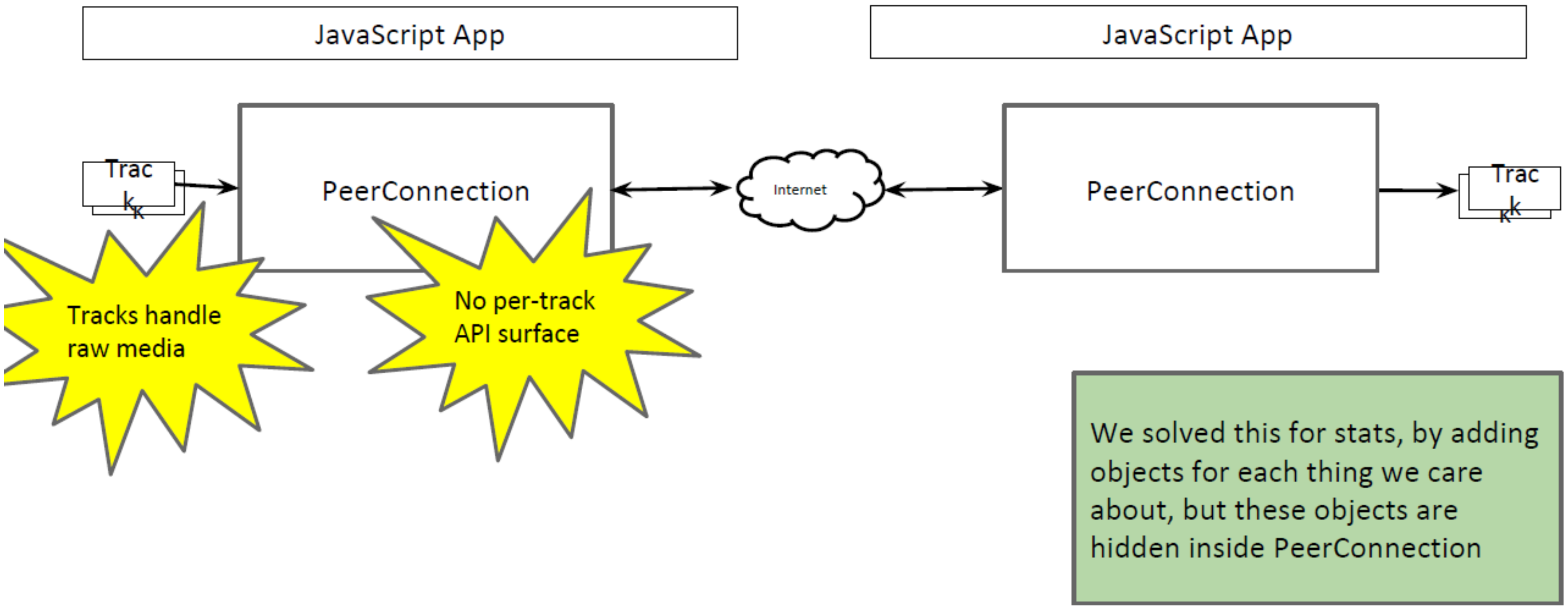
WebRTC 1.0 object model

Bernard Aboba
Principal Architect at Microsoft

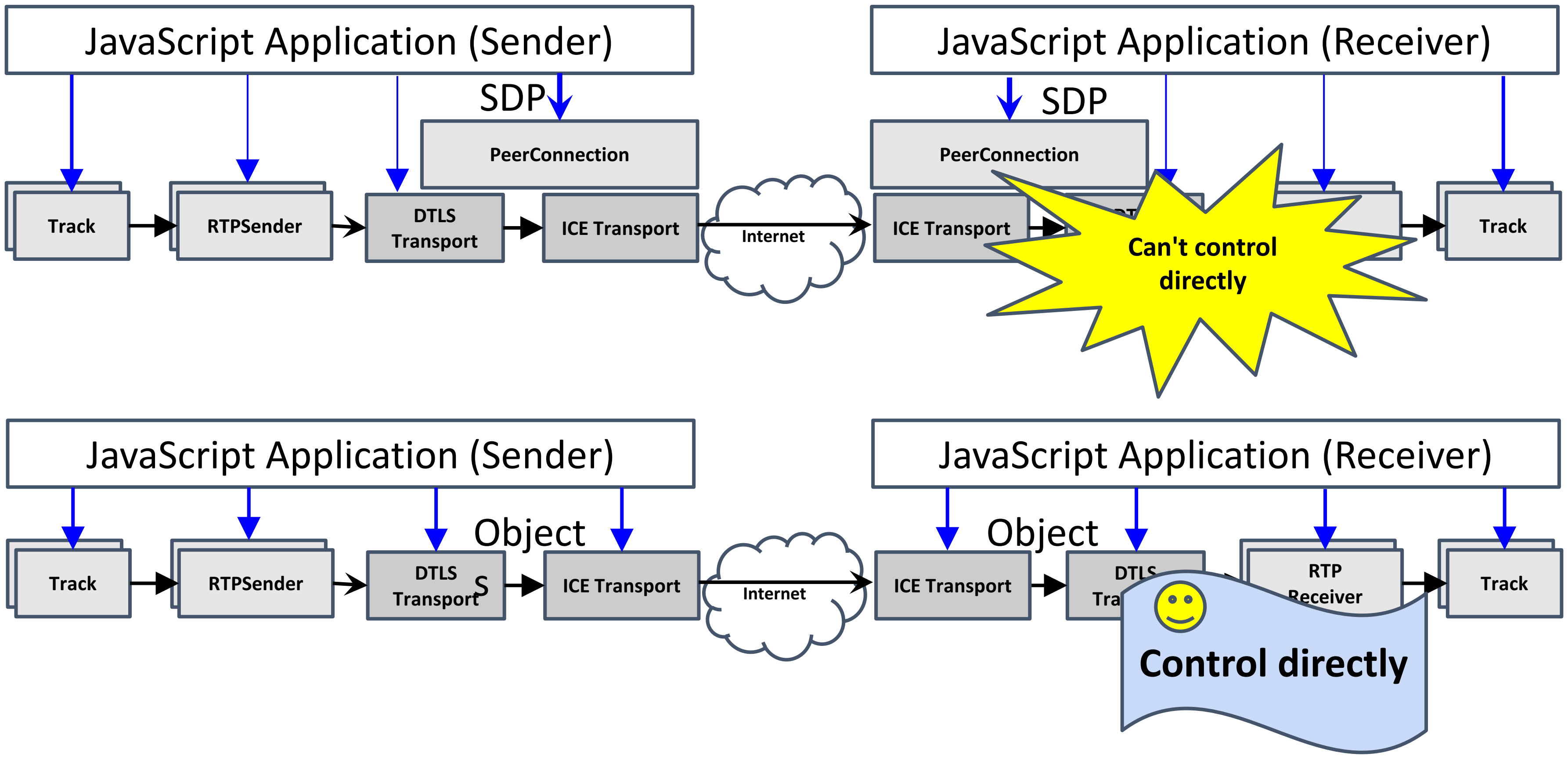
Robin Raymond
CTO at Optical Tone

Why is there an object model in WebRTC 1.0?

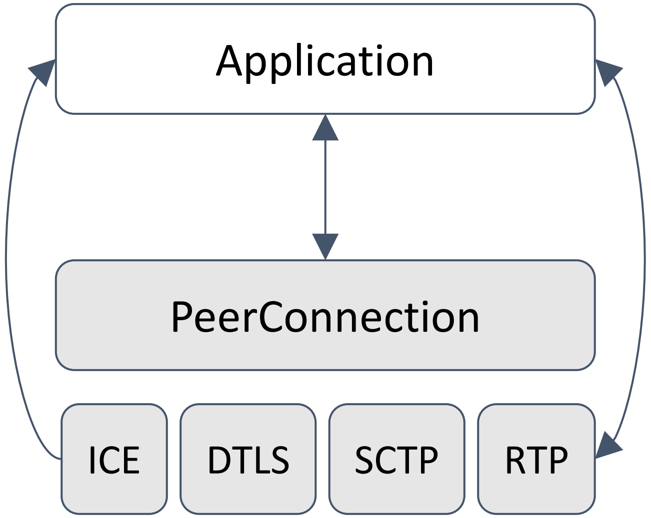


- Need a way to tweak parameters on individual tracks sent over the wire
  - Bitrate
  - Framerate
  - Direction (sendonly/recvonly etc.)
- Existing control surfaces insufficient:
  - createOffer params - not per-track
  - AddStream params - not modifiable post-add
  - MST constraints - affects raw media, not encoding

