConnection

 start(false);

 // feed the received offer into the PeerConnection

 pc.setRemoteDescription(msg.type,SessionDescription(msg.sdp));

 break;

 case "answer":

 pc.setRemoteDescription(msg.type,SessionDescription(msg.sdp));

 break;

 case "candidate":

 pc.addIceCandidate(IceCandidate(msg.sdp));

 break;

 }

};

1. **7. Advanced Example**

This example shows the more comples functionality.

TODO

1. **8. Peer-to-peer Data API**

The Peer-to-peer Data API lets a web application send and receive generic application data peer-to-peer.

Open issues (this should not be considered as a complete list of open issues)

* Data channel setup signaling (signaling via SDP and application specific signaling channel or first channel via SDP and consecutive channels via internal signalling).
* What can be shared with the WebSocket API specification regarding actual interfaces.
1. **8.1 DataChannel**

The [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) interface represents a bi-directional data channel between two peers. A [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) is created via a factory method on a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object. The corresponding [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object is then dispatched at the other peer if the channel setup was successful.

Each [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) has an associated *underlying data transport* that is used to transport actual data to the other peer. The transport properties of the [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport), such as reliability mode, are configured by the peer taking the initiative to create the channel. The other peer cannot change any transport properties of a offered data channel. The actual wire protocol between the peers is out of the scope for this specification. Open Issues: this needs to explain how the configuration state is passed between the peers. Open Issues: this type of design where one side can pick anything and the other side much support everything has proven to make future upgrades very difficult.

A [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) created with [createDataChannel()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-createdatachannel) *must* initially be in the [CONNECTING](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CONNECTING) (0) state. If the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object’s [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) is successfully set up, the user agent *must* [announce the DataChannel as open](http://dev.w3.org/2011/webrtc/editor/webrtc.html#announce-datachannel-open).

When the user agent is to *announce a DataChannel as open*, the user agent *must* queue a task to run the following steps:

1. If the associated [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), abort these steps.
2. Let *channel* be the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object to be announced.
3. Set *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) attribute to [OPEN](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-OPEN) (1).
4. Fire a simple event named [open](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-open) at *channel*.

When an [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) has been established, the user agent, of the peer that did not initiate the creation process *must* queue a task to run the following steps:

1. If the associated [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), abort these steps.
2. Let *configuration* be an information bundle with key-value pairs, received from the other peer as a part of the process to establish the underlying data channel.
3. Let *channel* be a newly created [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object.
4. Initialize *channel*’s [label](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-label) attribute to value that corresponds to the "label" key in *configuration*.
5. Initialize *channel*’s [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-reliable) attribute to true.
6. If *configuration* contains a key named "reliable", set *channel*’s [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-reliable) attribute to the corresponding value.
7. Set *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) attribute to [OPEN](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-OPEN) (1).
8. [Fire a datachannel event](http://dev.w3.org/2011/webrtc/editor/webrtc.html#fire-a-datachannel-event) named [datachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-peerconnection-datachannel) with *channel* at the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object.

When the *process of tearing down a* [*DataChannel*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) *object’s* [*underlying data transport*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) is initiated, the user agent *must* run the following steps:

1. If the associated [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object’s [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-CLOSED) (3), abort these steps.
2. Let *channel* be the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object which is about to be closed.
3. If *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) is [CLOSING](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSED) (2) or [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSED) (3), then abort these steps.
4. Set *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) attribute to [CLOSING](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSING) (2).
5. Queue a task to run the following steps:
	1. Close *channel*’s [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport).

The data transport protocol will specify what happens to, e.g. buffered data, when the data transport is closed.

* 1. Set *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) attribute to [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSED) (3).
	2. Fire a simple event named [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-close) at *channel*.

interface DataChannel {

 readonly attribute DOMString [label](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-label);

 readonly attribute boolean [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-reliable);

 const unsigned short [CONNECTING](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CONNECTING) = 0;

 const unsigned short [OPEN](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-OPEN) = 1;

 const unsigned short [CLOSING](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSING) = 2;

 const unsigned short [CLOSED](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-CLOSED) = 3;

 readonly attribute unsigned short [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-readyState);

 readonly attribute unsigned long [bufferedAmount](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-bufferedAmount);

 [TreatNonCallableAsNull]

 attribute Function? [onopen](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-onopen);

 [TreatNonCallableAsNull]

 attribute Function? [onerror](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-onerror);

