

WebRTC 1.0: Real-time Communication Between Browsers

W3C Editor's Draft 20 September 2012

**This version:**

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

**Latest published version:**

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

**Latest editor's draft:**

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

Abstract

This document defines a set of ECMAScript APIs in WebIDL to allow media to be sent over the network to another browser or device implementing the appropriate set of real-time protocols, and media to be received from another browser or device. 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.

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).

Table of Contents

* [1. Introduction](http://dev.w3.org/2011/webrtc/editor/webrtc.html#intro)
* [2. Conformance](http://dev.w3.org/2011/webrtc/editor/webrtc.html#conformance)
* [3. Terminology](http://dev.w3.org/2011/webrtc/editor/webrtc.html#terminology)
* [4. Network Stream API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#network-stream-api)
	+ [4.1 Introduction](http://dev.w3.org/2011/webrtc/editor/webrtc.html#introduction)
	+ [4.2 Interface definitions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#interface-definitions)
		- [4.2.1 MediaStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastream)
			* [4.2.1.1 label](http://dev.w3.org/2011/webrtc/editor/webrtc.html#label)
			* [4.2.1.2 Events on MediaStream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#events-on-mediastream)
		- [4.2.2 MediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamtrack)
	+ [4.3 AudioMediaStreamTrack](http://dev.w3.org/2011/webrtc/editor/webrtc.html#audiomediastreamtrack)
		- [4.3.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes)
		- [4.3.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods)
* [5. Peer-to-peer connections](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-connections)
	+ [5.1 RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection)
		- [5.1.1 RTCSdpType](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcsdptype)
		- [5.1.2 RTCSessionDescription Class](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcsessiondescription-class)
			* [5.1.2.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-1)
			* [5.1.2.2 Dictionary RTCSessionDescriptionInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcsessiondescriptioninit-members)
		- [5.1.3 RTCSessionDescriptionCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcsessiondescriptioncallback)
			* [5.1.3.1 Callback RTCSessionDescriptionCallback Parameters](http://dev.w3.org/2011/webrtc/editor/webrtc.html#callback-rtcsessiondescriptioncallback-parameters)
		- [5.1.4 RTCStatsCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcstatscallback)
			* [5.1.4.1 Callback RTCStatsCallback Parameters](http://dev.w3.org/2011/webrtc/editor/webrtc.html#callback-rtcstatscallback-parameters)
		- [5.1.5 RTCVoidCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcvoidcallback)
		- [5.1.6 RTCPeerConnectionErrorCallback](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnectionerrorcallback)
			* [5.1.6.1 Callback RTCPeerConnectionErrorCallback Parameters](http://dev.w3.org/2011/webrtc/editor/webrtc.html#callback-rtcpeerconnectionerrorcallback-parameters)
		- [5.1.7 RTCPeerState Enum](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerstate-enum)
		- [5.1.8 RTCGatheringState Enum](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcgatheringstate-enum)
		- [5.1.9 RTCIceState Enum](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcicestate-enum)
		- [5.1.10 RTCIceCandidate Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcicecandidate-type)
			* [5.1.10.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-2)
			* [5.1.10.2 Dictionary RTCIceCandidateInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcicecandidateinit-members)
		- [5.1.11 RTCIceServer Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtciceserver-type)
			* [5.1.11.1 Dictionary RTCIceServer Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtciceserver-members)
		- [5.1.12 RTCConfiguration Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcconfiguration-type)
			* [5.1.12.1 Dictionary RTCConfiguration Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcconfiguration-members)
		- [5.1.13 RTCIdentityAssertion Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcidentityassertion-type)
			* [5.1.13.1 Dictionary RTCIdentityAssertion Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcidentityassertion-members)
		- [5.1.14 RTCStatsElement dictionary](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcstatselement-dictionary)
			* [5.1.14.1 Dictionary RTCStatsElement Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcstatselement-members)
		- [5.1.15 RTCStatsReport Type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcstatsreport-type)
			* [5.1.15.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-3)
			* [5.1.15.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-1)
		- [5.1.16 RTCPeerConnection Interface](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-interface)
		- [5.1.17 Garbage collection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#garbage-collection)
* [6. IANA Registrations](http://dev.w3.org/2011/webrtc/editor/webrtc.html#iana-registrations)
	+ [6.1 Constraints](http://dev.w3.org/2011/webrtc/editor/webrtc.html#constraints)
* [7. Simple Example](http://dev.w3.org/2011/webrtc/editor/webrtc.html#simple-example)
* [8. Advanced Example](http://dev.w3.org/2011/webrtc/editor/webrtc.html#advanced-example)
* [9. Call Flow Browser to Browser](http://dev.w3.org/2011/webrtc/editor/webrtc.html#call-flow-browser-to-browser)
* [10. Call Flow Browser to MCU](http://dev.w3.org/2011/webrtc/editor/webrtc.html#call-flow-browser-to-mcu)
* [11. Peer-to-peer Data API](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-to-peer-data-api)
	+ [11.1 DataChannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#datachannel)
		- [11.1.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-4)
		- [11.1.2 Methods](http://dev.w3.org/2011/webrtc/editor/webrtc.html#methods-2)
		- [11.1.3 Dictionary DataChannelInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-datachannelinit-members)
	+ [11.2 Examples](http://dev.w3.org/2011/webrtc/editor/webrtc.html#examples)
	+ [11.3 Garbage Collection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#garbage-collection-1)
* [12. Statistics Model](http://dev.w3.org/2011/webrtc/editor/webrtc.html#statistics-model)
	+ [12.1 Example](http://dev.w3.org/2011/webrtc/editor/webrtc.html#example)
* [13. Identity Provider Interaction](http://dev.w3.org/2011/webrtc/editor/webrtc.html#identity-provider-interaction)
	+ [13.1 Peer-Connection/IdP Communications](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peer-connection-idp-communications)
	+ [13.2 Requesting Assertions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#requesting-assertions)
	+ [13.3 Verifying Assertions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#verifying-assertions)
	+ [13.4 Examples](http://dev.w3.org/2011/webrtc/editor/webrtc.html#examples-1)
* [14. Event definitions](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-definitions)
	+ [14.1 RTCPeerConnectionIceEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnectioniceevent)
		- [14.1.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-5)
		- [14.1.2 Dictionary RTCPeerConnectionIceEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-rtcpeerconnectioniceeventinit-members)
	+ [14.2 MediaStreamEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#mediastreamevent)
		- [14.2.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-6)
		- [14.2.2 Dictionary MediaStreamEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-mediastreameventinit-members)
	+ [14.3 DataChannelEvent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#datachannelevent)
		- [14.3.1 Attributes](http://dev.w3.org/2011/webrtc/editor/webrtc.html#attributes-7)
		- [14.3.2 Dictionary DataChannelEventInit Members](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dictionary-datachanneleventinit-members)
* [15. Event summary](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-summary)
* [16. Security Considerations](http://dev.w3.org/2011/webrtc/editor/webrtc.html#security-considerations)
* [17. 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. 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/).

2. 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)].

This specification defines conformance criteria that apply to a single product: the ***user agent*** that implements the interfaces that it contains.

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.

