RESTful Network API for WebRTC Signaling

Approved Version 1.0 – 31 Jan 2017

Open Mobile Alliance

OMA-TS-REST_NetAPI_WebRTCSignaling-V1_0-20170131-A

Contents

Figures

Tables

1. Scope

This specification defines a RESTful API for WebRTC Signaling using HTTP protocol bindings.

2. References

2.1 Normative References

[Autho4API_10]	"Authorization Framework for Network APIs", Open Mobile Alliance™, OMA-ER-Autho4API-V1_0, URL:http://www.openmobilealliance.org/
[IETF_Msid_draft]	"Cross Session Stream Identification in the Session Description Protocol", H. Alvestrand, November 2013, URL:http://tools.ietf.org/html/draft-ietf-mmusic-msid-02 Note: The referenced IETF draft is a work in progress, subject to change without notice.
[IETF_RTCWeb_JSEP]	"Javascript Session Establishment Protocol", J. Uberti, C. Jennings, October 22, 2013, URL:http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-05 Note: The referenced IETF draft is a work in progress, subject to change without notice.
[IETF_SCTP_SDP_draft]	"Stream Control Transmission Protocol (SCTP)-Based Media Transport in the Session Description Protocol (SDP)", S. Loreto and G. Camarillo, October 21, 2013, URL:http://tools.ietf.org/html/draft-ietf-mmusic-sctp-sdp-05 Note: The referenced IETF draft is a work in progress, subject to change without notice.
[REQ_RCS_API]	"Rich Communication Suite RCS API Detailed Requirements Version 2.3", URL:http://www.gsma.com/rcs/wp-content/uploads/2012/10/RCS_API_requirements_v2_3.pdf/
[REST_NetAPI_ACR]	"RESTful Network API for Anonymous Customer Reference Management", Open Mobile Alliance™, OMA-TS-REST_NetAPI_ACR-V1_0, URL:http://www.openmobilealliance.org/
[REST_NetAPI_Common]	"Common definitions for RESTful Network APIs", Open Mobile Alliance™, OMA-TS-REST_NetAPI_Common-V1_0, URL:http://www.openmobilealliance.org/
[REST_NetAPI_Notification Channel]	"RESTful Network API for Notification Channel", Open Mobile Alliance™, OMA-TS-REST_NetAPI_NotificationChannel-V1_0, URL:http://www.openmobilealliance.org/
[REST_SUP_WRTCSig]	"XML schema for the RESTful Network API for WebRTC Signaling", Open Mobile Alliance™, OMA-SUP-XSD_rest_netapi_webrtcsignaling-V1_0, URL:http://www.openmobilealliance.org/
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt
[RFC2616]	"Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding et. al, January 1999, URL:http://www.ietf.org/rfc/rfc2616.txt
[RFC3264]	"An Offer/Answer Model with the Session Description Protocol (SDP)", J. Rosenberg and H. Schulzrinne, June 2002, URL:http://www.ietf.org/rfc/rfc3264.txt
[RFC3388]	"Grouping of Media Lines in the Session Description Protocol (SDP)", G. Camarillo et al., December 2002, URL:http://www.ietf.org/rfc/rfc3388.txt
[RFC3966]	"The tel URI for Telephone Numbers", H.Schulzrinne, December 2004, URL:http://www.ietf.org/rfc/rfc3966.txt
[RFC3986]	"Uniform Resource Identifier (URI): Generic Syntax", R. Fielding et. al, January 2005, URL:http://www.ietf.org/rfc/rfc3986.txt
[RFC4566]	"SDP: Session Description Protocol", M. Handley et al., July 2006, URL:http://www.ietf.org/rfc/rfc4566.txt
[RFC4627]	"The application/json Media Type for JavaScript Object Notation (JSON)", D. Crockford, July 2006, URL:http://www.ietf.org/rfc/rfc4627.txt
[RFC5245]	"Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.", J. Rosenberg, April 2010, URL:http://www.ietf.org/rfc/rfc5245.txt
[RFC5888]	"The Session Description Protocol (SDP) Grouping Framework", G. Camarillo and H. Schulzrinne, June 2010, URL:http://www.ietf.org/rfc/rfc5888.txt
[SCRRULES]	"SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/

[W3C_WebRTC]	WebRTC 1.0: Real-time Communication Between Browsers, W3C Working Draft, 21 August 2012, The World Wide Web Consortium, URL:http://www.w3.org/TR/webrtc/ Note: The referenced W3C draft is a work in progress, subject to change without notice.
[XMLSchema1]	W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures Second Edition, W3C Recommendation 5 April 2012, URL:http://www.w3.org/TR/xmlschema11-1/
[XMLSchema2]	W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Recommendation 5 April 2012, URL:http://www.w3.org/TR/xmlschema11-2/

2.2 Informative References

[IETF_RTCWeb_Overview]	"Overview: Real Time Protocols for Brower-based Applications", H. Alvestrand, September 3, 20, 2013, URL:http://tools.ietf.org/html/draft-ietf-rtcweb-overview-08 Note: The referenced IETF draft is a work in progress, subject to change without notice.
[IETF_RTCWeb_RTP]	"Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", C. Perkins et al., October 21, 2013, URL:http://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage-10 Note: The referenced IETF draft is a work in progress, subject to change without notice.
[OMADICT]	"Dictionary for OMA Specifications", Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_9, URL:http://www.openmobilealliance.org/
[REST_WP]	"Guidelines for RESTful Network APIs", Open Mobile Alliance™, OMA-WP-Guidelines_for_RESTful_Network_APIs, URL:http://www.openmobilealliance.org/
[RFC3261]	"SIP: Session Initiation Protocol", J. Rosenberg et al., June 2002, URL:http://www.ietf.org/rfc/rfc3261.txt
[RFC3262]	"Reliability of Provisional Responses in the Session Initiation Protocol (SIP)", J. Rosenberg and H. Schulzrinne, June 2002, URL:http://www.ietf.org/rfc/rfc3262.txt
[RFC3312]	"Integration of Resource Management and Session Initiation Protocol (SIP)", G. Camarillo et al., October 2002, URL:http://www.ietf.org/rfc/rfc3312.txt
[RFC6337]	"Session Initiation Protocol (SIP) Usage of the Offer/Answer Model", S. Okumura et al., August 2011, URL:http://www.ietf.org/rfc/rfc6337.txt

3. Terminology and Conventions

3.1 Conventions

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].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

For the purpose of this TS, all definitions from the OMA Dictionary apply [OMADICT].

Client-side Notification URL	An HTTP URL exposed by a client, on which it is capable of receiving notifications and that can be used by the client when subscribing to notifications.
Long Polling	A variation of the traditional polling technique, where the server does not reply to a request unless a particular event, status or timeout has occurred. Once the server has sent a response, it closes the connection, and typically the client immediately sends a new request. This allows the emulation of an information push from a server to a client.
Notification Channel	A channel created on the request of the client and used to deliver notifications from a server to a client. The channel is represented as a resource and provides means for the server to post notifications and for the client to receive them via specified delivery mechanisms. For example in the case of Long Polling the channel resource is defined by a pair of URLs. One of the URLs is used by the client as a call-back URL when subscribing for notifications. The other URL is used by the client to retrieve notifications from the Notification Server.
Notification Server	A server that is capable of creating and maintaining Notification Channels.
Originator	The party that initiates a session.
Participant	A party that participates in a session, including the Originator.
Server-side Notification URL	An HTTP URL exposed by a Notification Server, that identifies a Notification Channel and that can be used by a client when subscribing to notifications.
Terminating Participant	A Participant in a session that is not the Originator.
Update Originator	The Participant that requests an update of the session parameters.
Update Recipient	The Participant that receives an update request.

3.3 Abbreviations

ACR	Anonymous Customer Reference
API	Application Programming Interface
CDATA	Character Data
HTTP	HyperText Transfer Protocol
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network

JSEP	Javascript Session Establishment Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
MSISDN	Mobile Subscriber ISDN Number
OMA	Open Mobile Alliance
REST	REpresentational State Transfer
RTCWeb	Real-Time Communication on the Web
SCR	Static Conformance Requirements
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SRTP	Secure Real-Time Transport Protocol
TS	Technical Specification
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Universal character set Transformation Format
W3C	World-Wide Web Consortium
WebRTC	Web Real-Time Communication
WP	White Paper
XML	eXtensible Markup Language
XSD	XML Schema Definition

4. Introduction

The Technical Specification of the RESTful Network API for WebRTC Signaling contains HTTP protocol bindings for WebRTC Signaling functionality, using the REST architectural style. The specification provides resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (i.e. XML, JSON).

4.1 Version 1.0

Version 1.0 of this specification supports the following operations:

– Managing subscriptions to event notifications related to WebRTC Signaling

– Creating and terminating WebRTC sessions

– Inviting a party to a WebRTC session, accepting, cancelling and rejecting such a session invitation

– Indicating that the invited party is being alerted (“Ringing”)

– Updating a WebRTC session, accepting, cancelling and rejecting session update requests

– Retrieving information about WebRTC sessions

– Retrieving and updating the ICE status of a session

– Sending and receiving event notifications related to WebRTC sessions

In addition, this specification provides:

– Support for scope values used with the authorization framework defined in [Autho4API_10]

– Support for Anonymous Customer Reference (ACR) as an end user identifier

– Support for “acr:auth” as a reserved keyword in an ACR

5. WebRTC Signaling API definition

This section is organized to support a comprehensive understanding of the WebRTC Signaling API design. It specifies the definition of all resources, definition of all data structures, and definitions of all operations permitted on the specified resources.

This Network API provides a method for the signaling of voice and video sessions over IP under the assumption that the applications which use this signaling are aligned with JSEP [IETF_RTCWeb_JSEP], e.g. as specified by WebRTC [W3C_WebRTC] which defines a Javascript API for use in the web browser. RTCWeb/WebRTC is a suite of IETF and W3C standards (see e.g. [IETF_RTCWeb_Overview]) primarily developed to allow web browsers to act as end points (source and/or sink) of real-time media (containing audio, video, and data channels) in a peer-to-peer fashion, or to communicate with another entity such as a media gateway or a media application. Media streams are transmitted over SRTP [IETF_RTCWeb_RTP].

The IETF RTCWeb specifications fully define how the media are transmitted. However, they do only partially specify the signaling [IETF_RTCWeb_JSEP]. In particular, JSEP requires SDP [RFC4566] to be used to describe the media streams involved in the session, and the offer-answer model [RFC3264] to negotiate the media. The offer-answer model mandates that an offer from one end point is followed by an answer from the other end point, after which a new offer can be initiated by any of the end points. Any other sequence (e.g. an answer followed by an answer) is considered a conflict. On top of the offer/answer model, JSEP introduces the concept of a provisional answer ("pranswer") which adds states to the offer-answer state machine (see [W3C_WebRTC] for a graphical representation). The concept of provisional answers is helpful in situations when the server needs to send multiple answers to the application. Provisional answers are also useful when the server would have to otherwise convert an offer into an answer (and vice versa) as part of its mediation role (see H.1.4 for an example). Finally, JSEP uses ICE [RFC5245] to penetrate firewalls.

The above three items defined in JSEP (i.e. SDP, offer/answer with the additional pranswer state and ICE) determine which information needs to be provided by the application to the WebRTC endpoint (usually the browser), and by the WebRTC endpoint to the application, in order to set up the end point for a communication session. It is however not specified how this information is transmitted from one endpoint to the other. This gap is covered by the present specification, which defines a RESTful Network API that allows a web application (e.g. a JavaScript running in a WebRTC-enabled browser) to signal a video and/or voice session over IP with another communication endpoint in the network.

Common data types, naming conventions, fault definitions and namespaces are defined in [REST_NetAPI_Common].

The remainder of this document is structured as follows:

Section 5 starts with a diagram representing the resources hierarchy followed by a table listing all the resources (and their URL) used by this API, along with the data structure and the supported HTTP verbs (section 5.1). What follows are the data structures (section 5.2). A sample of typical use cases is included in section 5.3, described as high level flow diagrams.

Section 6 contains detailed specification for each of the resources. Each such subsection defines the resource, the request URL variables that are common for all HTTP methods, and the supported HTTP verbs. For each supported HTTP verb, a description of the functionality is provided, along with an example of a request and an example of a response. For each unsupported HTTP verb, the returned HTTP error status is specified, as well as what should be returned in the Allow header.