WebRTC 1.0 to ORTC

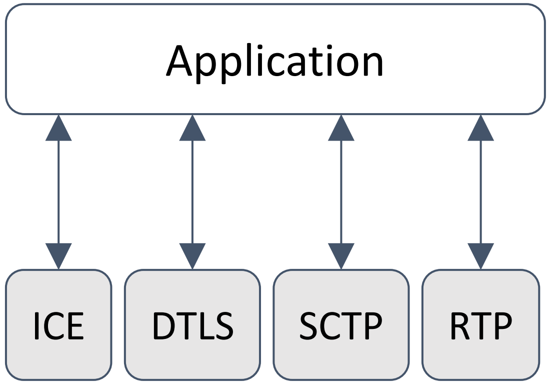


WebRTC now



Objects all read-only
Indirectly controlled via PeerConnection
Some direct control

ORTC



Same objects as WebRTC
Full direct control
PeerConnection optional (via JS lib)

WebRTC 1.0 spec additions

```
PeerConnection
      .getSenders()
      .getReceivers()
      .addTransceiver(kind)
      .sctp
      ...

RtpSender
      .track
      .transport
      .getCapabilities()
      .getParameters()
      .setParameters(params)
      .replaceTrack(track)
      ...

RtpReceiver
      .track
      .transport
      .getCapabilities()
      .getParameters()
      .getContributingSources()
      .getSynchronizationSources()

      ...
```

```
DtlsTransport
      .transport
      .state
      .getRemoteCertificates()
      .onstatechange
      ...
IceTransport
      .state
      .getLocalParameters(),
      .getRemoteParameters(),
      .getLocalCandidates()
      .getRemoteCandidates()
      .getSelectedCandidatePair()
      .onstatechange
      ...
SctpTransport
      .transport
DataChannel
      .transport
```

```
RtpParameters
  .codecs
  .encodings
  ...
RtpCodecParameters
  (read only)
  .mimeType
  .payloadType
  ...
RtpEncodingParameters
  .active
  .maxBitrate
  .maxFramerate
  .rid (read only)
  .resolutionScaleDownBy
  .ssrc (read only)
  ...
IceParameters
  (read only)
  .usernameFragment
  .password
DtlsParameters
  ...
```

Source: https://cdn.rawgit.com/w3c/webrtc-pc/master/webrtc.html

What you can do with WebRTC 1.0 objects

- "Warm up" media path while the getting a track and ringing
- Change the send codec (without SDP munging)
- Change the camera source instantly (front to back)
- Enable/disable sending of media instantly (without signalling)
- Set a maximum bitrate or maximum framerate
- Obtain detailed status of individual ICE and DTLS transports
- Send simulcast
- Receive simulcast (optional)

Example: Warmup

```
// Webrtc
  var audio = pc.addTransceiver("audio");
  var video = pc.addTransceiver("video");
  renderTrack(video.receiver.track);
  renderTrack(audio.receiver.track);
  // ... do getUserMedia and offer/answer
  // ... wait for "real answer"
  audio.sender.replaceTrack(audioTrack);
  video.sender.replaceTrack(videoTrack);
```

```
// ORTC
  var audioSender = new RTCRtpSender(...);
  var videoSender = new RTCRtpSender(...);
  var audioReceiver = new RTCRtpReceiver(...);
  var videoReceiver = new RTCRtpReceiver(...);
  renderTrack(video.receiver.track);
  renderTrack(audio.receiver.track);
  // ... do getUserMedia and call signalling
  // ... wait for "real answer"
  videoSender.setTrack(videoTrack);
  videoSender.send(...);
  audioSender.setTrack(audioTrack);
  audioSender.send(...);
```