3. Terminology

The [EventHandler](http://dev.w3.org/html5/spec/webappapis.html#eventhandler) interface represents a callback used for event handlers as defined in [[HTML5](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-HTML5)].

The concepts [***queue a task***](http://dev.w3.org/html5/spec/webappapis.html#queue-a-task) and [***fires a simple event***](http://dev.w3.org/html5/spec/webappapis.html#fire-a-simple-event) are defined in [[HTML5](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-HTML5)].

The terms [***event handlers***](http://dev.w3.org/html5/spec/webappapis.html#event-handlers) and [***event handler event types***](http://dev.w3.org/html5/spec/webappapis.html#event-handler-event-type) are defined in [[HTML5](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-HTML5)].

4. Network Stream API

4.1 Introduction

The MediaStream interface, as defined in the [[GETUSERMEDIA](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-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. A peer is defined as a user agent that supports this specification.

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

4.2 Interface definitions

NOTE

In this section, we only specify aspects of the following objects that are relevant when used along with a [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection). 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.

**4.2.1 MediaStream**

*4.2.1.1 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 [RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection) 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.

NOTE

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 agent to a remote peer using [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection),and then sent back to the original user agent in the same manner, in which case the original user agent will have multiple streams with the same label (the locally-generated one and the one received from the remote peer).

*4.2.1.2 Events on MediaStream*

A new media track may be associated with an existing MediaStream. For example, if a remote peer adds a new MediaStreamTrack object to one of the track lists of a MediaStream that is being sent over a**[RTCPeerConnection](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22idl-def-RTCPeerConnection)**, this is observed on the local user agent. 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 aMediaStreamTrackList 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/getusermedia.html#dom-mediastreamtrack-kind) attribute equals "audio", add it to the MediaStream object’s [audioTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-audiotracks) MediaStreamTrackList object.
3. If *track’s* [kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind) attribute equals "video", add it to the MediaStream object’s [videoTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-videotracks) MediaStreamTrackList object.
4. Fire a track event named [addtrack](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtracklist-addtrack) with the newly created *track* at the MediaStreamTrackList object.

An existing media track 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 named [removetrack](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#event-mediastreamtracklist-removetrack) with *track* at the MediaStreamTrackList object.

The event source for the onended event in the networked case is the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object.

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

ISSUE 1

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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) , 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#event-mediastreamtrack-muted) (1) until media data arrives.

In addition, a MediaStreamTrack has its readyState set to muted on the remote peer if the local user agent disables the corresponding MediaStreamTrack in the MediaStream that is being sent. When the addstream event triggers on a [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection), all MediaStreamTrack objects in the resulting MediaStream are muted until media data can be read from the RTP source.

ISSUE 2

ISSUE: How do you know when it has been disabled? This seems like an SDP question, not a media-levelquestion.

4.3 AudioMediaStreamTrack

NOTE

The DTMF API is having a bunch of list discussion and will probably change.

The [**AudioMediaStreamTrack**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-AudioMediaStreamTrack) is a specialization 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*);

};

**4.3.1 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.

**4.3.2 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 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.

ISSUE 3

ISSUE: How are invalid values handled?

If insertDTMF is called on the same object while an existing task for this object to 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 sent.

NOTE

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.

NOTE

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

5. Peer-to-peer connections

A [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)(*configuration* ) creates a [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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 RTCPeerConnection object has an associated ***ICE agent***, ***RTCPeerConnection readiness state***, and ICE state. These are initialized when the object is created.

When the ***RTCPeerConnection()*** 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 [RTCPeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22rtcpeerconnection-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 RTCPeerConnection object gets more information, it can adjust the number of components.
2. Set *connection*’s [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) to new.
3. Set *connection*’s [RTCPeerConnection ice state](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22rtcpeerconnection-readiness-state) to new.
4. Let *connection*’s [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-localStreams) attribute be an empty read-only MediaStream array.
5. Let *connection*’s [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-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 RTCPeerConnection object, the following procedures are followed:

1. If *iceState* is "new" and the IceTransports constraint is not set to "none", it must queue a task to start gathering ICE address and set the *iceState* to "gathering".
2. If the ICE Agent has found one or more candidate pairs for any MediaStreamTrack 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 MediaStreamTrack, the *iceState* is changed to "completed" and if no connection has been found for any MediaStreamTrack, the iceState is changed to "failed".

ISSUE 4

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?

1. If the *iceState* is "connected" or "completed" and both the local and remote session descriptions are set, the RTCPeerConnection state is set to "active".
2. If the *iceState* is "failed", a task is queued to calls the close method.

ISSUE 5

ISSUE:: CJ - this seems wrong to me.

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

NOTE

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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) expecting this media.
2. Create a MediaStream object to represent the media stream.
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/getusermedia.html#dom-mediastreamtrack-kind) attribute equals "audio", add it to the MediaStream object's [audioTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-audiotracks) MediaStreamTrackList object.
	3. If *track's* [kind](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastreamtrack-kind) attribute equals "video", add it to the MediaStream object's [videoTracks](http://dev.w3.org/2011/webrtc/editor/getusermedia.html#dom-mediastream-videotracks) MediaStreamTrackList object.

NOTE

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

NOTE

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 [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is 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-RTCPeerConnection-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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) finds that a stream from the remote peer has been removed, the user agent must follow these steps:

1. Let *connection* be the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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.

NOTE

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 [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is closed (3), abort these steps.
	2. Remove *stream* from *connection*’s [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object to need to initiate a new session description negotiation, an [negotiationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-negotiation) event is fired at the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection)object.

In particular, if a [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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/getusermedia.html#dom-mediastreamtracklist-add) method being invoked, the[**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object must fire the "negotiationneeded" event. Removal of media components must also trigger "negotiationneeded".

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.

5.1 RTCPeerConnection

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

**5.1.1 RTCSdpType**

The RTCSdpType enum describes the type of a [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) instance.

enum **RTCSdpType** {

 "offer",

 "pranswer",

 "answer"

};

|  |
| --- |
| **Enumeration description** |
| offer | An RTCSdpType 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 | An RTCSdpType 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 | An RTCSdpType 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". |

**5.1.2 RTCSessionDescription Class**

The ***RTCSessionDescription()*** constructor takes an optional dictionary argument, *descriptionInitDict*, whose content is used to initialize the new [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) object. If a dictionary key is not present in *descriptionInitDict*, the corresponding attribute will be initialized to null. If the constructor is run without the dictionary argument, all attributes will be initialized to null. This class is a future extensible carrier for the data contained in it and does not perform any substantive processing.

Objects implementing the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) interface must serialize with the serialization pattern "{ attribute }".

[Constructor (optional RTCSessionDescriptionInit descriptionInitDict)]

interface **RTCSessionDescription** {

 attribute [**RTCSdpType**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSdpType)? [type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCSessionDescription-type);

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

};
dictionary **RTCSessionDescriptionInit** {

 [**RTCSdpType**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSdpType) [type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCSessionDescriptionInit-type);

 DOMString [sdp](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22widl-RTCSessionDescriptionInit-sdp);

};

*5.1.2.1 Attributes*

**sdp** of type DOMString, nullable

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

**type** of type [*RTCSdpType*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSdpType), nullable

What type of SDP this RTCSessionDescription represents.

