Abstract

This document proposes mapping message representations between RTCWeb Javascript Session Establishment Protocol (JSEP) scheme and SIP messaging scheme. Such a signaling mapping is intended to enable Javascript to use SIP to establish a session between two RTCWeb enabled browsers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 1, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as
Internet-Draft            RTCWeb-JSEP-SIP-Mapping            August 2012

described in the Simplified BSD License.

Table of Contents

1. Introduction .......................... 3
   1.1. Terminology .......................... 3
2. Architecture Overview .................. 3
   2.1. Architecture Model .................. 3
   2.2. Session Management .................. 4
3. Media Setup ................................ 4
   3.1. Session Management ................. 4
       3.1.1. Initiate the Session ............ 4
       3.1.2. Accept the Session ............. 4
       3.1.3. Conform the Session ............ 5
       3.1.4. Terminate the Session .......... 5
       3.1.5. Confirm Session Termination ..... 5
   3.2. Media Management .................. 6
       3.2.1. Early Media ..................... 6
       3.2.2. Reliable Early Media ........... 6
       3.2.3. Updates to the Session ........ 7
   3.3. Querying capabilities .............. 8
       3.3.1. Requesting for Capabilities ..... 8
       3.3.2. Sharing Capabilities ......... 9
   3.4. Forking requests ................... 9
4. Mapping between SIP Message and JSEP API ... 9
   4.1. Map JSEP API to SIP Message .......... 9
   4.2. Map SIP Message to JSEP API .......... 10
5. SDP Information Exchange ............... 11
6. Example Message Flows .................. 11
   6.1. SIP Session ........................ 11
   6.2. Early Media ........................ 12
   6.3. Session Update ...................... 12
   6.4. Querying Capabilities ............... 13
7. Security Considerations ................ 14
8. IANA Considerations ................... 14
9. Acknowledgements ....................... 14
10. Normative References ................... 14
    Authors' Addresses ..................... 15
1. Introduction

In draft [I-D.ietf-rtcweb-jsep], it is mentioned that there are several options for the signalling mechanisms: ROAP (see [I-D.jennings-rtcweb-signaling]), XMPP/Jingle or SIP.

This document focuses on SIP and tries to explain how to use JSEP and SIP to exchange session descriptions.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Architecture Overview

2.1. Architecture Model

Figure 1 represents the overall architecture of realtime peer to peer media communication between two browsers. Here "Browser" is synonymous with "User Agent", and "Web App" is synonymous with "JavaScript".

```
+----------------------+
|          Server      |
| +----------------------+ |
| |  +----------------------+ |
| |  |  +----------------------+ |
| |  |  |  +----------------------+ |
| |  |  |  |  +----------------------+ |
| |  |  |  |  |  +----------------------+ |
| |  |  |  |  |  |  +----------------------+ |
| |  |  |  |  |  |  |  +----------------------+ |
| |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  v  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| Web App |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| +----------------------+  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  ^  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  SDP  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  | +----------------------+  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
| v  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  +----------------------+ |
```

Figure 1: JSEP-SIP Mapping Architecture
2.2. Session Management

The statement machine for session management is same as that described in section 17 of [RFC3261].

Section 3 explains how JS clients could use the SIP messages to manage the session. The detailed descriptions of SIP messages are defined in [RFC3261].

3. Media Setup

3.1. Session Management

3.1.1. Initiate the Session

To initiate a session, the Caller creates an offer and send the offer to Callee by using SIP "INVITE" request with SDP.

The JSEP APIs are defined in [webrtc-api] and [I-D.ietf-rtcweb-jsep].

JSEP API:

CallerJS->CallerUA: pc = new PeerConnection();

CallerJS->CallerUA: pc.addStream(localStream, null);

CallerJS->CallerUA: offer = pc.createOffer(null);

SIP message:

CallerJS->CalleeJS: SIP INVITE with SDP.

After receiving the SIP "INVITE" message, the Callee parses the offer, and applies the supplied offer as the remote description.

JSEP API:

CalleeJS->CalleeUA: peer.setRemoteDescription("offer",offer);

3.1.2. Accept the Session

If the Callee accepts a session, it creates the answer and sends it to Caller by using SIP OK response with SDP.

JSEP API:

CalleeJS->CalleeUA: peer.addStream(localStream, null);
CalleeJS-->CalleeUA: answer = peer.createAnswer(offer, null);

SIP message:
CalleeJS-->CallerJS: SIP 200 OK with SDP

3.1.3. Conform the Session

After receiving the session acceptance, SIP "200 OK" message, the Caller parses the received answer and applies the supplied answer as the remote description to the PeerConnection.

JSEP API:
CallerJS-->CallerUA: pc.setRemoteDescription("answer",answer);

Now Caller sends the session confirmation message to Callee by sending the SIP ACK request.

SIP message:
CallerJS-->CalleeJS: SIP ACK

3.1.4. Terminate the Session

To terminate a session, the Caller/Callee can close the peer connection by sending the session termination request to the Callee/Caller by using SIP "BYE" request.

