Migrating your native/mobile application to Unified Plan/WebRTC 1.0 API

Authors: steveanton, shampson

Background

The WebRTC specification has evolved over the last few years and with that we have a different SDP format (Unified Plan) and new APIs (RtpTransceivers among them) along with the removal of old APIs (AddStream et al.).

There's new functionality with the new API and the Unified Plan SDP format:

- Early media: after an offer is sent, the calling side can receive media before receiving the answer
- Support for multiple or no media streams for a given track
- With sender objects you can control media with sender.setParameters()
- With transceiver objects, you can more directly control certain aspects of SDP generation.

To keep the native API small and current, we'll be deprecating the non-standard "Plan B" style SDP and the APIs that have been removed from the standard. We realize that this will be a big change that affects a lot of developers, so this page serves to guide you through the process of updating the new APIs.

This Page

This page will describe how the API is changing relative to the current PeerConnection native APIs. Each section will cover a different aspect of the API and have the following format:

1. A table of API methods affected, listed for C++, Java, and Objective-C.
2. A description of the API changes pertaining to that section.
3. Examples written in pseudocode that illustrate how techniques using the old API can be migrated to the new API. It should be clear how to translate this into C++, Java, or Objective-C.

Any APIs not explicitly mentioned on this page are not affected by the change to Unified Plan/WebRTC 1.0.

Conceptual Changes

RtpTransceivers are the newest API concept that comes with the introduction of the Unified Plan SDP format. An RtpTransceiver combines both a sender and receiver and is associated with a single “m= section” in the SDP.

Jan-Ivar of Mozilla has written an excellent blog post exploring the RtpTransceiver API. Note that this is for the Web API, but the concepts directly map to the native APIs.

New RTCConfiguration Option: SdpSemantics

C++	**Java**	**Objective-C**

`RTTConfiguration`
`- sdp_semantics`
	`RTCConfiguration`
	`- sdpSemantics`
		`RTCConfiguration`
		`- sdpSemantics`

There is a new configuration option for PeerConnection which will enable compatibility with Unified Plan SDP and new APIs. There are two options for SdpSemantics:

- **PlanB**: PeerConnection will continue behaving as it does now. This is the default.
- **UnifiedPlan**: PeerConnection will create and set Unified Plan offers/answers. Some APIs will stop working and others will become available (see below).

The chosen SdpSemantics should be specified in RTCConfiguration when creating the PeerConnection and cannot be changed on the fly.

Which option should you use?

The Unified Plan semantics includes the latest spec-compliant format for SDP and includes new, more powerful APIs. In the future, these semantics will be the only supported and Plan B will be deprecated and removed.

You should continue to use Plan B semantics only if you send or receive more than one audio track or more than one video track *and* need to interoperate with existing WebRTC implementations that do not yet support Unified Plan (e.g., Chrome).

API Changes

There are no API differences with the Plan B SdpSemantics. With Unified Plan SdpSemantics, the API changes as described in the following sections:

Signaling API

C++	Java	Objective-C

`PeerConnectionInterface`
`- CreateOffer`
`- CreateAnswer`
`- SetLocalDescription`
`- SetRemoteDescription`
	`PeerConnection`
	`- createOffer`
	`- createAnswer`
	`- setLocalDescription`
	`- setRemoteDescription`
		`RTCPeerConnection`
		`- offerForConstraints`
		`- answerForConstraints`
		`- setLocalDescription`
		`- setRemoteDescription`

The methods for creating offers and answers will now produce SDP according to the latest JSEP standard. Notable changes from the existing SDP:

- Media sections (m= sections) will no longer always be in audio/video/data order. The order depends on the order which tracks/transceivers are added to the PeerConnection.
- Media sections will only have at most one track (i.e.., one group of a=ssrc lines).
- Media section IDs (a=mid) will no longer just be “audio” or “video”. The exact format should be treated as unspecified and will be unique for each media section.
- SSRCs for sending will still be signaled with a=ssrc for now.
- MSID will be signaled with “a=msid:<msid-id> <msid-appdata>” lines
    - To ensure backwards compatibility with Plan B for “simple” use cases (at most 1 audio track and at most 1 video track), MSIDs will also still be signaled with “a=ssrc:<ssrc-id> msid:identifier” lines .

The methods for applying local and remote session descriptions will now assume they are in the “Unified Plan” format. Notably, this means:

- Setting a session description with more than one “Plan B track” within one m section will throw an error. It’s not allowed within the same m section to have more than one a=ssrc:<ssrc-id> cname:<value> lines with different cname values.
- An RtpTransceiver will be created for each media section that cannot be matched to an existing transceiver.
- If “a=msid:<msid-id> <msid-appdata>” lines are signaled then “a=ssrc:<ssrc-id> msid:identifier” lines will be ignored

Please see the latest draft of the JSEP standard for more details about the format of “Unified Plan” SDP.