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

**sdp** of type DOMString

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

DOMString sdp

**5.1.3 RTCSessionDescriptionCallback**

callback **RTCSessionDescriptionCallback** = void ([**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) *sdp*);

*5.1.3.1 Callback*[***RTCSessionDescriptionCallback***](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback)*Parameters*

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

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

**5.1.4 RTCStatsCallback**

callback **RTCStatsCallback** = void ([**RTCStatsElement**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsElement)[] *statsElements*, MediaStreamTrack? *selector*);

*5.1.4.1 Callback*[***RTCStatsCallback***](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsCallback)*Parameters*

**statsElements of type array of**[**RTCStatsElement**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsElement)

The objects containing the stats result.

**selector of type MediaStreamTrack, nullable**

The selector object that the statistics was gathered for. Currently only MediaStreamTrack supported.

**5.1.5 RTCVoidCallback**

callback **RTCVoidCallback** = void ();

**5.1.6 RTCPeerConnectionErrorCallback**

callback **RTCPeerConnectionErrorCallback** = void (DOMString *errorInformation*);

*5.1.6.1 Callback*[***RTCPeerConnectionErrorCallback***](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback)*Parameters*

**errorInformation of type DOMString**

Information about what went wrong.

ISSUE 6

ISSUE: Should this be an enum?

**5.1.7 RTCPeerState Enum**

enum **RTCPeerState** {

 "new",

 "have-local-offer",

 "have-local-pranswer",

 "have-remote-pranswer",

 "active" (also could be called "open", "stable")",

 "closed"

};

|  |
| --- |
| **Enumeration description** |
| new | The object was just created, and no networking has yet occurred. |
| have-local-offer | A local description, of type "offer", has been supplied. |
| have-local-pranswer | A remote description of type "offer" has been supplied and a local description of type "pranswer" has been supplied. |
| have-remote-pranswer | A local description of type "offer" has been supplied and a remote description of type "pranswer" has been supplied. |
| active" (also could be called "open", "stable") | Both local and remote descriptions have been supplied, and the offer-answer exchange is complete. |
| closed | The connection is closed. |

The non normative peer state transitions are:

An example set of transitions might be:

Caller transition:

* new PeerConnection(): new
* setLocal(offer): have-local-offer
* setRemote(pranswer): have-remote-pranswer
* setRemote(answer): active
* close(): closed

Callee transition:

* new PeerConnection(): new
* setRemote(offer): received-offer
* setLocal(pranswer): have-local-pranswer
* setLocal(answer): active
* close(): closed

**5.1.8 RTCGatheringState Enum**

enum **RTCGatheringState** {

 "new",

 "gathering",

 "complete"

};

|  |
| --- |
| **Enumeration description** |
| new | The object was just created, and no networking has occurred yet. |
| gathering | The ICE engine is in the process of gathering candidates for this RTCPeerConnection. |
| complete | The ICE engine has completed gathering. Events such as adding a new interface or new TURN server could cause that state to go back to gathering. |

**5.1.9 RTCIceState Enum**

NOTE

There is active discussion around changing these states.

enum **RTCIceState** {

 "starting",

 "checking",

 "connected",

 "completed",

 "failed",

 "disconnected",

 "closed"

};

|  |
| --- |
| **Enumeration description** |
| starting | The ICE Agent is gathering addresses and/or waiting for remote candidates to be supplied. |
| checking | The ICE Agent has received remote candidates on at least one component, and is checking candidate pairs but has not yet found a connection. In addition to checking, it may also still be gathering. |
| connected | The ICE Agent has found a usable connection for all components but is still checking other candidate pairs to see if there is a better connection. It may also still be gathering. |
| completed | The ICE Agent has finished gathering and checking and found a connection for all components. |
| failed | The ICE Agent is finished checking all candidate pairs and failed to find a connection for at least one component. |
| disconnected | Liveness checks have failed for one or more components. This is more aggressive than failed, and may trigger intermittently (and resolve itself without action) on a flaky network. |
| closed | The ICE Agent has shut down and is no longer responding to STUN requests. |

States take either the value of any component or all components, as outlined below:

* checking occurs if ANY component has received a candidate and can start checking
* connected occurs if ALL components have established a working connection
* completed occurs if ALL components have finalized the running of their ICE process
* failed occurs if ANY component has given up trying to connect
* disconnected occurs if ANY component has failed liveness checks
* closed occurs only if PeerConnection.close() has been called.

NOTE

The WG is discussing if starting/checking should be one state or two.

If a component is discarded as a result of signaling (e.g. RTCP mux or BUNDLE), the state may advance directly from checking to completed.

An example transition might look like:

* new PeerConnection(): Starting
* (Starting, remote candidates received): Checking
* (Checking, found usable connection): Connected
* (Checking, gave up): Failed
* (Connected, finished all checks): Completed
* (Completed, lost connectivity): Disconnected
* (any state, ICE restart occurs): Starting
* close(): Closed

The non normative ICE state transitions are:

**5.1.10 RTCIceCandidate Type**

The ***RTCIceCandidate()*** constructor takes an optional dictionary argument, *candidateInitDict*, whose content is used to initialize the new [**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) object. If a dictionary key is not present in*candidateInitDict*, the corresponding attribute will be initialized to null. If the constructor is run without the dictionary argument, all attributes will be initialized to null. This class is a future extensible carrier for the data contained in it and does not perform any substantive processing.

Objects implementing the [**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) interface must serialize with the serialization pattern "{ attribute }".

[Constructor (optional RTCIceCandidateInit candidateInitDict)]

interface **RTCIceCandidate** {

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

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

 attribute unsigned short? [sdpMLineIndex](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCIceCandidate-sdpMLineIndex);

};
dictionary **RTCIceCandidateInit** {

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

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

 unsigned short [sdpMLineIndex](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCIceCandidateInit-sdpMLineIndex);

};

*5.1.10.1 Attributes*

**candidate** of type DOMString, nullable

This carries the candidate-attribute as defined in section 15.1 of [[ICE](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-ICE)].

**sdpMLineIndex** of type unsigned short, nullable

This indicates the index (starting at zero) of m-line in the SDP this candidate is associated with.

**sdpMid** of type DOMString, nullable

If present, this contains the identifier of the "media stream identification" as defined in [RFC 3388] for m-line this candidate is associated with.

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

**candidate** of type DOMString

DOMString sdpMid

**sdpMLineIndex** of type unsigned short

**sdpMid** of type DOMString

unsigned short sdpMLineIndex

**5.1.11 RTCIceServer Type**

dictionary **RTCIceServer** {

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

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

};

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

**credential** of type nullable DOMString

If the url element of the internal array is TURN URI, then this is the credential to use with that TURN server.

**url** of type DOMString

A stun or turn URI 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)].

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 array of RTCIceServer objects is:

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

**5.1.12 RTCConfiguration Type**

dictionary **RTCConfiguration** {

 [**RTCIceServer**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceServer)[] [iceServers](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCConfiguration-iceServers);

};

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

**iceServers** of type array of [*RTCIceServer*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceServer)

An array of containing the STUN and TURN servers provided by the JS that can be used by ICE.