All examples in section 6 use XML as the format for the message body, while JSON examples are provided in Appendix D.

Section 7 contains fault definition details such as Service Exceptions and Policy Exceptions.

Appendix B provides the Static Conformance Requirements (SCR).

Appendix C provides application/x-www-form-urlencoded examples, where applicable.

Appendix E provides the operations mapping to a pre-existing baseline specification, where applicable.

Appendix F provides a list of all Light-weight Resources, where applicable.

Appendix G defines authorization aspects to control access to the resources defined in this specification.

Appendix H contains additional flows which illustrate mappings between REST requests and SIP protocol messages.

Note: Throughout this document, client and application, audio and voice, as well as WebRTC and RTCWeb, can be used interchangeably.

5.1 Resources Summary

This section summarizes all the resources used by the RESTful Network API for WebRTC Signaling.

The "apiVersion" URL variable SHALL have the value "v1" to indicate that the API corresponds to this version of the specification. See [REST_NetAPI_Common] which specifies the semantics of this variable.

The figure below visualizes the resource structure defined by this specification. Note that those nodes in the resource tree which have associated HTTP methods defined in this specification are depicted by solid boxes.

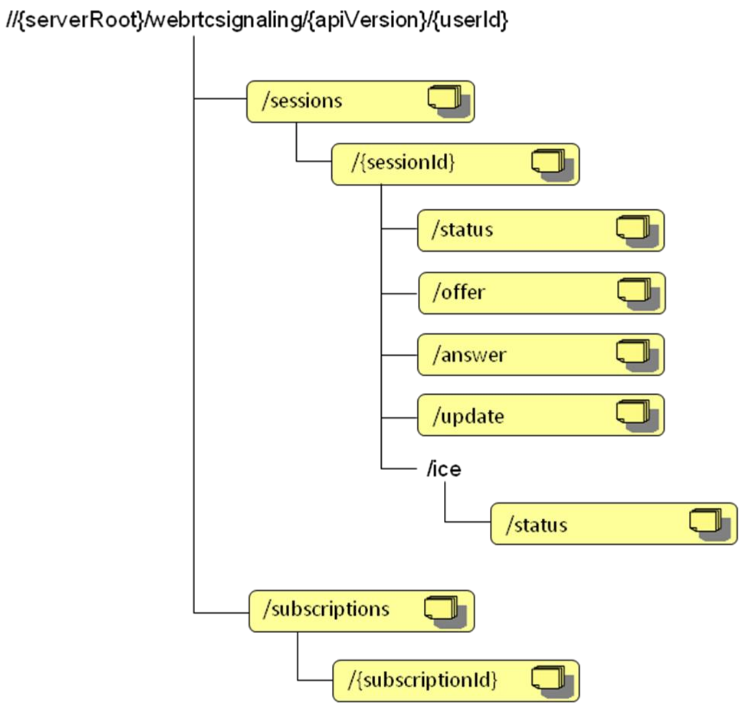

Figure 1: Resource structure defined by this specification

The following tables give a detailed overview of the resources defined in this specification, the data type of their representation and the allowed HTTP methods.

Purpose: To allow client to manage subscriptions for notifications of new WebRTC sessions or changes to existing sessions

Resource	**URL** **Base URL: http://{serverRoot}/webrtcsignaling/{apiVersion}**	**Data Structures**	**HTTP verbs**			
			GET	**PUT**	**POST**	**DELETE**
All subscriptions to WebRTC signaling notifications	/{userId}/subscriptions	WrtcsSubscriptionList (Used for GET) WrtcsNotificationSubscription (Used for POST) common:ResourceReference (optional alternative for POST response)	Retrieves the list of active WebRTC Signaling notification subscriptions	no	Creates a new subscription for notification for audio and/or video sessions	no
Individual subscription to WebRTC signaling notifications	/{userId}/subscriptions/{subscriptionId}	WrtcsNotificationSubscription (Used for GET/PUT)	Retrieves an individual audio and/or video subscription	no	no	Terminates a n individual audio and/or video subscription

Purpose: To allow client to manage audio and/or video sessions

Resource	**URL** **Base URL: http://{serverRoot}/webrtcsignaling/{apiVersion}**	**Data Structures**	**HTTP verbs**			
			GET	**PUT**	**POST**	**DELETE**
All WebRTC sessions	/{userId}/sessions	WrtcsSession common:ResourceReference (optional alternative for POST response)	no	no	Create a new audio and/or video session	no

Individual WebRTC session	/{userId}/sessions/{sessionId}	WrtcsSession	Retrieve an audio and/or video session	no	no	Terminate an audio and/or video session Reject invitation (Terminating Participant) Cancel invitation (Originator)
Status of a WebRTC session	/{userId}/sessions/{sessionId}/status	WrtcsSessionStatus	Retrieve the status	Indicate alerting of the user (“Ringing”) Accept a session invitation	no	no
Initial or most recent offer in a WebRTC session	/{userId}/sessions/{sessionId}/offer	WrtcsOffer	Retrieve the offer	Provide an offer to an offerless session invitation	no	no
Most recent answer in a WebRTC session	/{userId}/sessions/{sessionId}/answer	WrtcsAnswer	Retrieve the answer	Provide an answer to a session invitation or session modification	no	no
Update offer in a WebRTC session	/{userId}/sessions/{sessionId}/update	WrtcsOffer	Retrieve the update offer	Initiate an update	no	Cancel an update (Update Originator) Decline an update (Update Recipient)
ICE status of a WebRTC session	/{userId}/sessions/{sessionId}/ice/status	WrtcsIceStatus	Retrieve the ICE status	Update the ICE status	no	no

Purpose: To allow client to receive notifications regarding audio and/or video sessions

Resource	URL Base URL: <Specified by the client>	Data Structures	HTTP verbs			
			GET	PUT	POST	DELETE
Client notification about WebRTC signaling events	Specified by client when subscription is created or provisioned	WrtcsEventNotification	no	no	This operation notifies a client about audio and/or video session event	no
Client notification about WebRTC session invitation	Specified by client when subscription is created or provisioned	WrtcsSessionInvitationNotification	no	no	This operation notifies a client about audio and/or video session invitation	no
Client notification about session invitation acceptance or session update acceptance	Specified by client when subscription is created or provisioned	WrtcsAcceptanceNotification	no	no	This operation notifies a client about a session invitation acceptance by the Terminating Participant, or the session update acceptance by the Update Recipient.	no
Client notification about update offer in a WebRTC session	Specified by client when subscription is created or provisioned	WrtcsOfferNotification	no	no	This operation notifies a client about a new offer	no
Client notification about answer in a WebRTC session	Specified by client when subscription is created or provisioned	WrtcsAnswerNotification	no	no	This operation notifies a client about an answer	no

Client notification about subscription cancellation	Specified by client when subscription is created or provisioned	WrtcsSubscriptionCancellationNotification	no	no	This operation notifies a client about the cancellation of a subscription	no
Client notification about conflicts	Specified by client when subscription is created or provisioned	WrtcsConflictNotification	no	no	This operation notifies the client about a conflict that violates the offer-answer sequence rules	no

5.2 Data Types

5.2.1 XML Namespaces

The XML namespace for the WebRTC Signaling data types is:

urn:oma:xml:rest:netapi:webrtcsignaling:1

The 'xsd' namespace prefix is used in the present document to refer to the XML Schema data types defined in XML Schema [XMLSchema1, XMLSchema2]. The 'common' namespace prefix is used in the present document to refer to the data types defined in [REST_NetAPI_Common] .The use of namespace prefixes such as 'xsd' is not semantically significant.

The XML schema for the data structures defined in the section below is given in [REST_SUP_WRTCSig].

5.2.2 Structures

The subsections of this section define the data structures used in the WebRTC Signaling API.

Some of the structures can be instantiated as so-called root elements.

For structures that contain elements which describe a user identifier, the statements in section 6 regarding 'tel', 'sip' and 'acr' URI schemes apply.

5.2.2.1 Type: WrtcsSubscriptionList

This type represents a list of subscriptions to notifications regarding WebRTC signaling events.

Element	Type	Optional	Description
wrtcsNotificationSubscription	WrtcsNotificationSubscription[0..unbounded]	Yes	Array of notification subscriptions.
resourceURL	xsd:anyURI	No	Self referring URL.

A root element named wrtcsSubscriptionList of type WrtcsSubscriptionList is allowed in response bodies.

5.2.2.2 Type: WrtcsNotificationSubscription

This type represents a subscription to notifications regarding WebRTC Signaling events targeted at a particular user.

Element	Type	Optional	Description
callbackReference	common:CallbackReference	No	Client's Notification URL and OPTIONAL callbackData.
duration	xsd:int	Yes	Period of time (in seconds) notifications are provided for. If set to “0” (zero), a default duration time, which is specified by the service policy, will be used. If the parameter is omitted, the notifications will continue until the maximum duration time, which is specified by the service policy, unless the notifications are stopped by deletion of subscription for notifications. This element MAY be given by the client during resource creation in order to signal the desired lifetime of the subscription. The server SHOULD return in this element the period of time for which the subscription will still be valid.
clientCorrelator	xsd:string	Yes	A correlator that the client can use to tag this particular resource representation during a request to create a resource on the server. This element MAY be present. Note: this allows the client to recover from communication failures during resource creation and therefore avoids duplicate subscriptions in such situations. In case the element is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.
resourceURL	xsd:anyURI	Yes	Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification. The resourceURL MUST also be included in responses to any HTTP method that returns an entity body, and in PUT requests.

A root element named wrtcsNotificationSubscription of type WrtcsNotificationSubscription is allowed in request and/or response bodies.

5.2.2.3 Type: WrtcsSession

This type represents a WebRTC session.

Element	Type	Optional	Description
originatorAddress	xsd:anyURI	Yes	The address (e.g. 'sip' URI, 'tel' URI, 'acr' URI) of the Originator. If originatorAddress is also part of the request URL, the two MUST have the same value. This element MAY be omitted by the client, in which case it SHALL be filled in by the server.
originatorName	xsd:string	Yes	Human readable name of the Originator. If this is omitted by the client it MAY be filled in by the server. The server MAY modify this field according to policies, e.g. to prevent spoofing.
tParticipantAddress	xsd:anyURI	No	The address (e.g. 'sip' URI, 'tel' URI, 'acr' URI) of the Terminating Participant. If tParticipantAddress is also part of the request URL, the two MUST have the same value.
tParticipantName	xsd:string	Yes	Human readable name of the Terminating Participant. This element MAY be omitted in resource-creating requests. The server MAY modify this field according to policies, e.g. to provide missing values.
status	SessionStatus	Yes	Status of the session. MAY be omitted in resource creation request, and MUST be included in all responses. Default: Initiated.
offer	WrtcsOffer	Yes	The offer, which MUST be present in a request from the application to the server to create a session. Note that the offer can be absent in a session created by the server as part of an offerless INVITE [RFC3261].
answer	WrtcsAnswer	Yes	The answer. This element is not present in case there is no answer yet, or the session invitation has been declined by the Terminating Participant. This element MUST NOT be present in a request from the application to the server to create a session.

update	WrtcsOffer	Yes	The last pending session update request. Once an update request has been accepted by the Update Recipient, it moves into the “offer” element. Once an update request has been rejected by the Update Recipient it is removed from the “WrtcsSession” structure. This element MUST NOT be present in a request from the application to the server to create a session.
clientCorrelator	xsd:string	Yes	A correlator that the client can use to tag this particular resource representation during a request to create a resource on the server. This element SHOULD be present. Note: this allows the client to recover from communication failures during resource creation and therefore avoids duplicate session creations in such situations. In case the element is present, the server SHALL not alter its value, and SHALL provide it as part of the representation of this resource. In case the field is not present, the server SHALL NOT generate it.
resourceURL	xsd:anyURI	Yes	Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification. The resourceURL MUST also be included in responses to any HTTP method that returns an entity body, and in PUT requests.

A root element named wrtcsSession of type WrtcsSession is allowed in request and/or response bodies.

5.2.2.4 Type: WrtcsAnswer

This type represents an answer in WebRTC signaling.