 [TreatNonCallableAsNull]

 attribute Function? [onclose](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-onclose);

 void [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-close-void) ();

 [TreatNonCallableAsNull]

 attribute Function? [onmessage](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-onmessage);

 attribute DOMString [binaryType](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-binaryType);

 void [send](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-send-void-DOMString-data) (DOMString data);

 void [send](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-send-void-ArrayBuffer-data) (ArrayBuffer data);

 void [send](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannel-send-void-Blob-data) (Blob data);

};

**8.1.1 Attributes**

binaryType of type DOMString

FIXME: align behavior with WebSocket API

bufferedAmount of type unsigned long, readonly

FIXME: align behavior with WebSocket API

label of type DOMString, readonly

The *DataChannel.label* attribute represents a label that can be used to distinguish this [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object from other [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) objects. The attribute *must* return the value to which it was set when the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object was created.

onclose of type Function, nullable

This event handler, of type [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-close) , *must* be supported by all objects implementing the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) interface.

onerror of type Function, nullable

This event handler, of type [error](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-error) , *must* be supported by all objects implementing the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) interface.

onmessage of type Function, nullable

This event handler, of type [message](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-message) , *must* be supported by all objects implementing the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) interface.

onopen of type Function, nullable

This event handler, of type [open](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-open) , *must* be supported by all objects implementing the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) interface.

readyState of type unsigned short, readonly

The *DataChannel.readyState* attribute represents the state of the DataChannel object. It *must* return the value to which the user agent last set it (as defined by the processing model algorithms). The attribute can have the following values: *CONNECTING*, *OPEN*, *CLOSING* or *CLOSED*.

reliable of type boolean, readonly

The *DataChannel.reliable* attribute returns true if the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) is reliable, and false otherwise. The attribute *must* return the value to which it was set when the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) was created.

**8.1.2 Methods**

close

Closes the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) . It may be called regardless if the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object was created by this peer or the remote peer.

When the *close()* method is called, the user agent *must* initiate [the process of tearing down](http://dev.w3.org/2011/webrtc/editor/webrtc.html#tear-down-data-transport) the DataChannel object’s [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport).

*No parameters.*

*Return type:* void

send

FIXME: align behavior with WebSocket API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| data | DOMString | ✘ | ✘ |  |

*Return type:* void

send

FIXME: align behavior with WebSocket API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| data | ArrayBuffer | ✘ | ✘ |  |

*Return type:* void

send

FIXME: align behavior with WebSocket API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Nullable** | **Optional** | **Description** |
| data | Blob | ✘ | ✘ |  |

*Return type:* void

**8.1.3 Constants**

CLOSED of type unsigned short

The [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) has been closed or could not be established.

CLOSING of type unsigned short

The process of closing down the [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) has started.

CONNECTING of type unsigned short

The user agent is attempting to establish the [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport). This is the initial state of a [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object created with [createDataChannel()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-createdatachannel) .

OPEN of type unsigned short

TODO - theses constants need to be changed to an enum.

The [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) is established and communication is possible. This is the initial state of a [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object dispatched as a part of a [DataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelEvent) .

dictionary DataChannelInit {

 boolean [reliable](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannelInit-reliable);

};

**8.1.4 Dictionary** [**DataChannelInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelInit) **Members**

reliable of type boolean

-

1. **8.2 Examples**

This simple example shows how to create a DataChannel, register an event listener to handle incoming data, and how to send a message.

var chan = peerConn.createDataChannel("mylabel");

chan.onmessage = function (evt) {

 // use evt.data };

 chan.send("hello");

This simple example shows how to register an event listener to handle the case when a remote peer creates a new DataChannel.

peerConn.ondatachannel = function (evt) {

 var chan = evt.channel;

 chan.onmessage = function (evt) {

 // use evt.data

 };

 chan.onclose = function () {

 // remote side closed the data channel

 };

};

>

1. **9. Garbage collection**

A Window object *has a strong reference* to any [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) objects created from the constructor whose global object is that Window object.

1. **10. Event definitions**
2. **10.1 PeerConnectionEvent**

Several of the PeerConnection events use the [PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEvent) interface.

*Firing a PeerConnectionEvent event named e* with a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) *peer* means that an event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the [PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEvent) interface with the peer attribute set to the *PeerConnectio* object, *must* be created and dispatched at the given target.