**5.1.13 RTCIdentityAssertion Type**

dictionary **RTCIdentityAssertion** {

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

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

};

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

**idp** of type DOMString

The identity provider, identified as a domain name.

**name** of type DOMString

An RFC822-conformant [TODO: REF] representation of the verified peer identity. This identity will have been verified via the procedures described in [RTCWEB-SECURITY-ARCH].

**5.1.14 RTCStatsElement dictionary**

Each RTCStatsElement object consists of two RTCStatsReport objects, one corresponding to local stats and one to remote stats.

dictionary **RTCStatsElement** {

 [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) [local](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCStatsElement-local);

 [**RTCStatsReport**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsReport) [remote](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCStatsElement-remote);

};

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

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

The stats corresponding to local properties.

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

The stats corresponding to remote properties.

**5.1.15 RTCStatsReport Type**

Each RTCStatsReport has a timestamp. Individual statistics are accessed by passing string names to the [getValue()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-rtcstatsreport-getvalue) method. Note that while stats names are standardized [[OPEN ISSUE: Need to define an IANA registry for this and populate with pointers to existing things such as the RTCP statistics.]], any given implementation may be using experimental values or values not yet known to the Web application. Thus, applications must be prepared to deal with unknown stats.

Stats need to be synchronized with each other in order to yield reasonable values in computation; for instance, if “bytesSent” and “packetsSent” are both reported, they both need to be reported over the same interval, so that “average packet size” can be computed as “bytes / packets” - if the intervals are different, this will yield errors. Thus implementations must return synchronized values for all stats in a RTCStatsReport.

interface **RTCStatsReport** {

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

 any [getValue](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCStatsReport-getValue-any-DOMString-statName) (DOMString *statName*);

};

*5.1.15.1 Attributes*

**timestamp** of type long, readonly

The timestamp in milliseconds since the UNIX epoch (Jan 1, 1970, UTC).

*5.1.15.2 Methods*

**getValue**

The ***getValue()*** method returns the value for the statistic that corresponds to *statName*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| statName | DOMString | ✘ | ✘ |  |

*Return type:*any

**5.1.16 RTCPeerConnection Interface**

typedef MediaStream[] MediaStreamArray;

[Constructor (RTCConfiguration configuration, optional MediaConstraints

 constraints)]

interface **RTCPeerConnection** : *EventTarget*  {

 void [createOffer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-createOffer-void-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback-MediaConstraints-constraints) ([**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) *successCallback*, optional [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*, optional MediaConstraints *constraints*);

 void [createAnswer](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-createAnswer-void-RTCSessionDescriptionCallback-successCallback-RTCPeerConnectionErrorCallback--failureCallback---null-MediaConstraints-constraints---null) ([**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) *successCallback*, optional RTCPeerConnectionErrorCallback? failureCallback = *null*, optional MediaConstraints constraints = *null*);

 void [setLocalDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-setLocalDescription-void-RTCSessionDescription-description-RTCVoidCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) *description*, optional [**RTCVoidCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCVoidCallback) *successCallback*, optional [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

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

 void [setRemoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-setRemoteDescription-void-RTCSessionDescription-description-RTCVoidCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback) ([**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) *description*, optional [**RTCVoidCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCVoidCallback) *successCallback*, optional [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

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

 readonly attribute [**RTCPeerState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerState) [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-readyState);

 void [updateIce](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-updateIce-void-RTCConfiguration--configuration---null-MediaConstraints--constraints---null) (optional RTCConfiguration? configuration = *null*, optional MediaConstraints? constraints = *null*);

 void [addIceCandidate](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-addIceCandidate-void-RTCIceCandidate-candidate) ([**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) *candidate*);

 readonly attribute [**RTCGatheringState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCGatheringState) [iceGatheringState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-iceGatheringState);

 readonly attribute [**RTCIceState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceState) [iceState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-iceState);

 readonly attribute [**MediaStreamArray**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamArray) [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-localStreams);

 readonly attribute [**MediaStreamArray**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamArray) [remoteStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-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-RTCPeerConnection-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 EventHandler [ondatachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-ondatachannel);

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

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

 void [setIdentityProvider](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-setIdentityProvider-void-DOMString-provider-DOMString-protocol-optional-DOMString-username) (DOMString *provider*, optional DOMString *protocol*, optional optional DOMString *username*);

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

 readonly attribute [**RTCIdentityAssertion**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIdentityAssertion)? [peerIdentity](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-peerIdentity);

 void [getStats](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-getStats-void-MediaStreamTrack-selector-RTCStatsCallback-successCallback-RTCPeerConnectionErrorCallback-failureCallback) (MediaStreamTrack? *selector*, [**RTCStatsCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsCallback) *successCallback*, optional [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) *failureCallback*);

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

 attribute EventHandler [onnegotationneeded](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onnegotationneeded);

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

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

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

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

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

 attribute EventHandler [ongatheringchange](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-ongatheringchange);

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

 attribute EventHandler [onidentityresult](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-onidentityresult);

};

*Attributes*

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

The ***iceGatheringState*** attribute must return the gathering state of the [RTCPeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-ice-agent) connection state.

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

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

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

The ***localDescription*** attribute must return the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) that was most recently passed to [setLocalDescription()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-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 [*MediaStreamArray*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamArray), 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 EventHandler

This event handler, of event handler event type [addstream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-addstream), must be fired by all objects implementing the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) interface It is called any time a MediaStream is added by the remote peer. This will be fired only as a result of setRemoteDescription. Onnaddstream happens as early as possible after the setRemoteDescription. This callback does not wait for a given media stream to be accepted or rejected via SDP negotiation. Later, when the SDP accepts something, you get the addTrack callback. Later if SDP ended a media flow, that would result in trackEnded callback.

**ondatachannel** of type EventHandler

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

**ongatheringchange** of type EventHandler

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

**onicecandidate** of type EventHandler

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

**onicechange** of type EventHandler

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

**onidentityresult** of type EventHandler

This event handler, of event handler event type [identityresult](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-identityresult), must be fired by all objects implementing the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) interface. It is called any time an identity verification succeeds or fails.

**onnegotationneeded** of type EventHandler

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

**onopen** of type EventHandler

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

NOTE

Open issue if the "onopen" is needed or not.

**onremovestream** of type EventHandler

This event handler, of event handler event type [removestream](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-mediastream-removestream), must be fired by all objects implementing the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) interface. It is called any time a MediaStream is removed by the remote peer. This will be fired only as a result of setRemoteDescription.

**onstatechange** of type EventHandler

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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) interface. It is called any time the readyState changes, i.e., from a call to setLocalDescription, setRemoteDescription, or code. It does not fire for the initial state change into new

**peerIdentity** of type [*RTCIdentityAssertion*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIdentityAssertion), readonly, nullable

Contains the peer identity assertion information if an identity assertion was provided and verified.

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

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

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

The ***remoteDescription*** attribute must return the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) that was most recently passed to [setRemoteDescription()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-setremotedescription), plus any remote candidates that have been supplied via[addIceCandidate()](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-addicecandidate) since then.

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

**remoteStreams** of type [*MediaStreamArray*](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-MediaStreamArray), 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.