SIP message:
CallerJS-->CalleeJS: SIP BYE.

3.1.5. Confirm Session Termination

On receiving the session termination message, the User Agent Server (Caller or Callee that receiving the BYE request) closes the peer connection and responds with a success response to terminate the call with 200 OK response.

JSEP API:
CalleeJS-->CalleeUA: peer.close().

SIP message:
CalleeJS-->CallerJS: SIP 200 OK.
After receiving the confirmation for session termination request, the Caller will release the resources in browser through the following JSEP API.

JSEP API:

CallerJS->CallerUA: pc.close().

### 3.2. Media Management

#### 3.2.1. Early Media

According to [RFC3261], after receiving the invite, Callee MAY respond with a single or multiple provisional responses (1XX responses other than 100) with or with out SDP. To establish early media between the two peers, these provisional responses must contain SDP.

The corresponding JSEP Api is similar to that in Section 3.1.2.

SIP message:

CalleeJS->CallerJS: SIP 183 "Session Progress" With SDP.

#### 3.2.2. Reliable Early Media

##### 3.2.2.1. Provisional Acknowledgement

SIP supports reliability for early media through SIP PRACK message. Based of the specifications mentioned for SIP reliability [RFC3262], Caller can respond with PRACK request for the 183 provisional response.

The corresponding JSEP API is similar to that in Section 3.1.2.

SIP message:

CallerJS->CalleeJS: SIP PRACK.

After receiving the SIP UPDATE message, the callee can parse the session information, and apply the supplied offer as the remote description.

JSEP API:

CalleeJS->CalleeUA: peer.setRemoteDescription("offer", offer);
3.2.2.2. Accept Provisional Acknowledgement

On receiving the Provisional Acknowledgement message from Caller, Callee will accept it by responding with SIP 200 OK response for PRACK.

The corresponding JSEP API is similar to that in Section 3.1.3.

SIP message:

CalleeJS->CallerJS: SIP 200 OK.

After receiving the SIP 200 OK message, the caller can parse the session information, and apply the supplied answer as the remote description.

JSEP API:

CallerJS->CallerUA: peer.setRemoteDescription("answer",answer);

3.2.3. Updates to the Session

SIP sessions can be updated to add or remove media or to put one party on hold by the other party.

3.2.3.1. Update session with re-Invite

Session updates can be done with re-Invite messages according to [RFC3261]. The processing of re-invites is same as that for the normal invite as explained in Section 3.1.1, Section 3.1.2 and Section 3.1.3.

SIP message:

CallerJS->CalleeJS: SIP INVITE W/ SDP.

CalleeJS->CallerJS: SIP 200 OK W/ SDP.

CallerJS->CalleeJS: SIP ACK.

3.2.3.2. Update session with Update request

Session updates can also be done with Update requests defined in [RFC3311].
### 3.2.3.2.1. Sending Update Request

Session can be refreshed by Caller or Callee by sending the SIP Update request.

**JSEP API:**

CallerJS->CallerUA: `offer = pc.createOffer(null);`

**SIP message:**

CallerJS->CalleeJS: SIP UPDATE.

### 3.2.3.2.2. Accepting the Update Request

The User Agent Server that received the Update request, processes it according to rules specified in [RFC3261] and [RFC3311], and sends the SIP 200 OK successful response back to the sender of the request.

**JSEP API:**

CalleeJS->CalleeUA: `answer = peer.createAnswer(offer, null);`

**SIP message:**

CalleeJS->CallerJS: SIP 200 OK

### 3.3. Queriyng capabilities

Using SIP OPTIONS request Caller can get the capabilities of the Callee, for establishing the session, before the session establishment. In order to support this feature, the browser SHOULD support the other approach for JSEP implementation as specified in 3 of [I-D.ietf-rtcweb-jsep].

#### 3.3.1. Requesting for Capabilities

CallerJS can query for the capabilities of the CalleeJS according to the specifications provided in [RFC3261] by sending SIP OPTIONS request.

**SIP message:**

CallerJS->CalleeJS: SIP OPTIONS
3.3.2. Sharing Capabilities

User Agent server, that received the OPTIONS request will share its capabilities with the requester by sending its capabilities in the 200 OK response.

JSEP API:

CalleeJS->CalleeUA: capabilities = getCapabilities();

SIP message:

CalleeJS->CallerJS: 200 OK W/ SDP.

3.4. Forking requests

According to [I-D.ietf-rtcweb-jsep], JSEP supports forking of forking of requests. The use case for this scenario to happen can be on receiving a SIP redirect request or when the user wants to establish a multiparty session or any other use case which is out of the scope of this document. This feature can be implemented by sending multiple SIP Invite messages from CallerJS to multiple CalleeJS, following the steps [2.1, 2.2] based on the inputs received.