Element	Type	Optional	Description
type	OfferAnswerType	Yes	The type of the answer (i.e. whether this is a local or remote answer). This element is populated by the server and MUST NOT be populated by the application.
isProvisional	xsd:boolean	No	If set to “true”, this element signals that the answer is provisional (i.e. a pranswer according to [W3C_WebRTC]). If set to “false”, this element signals that the answer is final. The application SHALL always set this element to “false”. Note that it is assumed that an answer generated by the application cannot be provisional, however, that the server is allowed to mark answers as provisional.
sdp	xsd:string	Choice	An inlined session description in SDP format [RFC4566]. If XML syntax is used, the content of this element SHALL be embedded in a CDATA section.
sdpBase64	xsd:base64Binary	Choice	An inlined session description in SDP format [RFC4566] represented in the UTF-8 encoding, base64-encoded.
mediaIndicator	MediaIndicator [0..unbounded]	Yes	An indication of the media described in the offer or answer. This element SHOULD be instantiated by the server and MUST NOT be instantiated by the client.
allowVideoUpgrade	xsd:boolean	Yes	This OPTIONAL element signals whether the answerer allows upgrading an audio-only session to an audio/video session. • true: The audio-only call can be upgraded to an audio&video call • false: The audio-only call cannot be upgraded to an audio&video call • not present: It is unknown whether the audio-only call can be upgraded to an audio&video call It depends on the actual underlying network(s) whether or not this information can be conveyed end-to-end.

A root element named wrtcsAnswer of type WrtcsAnswer is allowed in request and/or response bodies.

XSD modelling uses a “choice” to select either “sdp” or “sdpBase64”, but neither both nor none of them.

5.2.2.5 Type: WrtcsOffer

This type represents an offer in WebRTC signaling. Such an offer is either the initial offer in a session, or an update request).

Element	Type	Optional	Description
type	OfferAnswerType	Yes	The type of the offer (i.e. whether this is a local or remote offer). This element is populated by the server and MUST NOT be populated by the application.
holdAlerting	xsd:boolean	Yes	If this element is present and set to “true”, the application is requested not to alert the user yet until another offer is provided with this flag set to “false” or absent. This element is only meaningful in notifications. Hence, it is filled by the server and has no meaning in requests from the client. Note: The purpose of this flag is to avoid alerting the user as long as there is no path for the media of this call.
sdp	xsd:string	Choice	An inlined session description in SDP format [RFC4566]. If XML syntax is used, the content of this element SHALL be embedded in a CDATA section.
sdpBase64	xsd:base64Binary	Choice	An inlined session description in SDP format [RFC4566] represented in the UTF-8 encoding, base64-encoded.
mediaIndicator	MediaIndicator [0..unbounded]	Yes	An indication of the media described in the offer or answer. This element SHOULD be instantiated by the server and MUST NOT be instantiated by the client.
serviceType	xsd:string	Yes	A string indicating the service type. This element is deployment-specific and can be detailed in profiles if needed. It depends on the actual underlying network protocols and/or proxies whether this information can be conveyed end-to-end.

allowVideoUpgrade	xsd:boolean	Yes	This OPTIONAL element signals whether the offerer allows upgrading an audio-only session to an audio/video session. • true: The audio-only call can be upgraded to an audio&video call • false: The audio-only call cannot be upgraded to an audio&video call • not present: It is unknown whether the audio-only call can be upgraded to an audio&video call It depends on the actual underlying network protocols and/or proxies whether or not this information can be conveyed end-to-end.

A root element named wrtcsOffer of type WrtcsOffer is allowed in request bodies.

XSD modelling uses a “choice” to select either “sdp” or “sdpBase64”, but neither both nor none of them.

5.2.2.6 Type: MediaIndicator

This type represents a media indicator. Typically, this corresponds to one distinct stream of media (audio, video) as usually indicated in an m-line in SDP [RFC4566].

Element	Type	Optional	Description
type	MediaType	No	Indicates whether this is an audio, video or data stream.
entryIdx	xsd:unsignedInt	No	The index of the entry in the SDP for correlation purposes, starting at 0.
entryId	xsd:string	Yes	The identifier of the entry in the SDP for correlation purposes. Maps to the a=mid SDP attribute [RFC5888]
streamId	xsd:string	Yes	The identifier of the media stream in the SDP for correlation purposes. Maps to the a=msid SDP attribute [IETF_Msid_draft]. MUST be present if trackId is present.
trackId	xsd:string	Yes	The identifier of the media stream track in the SDP for correlation purposes. Maps to the a=msid SDP attribute [IETF_Msid_draft]. MUST be present if streamId is present.
payload	PayloadIndicator [0..unbounded]	Yes	The payload type from the SDP. MUST be instantiated if “type” is equal to “audio” or “video”.
direction	MediaDirection	Yes	The direction of the media. MUST be instantiated if “type” is equal to “audio” or “video”. The default is “SendRecv”.

5.2.2.7 Type: PayloadIndicator

This type represents a payload indicator. Typically, this corresponds to the payload type number and associated format parameters in SDP [RFC4566].

Element	Type	Optional	Description
payloadType	xsd:unsignedInt	No	Payload type identifier from SDP [RFC4566].
encoding	xsd:string	Yes	Encoding of the media. Maps to the “a=rtpmap” information in the SDP as defined in [RFC4566] excluding the <payload type> field.
formatParams	xsd:string	Yes	Media format parameters. Maps to the “a=fmtp” information in the SDP as defined in [RFC4566] excluding the <payload type> field.

5.2.2.8 Type: WrtcsIceStatus

This type represents the ICE status.

Element	Type	Optional	Description
status	IceStatus	No	The ICE status.

A root element named wrtcsIceStatus of type WrtcsIceStatus is allowed in request and/or response bodies.

The application MUST report the finalization of ICE to the server, using this structure.

5.2.2.9 Type: WrtcsSessionStatus

This type represents the session status.

Element	Type	Optional	Description
status	SessionStatus	No	The session status.

A root element named wrtcsSessionStatus of type WrtcsSessionStatus is allowed in request and/or response bodies.

5.2.2.10 Type: WrtcsEventNotification

This type represents a general notification related to WebRTC signaling.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	The 'callbackData' element if it was passed by the application in the 'callbackReference' element when creating a subscription to notifications about WebRTC signaling events. See [REST_NetAPI_Common]
link	common:Link [0..unbounded]	Yes	Links to other resources that are in relationship to the notification (e.g. related WebRTC session). Depending on the value of eventType, the server MUST include links as defined by the actual Notification resource in section 6.10. Further, the server SHOULD include a link to the related subscription.
eventType	EventType	No	Type of event.
eventDescription	xsd:string	Yes	Textual description of the event.

A root element named wrtcsEventNotification of type WrtcsEventNotification is allowed in notification request bodies.

5.2.2.11 Type: WrtcsSessionInvitationNotification

This type represents the notification for a session invitation.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	The 'callbackData' element if it was passed by the application in the 'callbackReference' element when creating a subscription to notifications about WebRTC signaling events. See [REST_NetAPI_Common]
link	common:Link [0..unbounded]	Yes	Links to other resources that are in relationship to the notification (e.g. related WebRTC session). The server MUST include links as defined by the actual Notification resource in section 6.11. Further, the server SHOULD include a link to the related subscription.
originatorAddress	xsd:anyURI	Yes	The address (e.g. 'sip' URI, 'tel' URI, 'acr' URI) of the Originator. If this element is missing, the Originator is unknown.
originatorName	xsd:string	Yes	Human readable name of the Originator.
tParticipantAddress	xsd:anyURI	Yes	The address (e.g. 'sip' URI, 'tel' URI, 'acr' URI) of the Terminating Participant.

tParticipantName	xsd:string	Yes	Human readable name of the Terminating Participant.
forceConnectingWithoutMedia	xsd:boolean	Yes	If this element is true, the application MUST go ahead and immediately alert the user and obtain user response to be able to connect, even though ICE checks have not been completed. Default: false. Note: This is necessary to support networks which can only initiate ICE checks once the user has accepted the call invitation.
offer	WrtcsOffer	Yes	The actual offer from the Originator. This MUST be present, unless the notification represents an offerless INVITE [RFC3261].

A root element named wrtcsSessionInvitationNotification of type WrtcsSessionInvitationNotification is allowed in notification request bodies.

5.2.2.12 Type: WrtcsAcceptanceNotification

This type represents the notification about acceptance of a session invitation / session update.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	The ‘callbackData’ element if it was passed by the application in the ‘callbackReference’ element when creating a subscription to notifications about WebRTC signaling events. See [REST_NetAPI_Common]
link	common:Link [0..unbounded]	Yes	Links to other resources that are in relationship to the notification (e.g. related WebRTC session). The server MUST include links as defined by the actual Notification resource in section 6.12. Further, the server SHOULD include a link to the related subscription.
answer	WrtcsAnswer	Yes	The actual answer from the Terminating Participant or Update Recipient. Note that it depends on the network status whether or not this element is present. If it is not present, the server MUST have provided an answer to the client already in an earlier WrtcsAnswerNotification.

A root element named wrtcsAcceptanceNotification of type WrtcsAcceptanceNotification is allowed in notification request bodies.

5.2.2.13 Type: WrtcsOfferNotification

This type represents the notifications that carry an offer from the network to the application.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	The ‘callbackData’ element if it was passed by the application in the ‘callbackReference’ element when creating a subscription to notifications about WebRTC signaling events. See [REST_NetAPI_Common]
link	common:Link [0..unbounded]	Yes	Links to other resources that are in relationship to the notification (e.g. related WebRTC session). The server MUST include links as defined by the actual Notification resource in section 6.13. Further, the server SHOULD include a link to the related subscription.
offer	WrtcsOffer	No	The actual offer.

A root element named wrtcsOfferNotification of type WrtcsOfferNotification is allowed in notification request bodies.

5.2.2.14 Type: WrtcsAnswerNotification

This type represents the notifications that carry an answer from the network to the application.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	The ‘callbackData’ element if it was passed by the application in the ‘callbackReference’ element when creating a subscription to notifications about WebRTC signaling events. See [REST_NetAPI_Common]
link	common:Link [0..unbounded]	Yes	Links to other resources that are in relationship to the notification (e.g. related WebRTC session). The server MUST include links as defined by the actual Notification resource in section 6.14. Further, the server SHOULD include a link to the related subscription.
answer	WrtcsAnswer	No	The actual (provisional or final) answer.

A root element named wrtcsAnswerNotification of type WrtcsAnswerNotification is allowed in notification request bodies.

5.2.2.15 Type: WrtcsSubscriptionCancellationNotification

This type represents subscription cancellation notifications.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	CallbackData if passed by the application in the receiptRequest element during the associated subscription operation. See [REST_NetAPI_Common] for details.
link	common:Link[1..unbounded]	No	Link to other resources that are in relationship with the resource. There MUST be a link to the subscription that is cancelled (see section 6.15).
reason	common:ServiceError	Yes	Reason why subscription is being discontinued. SHOULD be present if the reason is different from a regular expiry of the subscription.

A root element named wrtcsSubscriptionCancellationNotification of type WrtcsSubscriptionCancellationNotification is allowed in notification request bodies.

5.2.2.16 Type: WrtcsConflictNotification

This type represents conflict notifications.

Element	Type	Optional	Description
callbackData	xsd:string	Yes	CallbackData if passed by the application in the receiptRequest element during the associated subscription operation. See [REST_NetAPI_Common] for details.
link	common:Link[1..unbounded]	No	Link to other resources that are in relationship with the resource. The server MUST include links as defined by the actual Notification resource in section 6.16. Further, the server SHOULD include a link to the related subscription.
reason	common:ServiceError	Yes	Exception payload that indicates an offer conflict.

A root element named wrtcsConflictNotification of type WrtcsConflictNotification is allowed in notification request bodies.

5.2.3 Enumerations

The subsections of this section define the enumerations used in the WebRTC Signaling API.

5.2.3.1 Enumeration: EventType

This enumeration is defines the types of events. It is used in notifications.

Enumeration	Description
Cancelled	The Originator has cancelled the session during the invite phase, or has cancelled an unanswered update offer.
SessionEnded	The session has ended.
Declined	The Terminating Participant has declined the session invitation, or the Update Recipient has declined the update.
NoAnswer	The session invitation to the Terminating Participant has timed out.
NotReachable	The Terminating Participant could not be reached or is unknown.
Ringing	The Terminating Participant is being alerted of the incoming call invitation (“phone ringing”).
Busy	The Terminating Participant is busy.

5.2.3.2 Enumeration: SessionStatus

This enumeration defines the status of a WebRTC session.