Example: Change camera

```
// Webrtc
  var sender = pc.addTrack(video1);
  sender.replaceTrack(video2);
  sender.replaceTrack(video1);
```

```
// ORTC
  var videoSender = new RTCRtpSender(...);
  videoSender.setTrack(video2);
  videoSender.setTrack(video1);
```

Example: Change send codecs

```
// Webrtc
  var sender = pc.addTrack(...);
  // After every call to
  // setLocalDescription and
  // setRemoteDescription:
  var p = sender.getParameters();
  p.codecs = reorderCodecs(p.codecs);
  sender.setParameters(p);
```

```
// ORTC
  var sender = new RTCRtpSender(...);
  // Only once, but choose the PTs
  var codecs = reorderCodecs(
    sender.getCapabilities().codecs);
  sender.send({codecs: codecs, ...});
```

Example: Set max bitrate

```
// Webrtc
  var sender = pc.addTrack(...);
  var p = sender.getParameters();
  // bps
  p.encodings[0].maxBitrate = 1000000;
  sender.setParameters(p);
```

```
// ORTC
  var sender = new RTCRtpSender(...);
  sender.send({
    encodings: [{maxBitrate: 1000000,
  ...}]
  });
```

Example: Enable/disable media

```
// Webrtc

  var sender = pc.addTrack(...);
  var p = sender.getParameters();
  p.encodings[0].active = false;
  sender.setParameters(p);
  p.encodings[0].active = true;
  sender.setParameters(p);
```

```
// ORTC
  var sender = new RTCRtpSender(...);
  sender.send({
    encodings: [{active: false, ...}]
  });
  sender.send({
    encodings: [{active: true, ...}]
  });
```

WebRTC 1.0 Object Model

- RtpSender* (Section 5.2)
- RtpReceiver (Section 5.3)
- RtpTransceiver (Section 5.4)
- DtlsTransport (Section 5.5)
- IceTransport (Section 5.6)
- SctpTransport (Section 6.1.1)
- DataChannel (Section 6.2)
- DTMFSender (Section 7.2)

* all objects are prefixed with “RTC”

Differences from ORTC

Objects not in WebRTC 1.0 object model (or in Edge):

- IceGatherer
  - WebRTC 1.0: Gathering within IceTransport (no forking)
- IceTransportController
  - WebRTC 1.0: Freezing controlled by SDP m-lines.
- RtpListener
  - WebRTC 1.0: Unroutable packets dropped.
- QuicTransport
  - WebRTC 1.0: No stream abstraction

Objects in WebRTC 1.0 Object Model not in ORTC:

- RtpTransceiver

RTCPeerConnection Interface

WebIDL

```
[Constructor(optional RTCConfiguration configuration)]
interface RTCPeerConnection : EventTarget {
    Promise<RTCSessionDescriptionInit> createOffer(optional RTCOfferOptions options);
    Promise<RTCSessionDescriptionInit> createAnswer(optional RTCAnswerOptions options);
    Promise<void>                      setLocalDescription(RTCSessionDescriptionInit
description);
    readonly attribute RTCSessionDescription? localDescription;
    readonly attribute RTCSessionDescription? currentLocalDescription;
    readonly attribute RTCSessionDescription? pendingLocalDescription;
    Promise<void>                      setRemoteDescription(RTCSessionDescriptionInit
description);
    readonly attribute RTCSessionDescription? remoteDescription;
    readonly attribute RTCSessionDescription? currentRemoteDescription;
    readonly attribute RTCSessionDescription? pendingRemoteDescription;
    Promise<void>                      addIceCandidate((RTCIceCandidateInit or
RTCIceCandidate) candidate);
    readonly attribute RTCSignalingState      signalingState;
    readonly attribute RTCIceGatheringState   iceGatheringState;
    readonly attribute RTCIceConnectionState  iceConnectionState;
    readonly attribute RTCPeerConnectionState connectionState;
    readonly attribute boolean?               canTrickleIceCandidates;
    static sequence<RTCIceServer>      getDefaultIceServers();
    RTCConfiguration                   getConfiguration();
    void                               setConfiguration(RTCConfiguration configuration);
    void                               close();
             attribute EventHandler           onnegotiationneeded;
             attribute EventHandler           onicecandidate;
             attribute EventHandler           onicecandidateerror;
             attribute EventHandler           onsignalingstatechange;
             attribute EventHandler           oniceconnectionstatechange;
             attribute EventHandler           onicegatheringstatechange;
             attribute EventHandler           onconnectionstatechange;
};
```