[Constructor(DOMString type, PeerConnectionEventInit eventInitDict)]

interface PeerConnectionEvent : Event {

 readonly attribute [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) [peer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionEvent-peer);

};
dictionary PeerConnectionEventInit : EventInit {

 [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) [peer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionEventInit-peer);

};

**10.1.1 Attributes**

peer of type [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection), readonly

The *peer* attribute represents the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object associated with the event.

**10.1.2 Dictionary** [**PeerConnectionEventInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEventInit) **Members**

peer of type [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection)

1. **10.2 PeerConnectionIceEvent**

The onicecandidate event of the PeerConnection uses the [PeerConnectionIceEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionIceEvent) interface.

*Firing a PeerConnectionIceEvent event named e* with a [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) *peer* means that an event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the PeerConnectionIceEvent interface with the peer attribute set to the *PeerConnection* object, and the candidate attribute set to the new ICE candiate *must* be created and dispatched at the given target.

[Constructor(DOMString type, PeerConnectionIceEventInit eventInitDict)]

interface PeerConnectionIceEvent : Event {

 readonly attribute [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) [peer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionIceEvent-peer);

 readonly attribute [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceCandidate) [candidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionIceEvent-candidate);

};

**10.2.1 Attributes**

candidate of type [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceCandidate), readonly

The candidate attribute is the [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-icecandidate) object with the new ICE candiate that caused the event.

peer of type [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection), readonly

The peer attribute represents the [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) object associated with the event.

dictionary PeerConnectionEventIceInit : EventInit {

 [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) [peer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionEventIceInit-peer);

 [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceCandidate) [candidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnectionEventIceInit-candidate);

};

**10.2.2 Dictionary** [**PeerConnectionEventIceInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEventIceInit) **Members**

candidate of type [IceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-IceCandidate)

peer of type [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection)

1. **10.3 MediaStreamTrackEvent**

The [addtrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastreamtracklist-addtrack) and [removetrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastreamtracklist-removetrack) events use the [MediaStreamTrackEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamTrackEvent) interface.

*Firing a track event named e* with a MediaStreamTrack *track* means that an event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the [MediaStreamTrackEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamTrackEvent) interface with the [track](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamtrackevent-track) attribute set to *track*, *must* be created and dispatched at the given target.

[Constructor(DOMString type, MediaStreamTrackEventInit eventInitDict)]

interface MediaStreamTrackEvent : Event {

 readonly attribute MediaStreamTrack [track](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-MediaStreamTrackEvent-track);

};
dictionary MediaStreamTrackEventInit : EventInit {

 readonly MediaStreamTrack? [track](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-MediaStreamTrackEventInit-track);

};

**10.3.1 Attributes**

track of type MediaStreamTrack, readonly

The *track* attribute represents the MediaStreamTrack object associated with the event.

**10.3.2 Dictionary** [**MediaStreamTrackEventInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamTrackEventInit) **Members**

track of type readonly MediaStreamTrack, nullable

1. **10.4 MediaStreamEvent**

The [addstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-addstream) and [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream) events use the [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent) interface.

*Firing a stream event named e* with a MediaStream *stream* means that an event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent) interface with the [stream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastreamevent-stream) attribute set to *stream*, *must* be created and dispatched at the given target.

[Constructor(DOMString type, MediaStreamEventInit eventInitDict)]

interface MediaStreamEvent : Event {

 readonly attribute MediaStream? [stream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-MediaStreamEvent-stream);

};
dictionary MediaStreamEventInit : EventInit {

 MediaStream [stream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-MediaStreamEventInit-stream);

};

**10.4.1 Attributes**

stream of type MediaStream, readonly, nullable

The *stream* attribute represents the MediaStream object associated with the event.

**10.4.2 Dictionary** [**MediaStreamEventInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEventInit) **Members**

stream of type MediaStream

1. **10.5 DataChannelEvent**

The [datachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-peerconnection-datachannel) event use the [DataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelEvent) interface.

*Firing a datachannel event named e* with a [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) *channel* means that an event with the name *e*, which does not bubble (except where otherwise stated) and is not cancelable (except where otherwise stated), and which uses the [DataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelEvent) interface with the [channel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannelevent-channel) attribute set to *channel*, *must* be created and dispatched at the given target.