Enumeration	Description
Initiated	The session was initiated but is not yet connected.
Ringing	The terminating participant is being alerted.
Connected	The session is established.
Closed	The session was closed. Resources representing closed sessions can be removed from the server immediately, or after a time period defined by service provider policies. Note that this state is e.g. reached if the remote Participant closes a session, or if the Terminating Participant does not accept a session invitation.

5.2.3.3 Enumeration: IceStatus

This enumeration provides the possible values of the ICE status in a WebRTC session based on the definitions in [W3C_WebRTC].

Enumeration	Description
New	The ICE status is “new” [W3C_WebRTC].
Checking	The ICE status is “checking” [W3C_WebRTC].
Connected	ICE connectivity checks have established one connection for each flow [W3C_WebRTC].
Completed	The ICE status is “completed” [W3C_WebRTC].
Failed	ICE connectivity checks have finished and connectivity could *not* be established for all flows (but possibly for some) [W3C_WebRTC].
Disconnected	The ICE status is “disconnected” [W3C_WebRTC].
Closed	The ICE status is “closed” [W3C_WebRTC].

5.2.3.4 Enumeration: MediaType

This enumeration defines the possible media types in the MediaIndicator data type.

Enumeration	Description
Audio	Represents an audio stream (m=audio in SDP [RFC4566]).
Video	Represents a video stream (m=video in SDP [RFC4566]).
Data	Represents a data channel [IETF_SCTP_SDP_draft].

5.2.3.5 Enumeration: MediaDirection

This enumeration defines the possible media directions in the MediaIndicator data type.

Enumeration	Description
SendRecv	The stream is bidirectional [RFC3264].
SendOnly	The stream is send-only [RFC3264].
RecvOnly	The stream is receive-only [RFC3264].
Inactive	The stream is currently not active [RFC3264].

5.2.3.6 Enumeration: OfferAnswerType

This enumeration determines whether an offer resp. answer is local or remote.

Enumeration	Description
Local	The offer or answer is a local one.
Remote	The offer or answer is a remote one.

5.2.4 Values of the Link “rel” attribute

The “rel” attribute of the Link element is a free string set by the server implementation, to indicate a relationship between the current resource and an external resource. The following are possible strings (list is non-exhaustive, and can be extended):

- WrtcsSubscriptionList
- WrtcsNotificationSubscription
- WrtcsSession
- WrtcsAnswer
- WrtcsOffer
- WrtcsIceStatus
- WrtcsSessionStatus

These values indicate the kind of resource that the link points to.

5.3 Sequence Diagrams

The following subsections describe the resources, methods and steps involved in typical scenarios.

In a sequence diagram, a step which involves delivering a notification is labeled with “POST or NOTIFY”, where “POST” refers to delivery via the HTTP POST method, and “NOTIFY” refers to delivery using the Notification Channel [REST_NetAPI_NotificationChannel].

The WebRTC Signaling API has been designed to work based on SDP, offer/answer with the additional pranswer state and ICE as defined by JSEP. A web browser which exposes the WebRTC API [W3C_WebRTC] is a typical runtime environment for an application using the WebRTC Signaling API, even though other instances such as native or server-side applications can use it as well..If the application is running in a WebRTC-enabled web browser, the web browser serves as the media engine for the session. By design, as the WebRTC Signaling API only covers the signalling part, the API needs to make assumptions w.r.t. the media engine. To illustrate how application, Network API and media engine interwork, the browser runtime environment which includes a media engine with an interface based on the W3C WebRTC API has been chosen to be depicted in the flow diagrams in this section. Therefore, many of the diagrams in this section include the web browser as an actor, and mention the PeerConnection object of the WebRTC API. Note that any media engine which is aligned with the JSEP subset mentioned above can be used in a deployment instead, however, at the time of writing, the WebRTC-enabled browser is the only well-specified instance of such a media engine.

This version of the specification has been designed under the assumption that the server acts as a gateway towards a SIP [RFC3261] infrastructure. Appendix H provides a mapping from API calls to SIP messages. In the flows in this section, only the API view is shown, i.e. the messages between the server and the SIP infrastructure are not depicted.

The flows in the sections below contain messages and participants that are defined in this specification, as well as those that are not defined in this specification, but that informatively show the interworking with external components and systems. The legend below introduces the graphical styles used to distinguish between these categories.

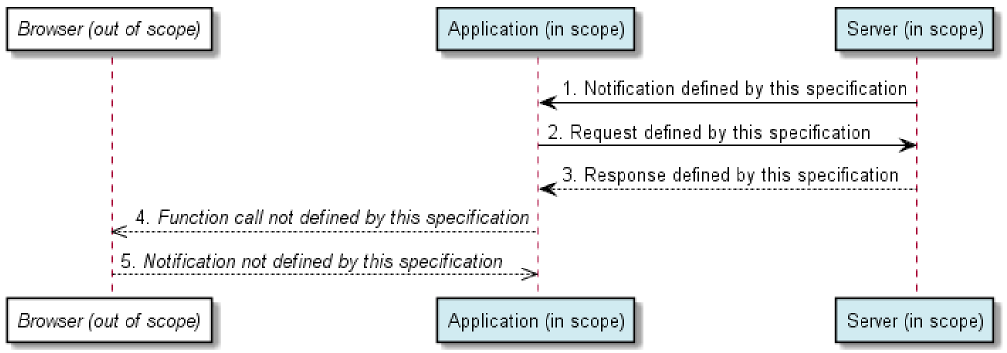
Browser (out of scope)
Application (in scope)
Server (in scope)
1. Notification defined by this specification
2. Request defined by this specification
3. Response defined by this specification
4. Function call not defined by this specification
5. Notification not defined by this specification

Figure 2: Legend for the sequence diagrams

5.3.1 Subscribing to and unsubscribing from WebRTC signaling notifications

This figure below shows a scenario for an application subscribing to and unsubscribing from WebRTC signaling notifications.

The notification URL passed by the client during the subscription step can be a Client-side Notification URL, or a Server-side Notification URL. Refer to [REST_NetAPI_NotificationChannel] for sequence flows illustrating the creation of a Notification Channel and obtaining a Server-side Notification URL on the server-side, and the use of that Notification Channel by the client.

The resources:

– To subscribe to WebRTC signaling notifications, create a new resource under **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/subscriptions**

– To cancel subscription to WebRTC signalingnotifications delete the resource under **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/subscriptions/{subscriptionId}**

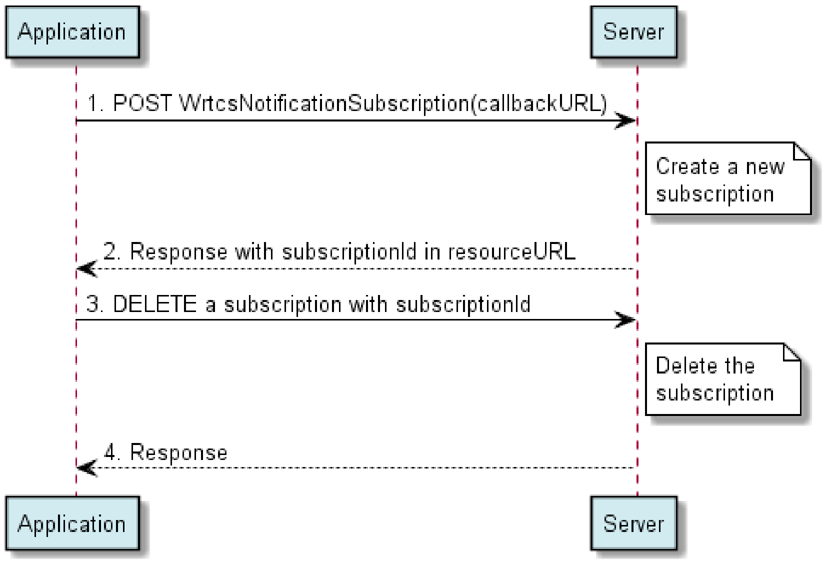
Application
Server
1. POST WrtcsNotificationSubscription(callbackURL)
Create a new subscription
2. Response with subscriptionId in resourceURL
3. DELETE a subscription with subscriptionId
Delete the subscription
4. Response

Figure 3: Subscribing to and unsubscribing from WebRTC signaling notifications

Outline of the flows:

1. An application subscribes to WebRTC signaling notifications using the POST method to submit the WrtcsNotificationSubscription data structure to the resource containing all subscriptions.
2. The server returns a response which comprises the result resource URL containing the subscriptionId.
3. The application stops receiving notifications using DELETE with the resource URL containing the subscriptionId.
4. The server returns a response.

5.3.2 Handling offers and answers

The WebRTC Signaling API is based on the offer-answer model. This means that the communicating parties control the media session by exchanging offers and answers which request and confirm changes to the connectivity, the media formats, the media flows used in the session (e.g. audio-only or also video) etc. In addition to offers and answers [RFC3264] which always occur in pairs in SIP [RFC3261], the WebRTC specification [W3C_WebRTC] also allows provisional answers (pranswer). To respond to an offer, zero or more pranswers can be provided before the final answer. Pranswers MAY be provided by the server to the application in a notification, but MUST NOT be provided by the application to the server. It is the responsibility of the server to ensure the correct mapping between SIP answers and WebRTC pranswer / answer primitives. The application MUST NOT generate answers of type pranswer.

The figure below shows how an application MUST handle offers, pranswers and answers. It is assumed as a prerequisite for the flow below that the session has been created previously, and that any previous offer-answer exchange has been completed with an answer (i.e. there is no outstanding answer).

The resources:

- To submit an answer to an incoming offer, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/answer**
- To submit an offer to modify the current session, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/update**

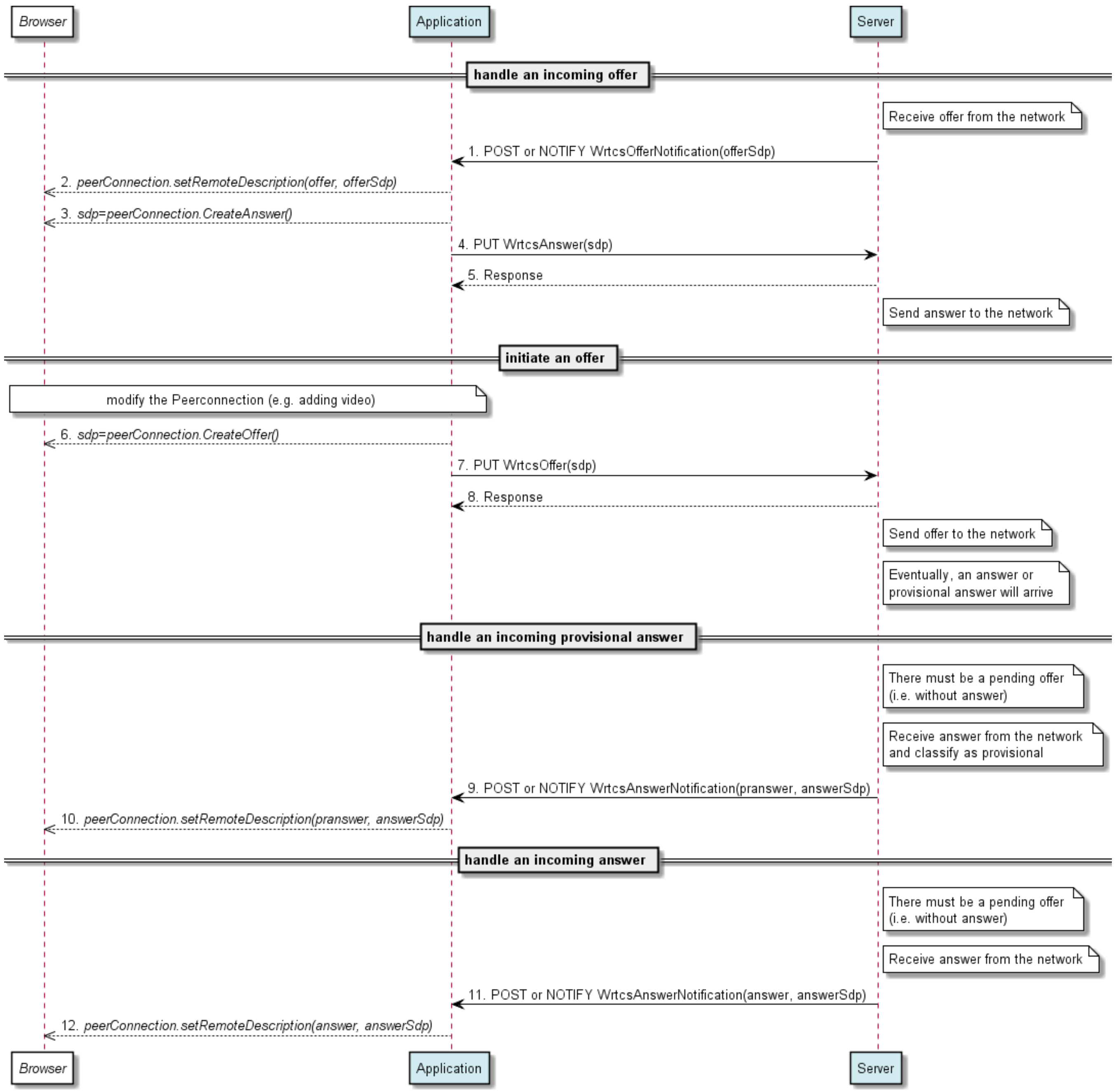
Browser
Application
Server
handle an incoming offer
Receive offer from the network
1. POST or NOTIFY WrtcsOfferNotification(offerSdp)
2. peerConnection.setRemoteDescription(offer, offerSdp)
3. sdp=peerConnection.CreateAnswer()
4. PUT WrtcsAnswer(sdp)
5. Response
Send answer to the network
initiate an offer
modify the Peerconnection (e.g. adding video)
6. sdp=peerConnection.CreateOffer()
7. PUT WrtcsOffer(sdp)
8. Response
Send offer to the network
Eventually, an answer or provisional answer will arrive
handle an incoming provisional answer
There must be a pending offer (i.e. without answer)
Receive answer from the network and classify as provisional
9. POST or NOTIFY WrtcsAnswerNotification(pranswer, answerSdp)
10. peerConnection.setRemoteDescription(pranswer, answerSdp)
handle an incoming answer
There must be a pending offer (i.e. without answer)
Receive answer from the network
11. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp)
12. peerConnection.setRemoteDescription(answer, answerSdp)
Browser
Application
Server