*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 | [**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) | ✘ | ✘ |  |

*Return type:*void

**addStream**

Adds a new stream to the RTCPeerConnection.

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

1. If the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is closed (3), throw an INVALID\_STATE\_ERR exception.
2. If *stream* is already in the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-localStreams) object, then abort these steps.
3. Add *stream* to the end of the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [localStreams](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-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 negotiationneeded event.

ISSUE 8

ISSUE: Should this fire if the RTCPeerConnection 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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html%22%20%5Cl%20%22rtcpeerconnection-readiness-state) is closed (3), throw an INVALID\_STATE\_ERR exception.
2. Destroy the [RTCPeerConnection ICE Agent](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-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 [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) to closed (3).

*No parameters.*

*Return type:*void

**createAnswer**

The createAnswer method generates an [[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 in the remote configuration. Like createOffer, the returned blob contains descriptions of the local MediaStreams attached to this RTCPeerConnection, 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 corresponding offer, specifies how the media plane should be established. The generation of the SDP must follow the appropriate process for generating an 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 needed to get the ICE user name fragment and password.

An answer can be marked as provisional, as described in [[RTCWEB-JSEP](http://dev.w3.org/2011/webrtc/editor/webrtc.html#bib-RTCWEB-JSEP)], by setting the [type](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCSessionDescription-type) to "pranswer".

If the PeerConnection is configured to generate Identity assertions, then the session description shall contain an appropriate assertion.

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| successCallback | [**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) | ✘ | ✘ |  |
| null | RTCPeerConnectionErrorCallback? failureCallback = | ✘ | ✔ |  |
| null | MediaConstraints constraints = | ✘ | ✔ |  |

*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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is 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 3264 offer with the supported configurations for the session, including descriptions of the local MediaStreams attached to this RTCPeerConnection, 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.

If the PeerConnection is configured to generate Identity assertions, then the session description shall contain an appropriate assertion.

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

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

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 | [**RTCSessionDescriptionCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescriptionCallback) | ✘ | ✘ |  |
| failureCallback | [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |
| constraints | MediaConstraints | ✘ | ✔ |  |

*Return type:*void

**getIdentityAssertion**

Initiates the process of obtaining an identity assertion. Applications need not make this call. It is merely intended to allow them to start the process of obtaining identity assertions before a call is initiated. If an identity is needed, either because the browser has been configured with a default identity provider or because ***setidentityprovider()*** method was called, then an identity will be automatically requested when an offer or answer is created.

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%22%20%5Cl%20%22widl-PeerConnection-CLOSED) (3), abort these steps.
2. Instantiate a new IdP proxy and request an identity assertion.

*No parameters.*

*Return type:*void

**getStats**

When the ***getStats()*** method is invoked, the user agent must queue a task to run the following substeps:

1. If the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object's [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is closed (3), throw an INVALID\_STATE\_ERR exception.
2. Gather the stats indicated by the selector. If the selector is invalid, call the failureCallback.
3. Call the successCallback, supplying the relevant statistics object.

The “selector” may be a MediaStreamTrack that is a member of a MediaStream on the incoming or outgoing streams. The callback reports on all relevant statistics for that selector. If the selector is blank or missing, stats for the whole PeerConnection are reported. TODO: Evaluate the need for other selectors than MediaStreamTrack.

The returned structure contains a list of StatsElements, each reporting stats for one object that the implementation thinks is relevant for the selector. One achieves the total for the selector by summing over all the elements; for instance, if a MediaStreamTrack is carried by multiple SSRCs over the network, the getStats function may return one StatsElement per SSRC (which can be distinguished by the value of the “ssrc” stats attribute).

A PC must return consistent stats for each element in the array, adding new elements to the end as needed; this is needed so that an application can simply correlate a value read at one moment to a value read at a later moment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| selector | MediaStreamTrack | ✔ | ✘ |  |
| successCallback | [**RTCStatsCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCStatsCallback) | ✘ | ✘ |  |
| failureCallback | [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |

*Return type:*void

**removeStream**

Removes the given stream from the localStream array in the RTCPeerConnection and fires negotiationneeded.

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

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

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

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

*Return type:*void

**setIdentityProvider**

Sets the identity provider to be used for a given PeerConnection object. Applications need not make this call; if the browser is already configured for an IdP, then that configured IdP will be used to get an assertion.

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

1. Set the current identity values to the triplet provider, protocol, username.
2. If the PeerConnection object's [PeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#peerconnection-readiness-state) is [active](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-PeerConnection-ACTIVE), and any of the identity settings have changed, 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%22%20%5Cl%20%22widl-PeerConnection-CLOSED) (3), abort these steps.
	2. Instantiate a new IdP proxy and request an identity assertion.
	3. If/when the assertion is obtained, fire a negotiationneeded event.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| provider | DOMString | ✘ | ✘ |  |
| protocol | DOMString | ✘ | ✔ |  |
| username | optional DOMString | ✘ | ✔ |  |

*Return type:*void

**setLocalDescription**

The ***setLocalDescription()*** method instructs the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) to apply the supplied [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) as the local description.

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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) can fully adopt the new local description, or roll back to the old description if the remote side denied the change.

ISSUE 7

ISSUE: 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. [localDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-localdescription) must return the previous description until the new description is successfully applied.

The failureCallback will be called if the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) is a valid description but cannot be applied at the media layer, e.g., if there are insufficient resources to apply the SDP. The user agent must roll back as necessary if the new description was partially applied when the failure occurred.

A TBD exception is thrown if the SDP content is invalid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| description | [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) | ✘ | ✘ |  |
| successCallback | [**RTCVoidCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCVoidCallback) | ✘ | ✔ |  |
| failureCallback | [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |

*Return type:*void

**setRemoteDescription**

The ***setRemoteDescription()*** method instructs the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) to apply the supplied [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) as the remote offer or answer. This API changes the local media state.

If a=identity attributes are present, the browser verifies the identity following the procedures in [XREF sec.identity-proxy-assertion-request].

Changes to the state of media transmission will occur when a final answer is successfully applied. [remoteDescription](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-peerconnection-remotedescription) must return the previous description until the new description is successfully applied.

The failureCallback will be called if the [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) is a valid description but cannot be applied at the media layer, e.g., if there are insufficient resources to apply the SDP. The user agent must roll back as necessary if the new description was partially applied when the failure occurred.

A TBD exception is thrown if the SDP content is invalid.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| description | [**RTCSessionDescription**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCSessionDescription) | ✘ | ✘ |  |
| successCallback | [**RTCVoidCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCVoidCallback) | ✘ | ✔ |  |
| failureCallback | [**RTCPeerConnectionErrorCallback**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionErrorCallback) | ✘ | ✔ |  |

*Return type:*void

**updateIce**

The updateIce method updates the ICE Agent process of gathering local candidates and pinging remote candidates. If there is a mandatory constraint called "IceTransports" it will control 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.

NOTE

This method was previously used to restart ICE. We should document the new procedure in the correct place.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Type | Nullable | Optional | Description |
| null | RTCConfiguration? configuration = | ✘ | ✔ |  |
| null | MediaConstraints? constraints = | ✘ | ✔ |  |

*Return type:*void

**5.1.17 Garbage collection**

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

6. 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. 6.1 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.

