“Unified Plan” Transition Guide (JavaScript)

Author: Henrik Boström (hbos@)

Background

The WebRTC specification has evolved over the years. Notable API changes in the past include the shift to addTrack() and other “track-based” APIs from the legacy addStream() and other “stream-based” APIs. The track-based APIs exposed senders and receivers, allowing replacing which track to send without renegotiation and getting and setting encoding parameters, but there is one more stage left in the evolution towards WebRTC 1.0.

The specification has settled on an SDP format called “Unified Plan” which is different than the Chrome’s SDP format, “Plan B”. Switching SDP format will impact many applications in possibly breaking ways, including what SDP format is generated/accepted and assumptions no longer holding (e.g. local and remote track IDs no longer match in some cases). It also adds a set of new APIs: transceivers. This guide outlines the differences between “Plan B” and “Unified Plan” to help developers prepare for the switch. Firefox already supports Unified Plan, which is the only SDP format they support. Other browsers will follow.

You should be cautious if your application does one of the following things:

- Has multiple audio tracks or multiple video tracks.
- Relies on assumptions about local and remote track IDs matching.
- Munges SDP, uses MCUs or SFUs or otherwise modifies SDP or uses SDP that is not generated by Chrome.

Controlling Which SDP Format Is Used

Specifying the sdpSemantics in Your Application

You can override the default behavior by explicitly specifying the *sdpSemantics* in the RTCPeerConnection constructor, allowing your application to control the SDP format of the client. This flag is Chrome-only; if the browser does not recognize the parameter it is simply ignored.

```
// Use Unified Plan or Plan B regardless of what the default browser behavior is.
new RTCPeerConnection({sdpSemantics:'unified-plan'});
new RTCPeerConnection({sdpSemantics:'plan-b'});
```

Unless your application is prepared for both cases, it is recommended that you explicitly set the *sdpSemantics* to avoid surprises when Chrome's default behavior changes.

Controlling the Default with chrome://flags

As of Chrome M71, chrome://flags contain the experiment "*WebRTC: Use Unified Plan SDP Semantics by default*". If you enable this option, or pass the command line argument --enable-features=RTCUnifiedPlanByDefault when launching Chrome, RTCPeerConnections will be constructed to use Unified Plan unless otherwise specified. (Prior to M71 one could have the same effect using --enable-*blink*-features=RTCUnifiedPlanByDefault.) This allows you to experiment on local instances of Chrome running applications that don't specify *sdpSemantics*.

Feature Detection

In M69 transceivers were added, but they are only supported when Unified Plan is used. Checking whether addTransceiver() throws an exception on a default-constructed RTCPeerConnection is one way to feature detect if Unified Plan is the default. In M70, getConfiguration() was added allowing you to explicitly check the value of *sdpSemantics*, similar to how this already showed up in chrome://webrtc-internals/. For more details on feature detection see this code snippet.

More sophisticated feature detection involves parsing SDP and counting m= sections and tracks within m= sections.

Differences Between Plan B and Unified Plan

How the SDP is Different

In Plan B, one m= section is used for audio and one m= section is used for video, with the media stream identification attribute, or *mid*, set to "audio" or "video" respectively. If multiple tracks of the same kind are in an offer, multiple a=ssrc lines are listed under *the same* m= section.

In Unified Plan, one m= section means one sending and/or one receiving track. The mid is its identifier. If multiple tracks are used, *multiple* m= sections are created.

This makes the two SDP formats incompatible when multiple tracks of the same media type is used. For example, a Unified Plan client might reject a Plan B offer with the exception: “Failed to set remote answer sdp: Media section has more than one track specified with a=ssrc lines which is not supported with Unified Plan.” Or a Plan B client might accept a Unified Plan offer, but only creating tracks and firing *ontrack* for the first m= sections, see code snippet.

Here’s an example of SDP offers in Plan B and Unified Plan when sending two audio tracks. The example has been reduced and ssrcs/ids modified for simplicity. For more details, see the complete original SDP diff and code producing the SDP dumps.

Plan B offer

```
...
a=group:BUNDLE audio
a=msid-semantic: WMS stream-id-2 stream-id-1
m=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
...
a=mid:audio
...
a=rtpmap:103 ISAC/16000
...
a=ssrc:10 cname:cname
a=ssrc:10 msid:stream-id-1 track-id-1
a=ssrc:10 mslabel:stream-id-1
a=ssrc:10 label:track-id-1
a=ssrc:11 cname:cname
a=ssrc:11 msid:stream-id-2 track-id-2
a=ssrc:11 mslabel:stream-id-2
a=ssrc:11 label:track-id-2
```

Unified Plan offer

```
...
a=group:BUNDLE 0 1
a=msid-semantic: WMS
m=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
...
a=mid:0
...
a=sendrecv
a=msid:- <track-id-1>
...
a=rtpmap:103 ISAC/16000
...
a=ssrc:10 cname:cname
a=ssrc:10 msid: track-id-1
a=ssrc:10 mslabel:
a=ssrc:10 label:track-id-1
m=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
...
```

```
a=mid:1
...
a=sendrecv
a=msid:- track-id-2
...
a=rtpmap:103 ISAC/16000
...
a=ssrc:11 cname:cname
a=ssrc:11 msid: track-id-2
a=ssrc:11 mslabel:
a=ssrc:11 label:track-id-2
```

Things to note:

- In Plan B, “a=mid:audio” was used and both tracks were specified under the same “m=audio” section.
- In Unified Plan, “a=mid:0” was used for the first track and “a=mid:1” was used for the second track. “a=audio” and a lot of “a=rtpmap” lines are listed twice, once for each m= section.
- In this example, because addTrack() was called without a stream argument, Unified Plan does not contain a stream ID; the “a=msid” lists the stream ID as “-”. In Plan B however, a randomized stream ID was generated for each track. If a stream is used, both formats list that stream ID.
- In Chrome’s Unified Plan, the “a=ssrc” lines with “msid”, “mslabel” and “label” (marked gray above) are there for *backwards compatibility* and are not required in Unified Plan SDP format.