Figure 4: Offer and answer handling

Outline of the flows when receiving an offer:

1. The application receives a WrtcsOfferNotification that carries the offer.
2. The application provides this offer as the remote description to the PeerConnection object in the browser which is associated with the session.
3. The application requests the PeerConnection object to create an answer.
4. The application provides the answer to the server by updating the resource representing the answer in the session.

5. The server returns a response. Subsequently, the server takes care of sending the answer in an appropriate way to the network infrastructure.

Outline of the flows when initiating an offer:

6. After modifying some aspect of the session locally in the browser (e.g. adding a video stream), the application requests the PeerConnection object in the browser that is associated with this session to create an offer which reflects the changes.
7. The application provides the offer to the server by updating the resource representing the update offer in the session.
8. The server returns a response. Subsequently, the server takes care of sending the offer in an appropriate way to the network infrastructure, and waits for an answer.

Outline of the flows when receiving a provisional answer. Note that subsequent to an offer, zero or more provisional answers may be received.

9. After the server has received an answer from the network infrastructure and has detected that it qualifies as a pranswer, the server sends to the application a WrtcsAnswerNotification that carries the provisional answer.
10. The application provides this provisional answer as the remote description to the PeerConnection object in the browser which is associated with the session.

Outline of the flows when receiving an answer. Note that subsequent to an answer and prior to the next offer, more provisional answers MUST NOT be received.

11. After the server has received an answer from the network infrastructure, the server sends to the application a WrtcsAnswerNotification that carries the answer.
12. The application provides this provisional answer as the remote description to the PeerConnection object in the browser which is associated with the session.

5.3.3 Normal signaling flow of a WebRTC session - Originator

The figure below shows a scenario for the signaling in a WebRTC session with successful result from the point of view of the Originator.

The resources:

- To start a WebRTC session, create a new resource with the WrtcsSession data structure under **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions**
- To report successful completion of the ICE procedures, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**
- To end a WebRTC session delete the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}**
- To indicate ICE status changes to the server, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

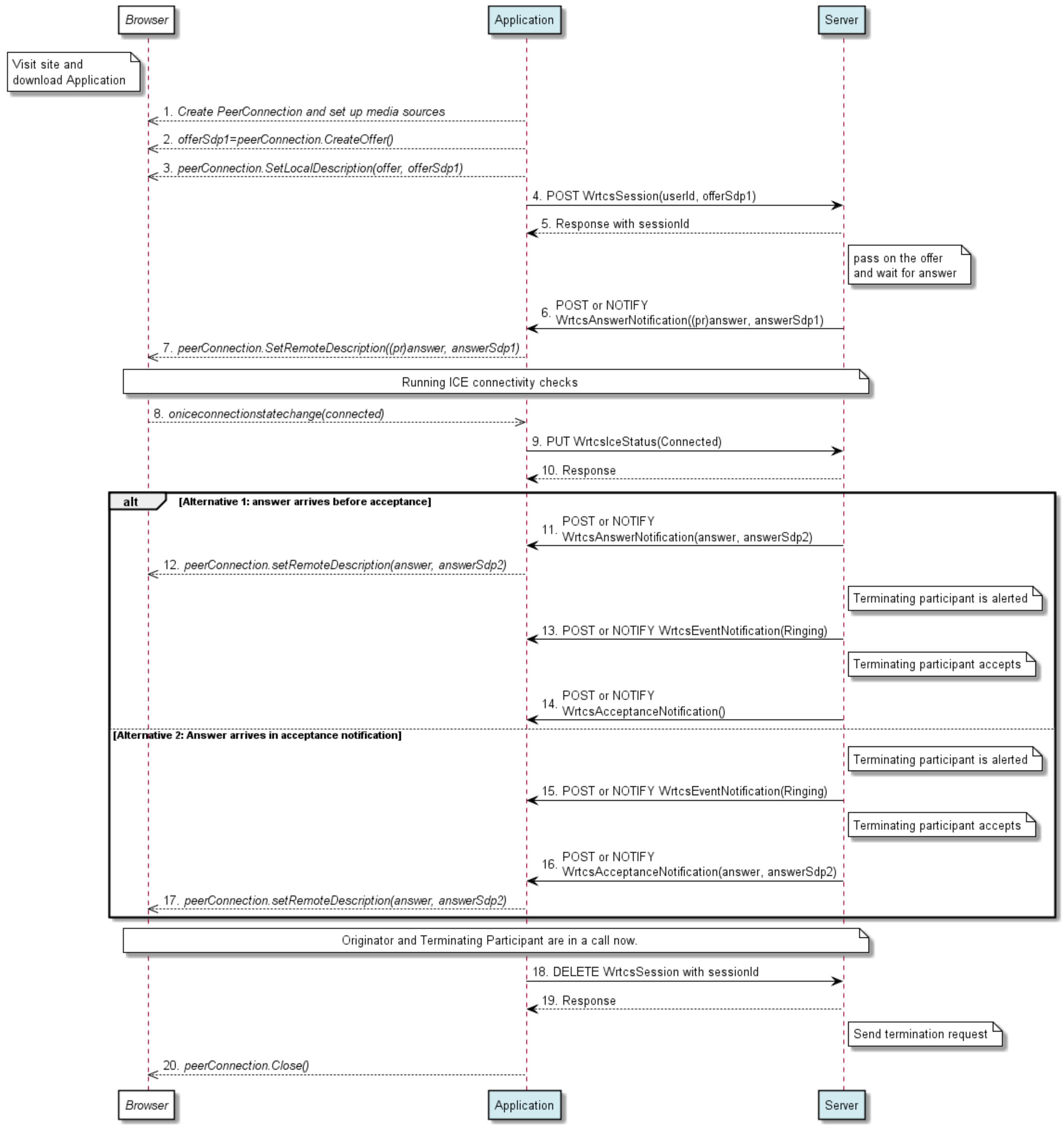
Browser
Application
Server
Visit site and download Application
1. Create PeerConnection and set up media sources
2. offerSdp1=peerConnection.CreateOffer()
3. peerConnection.SetLocalDescription(offer, offerSdp1)
4. POST WrtcsSession(userId, offerSdp1)
5. Response with sessionId
pass on the offer and wait for answer
6. POST or NOTIFY WrtcsAnswerNotification((pr)answer, answerSdp1)
7. peerConnection.SetRemoteDescription((pr)answer, answerSdp1)
Running ICE connectivity checks
8. oniceconnectionstatechange(connected)
9. PUT WrtcsIceStatus(Connected)
10. Response
alt
[Alternative 1: answer arrives before acceptance]
11. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp2)
12. peerConnection.setRemoteDescription(answer, answerSdp2)
Terminating participant is alerted
13. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
14. POST or NOTIFY WrtcsAcceptanceNotification()
[Alternative 2: Answer arrives in acceptance notification]
Terminating participant is alerted
15. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
16. POST or NOTIFY WrtcsAcceptanceNotification(answer, answerSdp2)
17. peerConnection.setRemoteDescription(answer, answerSdp2)
Originator and Terminating Participant are in a call now.
18. DELETE WrtcsSession with sessionId
19. Response
Send termination request
20. peerConnection.Close()
Browser
Application
Server

Figure 5: WebRTC session signaling - Originator

Outline of the flows:

1. The application creates a PeerConnection object in the browser and sets up the media sources.
2. The application retrieves from the PeerConnection object the initial offer that describes the media sources.
3. The application set the initial offer as the local session description in the PeerConnection object.
4. The application creates a new WebRTC session on the server using the POST method on the resource containg all WebRTC sessions, passing the identity of the Terminating Participant and the initial offer.

5. The server returns in the response to the POST request a resource URL that contains a session Id. This resource URL can be used in subsequent HTTP methods to identify the session.
6. Eventually, the server receives from the network infrastructure an answer or provisional answer to the initial offer which contains a session description that has been derived from the session description in the initial offer using the offer-answer model [RFC3264]. The server notifies the application of the answer or provisional answer by sending a WrtcsAnswerNotification to the application.
7. The application sets the received answer or provisional answer as the remote description in the PeerConnection object. Triggered by that, the browser starts the ICE connectivity checks.
8. The browser reports to the application that a media path has been established by doing ICE connectivity checks with the media endpoint that was signaled in the remote session description from the provisional answer. Note that this can be a media gateway, or the browser of the Terminating Participant, depending on network topology.
9. The application reports the successful ICE checks by updating the ICE status using the PUT method.
10. The server returns an HTTP response.

Alternative flow 1: Answer arrives before acceptance

11. Another answer from the Terminating Participant arrives, which is forwarded to the application using a WrtcsAnswerNotification. Note that if the answer in step 6 was not provisional, this step is omitted.
12. The application sets the received answer as the remote description in the PeerConnection object. Note that if the answer in step 6 was not provisional, this step is omitted.
13. Eventually, the server receives from the network infrastructure a message that the Terminating Participant is now being alerted of the incoming call. The server notifies the application of the fact that the Terminating Participant is being alerted by sending a WrtcsEventNotification of type “Ringing” to the application.
14. Eventually, the server receives from the network infrastructure a message that the Terminating Participant has accepted the invitation to the call. The server notifies the application of the fact that the Terminating Participant has accepted the call by sending a WrtcsAcceptanceNotification to the application. Since the answer was already sent, the notification does not contain an answer. The call is now established.

Alternative flow 2: Answer arrives in acceptance notification (see also H.1.1**)**

15. Eventually, the server receives from the network infrastructure a message that the Terminating Participant is now being alerted of the incoming call. The server notifies the application of the fact that the Terminating Participant is being alerted by sending a WrtcsEventNotification of type “Ringing” to the application.
16. Eventually, the server receives from the network infrastructure a message that the Terminating Participant has accepted the invitation to the call. This notification also includes the answer, assuming the previous answer was provisional. The server notifies the application of the fact that the Terminating Participant has accepted the call by sending a WrtcsAcceptanceNotification to the application.
17. The application sets the received session description as the remote description in the PeerConnection object. The call is now established.

End of alternatives.

18. To terminate the call, the application uses the DELETE method on the session resource, addressed by the resourceURL containing the session Id.
19. The server indicates in the response that the deletion was successful, and sends a termination request towards the network infrastructure.
20. The application cleans up the local browser resources and terminates the media capturing / rendering by invoking the “Close” method of the PeerConnection object.