ISSUE 9

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 RTCPeerConnection 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 RTCPeerConnection 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 RTCPeerConnection may wish to receive video but it is not going to send any video. The RTCPeerConnection needs to know if it should signal to the remote side if it wishes to receive video or not. This constraint 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 RTCPeerConnection may wish to receive audio but it is not going to send any audio. The RTCPeerConnection 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 their 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 constraint indicates which candidates the ICE engine is allowed to 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 use 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.

**RequestIdentity**

This is a enum type constraint that can take the values "yes", "no", and "ifconfigured". The default is a non mandatory "ifconfigured".

This constraint indicates whether an identity should be be requested. The constraint may be used with either of the createOffer(), createAnswer() calls or with the constructor.The value "yes" means that an identity must be requested. The value "no" means that no identity is to be requested. The value "ifconfigured" means that an identity will be requested if either the user has configured an identity in the browser or if the setIdentityProvider() call has been made by the JS. As this is the default value, an identity will be requested if and only if the user has configured an IdP in some way. Note that as long as DTLS-SRTP is in used, fingerprints will be sent regardless of the value of this constraint.

TODO items - need to register with IANA.

7. 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.

EXAMPLE 1

var signalingChannel = createSignalingChannel();

var pc;

var configuration = ...;

// run start(true) to initiate a call

function start(isCaller) {

 pc = new RTCPeerConnection(configuration);

 // send any ice candidates to the other peer

 pc.onicecandidate = function (evt) {

 signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));

 };

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

 pc.onaddstream = function (evt) {

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

 };

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

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

 selfView.src = URL.createObjectURL(stream);

 pc.addStream(stream);

 if (isCaller)

 pc.createOffer(gotDescription);

 else

 pc.createAnswer(gotDescription);

 function gotDescription(desc) {

 pc.setLocalDescription(desc);

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

 }

 });

}

signalingChannel.onmessage = function (evt) {

 if (!pc)

 start(false);

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

 if (signal.sdp)

 pc.setRemoteDescription(new RTCSessionDescription(signal.sdp));

 else

 pc.addIceCandidate(new RTCIceCandidate(signal.candidate));

};

8. Advanced Example

This example shows the more complete functionality.

EXAMPLE 2

TODO

9. Call Flow Browser to Browser

NOTE

Editor Note: This example flow needs to be discussed on the list and is likely wrong in many ways.

This shows an example of one possible call flow between two browsers. This does not show every callback that gets fired but instead tries to reduce it down to only show the key events and messages.

The following flow show a more complete set of the callbacks and events that happen.

10. Call Flow Browser to MCU

NOTE

Editor Note: This example flow needs to be discussed on the list and is likely wrong in many ways.

This shows an example of one possible call flow between a centralized conferencing server and a browser. This does not show every callback that gets fired but instead tries to reduce it down to only show the key events and messages.

11. Peer-to-peer Data API

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

ISSUE 10: More Open Issues

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

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

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 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 theDataChannel 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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is 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.
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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is 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.
8. Fire a datachannel event named [datachannel](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-peerconnection-datachannel) with *channel* at the [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) object’s [RTCPeerConnection readiness state](http://dev.w3.org/2011/webrtc/editor/webrtc.html#rtcpeerconnection-readiness-state) is closed, 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 or closed, then abort these steps.
4. Set *channel*’s [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) attribute to closing.
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).

NOTE

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 (3).
	2. Fire a simple event named [close](http://dev.w3.org/2011/webrtc/editor/webrtc.html#event-datachannel-close) at *channel*.

interface **DataChannel** : *EventTarget* {

 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);

 readonly attribute [**DataChannelState**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannelState) [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);

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

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

 attribute EventHandler [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) ();

 attribute EventHandler [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*);

};

**11.1.1 Attributes**

**binaryType** of type DOMString

NOTE

FIXME: align behavior with WebSocket API

**bufferedAmount** of type unsigned long, readonly

NOTE

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 EventHandler

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 EventHandler

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 EventHandler

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 EventHandler

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

**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.

**11.1.2 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**

NOTE

FIXME: align behavior with WebSocket API

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

*Return type:*void

**send**

NOTE

FIXME: align behavior with WebSocket API

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

*Return type:*void

**send**

NOTE

FIXME: align behavior with WebSocket API

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

*Return type:*void

dictionary **DataChannelInit** {

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

};

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

**reliable** of type boolean

FIXME: write description

enum **DataChannelState** {

 "connecting",

 "open",

 "closing",

 "closed"

};

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

11.2 Examples

This example shows how to create a [**DataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object and perform the offer/answer exchange required to connect the channel to the other peer. The [**DataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) is used in the context of a simple chat application and listeners are attached to monitor when the channel is ready, messages are received and when the channel is closed.

NOTE

This example uses the negotiationneeded event to initiate the offer/answer dialog. The exact behavior surrounding the negotiationneeded event is not specified in detail at the moment. This example can hopefully help to drive that discussion. An assumption made in this example is that the event only triggeres when a new negotiation should be started. This means that an action (such as addStream()) that normally would have fired the negotiationneeded event will not do so during an ongoing offer/answer dialog.

EXAMPLE 3

var signalingChannel = createSignalingChannel();

var pc;

var configuration = "...";

var channel;

// call start(true) to initiate

function start(isInitiator) {

 pc = new PeerConnection(configuration);

 // send any ice candidates to the other peer

 pc.onicecandidate = function (evt) {

 signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));

 };

 // let the "negotiationneeded" event trigger negotiation

 pc.onnegotiationneeded = function () {

 pc.createOffer(localDescCreated);

 }

 if (isInitiator) {

 // create data channel and setup chat

 channel = pc.createDataChannel("chat");

 setupChat();

 } else {

 // setup chat on incoming data channel

 pc.ondatachannel = function (evt) {

 channel = evt.channel;

 setupChat();

 };

 }

}

function localDescCreated(desc) {

 pc.setLocalDescription(desc, function () {

 signalingChannel.send(JSON.stringify({ "sdp": pc.localDescription }));

 });

}

signalingChannel.onmessage = function (evt) {

 if (!pc)

 start(false);

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

 if (message.sdp)

 pc.setRemoteDescription(new SessionDescription(message.sdp), function () {

 if (pc.remoteDescription.type == "offer")

 createAnswer(localDescCreated);

 });

 else

 pc.addIceCandidate(new IceCandidate(message.candidate));

};

function setupChat() {

 channel.onopen = function () {

 // e.g. enable send button

 enableChat(channel);

 };

 channel.onmessage = function (evt) {

 showChatMessage(evt.data);

 };

}

function sendChatMessage(msg) {

 channel.send(msg);

}

11.3 Garbage Collection

A [**DataChannel**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-DataChannel) object must not be garbage collected if its

* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) is connecting and at least one event listener is registered for open events, message events, error events, or close events.
* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) is open and at least one event listener is registered for message events, error events, or close events.
* [readyState](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dom-datachannel-readystate) is closing and at least one event listener is registered for error events, or close events.
* [underlying data transport](http://dev.w3.org/2011/webrtc/editor/webrtc.html#dfn-underlying-data-transport) is established and data is queued to be transmitted.