RTCPeerConnection Interface Extensions

WebIDL

```
partial interface RTCPeerConnection {
    sequence<RTCRtpSender>      getSenders();
    sequence<RTCRtpReceiver>    getReceivers();
    sequence<RTCRtpTransceiver> getTransceivers();
    RTCRtpSender                addTrack(MediaStreamTrack track,
                                         MediaStream... streams);
    void                        removeTrack(RTCRtpSender sender);
    RTCRtpTransceiver           addTransceiver((MediaStreamTrack or DOMString)
trackOrKind,
                                               optional RTCRtpTransceiverInit init);
    attribute EventHandler ontrack;
};
```

- RtpTransceiver (combination of sender and receiver) “vended” by addTransceiver()
- RtpSender “vended” by addTrack (addStream deprecated)
- RtpReceiver returned by a track event (see next slide) (onaddstream deprecated)
- RtpTransceiver(s), RtpSender(s), RtpReceiver(s) can be retrieved via getTransceivers, getSenders, getReceivers

RtpTransceiver Object

- Since SDP is bi-directional, RtpTransceiver pairs the RtpSender and RtpReceiver objects sharing an m-line.
  - setDirection enables “hold” scenarios (“sendrecv”, “sendonly”, “recvonly” or “inactive”)
  - Note: addTrack () returns an RtpSender and track event returns an RtpReceiver.
    - Result: RtpSender and RtpReceiver may not both exist at a given time.
- RtpTransceiver objects “vended” by pc.addTransceiver().
- RtpReceiver object can be accessed via transceiver.receiver, RtpSender object via transceiver.sender.
- pc.getTransceivers() returns the set of transceivers.
- pc.getSenders() returns an RTCPeerConnection’s “set of senders”
- pc.getReceivers() returns an RTCPeerConnection’s “set of receivers”.

Transceiver creation options (RtpTransceiverInit)

WebIDL

```
dictionary RTCRtpTransceiverInit {
    RTCRtpTransceiverDirection                   direction = "sendrecv";
    sequence<MediaStream>                        streams = [];
    sequence<RTCRtpEncodingParameters>           sendEncodings = [];
};
```

WebIDL

```
enum RTCRtpTransceiverDirection {
    "sendrecv",
    "sendonly",
    "recvonly",
    "inactive"
};
```

Concepts

- direction attribute useful in “hold” scenarios (can be changed via transceiver.setDirection()
- sendEncodings useful in advanced video scenarios
  - Examples: simulcast, bandwidth limitation (more later)

Track Event

WebIDL

```
[Constructor(DOMString type, RTCTrackEventInit eventInitDict)]
interface RTCTrackEvent : Event {
    readonly attribute RTCRtpReceiver            receiver;
    readonly attribute MediaStreamTrack          track;
    readonly attribute FrozenArray<MediaStream>  streams;
    readonly attribute RTCRtpTransceiver         transceiver;
};
```

Concepts

- event.receiver provides the RtpReceiver.
- event.track provides the remote track (same as receiver.track)
- event.streams provides the streams that the track is part of
  - Specification unclear whether this is always present or not.
- event.transceiver provides the transceiver (there is no receiver.transceiver attribute).