[Constructor(DOMString type, DataChannelEventInit eventInitDict)]

interface DataChannelEvent : Event {

 readonly attribute [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) [channel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannelEvent-channel);

};
dictionary DataChannelEventInit : EventInit {

 [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) [channel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-DataChannelEventInit-channel);

};

**10.5.1 Attributes**

channel of type [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel), readonly

The *channel* attribute represents the [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object associated with the event.

**10.5.2 Dictionary** [**DataChannelEventInit**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelEventInit) **Members**

channel of type [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel)

1. **11. Event summary**

*This section is non-normative.*

The following event fires on MediaStream objects:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| *ended*  | Event  | The MediaStream finished as a result of all tracks in the MediaStream [ending](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastreamtrack-ended). |

The following event fires on MediaStreamTrack objects:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| *muted*  | Event  | The MediaStreamTrack object's source is temporarily unable to provide data. |
| *unmuted*  | Event  | The MediaStreamTrack object's source is live again after having been temporarily unable to provide data. |
| *ended*  | Event  | The MediaStreamTrack object's source will no longer provide any data, either because the user revoked the permissions, or because the source device has been ejected, or because the remote peer stopped sending data, or because the [stop()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-mediastream-stop) method was invoked. |

The following event fires on MediaStreamTrackList objects:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| *addtrack*  | [MediaStreamTrackEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamTrackEvent)  | A new MediaStreamTrack has been added to this list. |
| *removetrack*  | [MediaStreamTrackEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamTrackEvent)  | A MediaStreamTrack has been removed from this list. |

The following event fires on [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) objects:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| *open*  | Event  | The [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object’s [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) has been established (or re-established). |
| *MessageEvent*  | Event  | A message was successfully received. TODO: Ref where MessageEvent is defined?  |
| *error*  | Event  | TODO. |
| *close*  | Event  | The [DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object’s [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) has was closed. |

The following events fire on [PeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnection) objects:

|  |  |  |
| --- | --- | --- |
| **Event name** | **Interface** | **Fired when...** |
| *connecting*  | Event  | TODO |
| *open*  | Event  | TODO |
| *addstream*  | [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent)  | A new stream has been added to the [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams) array. |
| *removestream*  | [MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamEvent)  | A stream has been removed from the [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-remoteStreams) array. |
| *renegotiationneeded*  | [PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEvent)  | The browser wishes to inform the application that session negotiation needs to be redone at some point in the near future. Open Issue: should this be moved to "Negotiation Needed" instead of "Re-Negotiation Needed"?  |
| *statechange*  | [PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEvent)  | TODO  |
| *icechange*  | [PeerConnectionEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionEvent)  | TODO  |
| *icecandidate*  | [PeerConnectionIceEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-PeerConnectionIceEvent)  | TODO  |

1. **12. Change Log**

This section will be removed before publication.

1. **Changes since May 28, 2012**
2. TODO
3. **Changes since April 27, 2012**
4. Major rewrite of PeerConnection section to line up with IETF JSEP draft.
5. Added simple PeerConnection example. Initial update of SessionDescription and IceCandidate to support serialization and construction.
6. **Changes since 21 April 2012**
7. Moved MediaStream and related definitions to getUserMedia.
8. Removed some left-overs from the old Data Stream API.
9. Initial import of new Data API.
10. Renamed "underlying data channel" to "underlying data transport". Fixed closing procedures. Fixed some typos.
11. **Changes since 12 January 2012**
12. Clarified the relation of Stream, Track, and Channel.
13. **Changes since 17 October 2011**
14. Tweak the introduction text and add a reference to the IETF RTCWEB group.
15. Changed the first argument to getUserMedia to be an object.
16. Added a MediaStreamHints object as a second argument to PeerConnection.addStream.
17. Added AudioMediaStreamTrack class and DTMF interface.
18. **Changes since 23 August 2011**
19. Separated the SDP and ICE Agent into separate agents and added explicit state attributes for each.
20. Removed the send method from PeerConenction and associated callback function.
21. Modified MediaStream() constructor to take a list of MediaStreamTrack objects instead of a MediaStream. Removed text about MediaStream parent and child relationship.
22. Added abstract.
23. Moved a few paragraphs from the MediaStreamTrack.label section to the MediaStream.label section (where they belong).
24. Split MediaStream.tracks into MediaStream.audioTracks and MediaStream.videoTracks.
25. Removed a sentence that implied that track access is limited to LocalMediaStream.
26. Updated a few getUserMedia()-examples to use MediaStreamOptions.
27. Replaced calls to URL.getObjectURL() with URL.createObjectURL() in example code.
28. Fixed some broken getUserMedia() links.
29. Introduced state handling on MediaStreamTrack (removed state handling from MediaStream).
30. Reintroduced onended on MediaStream to simplify checking if all tracks are ended.
31. Aligned the MediaStreamTrack ended event dispatching behavior with that of MediaStream.
32. Updated the LocalMediaStream.stop() algorithm to implicitly use the end track algorithm.
33. Replaced an occurrence the term finished track with ended track (to align with rest of spec).
34. Moved (and extended) the explanation about track references and media sources from LocalMediaStream to MediaStreamTrack.
35. Removed section "Obtaining local multimedia content".
36. Updated getUserMedia() calls in examples (changes in Media Capture TF spec).
37. Introduced MediaStreamTrackList interface with support for adding and removing tracks.
38. Updated the algorithm that is run when PeerConnection receives a stream (create new stream when negotiated instead of when data arrives).
39. **A. Acknowledgements**