12. Statistics Model

The basic statistics model is that the browser maintains a set of statistics indexed by selector. The “selector” may be a MediaStreamTrack that is a member of a MediaStream on the incoming or outgoing streams. The calling Web application provides the selector to the getStats() method and the browser returns (in the JavaScript) a set of statistics that it believes is relevant to the selector.

The statistics returned are designed in such a way that repeated queries yield the same statistics in the same place in the structure. Thus, a Web application can make measurements over a given time period by requesting measurements at the beginning and end of that period.

12.1 Example

Consider the case where the user is experiencing bad sound and the application wants to determine if the cause is is it packet loss. The sound track is audio track 0 of remote stream 0 of pc1. The following example code might be used:

EXAMPLE 4

var baseline, now;

var selector = pc.remoteStreams[0].audioTracks[0];

pc.getStats(selector, function (stats) {

 baseline = stats;

});

// ... wait a bit

setTimeout(function () {

 pc.getStats(selector, function (stats) {

 now = stats;

 processStats();

 });

}, aByte);

function processStats() {

 // Real code would:

 // - Check that timestamp of “local stats” and “remote stats”

 // are reasonably consistent.

 // - Sum up over all the elements rather than just accessing

 // element zero.

 var packetsSent = now[0].remote.getValue("packetsSent") -

 baseline[0].remote.getValue("packetsSent");

 var packetsReceived = now[0].local.getValue("packetsReceived") -

 baseline[0].local.getValue("packetsReceived");

 // if fractionLost is > 0.3, we have probably found the culprit

 var fractionLost = (packetsSent - packetsReceived) / packetsSent;

}

13. Identity Provider Interaction

WebRTC offers and answers (and hence the channels established by PeerConnection objects) can be authenticated by using web-based Identity Providers. The idea is that the entity sending the offer/answer acts as the Authenticating Party (AP) and obtains an identity assertion from the IdP which it attaches to the offer/answer. The consumer of the offer/answer (i.e., the PeerConnection on which setRemoteDescription() is called acts as the Relying Party (RP) and verifies the assertion.

The interaction with the IdP is designed to decouple the browser from any particular identity provider; the browser need only know how to load the IdP's JavaScript--which is deterministic from the IdP's identity--and the generic protocol for requesting and verifying assertions. The IdP provides whatever logic is necessary to bridge the generic protocol to the IdP's specific requirements. Thus, a single browser can support any number of identity protocols, including being forward compatible with IdPs which did not exist at the time the browser was written. The generic protocol details are described in [RTCWEB-SECURITY-ARCH]. This document specifies the procedures required to instantiate the IdP proxy, request identity assertions, and consume the results.

13.1 Peer-Connection/IdP Communications

In order to communicate with the IdP, the browser must instantiate an isolated interpreted context [TODO: What's the technical term?], such as an invisible IFRAME. The initial contents of the context are loaded from a URI derived from the IdP's domain name. [RTCWEB-SECURITY-ARCH; Section XXX].

For purposes of generating assertions, the IdP shall be chosen as follows:

1. If the setIdentityProvider() method has been called, the IdP provided shall be used.
2. If the setIdentityProvider() method has not been called, then the browser shall use an IdP configured into the browser. If more than one such IdP is configured, the browser should provide the user with a chooser interface.

In order to verify assertions, the IdP domain name and protocol shall be equal to the "domain" and "protocol" fields of the identity assertion.

The context must have a MessageChannel named window.TBD which is "entangled" to the PeerConnection and is unique to that subcontext. This channel is used for messaging between the PeerConnection and the IdP. All messages sent via this channel are strings, specifically the JSONified versions of JS structs.

All messages sent from the PeerConnection to the IdP context must have an origin of rtcweb://peerconnection/. The fact that ordinary Web pages cannot set their origin values arbitrarily is an essential security feature, as it stops attackers from requesting WebRTC-compatible identity assertions from IdPs. For this reason, the origin must be included in the identity assertion and verified by the consuming PeerConnection.

13.2 Requesting Assertions

The identity assertion request process involves the following steps.

1. The PeerConnection instantiates an IdP context as described in the previous section.
2. The IdP serves up the IdP JS code to the IdP context.
3. Once the IdP is loaded and ready to receive messages it sends a "READY" message ([RTCWEB-SECURITY-ARCH; Section 5.6.5.2]. Note that this does not imply that the user is logged in, merely that enough IdP state is booted up to be ready to handle PostMessage calls.
4. The IdP sends a "SIGN" message (Section 5.6.5.2.2) to the IdP proxy context. This message includes the material the PeerConnection desires to be bound to the user's identity.
5. If the user is not logged in, at this point the IdP will initiate the login process. For instance, it might pop up a dialog box inviting the user to enter their (IdP) username and password.
6. Once the user is logged in (potentially after the previous step), the IdP proxy generates an identity assertion (depending on the authentication protocol this may involve interacting with the IDP server).
7. Once the assertion is generated, the IdP proxy sends a response (Section 5.6.5.2.2) containing the assertion to the PeerConnection over the message channel.
8. The PeerConnection stores the assertion for use with future offers or answers. If the identity request was triggered by a createOffer() or createAnswer(), then the assertion is inserted in the offer/answer.

13.3 Verifying Assertions

The identity assertion request process involves the following steps.

1. The PeerConnection instantiates an IdP context as described in the previous section.
2. The IdP serves up the IdP JS code to the IdP context.
3. Once the IdP is loaded and ready to receive messages it sends a "READY" message ([RTCWEB-SECURITY-ARCH; Section 5.6.5.2]. Note that this does not imply that the user is logged in, merely that enough IdP state is booted up to be ready to handle PostMessage calls.
4. The IdP sends a "VERIFY" message (Section 5.6.5.2.2) to the IdP proxy context. This message includes assertion from the offer/answer which is to be verified.
5. The IdP proxy verifies the identity assertion (depending on the authentication protocol this may involve interacting with the IDP server).
6. Once the assertion is verified the IdP proxy sends a response containing the verified assertion results (Section 5.6.5.2.3) to the PeerConnection over the message channel.
7. The PeerConnection displays the assertion information in the browser UI and stores the assertion in the [peerIdentity](http://dev.w3.org/2011/webrtc/editor/webrtc.html#widl-RTCPeerConnection-peerIdentity) attribute for availability to the JS application. The assertion information to be displayed shall contain the domain name of the IdP and the identity returned by the IdP and must be displayed via some mechanism which cannot be spoofed by content. [[OPEN ISSUE: The identity information should also be available in the inspector interface defined in [RTCWEB-SECURITY-ARCH; Section 5.5].

13.4 Examples

The identity system is designed so that applications need not take any special action in order for users to generate and verify identity assertions; if a user has configured an IdP into their browser, then the browser will automatically request generate assertions and the other side will automatically verify them and display the results. However, applications may with to exercise tighter control over the identity system as shown by the following examples.

This example shows how to configure the identity provider and protocol.

EXAMPLE 5

pc.setIdentityProvider("example.com", "default", "alice@example.com");

This example shows how to consume identity assertions inside a Web application.