Appendix H describes how this flow can be mapped to SIP.Note that other combinations of pranswer and answer are possible depending on the message exchange taking place in the underlying infrastructure. In those cases, the general rules for handling offers, pranswers and answers in section 5.3.2 apply.

5.3.4 Normal signaling flow of a WebRTC session – Terminating Participant

The figure below shows a scenario for the signaling in a WebRTC session with successful result from the point of view of the Terminating Participant.

The resources:

- To accept a WebRTC session invitation, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**
- To provide an answer without accepting the session invitation, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/answer**
- To indicate that the user is being alerted of an incoming call, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**
- To end a WebRTC session delete the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}**
- To indicate to the server changes of the ICE status, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

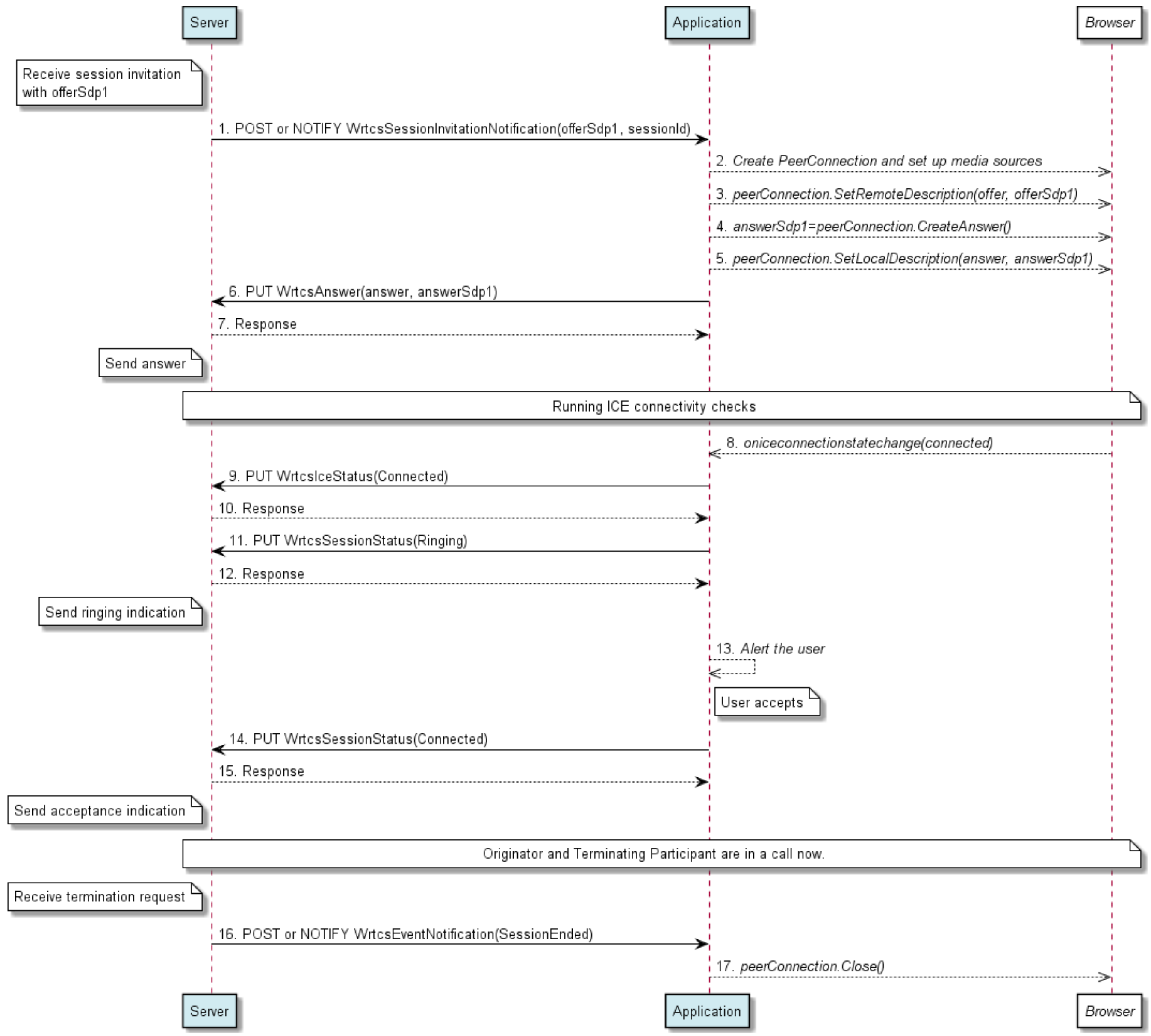
Server
Application
Browser
Receive session invitation with offerSdp1
1. POST or NOTIFY WrtcsSessionInvitationNotification(offerSdp1, sessionId)
2. Create PeerConnection and set up media sources
3. peerConnection.SetRemoteDescription(offer, offerSdp1)
4. answerSdp1=peerConnection.CreateAnswer()
5. peerConnection.SetLocalDescription(answer, answerSdp1)
6. PUT WrtcsAnswer(answer, answerSdp1)
7. Response
Send answer
Running ICE connectivity checks
8. oniceconnectionstatechange(connected)
9. PUT WrtcsIceStatus(Connected)
10. Response
11. PUT WrtcsSessionStatus(Ringing)
12. Response
Send ringing indication
13. Alert the user
User accepts
14. PUT WrtcsSessionStatus(Connected)
15. Response
Send acceptance indication
Originator and Terminating Participant are in a call now.
Receive termination request
16. POST or NOTIFY WrtcsEventNotification(SessionEnded)
17. peerConnection.Close()
Server
Application
Browser

Figure 6: WebRTC session signaling – Terminating Participant

Outline of the flows:

1. The server receives from the network infrastructure an invitation to participate in a real-time media session, containing an initial offer according to the offer-answer model [RFC3264]. The server informs the application of that invitation by sending it a WrtcsSessionInvitationNotification which includes the offer's session description and a resource URL that identifies the session to be used in subsequent requests.
2. The application creates a PeerConnection object in the browser, sets up the local media sources and adds these to the PeerConnection object.
3. The application sets the session description received in the initial offer as the remote description in the PeerConnection object.
4. The application retrieves an answer to the offer from the PeerConnection object.
5. The application sets the retrieved session description as the local description in the PeerConnection object, marked as answer. Triggered by that, the browser starts the ICE connectivity checks.

6. The application uses the PUT method on the resource containing the answer to provide that session description to the server as the answer.
7. The server returns an HTTP response to the request and sends the answer towards the network infrastructure, which takes care of routing the answer to the Originator.
8. The browser reports to the application that a media path has been established by doing ICE connectivity checks with the media endpoint that was signaled in the remote session description from the offer. Note that this can be a media gateway, or the browser of the Originator, depending on network topology.
9. The application reports the successful ICE checks by updating the ICE status using the PUT method.
10. The server returns an HTTP response to the request.
11. The application can now alert the user. First, the application reports to the server that it is alerting the user, by setting the session status resource to "Ringing" using the PUT method.
12. The server returns an HTTP response to the request. Subsequently, the server sends towards the network infrastructure an indication that the Terminating Participant is being alerted. The network infrastructure takes care of routing the information to the Originator.
13. The application now alerts the user, who eventually accepts the call.
14. The application uses the PUT method on the resource containing the session status to accept the call by updating the status.
15. The server returns a response to the request and sends the call acceptance indication towards the network infrastructure, which takes care of routing the information to the Originator. The call is now established.
16. Eventually, the server receives from the network infrastructure a message that the Originator is requesting to terminate the call. The server notifies the application of the call session termination by sending a WrtcsEventNotification of type "SessionEnded" to the application.
17. The application cleans up the local browser resources and terminates the media capturing / rendering by invoking the "Close" method of the PeerConnection object.

5.3.5 Signaling flow of a WebRTC session with delayed alerting

The following flow shows how the Terminating Participant's application responds to a session invitation in which the Originator has requested to delay the alerting of the Terminating Participant until the Originator sends an updated offer. Using this pattern, the Originator requests the Terminating Participant to delay the alerting of the user until the Originator reports that media connectivity is possible (e.g. ICE checks have been succeeded, or QoS reservations have been granted).

The flow is shown from the Terminating Participant's point of view. Note that it is assumed that the Originator's server handles the necessary steps at the Originator side, i.e. an Originator using the API defined by this specification will never have to generate an invitation requesting delayed alerting. It is however necessary that a terminating participant is able to handle invitation of this type.

The resources:

- To provide an answer to a session invitation without accepting it (yet), or to provide an answer to an offer, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/answer**

- To accept a WebRTC session invitation, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**

- To indicate that the user is being alerted of an incoming call, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**

- To indicate to the server changes of the ICE status, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

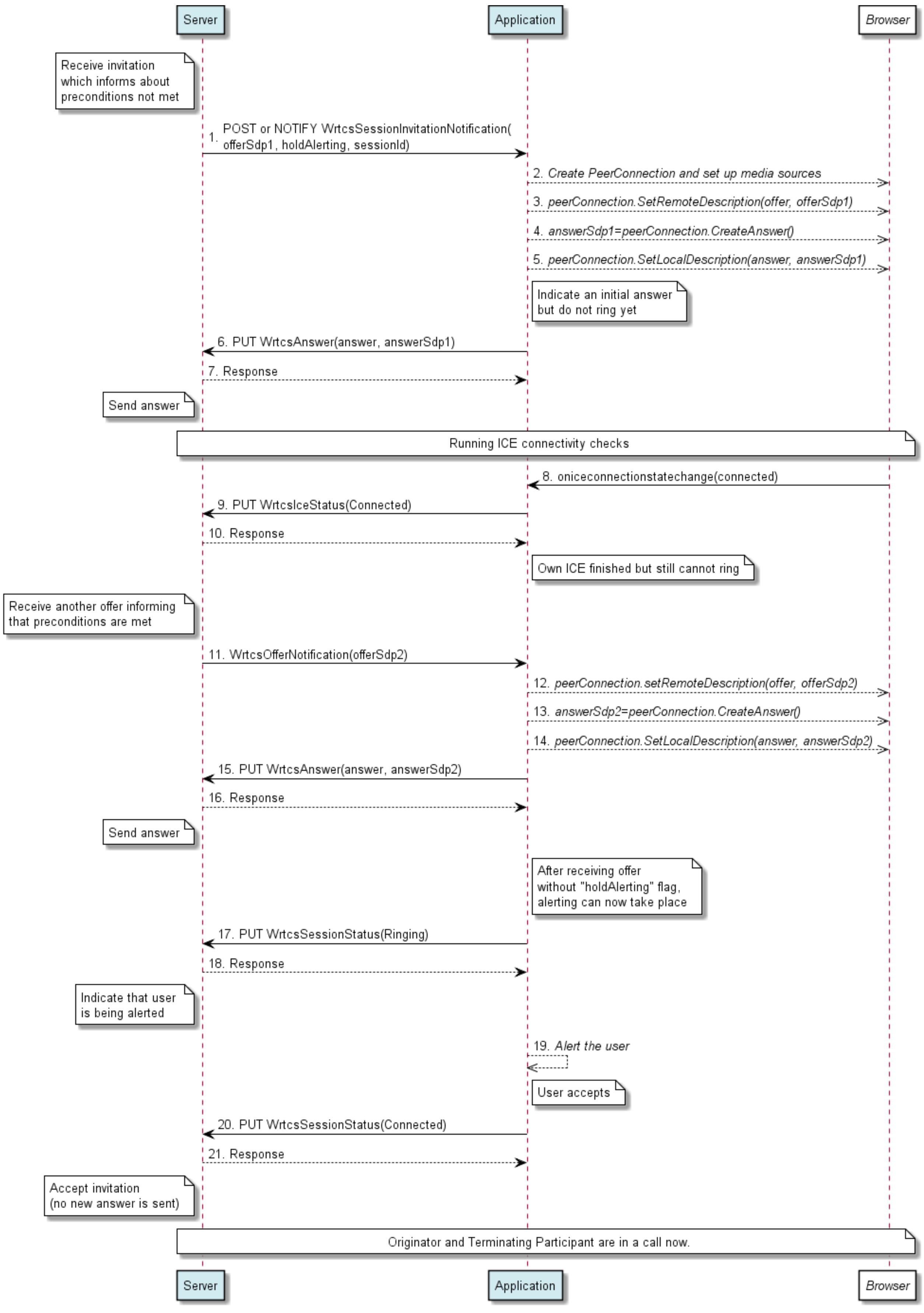
Server
Application
Browser
Receive invitation which informs about preconditions not met
1. POST or NOTIFY WrtcsSessionInvitationNotification(offerSdp1, holdAlerting, sessionId)
2. *Create PeerConnection and set up media sources*
3. *peerConnection.SetRemoteDescription(offer, offerSdp1)*
4. *answerSdp1=peerConnection.CreateAnswer()*
5. *peerConnection.SetLocalDescription(answer, answerSdp1)*
Indicate an initial answer but do not ring yet
6. PUT WrtcsAnswer(answer, answerSdp1)
7. Response
Send answer
Running ICE connectivity checks
8. oniceconnectionstatechange(connected)
9. PUT WrtcsIceStatus(Connected)
10. Response
Own ICE finished but still cannot ring
Receive another offer informing that preconditions are met
11. WrtcsOfferNotification(offerSdp2)
12. *peerConnection.setRemoteDescription(offer, offerSdp2)*
13. *answerSdp2=peerConnection.CreateAnswer()*
14. *peerConnection.SetLocalDescription(answer, answerSdp2)*
15. PUT WrtcsAnswer(answer, answerSdp2)
16. Response
Send answer
After receiving offer without "holdAlerting" flag, alerting can now take place
17. PUT WrtcsSessionStatus(Ringing)
18. Response
Indicate that user is being alerted
19. *Alert the user*
User accepts
20. PUT WrtcsSessionStatus(Connected)
21. Response
Accept invitation (no new answer is sent)
Originator and Terminating Participant are in a call now.
Server
Application
Browser