Offer to Receive Options/Constraints

C++	Java	Objective-C

`PeerConnectionInterface`
`- CreateOffer`
	`PeerConnection`
	`- createOffer`
		`RTCPeerConnection`
		`- offerForConstraints`

- CreateAnswer RTCOfferAnswerOptions - offer_to_receive_audio - offer_to_receive_video	- createAnswer MediaConstraints - “offerToReceiveAudio” - “offerToReceiveVideo”	- answerForConstraints RTCMediaConstraints - kRTCMediaConstraintsOfferToReceiveAudio - kRTCMediaConstraintsOfferToReceiveVideo

These constraints are passed to PeerConnection’s methods for creating SDP to add placeholder media sections in case a sending track would not have already created one.

With Unified Plan semantics, these are still supported for creating offers using a backwards compatibility shim (basically creating an RtpTransceiver if one doesn’t exist). It is recommended to switch to the RtpTransceiver API (example below). They are no longer supported when creating answers.

Examples (pseudocode)

Offer to receive video but not send video.	
PlanB Semantics	UnifiedPlan Semantics

CreateOffer(offer_to_receive_video=1);	AddTransceiver(AUDIO, direction=recvonly);
	CreateOffer();

Reject an incoming audio media section.	
PlanB Semantics	UnifiedPlan Semantics

pc.SetRemoteDescription(offer);
pc.CreateAnswer(offer_to_receive_audio=0);
	pc.SetRemoteDescription(offer);
	for (transceiver in pc.GetTransceivers()) {
	  if (transceiver.media_type == AUDIO) {
	    transceiver.Stop();
	  }
	}
	pc.CreateAnswer();

Stream API

C++	**Java**	**Objective-C**

`PeerConnectionInterface`
`- AddStream`
`- RemoveStream`
`- local_streams`
`- remote_streams`
	`PeerConnection`
	`- addStream`
	`- removeStream`
		`RTCPeerConnection`
		`- addStream`
		`- removeStream`
		`- localStreams`

The PeerConnection methods for adding, removing and retrieving local and remote streams are no longer supported with UnifiedPlan and will crash if an application tries to call them. The API has moved to work in terms of tracks instead of streams (which are a collection of tracks) and the replacement code is very similar. Please see below for using GetSenders/GetReceivers as a replacement for local_streams() and remote_streams().

Examples (pseudocode)

Add stream to a PeerConnection.	
PlanB Semantics	UnifiedPlan Semantics

`stream = Stream(“stream_id”);`
`stream.AddAudioTrack(audio_track);`
`stream.AddVideoTrack(video_track);`
`pc.AddStream(stream);`
	`pc.AddTrack(audio_track,`
	`{“stream_id”});`
	`pc.AddTrack(video_track,`
	`{“stream_id”});`

Remove stream from a PeerConnection.	
PlanB Semantics	UnifiedPlan Semantics

`stream = Stream(“stream_id”);`
`stream.AddAudioTrack(audio_track);`
`stream.AddVideoTrack(video_track);`
`pc.AddStream(stream);`
`...`
`pc.RemoveStream(stream);`
	`audio_sender =`
	`pc.AddTrack(audio_track,`
	`{“stream_id”});`
	`video_sender =`
	`pc.AddTrack(video_track,`
	`{“stream_id”});`
	`...`
	`pc.RemoveTrack(audio_sender);`
	`pc.RemoveTrack(video_sender);`

CreateSender API

C++	Java	Objective-C
`PeerConnectionInterface - CreateSender`	`PeerConnection - createSender`	`RTCPeerConnection - senderWithKind`

CreateSender is no longer implemented and will crash now. The AddTransceiver API replicates this behavior and also creates a receiver (if you don't want to receive you can set the transceiver to send only).

Examples (pseudocode)

Add an audio sender with no track.	
PlanB Semantics	UnifiedPlan Semantics
`sender = pc.CreateSender(AUDIO);`	`transceiver = pc.AddTransceiver(AUDIO);` `sender = transceiver.sender;`

Track API

C++	Java	Objective-C
`PeerConnectionInterface - AddTrack`	`PeerConnection - addTrack`	`RTCPeerConnection - addTrack`

AddTrack now operates in terms of RtpTransceivers as described by the latest WebRTC specification. This means that adding a track will add both a sender and receiver (a transceiver), and AddTrack can possibly reuse an existing RtpTransceiver. AddTrack supports specifying zero or many stream labels to be associated with the track that will be put in the SDP.