Sending one audio track and one video track is compatible between clients on different SDP formats though, see code and diff.

The RTCRtpTransceiver APIs and Changes in Behavior

A transceiver represents an m= section in SDP. Because an m= section can be used for both sending and receiving, a transceiver *always* consists of both a sender and a receiver. The direction can change between “inactive”, “sendonly”, “recvonly” and “sendrecv”. A track can be attached to the sender, it may be replaced (with replaceTrack()) or the sender can have no track attached.

When a track is added to a peer connection with addTrack() or addTransceiver(), a transceiver is created (or reused) and the track is attached to the sender. Because a transceiver *may* be used for receiving, a receiving track is always created and attached to the receiver. Unlike the sender, the receiver’s track is always there and cannot be replaced, but it is muted by default.

setLocalDescription() and setRemoteDescription() permanently ties transceivers to m= sections by setting the value of *transceiver.mid* to the m=section’s mid. This is a string identifier that can be used by both endpoints to correlate transceivers and thus which track is which. Applying remote SDP may cause transceivers to be created or updated and RTCPeerConnection.ontrack

to fire for each remote track that should now receive. The track in question is a receiver's track, which may or may not have been created at an earlier stage due to reusing transceivers, such as one created at addTrack(). The track will unmute; if a subsequent setRemoteDescription() changes the transceiver's direction not to receive anymore, the track is muted again.

Local and Remote Track IDs Do Not Match

In Plan B, a sender is created for each local track that is added and a receiver is created for each remote track that is negotiated, the track IDs of local and remote endpoints match.

In Unified Plan, senders and receivers are created in pairs of transceivers, and a *transceiver.receiver.track* may have been created long before remote SDP offers a track. As such, the track that is fired in RTCPeerConnection.ontrack is no longer guaranteed to have an ID matching the sender-side version of the track. Furthermore, because of addTransceiver() and replaceTrack(), a track may be sent multiple times. Track IDs are misleading, and shouldn't be assumed to match. Instead, *transceiver.mid* should be used to correlate local and remote tracks. You may have to "setLocalDescription(answer)" before the *mid* is known.

Remote Tracks Are Never Truly "Removed"

In Plan B, RTCPeerConnection.removeTrack() removes the sender and track, and during negotiation the other endpoint removes the corresponding receiver and track.

In Unified Plan, removeTrack() changes the direction of the transceiver and nulls the sender's track. During negotiation, the other endpoint changes the corresponding transceiver's direction and mutes the track but it is never removed, and it may be reused in the future. See code snippet.

MediaStreamTrack.onended No Longer Fires

Because remote tracks may be muted but are never removed, MediaStreamTrack.onended no longer fires. Instead, rely on MediaStreamTrack.onmute to discover if a track is removed. See code snippet.

Legacy APIs

RTCPeerConnection.addStream() and removeStream() are non-standard APIs. These are shimmed on top of RTCPeerConnection.addTrack() and removeTrack() for backwards compatibility reasons. They will continue to work in Unified Plan, but standardized APIs should be preferred.

Likewise, the events "addstream" and "removestream" fired on an RTCPeerConnection (with event handlers "onaddstream" and "onremovestream") are legacy APIs that are not in the standard. For backwards compatibility reasons they are supported in Unified Plan as of M71, but RTCPeerConnection.ontrack, MediaStreamTrack.onmute and MediaStream.onaddtrack/onremovetrack should be preferred.

Transceiver direction, currentDirection and Early Media

Transceivers are bi-directional. When created, transceivers will by default have a “sendrecv” direction and a null currentDirection. The direction is what we are willing to negotiate, and currentDirection is what has currently been negotiated. If the answerer wants to send media and there is a “sendrecv” transceiver in the offer, the answer can use that transceiver to send media. This allows the answerer to start sending RTP packets immediately, giving the offerer access to media “early”. In Plan B, the offerer doesn’t have a receiver until the answer arrives.

This code snippet illustrates the states transceivers may go through during offers and answers.

Offering to Receive Media

The answerer is not allowed to add new m= sections in the answer SDP. This means that if the offerer is not offering to send anything, there’s typically no “sendrecv” m= sections that the answerer can use to send media on. Renegotiation is needed on the answerer’s side, adding more round trip times before sending is possible. This section describes how to offer to receive media, making it possible to receive Early Media in the initial offer/answer cycle.

In Plan B, the legacy createOffer()-constraints offerToReceiveAudio/offerToReceiveVideo would create m= sections for it to be possible to receive any number of tracks. In Unified Plan this constraint only works for a single track per kind as it only creates one audio and/or video transceiver; the spec advises using addTransceiver() instead. Chrome only partially supports this constraint; it works for offering but the createOffer()-transceiver is not immediately visible, leading to some unexpected behavior.

The Unified Plan way of offering to receive media is to use addTransceiver() to create as many “recvonly”-transceiver as you are willing to receive. This code snippet illustrates how.