Figure 7: Signaling flow of a WebRTC session with delayed alerting

Outline of the flows:

1. The server receives from the network infrastructure an invitation to participate in a real-time media session, containing an initial offer according to the offer-answer model [RFC3264]. This offer includes information that certain preconditions for opening the media connection such as QoS reservation or ICE checks, are not yet met. The server informs the application of that offer by sending it a WrtcsSessionInvitationNotification which includes the offer's session description, a resource URL that identifies the session to be used in subsequent requests, and an indicator that tells the application that it can go ahead processing the offer contained in the invitation, but should refrain from alerting the user.
2. The application creates a PeerConnection object in the browser, sets up the local media sources and adds these to the PeerConnection object.
3. The application sets the session description received in the initial offer as the remote description in the PeerConnection object.
4. The application retrieves an answer to the offer from the PeerConnection object.
5. The application sets the retrieved session description as the local description in the PeerConnection object, marked as answer. Triggered by that, the browser starts the ICE connectivity checks.
6. The application uses the PUT method on the resource containing the answer to provide that session description to the server as the answer.
7. The server returns an HTTP response to the request and sends the answer towards the network infrastructure, which takes care of routing the answer to the Originator. Subsequently, ICE connectivity checks are performed to set up media connectivity.
8. The browser reports to the application that a media path has been established by doing ICE connectivity checks with the media endpoint that was signaled in the remote session description from the offer. Note that this can be a media gateway, or the browser of the Originator, depending on network topology.
9. The application reports the successful ICE checks by updating the ICE status using the PUT method.
10. The server returns an HTTP response to the request. Subsequently, the application's own ICE procedures have finished, and a media path is available for the call leg of the Terminating Participant. However, there is no information yet available whether there is also media connectivity available for the call leg of the Originator; therefore, the application still cannot alert the user.
11. Eventually, the server receives information from the network infrastructure that media connectivity is now also available for the Originator's call leg. The server sends to the application a WrtcsOfferNotification which contains an updated offer without the "holdAlerting" flag instantiated.
12. The application sets the session description received in the updated offer as the remote description in the PeerConnection object.
13. The application retrieves an answer to the offer from the PeerConnection object.
14. The application sets the retrieved session description as the local description in the PeerConnection object, marked as answer.
15. The application uses the PUT method on the resource containing the answer to provide that session description to the server as the answer.
16. The server returns an HTTP response to the request and sends the answer towards the network infrastructure, which takes care of routing the answer to the Originator.
17. As the Originator has withdrawn the request to hold alerting, and as locally the ICE procedures have also succeeded, the application can now alert the user. First, the application reports to the server that it is alerting the user, by setting the session status resource to "Ringing" using the PUT method.

18. The server returns an HTTP response to the request. Subsequently, the server sends towards the network infrastructure an indication that the Terminating Participant is being alerted. The network infrastructure takes care of routing the information to the Originator.
19. The application now alerts the user, who eventually accepts the call.
20. The application uses the PUT method on the resource containing the session status to accept the call by updating the status.
21. The server returns a response to the request and sends the call acceptance indication towards the network infrastructure, which takes care of routing the information to the Originator. The call is now established.

Appendix H describes how this flow can be mapped to SIP.

5.3.6 Signaling flow with an offerless session invitation

The figure below shows a scenario where the Terminating Participant receives a WebRTC session invitation that contains no offer. Such invitations occur for instance in third-party call control scenarios, when the network expects the Terminating Participant to make an offer as response to the invitation. The flow is shown from the Terminating Participant's point of view.

Note: In third-party call control scenarios, the invites both call participants (so, strictly speaking, there are two Terminating Participants but no Originator). To do this, the network typically first invites one participant and asks it to declare its media properties, i.e. to provide an offer. This mechanism is known as "offerless invite". The offer received by the network in the response to the invitation is then included in the invitation sent to the second participant.

Note: Not all networks are able to forward an offer before the invited participant has actually accepted the invitation. Even though the call set-up takes longer in such networks and may lead to so-called "ghost-rings" (the participant is alerted of an incoming call but the establishment of the media path will fail later, making the call impossible), this API supports also such networks. The need to go ahead with call setup without waiting for an established media path (aka ICE completion) is signalled to the application by the flag "forceConnectingWithoutMedia", enforcing the application to immediately alert the user, rather than waiting for the establishment of a media path. However, due to the disadvantages outlined above, it is strongly advised that networks which are used with the RESTful Network API for WebRTC Signaling allow forwarding an offer as response to an offerless session invitation before the user actually is alerted.

The resources:

- To provide an offer responding to an offerless session invitation, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/offer**
- To accept a WebRTC session invitation, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**
- To indicate that the user is being alerted of an incoming call, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status**
- To indicate to the server changes of the ICE status, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

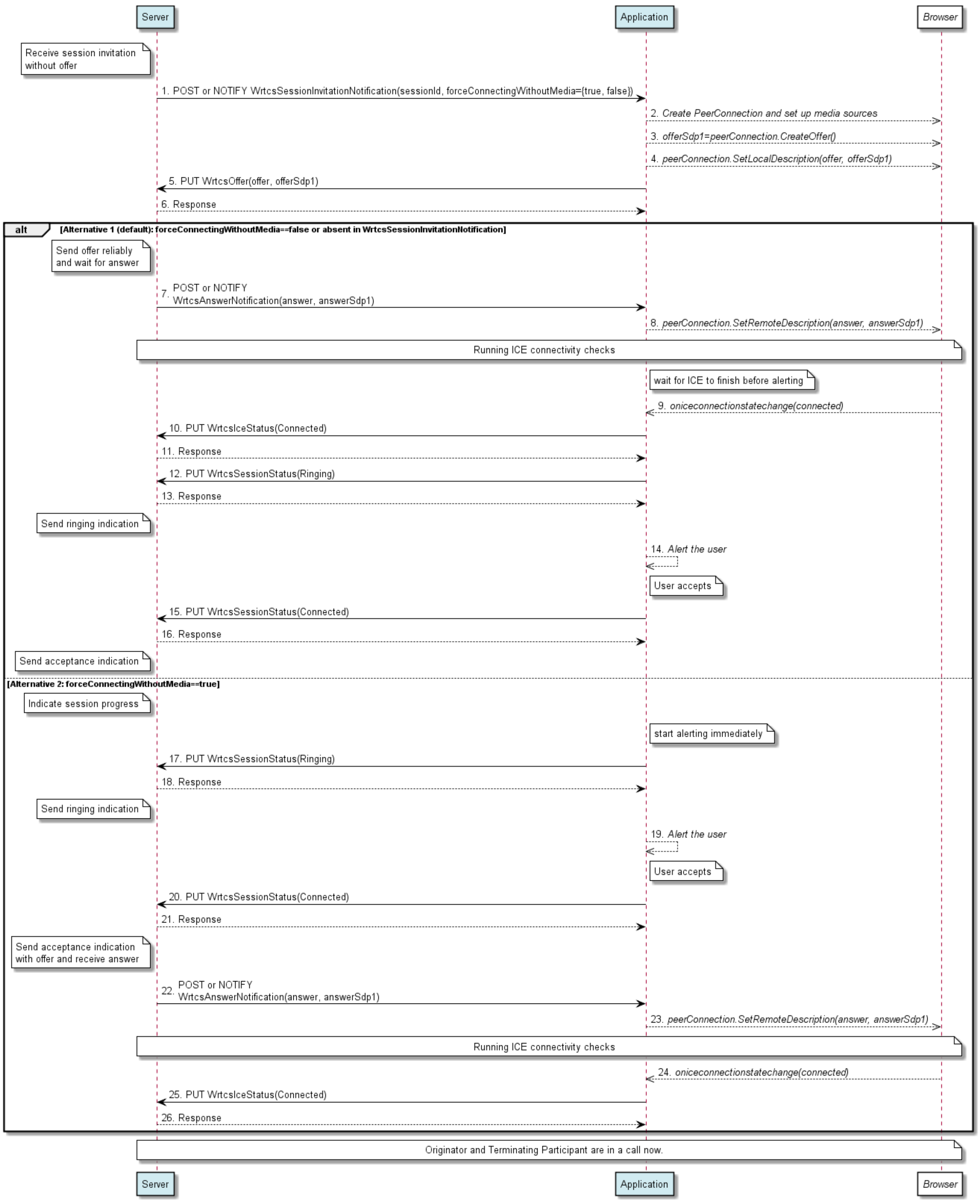
Server
Application
Browser
Receive session invitation without offer
1. POST or NOTIFY WrtcsSessionInvitationNotification(sessionId, forceConnectingWithoutMedia={true, false})
2. Create PeerConnection and set up media sources
3. offerSdp1=peerConnection.CreateOffer()
4. peerConnection.SetLocalDescription(offer, offerSdp1)
5. PUT WrtcsOffer(offer, offerSdp1)
6. Response
alt
[Alternative 1 (default): forceConnectingWithoutMedia==false or absent in WrtcsSessionInvitationNotification]
Send offer reliably and wait for answer
7. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp1)
8. peerConnection.SetRemoteDescription(answer, answerSdp1)
Running ICE connectivity checks
wait for ICE to finish before alerting
9. oniceconnectionstatechange(connected)
10. PUT WrtcsIceStatus(Connected)
11. Response
12. PUT WrtcsSessionStatus(Ringing)
13. Response
Send ringing indication
14. Alert the user
User accepts
15. PUT WrtcsSessionStatus(Connected)
16. Response
Send acceptance indication
[Alternative 2: forceConnectingWithoutMedia==true]
Indicate session progress
start alerting immediately
17. PUT WrtcsSessionStatus(Ringing)
18. Response
Send ringing indication
19. Alert the user
User accepts
20. PUT WrtcsSessionStatus(Connected)
21. Response
Send acceptance indication with offer and receive answer
22. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp1)
23. peerConnection.SetRemoteDescription(answer, answerSdp1)
Running ICE connectivity checks
24. oniceconnectionstatechange(connected)
25. PUT WrtcsIceStatus(Connected)
26. Response
Originator and Terminating Participant are in a call now.
Server
Application
Browser

Figure 8: Signaling flow with an offerless session invitation

Outline of the flows:

1. The server receives from the network infrastructure an invitation to participate in a real-time media session. This invitation does not include an offer. The server informs the application of that invitation by sending it a WrtcsSessionInvitationNotification which includes a resource URL that identifies the session to be used in subsequent requests, but no offer. If the server intends to enforce the application to go ahead immediately with alerting the user and connecting the call without waiting for the establishment of a media path, the server signals the "forceConnectingWithoutMedia" flag and sets it to true.
2. The application creates a PeerConnection object in the browser, sets up the local media sources and adds these to the PeerConnection object.
3. The application retrieves a session description from the PeerConnection object as an offer.
4. The application sets the retrieved session description as the local description in the PeerConnection object, marked as offer.
5. The application uses the PUT method on the resource containing the offer to provide that session description to the server as the offer.
6. The server returns an HTTP response to the request.