Track IDs

Because AddTrack will implicitly create a receiver and a track (for early media), the remote track ID will be assigned at "AddTrack" time, rather than "SetRemoteDescription" time. This means that track IDs may not match between the sending and receiving PeerConnections (specifically, if the receiving PeerConnection calls AddTrack before SetRemoteDescription). This is described in more detail in the previously mentioned blog post by Jan-Ivar.

If you were previously relying on track IDs being symmetrical, you should now correlate tracks using either the transceiver's MID (assigned after SetLocalDescription) or order.

Examples (pseudocode)

Identify camera vs. screenshare tracks

PlanB Semantics

```
// Offerer side
pc.AddTrack(cameraTrack);
pc.AddTrack(screenshareTrack);

offer = pc.CreateOffer();
pc.SetLocalDescription(offer);
SendOffer(offer, cameraTrack.id,
screenshareTrack.id);
```

```
// Answerer side
void OnOffer(offer, cameraTrackId,
screenshareTrackId) {
  cameraTrackId_ = cameraTrackId;
  screenshareTrackId_ =
screenshareTrackId;
  pc.SetRemoteDescription(offer);
  ...
}

void OnAddTrack(receiver) {
  if (receiver.track.id ==
cameraTrackId_) {
    cameraTrack_ = receiver.track;
  }
  if (receiver.track.id ==
screenshareTrackId_) {
    screenshareTrack_ =
receiver.track;
  }
}
```

UnifiedPlan Semantics

```
// Offerer side
cameraTransceiver =
pc.AddTransceiver(cameraTrack);
screenshareTransceiver =
pc.AddTransceiver(screenshareTrack);

offer = pc.CreateOffer();
pc.SetLocalDescription(offer);
SendOffer(offer, cameraTransceiver.mid,
screenshareTransceiver.mid);
```

```
// Answerer side
void OnOffer(offer, cameraMid,
screenshareMid) {
  cameraMid_ = cameraMid;
  screenshareMid_ = screenshareMid;
  pc.SetRemoteDescription(offer);
  ...
}

void OnTrack(transceiver) {
  if (transceiver.mid == cameraMid_) {
    cameraTrack_ =
transceiver.receiver.track;
  }
  if (transceiver.mid ==
screenshareMid_) {
    screenshareTrack_ =
transceiver.receiver.track;
  }
}
```

RTP Media API (RtpSender/RtpReceiver)

C++	**Java**	**Objective-C**

`PeerConnectionInterface`	`PeerConnection`	`RTCPeerConnection`
`- GetSenders`	`- getSenders`	`- senders`
`- GetReceivers`	`- getReceivers`	`- receivers`

Each RtpTransceiver has exactly one RtpSender and one RtpReceiver always, and no RtpSenders or RtpReceivers can be created outside an RtpTransceiver. Therefore, GetTransceivers().length = GetSenders().length = GetReceivers().length. Furthermore, senders and receivers will no longer be removed when setting local/remote descriptions; rather, the RtpTransceiver's currentDirection property will transition to a direction that does not include sending and/or receiving. GetTransceivers() will include all RtpTransceivers ever created; likewise for GetSenders() and GetReceivers(). If you wish to get all active senders and/or receivers (similar to the behavior with Plan B), see the examples below.

Examples (pseudocode)

Get all active senders.	
PlanB Semantics	UnifiedPlan Semantics
`active_senders = pc.GetSenders();`	`active_senders = [];`
	`for (transceiver in`
	`pc.GetTransceivers()) {`
	`  if (transceiver.direction == SENDRECV`
	`or SENDONLY) {`
	`    active_senders +=`
	`transceiver.sender;`
	`  }`
	`}`

Get all active receivers.	
PlanB Semantics	UnifiedPlan Semantics
`active_receivers =`	`active_receivers = []`
`pc.GetReceivers();`	`for (transceiver in`
	`pc.GetTransceivers()) {`
	`  if (transceiver.current_direction is`
	`set && transceiver.current_direction ==`
	`SENDRECV or RECVONLY) {`
	`    active_receivers +=`
	`transciever.receiver;`
	`  }`
	`}`

Transceiver API

C++	Java	Objective-C

`PeerConnectionInterface`
`- AddTransceiver`
`- GetTransceivers`
`PeerConnectionObserver`
`- OnTrack`
	`PeerConnection`
	`- addTransceiver`
	`- getTransceivers`
	`PeerConnection.Observer`
	`- onTrack`
		`RTCPeerConnection`
		`- addTransceiverWithTrack`
		`- addTransceiverOfType`
		`- transceivers`
		`RTCPeerConnectionDelegate`
		`- didStartReceivingOnTransceiver`

The RtpTransceiver API as described in the latest WebRTC specification has been added. The API can be accessed by the AddTransceiver and GetTransceivers methods on PeerConnection. Additionally, the new OnTrack callback has been added which is intended to replace OnAddTrack. OnTrack is called from SetRemoteDescription when the SDP indicates a transceiver that was not previously receiving a track is now receiving. Note that this means OnTrack may be called multiple times for the same transceiver if the direction changes back and forth (e.g., a call is placed on hold and resumed).