WebRTC 1.0: RtpTransceiver Interface

WebIDL

```
interface RTCRtpTransceiver {
    readonly attribute DOMString?                    mid;
    [SameObject]
    readonly attribute RTCRtpSender                  sender;
    [SameObject]
    readonly attribute RTCRtpReceiver                receiver;
    readonly attribute boolean                       stopped;
    readonly attribute RTCRtpTransceiverDirection    direction;
    readonly attribute RTCRtpTransceiverDirection?   currentDirection;
    void setDirection(RTCRtpTransceiverDirection direction);
    void stop();
    void setCodecPreferences(sequence<RTCRtpCodecCapability> codecs);
};
```

Concepts

- Each transceiver corresponds to an SDP m-line (mid)
- Transceivers always have sender and receiver attributes (not nullable)
- Transceivers are stop()'d as a unit (e.g. no sender.stop or receiver.stop)
- Transceiver direction set via setDirection)
- Codec preferences can be changed without SDP negotiation via setCodecPreferences().

WebRTC 1.0: RtpSender Interface

WebIDL

```
interface RTCRtpSender {
    readonly attribute MediaStreamTrack? track;
    readonly attribute RTCDtlsTransport? transport;
    readonly attribute RTCDtlsTransport? rtcpTransport;
    // Feature at risk
    static RTCRtpCapabilities getCapabilities(DOMString kind);
    Promise<void>             setParameters(optional RTCRtpParameters parameters);
    RTCRtpParameters          getParameters();
    Promise<void>             replaceTrack(MediaStreamTrack? withTrack);
    Promise<RTCStatsReport> getStats();
};
```

Differences from ORTC

- getParameters() method used to retrieve *parameters*. No equivalent in ORTC (application can store *parameters* passed to send(*parameters*)).
- setParameters method used to set *parameters*.
- replaceTrack() method instead of setTrack().
- No setTransport() method.
- No stop() method (can call transceiver.stop()).

WebRTC 1.0: RtpReceiver Interface

WebIDL

```
interface RTCRtpReceiver {
    readonly attribute MediaStreamTrack  track;
    readonly attribute RTCDtlsTransport? transport;
    readonly attribute RTCDtlsTransport? rtcpTransport;
    // Feature at risk
    static RTCRtpCapabilities                 getCapabilities(DOMString kind);
    RTCRtpParameters                          getParameters();
    sequence<RTCRtpContributingSource>        getContributingSources();
    sequence<RTCRtpSynchronizationSource> getSynchronizationSources();
    Promise<RTCStatsReport>                   getStats();
};
```

Differences from ORTC

- getParameters() method used to retrieve *parameters*. No equivalent in ORTC (application can store *parameters* passed to receive(*parameters*)).
- No setTransport() method.
- No stop() method (can call transceiver.stop()).

WebRTC 1.0: RTCDtlsTransport Interface

WebIDL

```
interface RTCDtlsTransport : EventTarget {
    readonly attribute RTCIceTransport       transport;
    readonly attribute RTCDtlsTransportState state;
    sequence<ArrayBuffer> getRemoteCertificates();
             attribute EventHandler          onstatechange;
             attribute EventHandler          onerror;
};
```

Differences from ORTC

- No getLocalParameters, getRemoteParameters, start or stop methods (handled in SDP)

SctpTransport Interface

WebIDL

```
partial interface RTCPeerConnection {
    readonly attribute RTCSctpTransport? sctp;
    RTCDataChannel createDataChannel(USVString label,
                                     optional RTCDataChannelInit dataChannelDict);
              attribute EventHandler       ondatachannel;
};
```

WebIDL

```
interface RTCSctpTransport {
    readonly attribute RTCDtlsTransport transport;
    readonly attribute unsigned long    maxMessageSize;
};
```

Differences from ORTC

- pc.sctp versus DataChannel.transport (ORTC)
- No SctpTransport.state
- No sctp.getCapabilities or sctp.stop