EXAMPLE 6

pc.onidentityresult = function(result) {

 console.log("IdP= " + pc.peerIdentity.idp +

 " identity=" + pc.peerIdentity.name);

};

14. Event definitions

14.1 RTCPeerConnectionIceEvent

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

***Firing a***[***RTCPeerConnectionIceEvent***](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionIceEvent)***event named e*** with an [**RTCIceCandidate**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCIceCandidate) *candidate* 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 RTCPeerConnectionIceEvent interface with the candidate attribute set to the new ICE candidate must be created and dispatched at the given target.

[Constructor(DOMString type, RTCPeerConnectionIceEventInit

 eventInitDict)]

interface **RTCPeerConnectionIceEvent** : *Event* {

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

};
dictionary **RTCPeerConnectionIceEventInit** : *EventInit* {

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

};

**14.1.1 Attributes**

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

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

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

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

14.2 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);

};

**14.2.1 Attributes**

**stream** of type MediaStream, readonly, nullable

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

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

**stream** of type MediaStream

14.3 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);

};

**14.3.1 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.

**14.3.2 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)

15. Event summary

*This section is non-normative.*

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 [**RTCPeerConnection**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnection) 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-RTCPeerConnection-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-RTCPeerConnection-remoteStreams) array. |
| ***negotiationneeded*** | Event | The browser wishes to inform the application that session negotiation needs to be done at some point in the near future. |
| ***statechange*** | Event | TODO |
| ***icechange*** | Event | TODO |
| ***icecandidate*** | [**RTCPeerConnectionIceEvent**](http://dev.w3.org/2011/webrtc/editor/webrtc.html#idl-def-RTCPeerConnectionIceEvent) | TODO |
| ***identityresult*** | RTCIdentityEvent | TODO |

16. Security Considerations

TBD.

17. Change Log

This section will be removed before publication.

Changes since Aug 16, 2012

1. Replaced stringifier with serializer on RTCSessionDescription and RTCIceCandidate (used when JSON.stringify() is called).
2. Removed offer and createProvisionalAnswer arguments from the createAnswer() method.
3. Removed restart argument from the updateIce() method.
4. Made DataChannel an EventTarget
5. Updated simple PeerConnection example to match spec changes.
6. Added section about DataChannel garbage collection.
7. Added stuff for identity proxy.
8. Added stuff for stats.
9. Added stuff peer and ice state reporting.
10. Minor changes to sequence diagrams.
11. Added a more complete DataChannel example
12. Various fixes from Dan's Idp API review.
13. Patched the Stats API.

Changes since Aug 13, 2012

1. Made the RTCSessionDescription and RTCIceCandidate constructors take dictionaries instead of a strings. Also added detailed stringifier algorithm.
2. Went through the list of issues (issue numbers are only valid with HEAD at fcda53c460). Closed (fixed/wontfix): 1, 8, 10, 13, 14, 16, 18, 19, 22, 23, 24. Converted to notes: 4, 12. Updated: 9.
3. Incorporate [changes proposed](http://lists.w3.org/Archives/Public/www-archive/2012Aug/0015.html) by Li Li.
4. Use an enum for DataChannelState and fix IDLs where using an optional argument also requires all previous optional arguments to have a default value.

Changes since Jul 20, 2012

1. Added RTC Prefix to names (including the notes below).
2. Moved to new definition of configuration and ice servers object.
3. Added correlating lines to candidate structure.
4. Converted setLocalDescription and setRemoteDescription to be asynchronous.
5. Added call flows.

Changes since Jul 13, 2012

1. Removed peer attribute from RTCPeerConnectionIceEvent (duplicates functionality of Event.target attribute).
2. Removed RTCIceCandidateCallback (no longer used).
3. Removed RTCPeerConnectionEvent (we use a simple event instead).
4. Removed RTCSdpType argument from setLocalDescription() and setRemoteDescription(). Updated simple example to match.

Changes since May 28, 2012

1. Changed names to use RTC Prefix.
2. Changed the data structure used to pass in STUN and TURN servers in configuration.
3. Updated simple RTCPeerConnection example (RTCPeerConnection constructor arguments; use icecandidate event).
4. Initial import of new Data API.
5. Removed some left-overs from the old Data Stream API.
6. Renamed "underlying data channel" to "underlying data transport". Fixed closing procedures. Fixed some typos.

Changes since April 27, 2012

1. Major rewrite of RTCPeerConnection section to line up with IETF JSEP draft.
2. Added simple RTCPeerConnection example. Initial update of RTCSessionDescription and RTCIceCandidate to support serialization and construction.

Changes since 21 April 2012

1. Moved MediaStream and related definitions to getUserMedia.
2. Removed section "Obtaining local multimedia content".
3. Updated getUserMedia() calls in examples (changes in Media Capture TF spec).
4. Introduced MediaStreamTrackList interface with support for adding and removing tracks.
5. Updated the algorithm that is run when RTCPeerConnection receives a stream (create new stream when negotiated instead of when data arrives).

Changes since 12 January 2012

1. Clarified the relation of Stream, Track, and Channel.

Changes since 17 October 2011

1. Tweak the introduction text and add a reference to the IETF RTCWEB group.
2. Changed the first argument to getUserMedia to be an object.
3. Added a MediaStreamHints object as a second argument to RTCPeerConnection.addStream.
4. Added AudioMediaStreamTrack class and DTMF interface.

Changes since 23 August 2011

1. Separated the SDP and ICE Agent into separate agents and added explicit state attributes for each.
2. Removed the send method from PeerConenction and associated callback function.
3. Modified MediaStream() constructor to take a list of MediaStreamTrack objects instead of a MediaStream. Removed text about MediaStream parent and child relationship.
4. Added abstract.
5. Moved a few paragraphs from the MediaStreamTrack.label section to the MediaStream.label section (where they belong).
6. Split MediaStream.tracks into MediaStream.audioTracks and MediaStream.videoTracks.
7. Removed a sentence that implied that track access is limited to LocalMediaStream.
8. Updated a few getUserMedia()-examples to use MediaStreamOptions.
9. Replaced calls to URL.getObjectURL() with URL.createObjectURL() in example code.
10. Fixed some broken getUserMedia() links.
11. Introduced state handling on MediaStreamTrack (removed state handling from MediaStream).
12. Reintroduced onended on MediaStream to simplify checking if all tracks are ended.
13. Aligned the MediaStreamTrack ended event dispatching behavior with that of MediaStream.
14. Updated the LocalMediaStream.stop() algorithm to implicitly use the end track algorithm.
15. Replaced an occurrence the term finished track with ended track (to align with rest of spec).
16. Moved (and extended) the explanation about track references and media sources from LocalMediaStream to MediaStreamTrack.

A. Acknowledgements

The editors wish to thank the Working Group chairs and Team Contact, Harald Alvestrand, Stefan Håkansson and Dominique Hazaël-Massieux, for their support. Substantial text in this specification was provided by many people including Harald Alvestrand, Justin Uberti, and Eric Rescorla.

B. References

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>

**[HTML5]**

Ian Hickson; David Hyatt. [*HTML5.*](http://www.w3.org/TR/html5) 29 March 2012. W3C Working Draft. (Work in progress.) URL: <http://www.w3.org/TR/html5>

**[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/>

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/>