Note that the RtpTransceiver API is only available with UnifiedPlan semantics. The PeerConnection methods that deal with transceivers will crash if called with PlanB semantics.

Transceiver reuse for receiving: if you would like to reuse a transceiver you created for receiving when setting the remote description, it needs to have been added via AddTrack. Otherwise, a separate one will be created. See the JSEP standard section 5.10.

Examples (pseudocode)

Hook up remote video rendering before creating an offer (early media).

```
track = CreateLocalVideoTrack();
transceiver = pc.AddTransceiver(track);
remote_track = transceiver.receiver.track;
HookUpRemoteTrackRenderer(remoteTrack);
```

This allows media to start rendering potentially before the answer is received from the remote peer. Note that this is only possible if the DTLS/ICE connection has already been established, and a new transceiver is being added to an existing PeerConnection.

Examples in AppRTC: Android | iOS (note: these use AddTrack but AddTransceiver also works here)

Use OnTrack instead of OnAddTrack.

```
pc.observer.OnTrack = (RtpTransceiver transceiver) {
  HookUpRemoteTrackRenderer(transceiver.receiver.track);
};
pc.SetRemoteDescription(offer with 1 video track);
// OnTrack fires.
```

Simulcast

Simulcast can still be enabled with UnifiedPlan semantics by munging the SDP as it is currently done with PlanB semantics. Future support will be added for setting up simulcast with the AddTransceiver API (using more than one send_encodings), but this is currently not supported. When this support is added it will implement a simulcast SDP according to draft-ietf-mmusic-sdp-simulcast-13.

Example for munging the SDP:
Creating an SDP offer with UnifiedPlan semantics will currently generate a video m section with the following a=ssrc lines:

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101 102 123 127 122 125 107 108 109 124
...
...
a=ssrc-group:FID 1 2
a=ssrc:1 cname:tREKXAufvG1TKxdp
a=ssrc:1 msid:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0 629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:1 mslabel:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
a=ssrc:1 label:629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:2 cname:tREKXAufvG1TKxdp
a=ssrc:2 msid:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0 629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:2 mslabel:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
a=ssrc:2 label:629b3901-27e6-4b5f-9705-f35bd47cec02
```

There are a few pieces worth noting:

- The ssrc numbers are simplified for this example (1 and 2)
- There is just one video stream with a primary ssrc of 1
- ssrc 2 is a rtx repair flow stream
- The “a=ssrc” lines with mslabel, label, and msid are for backwards compatibility and not necessary in a Unified Plan style SDP

In order to munge the SDP appropriately for simulcast you must use an “a=ssrc-group:SIM …” line as specified in RFC 5576, and “a=ssrc” lines for each simulcast layer stream & its rtx stream. The new (munged) SDP would look like this:

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101 102 123 127 122 125 107 108 109 124
...
...
a=ssrc-group:FID 1 2
a=ssrc-group:FID 3 4
a=ssrc-group:FID 5 6
a=ssrc-group:SIM 1 3 5
a=ssrc:1 cname:tREKXAufvG1TKxdp
a=ssrc:1 msid:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:1 mslabel:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
a=ssrc:1 label:629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:2 cname:tREKXAufvG1TKxdp
a=ssrc:2 msid:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:2 mslabel:aKWujDt3uY7cpqYXjUfqMMef2IP3YKCDGTh0
a=ssrc:2 label:629b3901-27e6-4b5f-9705-f35bd47cec02
a=ssrc:3 cname:tREKXAufvG1TKxdp
a=ssrc:4 cname:tREKXAufvG1TKxdp
a=ssrc:5 cname:tREKXAufvG1TKxdp
a=ssrc:6 cname:tREKXAufvG1TKxdp
```

The additions are highlighted above in green. The additions are:

- “a=ssrc:<ssrc-id> cname...” lines for each additional stream
- “a=ssrc-group:FID…” lines mapping the rtx stream ssrc to the simulcast layer’s ssrc it corresponds to
- “a=ssrc-group:SIM…” to define the ssrcs that correspond to the simulcast layers

After calling PeerConnection::SetLocalDescription() with the munged SDP, the simulcast layers will be enabled for that m-section (RtpSender). This can be checked by verifying that the corresponding video RtpSender has multiple encoding parameters - video_sender.getParameters().encodings > 1.