WebRTC 1.0: RTCIceTransport Interface

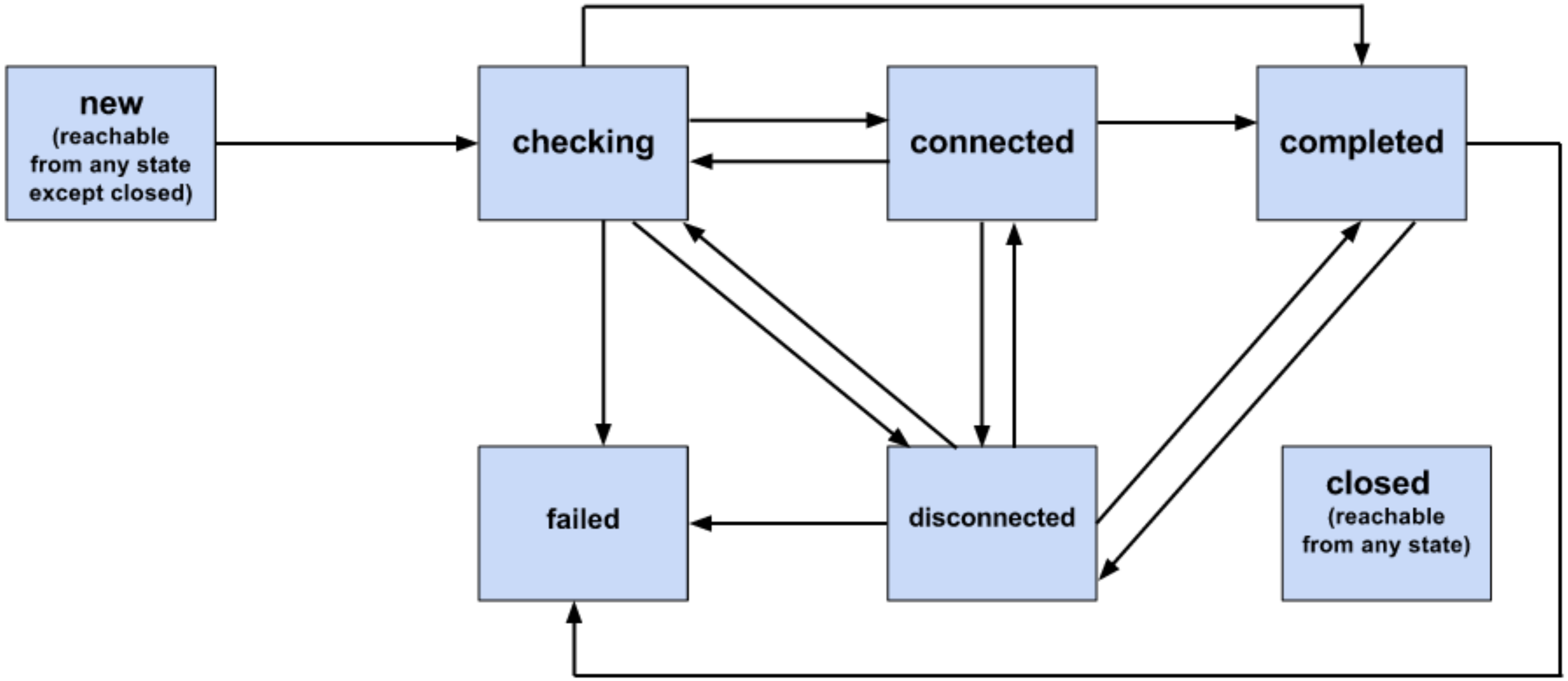
WebIDL

```
interface RTCIceTransport : EventTarget {
    readonly attribute RTCIceRole            role;
    readonly attribute RTCIceComponent       component;
    readonly attribute RTCIceTransportState  state;
    readonly attribute RTCIceGathererState   gatheringState;
    sequence<RTCIceCandidate> getLocalCandidates();
    sequence<RTCIceCandidate> getRemoteCandidates();
    RTCIceCandidatePair?      getSelectedCandidatePair();
    RTCIceParameters?         getLocalParameters();
    RTCIceParameters?         getRemoteParameters();
             attribute EventHandler          onstatechange;
             attribute EventHandler          ongatheringstatechange;
             attribute EventHandler          onselectedcandidatepairchange;
};
```

Differences from ORTC

- No forking support
- Combines IceGatherer and IceTransport functionality
- onselectedcandidatepairchange versus onpairchange
- pc.addIceCandidate versus IceTransport.addRemoteCandidate

