WebRTC 1.1 at 2015 ~~f2f~~TPAC?

First step after 1.0

Status of 1.0 (assuming lots of PRs go in)

- PeerConnection: has getSenders() and getReceivers()
- RtpSender
  - has settable parameters, active on/off, priority, maxBitrate, and payloadType
  - has read-only getCapabilities, parameters, codecs, header extensions, SSRCs,
  - has multiple encodings
- RtpReceiver: has readable parameters, same as RtpSender
- DtlsTransport: has state, certificates, all read-only
- IceTransport: has read-only state, selectedCandidatePair, role, component, gatheringState, local/remote parameters, local/remote candidates
- DataChannel: has SctpTransport
- SctpTransport: has DtlsTransport

All of the objects are there and read-only, with a little configuration possible. But we can't construct them or configure them fully.

Goal of first step after 1.0:

Construct and configure the WebRTC 1.0 objects without a PeerConnection and without SDP:

- IceTransport
- DtlsTransport
- RtpSender
- RtpReceiver
- SctpTransport
- DataChannel

Assumptions/Scope

- Don’t need to support RTP/RTCP non-mux
- No initial need to support forking

IceTransport + DtlsTransport

```
[Constructor()]
partial interface RTCIceTransport {
  void gather(RTCIceGatherOptions gatherOptions);  // IceServers + IceTransportPolicy
  void start(RTCIceParameters remoteParameters,
             optional RTCIceRole role);
  void addRemoteCandidate(RTCIceCandidate remoteCandidate);
  void stop();
  attribute EventHandler? onlocalcandidate;
}

[Constructor(RTCIceTransport transport)]
partial interface RTCDtlsTransport {
  void start(RTCDtlsParameters remoteParameters);
  void stop();
}
```

RtpSender + RtpReceiver

```
[Constructor(MediaStreamTrack track, RTCDtlsTransport transport)]
partial interface RTCRtpSender {
  void setTransport (RTCDtlsTransport transport);
  void send (RTCRtpParameters parameters);
  void stop ();
};

[Constructor(RTCDtlsTransport transport)]
partial interface RTCRtpReceiver {
  void setTransport (RTCDtlsTransport transport);
  void receive (RTCRtpParameters parameters);
  void stop ();
};
```

RTP Example

```
var ice = new IceTransport();
var dtls = new DtlsTransport(dtls);
var sender = new RtpSender(track, dtls);
var receiver = new RtpReceiver(dtls);
ice.gather({iceServers: [...]});
var rtpSendParameters = {codecs: [...], encodings: [...]};
// Signal out ice.getLocalParameters(), dtls.getLocalParameters(), rtpSendParameters
// Signal in remoteIceParameters, remoteDtlsParameters, rtpRecvParameters
ice.start(remoteIceParameters);
dtls.start(remoteDtlsParameters);
sender.send(rtpSendParameters);
receiver.receive(rtpReceiverParameters);
```

SctpTransport + DataChannel

```
[Constructor(RTCDtlsTransport transport)]
partial interface RTCSctpTransport : RTCDataTransport {
  static RTCSctpCapabilities getCapabilities ();
  void start (RTCSctpCapabilities remoteCaps);
  void stop ();
  attribute EventHandler ondatachannel;
}

[Constructor(RTCDataTransport transport, RTCDataChannelParameters parameters)]
partial interface RTCDataChannel {
}
```

Data Channel Example

```
var ice = new IceTranpsort();
var dtls = new DtlsTransport(dtls);
var sctp = new SctpTransport(dtls);
// Signal out ice.getLocalParameters(), dtls.getLocalParameters(), sctp.getCapabilities()
// Signal in remoteIceParameters, remoteDtlsParameters, remoteSctpCapabilities
ice.start(remoteIceParameters);
dtls.start(remoteDtlsParameters);
sctp.start(remoteSctpCapabilities);
sctp.ondatachannel = ...;
dc = new DataChannel(sctp, {label: "...", id: "..."});
dc.send("...");
```

Oh, and we need some more full dictionaries

```
dictionary RTCIceCandidate {
  DOMString              foundation;
  unsigned long          priority;
  DOMString              ip;
  RTCIceProtocol         protocol;
  unsigned short         port;
  RTCIceCandidateType    type;
  DOMString              relatedAddress = "";
  unsigned short         relatedPort;
};

dictionary RTCRtpCodecCapability {
  // ...
  unsigned long             clockRate;
  unsigned long             channels;
  payloadtype               preferredPayloadType;
  unsigned long             maxptime;
  sequence<RTCRtcpFeedback> rtcpFeedback;
  Dictionary                parameters;
  unsigned short            maxTemporalLayers = 0;
  unsigned short            maxSpatialLayers = 0;
};
```

```
dictionary RTCRtpHeaderExtension {
  // ...
  unsigned short preferredId;
}

dictionary RTCRtpParameters {
  DOMString muxId = "";
  // ...
}

dictionary RTCRtpEncodingParameters {
  DOMString           encodingId;
  sequence<DOMString> dependencyEncodingIds;
};

dictionary RTCRtpFecParameters {
  // ...
  DOMString mechanism;
};
```

And few other things (needs examples)

```
Promise<RTCStatsReport> getStats();  // Goes on everything

attribute eventhandler onerror;  // Goes on everything

[Constructor(RTCRtpSender sender)]
interface RTCDtmfSender {}

[Constructor(RTCDtlsTransport transport)]
interface RTCIdentity {
    // ...
    Promise<DOMString> getIdentityAssertion(DOMString provider, protocol, username);
    Promise<RTCIdentityAssertion> setIdentityAssertion(DOMString assertion);
};

// If you want proper ICE behavior with multiple IceTransports.
interface RTCIceTransportController {
    void addTransport (RTCIceTransport transport, optional unsigned long index);
};
```