The editors wish to thank the Working Group chairs, Harald Alvestrand and Stefan Håkansson, for their support.

1. **B. References**
2. **B.1 Normative references**

[GETUSERMEDIA]

D. Burnett, A. Narayanan. [*getusermedia: Getting access to local devices that can generate multimedia streams*](http://dev.w3.org/2011/webrtc/editor/getusermedia.html) 22 December 2011. W3C Editors draft (Work in progress.) URL: <http://dev.w3.org/2011/webrtc/editor/getusermedia.html>

[ICE]

J. Rosenberg. [*Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.*](http://tools.ietf.org/html/rfc5245) April 2010. Internet RFC 5245. URL: <http://tools.ietf.org/html/rfc5245>

[RFC2119]

S. Bradner. [*Key words for use in RFCs to Indicate Requirement Levels.*](http://www.ietf.org/rfc/rfc2119.txt) March 1997. Internet RFC 2119. URL: <http://www.ietf.org/rfc/rfc2119.txt>

[RTCWEB-CONSTRAINTS]

D. Burnett. [*IANA Registry for RTCWeb Media Constraints.*](http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/) URL: <http://datatracker.ietf.org/doc/draft-burnett-rtcweb-constraints-registry/>

[SDP]

J. Rosenberg, H. Schulzrinne. [*An Offer/Answer Model with the Session Description Protocol (SDP).*](http://tools.ietf.org/html/rfc3264) June 2002. Internet RFC 3264. URL: <http://tools.ietf.org/html/rfc3264>

[STUN]

J. Rosenberg, R. Mahy, P. Matthews, D. Wing. [*Session Traversal Utilities for NAT (STUN).*](http://tools.ietf.org/html/rfc5389) October 2008. Internet RFC 5389. URL: <http://tools.ietf.org/html/rfc5389>

[STUN-URI]

S. Nandakumar, G. Salgueiro, P. Jones, and M. Petit-Huguenin. [*URI Scheme for Session Traversal Utilities for NAT (STUN) Protocol.*](http://tools.ietf.org/html/draft-nandakumar-rtcweb-stun-uri) 12 March 2012. Internet Draft (work in progress). URL: <http://tools.ietf.org/html/draft-nandakumar-rtcweb-stun-uri>

[TURN]

P. Mahy, P. Matthews, J. Rosenberg. [*Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).*](http://tools.ietf.org/html/rfc5766) April 2010. Internet RFC 5766. URL: <http://tools.ietf.org/html/rfc5766>

[TURN-URI]

M. Petit-Huguenin, S. Nandakumar, G. Salgueiro, and P. Jones. [*Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers.*](http://tools.ietf.org/html/draft-petithuguenin-behave-turn-uris) 12 March 2012. Internet Draft (work in progress). URL: <http://tools.ietf.org/html/draft-petithuguenin-behave-turn-uris>

[WEBIDL]

Cameron McCormack. [*Web IDL.*](http://www.w3.org/TR/2011/WD-WebIDL-20110927/) 27 September 2011. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/2011/WD-WebIDL-20110927/>

1. **B.2 Informative references**

[RTCWEB-JSEP]

J. Uberti, C. Jennings. [*Javascript Session Establishment Protocol.*](http://datatracker.ietf.org/doc/draft-ietf-rtcweb-jsep/) URL: <http://datatracker.ietf.org/doc/draft-ietf-rtcweb-jsep/>