WebRTC 1.0: RTCIceTransportState Machine (non-normative)



Differences from ORTC

- No transition from “failed” to “checking” (“failed” state is terminal)
- “disconnected” state can be reached via transient connectivity loss (not just consent failure)
- No transition from “completed” to “connected” (connectivity loss/consent failure transitions to disconnected)
- Spec is unclear how “completed” state is reached (requires “end-of-candidates” indication for each IceTransport)

WebRTC 1.0: RTCRtpCapabilities

WebIDL

```
dictionary RTCRtpCapabilities {
    sequence<RTCRtpCodecCapability>           codecs;
    sequence<RTCRtpHeaderExtensionCapability> headerExtensions;
};
```

WebIDL

```
dictionary RTCRtpCodecCapability {
    DOMString      mimeType;
    unsigned long  clockRate;
    unsigned short channels;
    DOMString      sdpFmtpLine;
};
```

Differences from ORTC

- FEC mechanisms not included in capabilities.
- Codec parameters included in sdpFmtpLine

WebRTC 1.0: RTCRtpParameters

WebIDL

```
dictionary RTCRtpParameters {
    DOMString                                        transactionId;
    sequence<RTCRtpEncodingParameters>               encodings;
    sequence<RTCRtpHeaderExtensionParameters>        headerExtensions;
    RTCRtcpParameters                                rtcp;
    sequence<RTCRtpCodecParameters>                  codecs;
    RTCDegradationPreference                         degradationPreference;
};
```

WebIDL

```
dictionary RTCRtpCodecParameters {
    unsigned short payloadType;
    DOMString      mimeType;
    unsigned long  clockRate;
    unsigned short channels;
    DOMString      sdpFmtpLine;
};
```

Differences from ORTC

- Codec parameters included in sdpFmtpLine.

WebRTC 1.0: RTCRtpEncodingParameters

WebIDL

```
dictionary RTCRtpEncodingParameters {
    unsigned long         ssrc;
    RTCRtpRtxParameters   rtx;
    RTCRtpFecParameters   fec;
    RTCDtxStatus          dtx;
    boolean               active;
    RTCPriorityType       priority;
    unsigned long         ptime;
    unsigned long         maxBitrate;
    double                maxFramerate;
    DOMString             rid;
    double                scaleResolutionDownBy;
};
```

Attribute	Type	Receiver/Sender	Read/Write
ssrc	unsigned long	Receiver/Sender	Read-only
fec	RTCRtpFecParameters	Receiver/Sender	Read-only
dtx	RTCDtxStatus	Sender	Read/Write
rtx	RTCRtpRtxParameters	Receiver/Sender	Read-only
active	boolean	Sender	Read/Write
priority	RTCPriorityType	Sender	Read/Write
ptime	unsigned long	Sender	Read/Write
maxBitrate	unsigned long	Sender	Read/Write
maxFramerate	double	Sender	Read/Write
scaleResolutionDownBy	double	Sender	Read/Write
rid	DOMString	Receiver/Sender	Read-only

Differences from ORTC

- Functionality differences:
  - “dtx” in WebRTC 1.0: Whether discontinuous transmission will be turned on if it is negotiated (via codec-specific parameter or CN codec).
    - No implementations yet, so not added to ORTC.
  - “dependencyEncodingIds” and “frameRateScale” in ORTC: Support for scalable video coding.
    - SVC out of scope for WebRTC 1.0.
- Name differences:
  - “resolutionScale” (ORTC), “scaleResolutionDownBy” (WebRTC 1.0)
  - “encodingId” (ORTC), “rid” (WebRTC 1.0)

Demo: The Firefox Object Model

http://internaut.com:8080/~baboba/cluecon-tutorial/ff/js/main.js

- Substitutes for deprecated APIs
  - Use addTrack instead of addStream
  - Use ontrack instead of onaddstream
- Support for RtpSender/RtpReceiver objects only
  - Check the capability dumper: http://internaut.com:8080/~baboba/cluecon-tutorial/cap-dumper/
  - No support for transceivers, IceTransport, DtlsTransport, etc.

Questions?