4. Mapping between SIP Message and JSEP API

4.1. Map JSEP API to SIP Message

When CallerJS uses JSEP API to interact with CallerUA, mapping between JSEP API to SIP Message is required, to send the offer to CalleeJS. Figure 2 shows the mapping from JSEP APIs to SIP messages.
+-------------------------------------+--------------------------+
|         JSEP API                    | SIP Message              |
+-------------------------------------+--------------------------+
| createOffer(),setLocalDescription()| SIP:Invite               |
+-------------------------------------+--------------------------+
| createAnswer()                      | SIP:200 OK               |
+-------------------------------------+--------------------------+
| setRemoteDescription()              | SIP:ACK                  |
+-------------------------------------+--------------------------+
| close()                             | SIP:BYE                  |
+-------------------------------------+--------------------------+
| getCapabilities()                   | SIP:200 OK (for options) |
+-------------------------------------+--------------------------+
| addStream(),removeStream()          | SIP:Re-Invite            |
+-------------------------------------+--------------------------+
| PeerConnectionErrorCallBack()       | SIP:4xx, 5xx, 6xx        |
+-------------------------------------+--------------------------+

Figure 2: Map JSEP API to Jingle Message

4.2. Map SIP Message to JSEP API

When CalleeJS receives SIP Message, it needs to map it to the corresponding JSEP API, to interact with the CalleeUA. Figure 3 shows the mapping from JSEP APIs to SIP messages.

+--------------------------+--------------------------------------+
|   SIP Message            |        JSEP API                      |
+--------------------------+--------------------------------------+
| SIP: Invite              | setRemoteDescription(),createOffer() |
+--------------------------+--------------------------------------+
| SIP: 200 OK              | createAnswer()                       |
+--------------------------+--------------------------------------+
| SIP: ACK                 | setLocalDescription()                |
+--------------------------+--------------------------------------+
| SIP: BYE                 | close()                              |
+--------------------------+--------------------------------------+
| SIP: 200 OK (for options)| getCapabilities()                    |
+--------------------------+--------------------------------------+
| SIP: 4xx, 5xx, 6xx       | PeerConnectionErrorCallBack()        |
+--------------------------+--------------------------------------+

Figure 3: Map SIP Message to JSEP API
5. SDP Information Exchange

While using SIP messaging scheme, SDP retrieved from the peerconnection (as offer or answer) can be directly mapped to the SDP part of the SIP message.

6. Example Message Flows

6.1. SIP Session

Figure 4 represents the basic SIP session flow between two peers, CallerJS uses SIP INVITE method to initiate a session with CalleeJS, this method includes the SDP part in the body of the SIP Message. Then CalleeJS accepts the session using SIP 200 OK message with SDP. Now CallerJS confirms the receipt of session details with SIP ACK request. After the media session, CallerJS uses the SIP BYE request to terminate the session, and CalleeJS acknowledges with SIP 200 OK response.

```
   Call JS                              Callee JS

<p>| |
|                                                |</p>
<table>
<thead>
<tr>
<th>SIP: INVITE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>SIP: 200 OK</td>
</tr>
<tr>
<td>&lt;-----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SIP: ACK</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Media Session</td>
</tr>
<tr>
<td>&lt;-----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SIP: BYE</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;------------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 4: SIP Session
```

Message details go here...
6.2. Early Media

In Figure 5, CallerJS uses SIP INVITE method to initiate a session with CalleeJS, this method includes the SDP part in the body of the SIP Message. Then CalleeJS responds with the SIP 183 session progress provisional response. CallerJS acknowledges this provisional response with SIP PRACK request. Then callerJS confirms the early media session with SIP 200 OK response. SDP session negotiation can happen in any of these messages according to SIP specification [RFC3261] and SDP offer/answer specification [RFC3262]. Media session can be established without SIP reliable responses also, if the session negotiations is completed with offer/answer exchange by Invite and 183 response only.

![Figure 5: Early Media](image)

Message details go here...

6.3. Session Update

In Figure 6, CallerJS uses SIP RE-INVITE request with SDP to update session details like adding video to an existing session. CalleeJS accepts that by accepting the media update by SIP 200 OK response with SDP. CallerJS acknowledges this acceptance by SIP ACK request.
In Figure 7, CallerJS uses SIP UPDATE request with SDP to update session details like adding video to an existing session. CalleeJS accepts that by accepting the media updation by SIP 200 OK response with SDP.

Message details go here...

6.4. Querying Capabilities

In Figure 8, CallerJS uses SIP OPTIONS request to query the capabilities of the CalleeJS. On receiving this CalleeJS responds with the 200 OK message with all the features supported by it using the getCapabilities method according to SIP [RFC3261].
Figure 8: Query Capabilities

Message details go here...

7. Security Considerations

TBD.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

The author would like to thank Kiran Kumar, Bert greevenbosch, Justin Uberti for the reviews and feedbacks.

10. Normative References

[I-D.ietf-rtcweb-jsep]

[I-D.jennings-rtcweb-signaling]


Authors' Addresses

Kiran Kumar Guduru
S.V.University
D.No: 25-1-996, 6th Lane, Nethaji Nagar
Nellore  524004
India

Phone: +91-998-5312234
Email: g.kiranreddy4u@gmail.com

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang
Shenzhen  518129
P. R. China

Phone: +86-755-28971807
Email: likepeng@huawei.com