Alternative 1 (default): forceConnectingWithoutMedia=false or absent

7. After the server has sent the offer towards the network infrastructure, it waits for an answer. Eventually, the answer arrives from the network which is forwarded by the server to the application in a WrtcsAnswerNotification.
8. The application sets the session description contained in the notification as the remote description in the PeerConnection object, marked as answer. ICE connectivity checks can subsequently start. The application waits for the ICE connectivity checks to successfully finish before alerting the user.
9. The browser reports to the application that a media path has been established by doing ICE connectivity checks with the media endpoint that was signaled in the remote session description from the answer.
10. The application reports the successful ICE checks by updating the ICE status using the PUT method.
11. The server returns an HTTP response to the request.
12. As the ICE procedures have succeeded, the application can now alert the user. First, the application reports to the server that it is alerting the user, by setting the session status resource to "Ringing" using the PUT method.
13. The server returns an HTTP response to the request. Subsequently, the server sends towards the network infrastructure an indication that the participant is being alerted. The network infrastructure takes care of routing the information appropriately.
14. The application now alerts the user, who eventually accepts the call.
15. The application uses the PUT method on the resource containing the session status to accept the call by updating the status.
16. The server returns a response to the request and sends the call acceptance indication towards the network infrastructure, which takes care of routing the information appropriately.

Alternative 2: forceConnectingWithoutMedia=true

17. As requested by the server in step1 and signalled by the "forceConnectingWithoutMedia" flag, the application starts the alerting and connection process early. The application reports to the server that it is alerting the user, by setting the session status resource to "Ringing" using the PUT method.
18. The server returns an HTTP response to the request. Subsequently, the server sends towards the network infrastructure an indication that the participant is being alerted. The network infrastructure takes care of routing the information appropriately.
19. The application now alerts the user, who eventually accepts the call.
20. The application uses the PUT method on the resource containing the session status to accept the call by updating the status.

21. The server returns a response to the request and sends the offer and acceptance indication towards the network infrastructure, which takes care of routing the information appropriately. It then waits for an answer and confirmation of session acceptance.
22. Eventually, the answer arrives from the network which is forwarded by the server to the application in a WrtcsAnswerNotification.
23. The application sets the session description contained in the notification as the remote description in the PeerConnection object, marked as answer. ICE connectivity checks can subsequently start.
24. The browser reports to the application that a media path has been established by doing ICE connectivity checks with the media endpoint that was signaled in the remote session description from the answer
25. The application reports the successful ICE checks by updating the ICE status using the PUT method.
26. The server returns an HTTP response to the request.

End of alternatives.

The call is now established.

5.3.7 Signaling flow to cancel a WebRTC session invitation - Originator

The figure below shows a scenario where the Originator cancels a WebRTC session invitation before the Terminating Participant has accepted the invitation. The flow is shown from the Originator's point of view.

The resources:

– To cancel a WebRTC session, delete the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}**

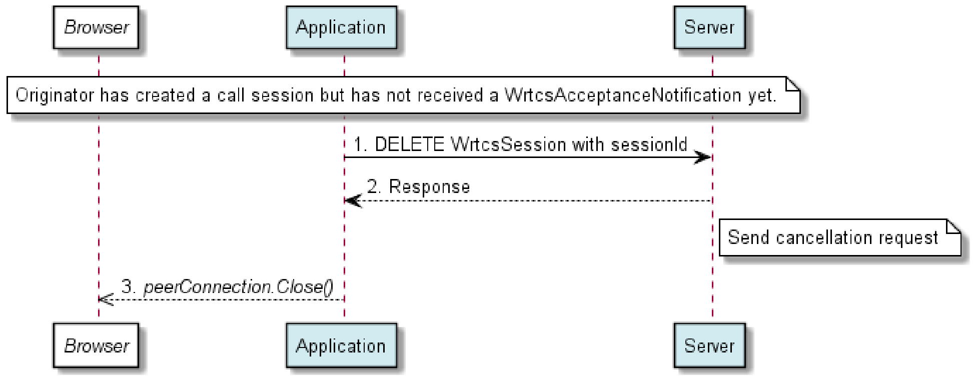
Browser
Application
Server
Originator has created a call session but has not received a WrtcsAcceptanceNotification yet.
1. DELETE WrtcsSession with sessionId
2. Response
Send cancellation request
3. peerConnection.Close()
Browser
Application
Server

Figure 9: Signaling flow to cancel a WebRTC session invitation - Originator

Outline of the flows:

It is assumed that the Originator has performed all steps up to including step 5 in section 5.3.3, but has not yet received a WrtcsAcceptanceNotification.

1. To cancel the call, the application uses the DELETE method on the session resource, addressed by the resourceURL containing the session Id.
2. The server indicates in the response that the deletion was successful, and sends a cancellation request towards the network infrastructure.
3. The application cleans up the local browser resources and terminates the media capturing / rendering by invoking the "Close" method of the PeerConnection object.

An update request (see 5.3.11) can be cancelled using the same flow.

5.3.8 Signaling flow to cancel a WebRTC session invitation – Terminating Participant

The figure below shows a scenario where the Originator cancels a WebRTC session invitation before the Terminating Participant has accepted the invitation. The flow is shown from the Terminating Participant's point of view.

There are no resources defined in this section, as the Terminating Participant can only react locally to the cancellation notification.

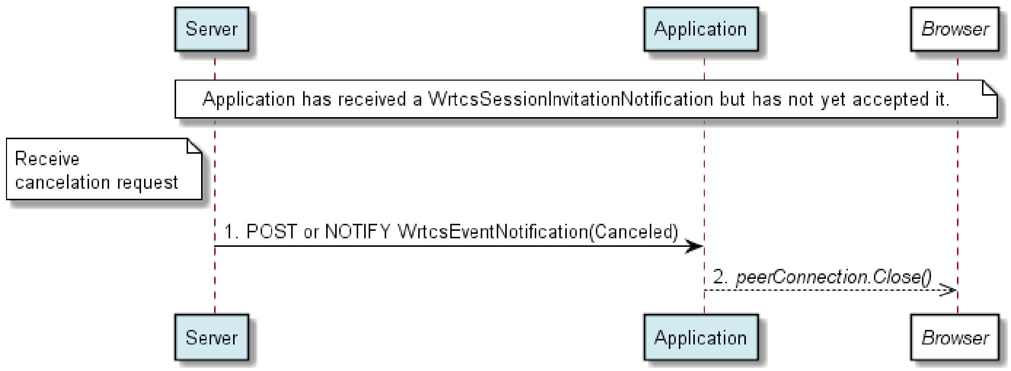
Server
Application
Browser
Application has received a WrtcsSessionInvitationNotification but has not yet accepted it.
Receive cancelation request
1. POST or NOTIFY WrtcsEventNotification(Canceled)
2. peerConnection.Close()
Server
Application
Browser

Figure 10: Signaling flow to cancel a WebRTC session invitation– Terminating Participant

Outline of the flows:

It is assumed that the Terminating Participant has received a WrtcsSessionInvitationNotification (step 1 in section 5.3.4) but has not yet successfully indicated acceptance of that invitation (step 15 in section 5.3.4)

1. Eventually, the server receives from the network infrastructure a message that the Originator is requesting to cancel the session invitation. The server notifies the application of the call session cancelation by sending a WrtcsEventNotification of type "Canceled" to the application.
2. The application cleans up the local browser resources and terminates the media capturing / rendering by invoking the "Close" method of the PeerConnection object.

An update request (see 5.3.12) can be cancelled using the same flow.

5.3.9 Signaling flow to reject a WebRTC session invitation – Terminating Participant

The figure below shows a scenario where the Terminating Participant rejects a WebRTC session invitation. The flow is shown from the Terminating Participant's point of view.

It is assumed that the Terminating Participant has received a WrtcsSessionInvitationNotification (step 1 in section 5.3.4) but has not yet successfully indicated acceptance of that invitation (step 15 in section 5.3.4)

The resources:

– To reject a WebRTC session invitation, delete the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}**

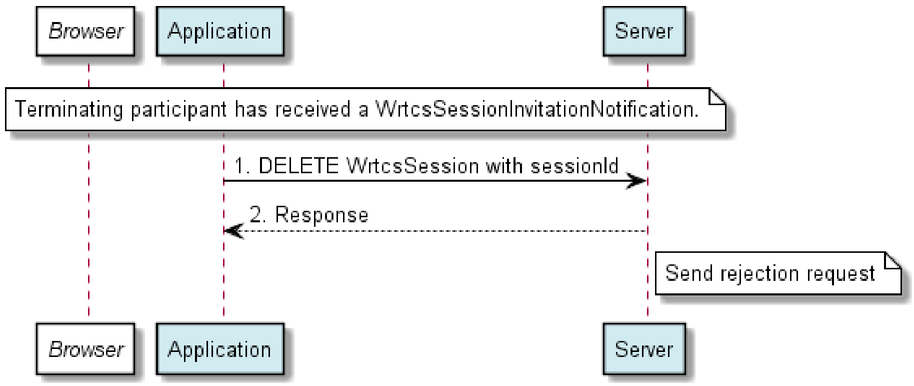
Browser
Application
Server
Terminating participant has received a WrtcsSessionInvitationNotification.
1. DELETE WrtcsSession with sessionId
2. Response
Send rejection request

Figure 11: Signaling flow to reject a WebRTC session invitation – Terminating Participant

Outline of the flows:

1. To reject the session invitation, the application uses the DELETE method on the session resource, addressed by the resourceURL containing the session Id.
2. The server indicates in the response that the deletion was successful, and sends a rejection request towards the network infrastructure.

5.3.10 Signaling flow to reject a WebRTC session invitation - Originator

The figure below shows a scenario where the Originator is informed that the Terminating Participant has declined a WebRTC session invitation. The flow is shown from the Originator's point of view.

There are no resources defined in this section, as the Originator can only react locally to the rejection notification.

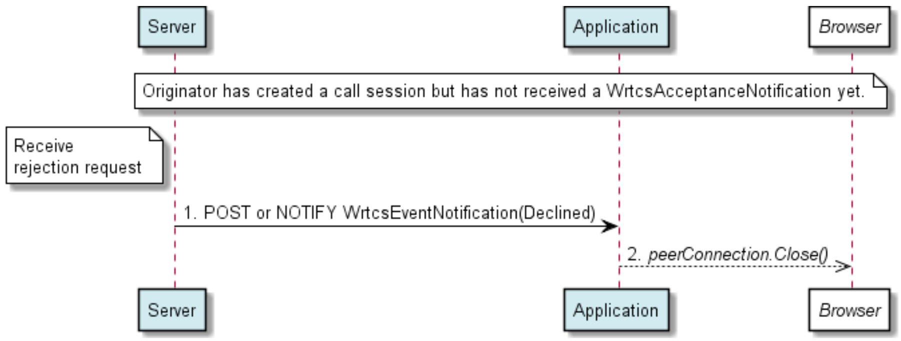
Server
Application
Browser
Originator has created a call session but has not received a WrtcsAcceptanceNotification yet.
Receive rejection request
1. POST or NOTIFY WrtcsEventNotification(Declined)
2. peerConnection.Close()

Figure 12: Signaling flow to reject a WebRTC session invitation - Originator

Outline of the flows:

It is assumed that the Originator has performed all steps up to including step 5 in section 5.3.3, but has not yet received a WrtcsAcceptanceNotification.

1. Eventually, the server receives from the network infrastructure a message that the Terminating Participant has rejected the session invitation. The server notifies the application of the session invitation rejection by sending a WrtcsEventNotification of type "Declined" to the application.
2. The application cleans up the local browser resources and terminates the media capturing / rendering by invoking the "Close" method of the PeerConnection object.

5.3.11 Signaling flow of a WebRTC session modification – Update Originator

The figure below shows a scenario where a WebRTC session is modified (e.g. to add or remove a video stream). The flow is shown from the Update Originator's point of view.

The resources:

- To modify a session, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/update**
- To indicate to the server changes of the ICE status, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

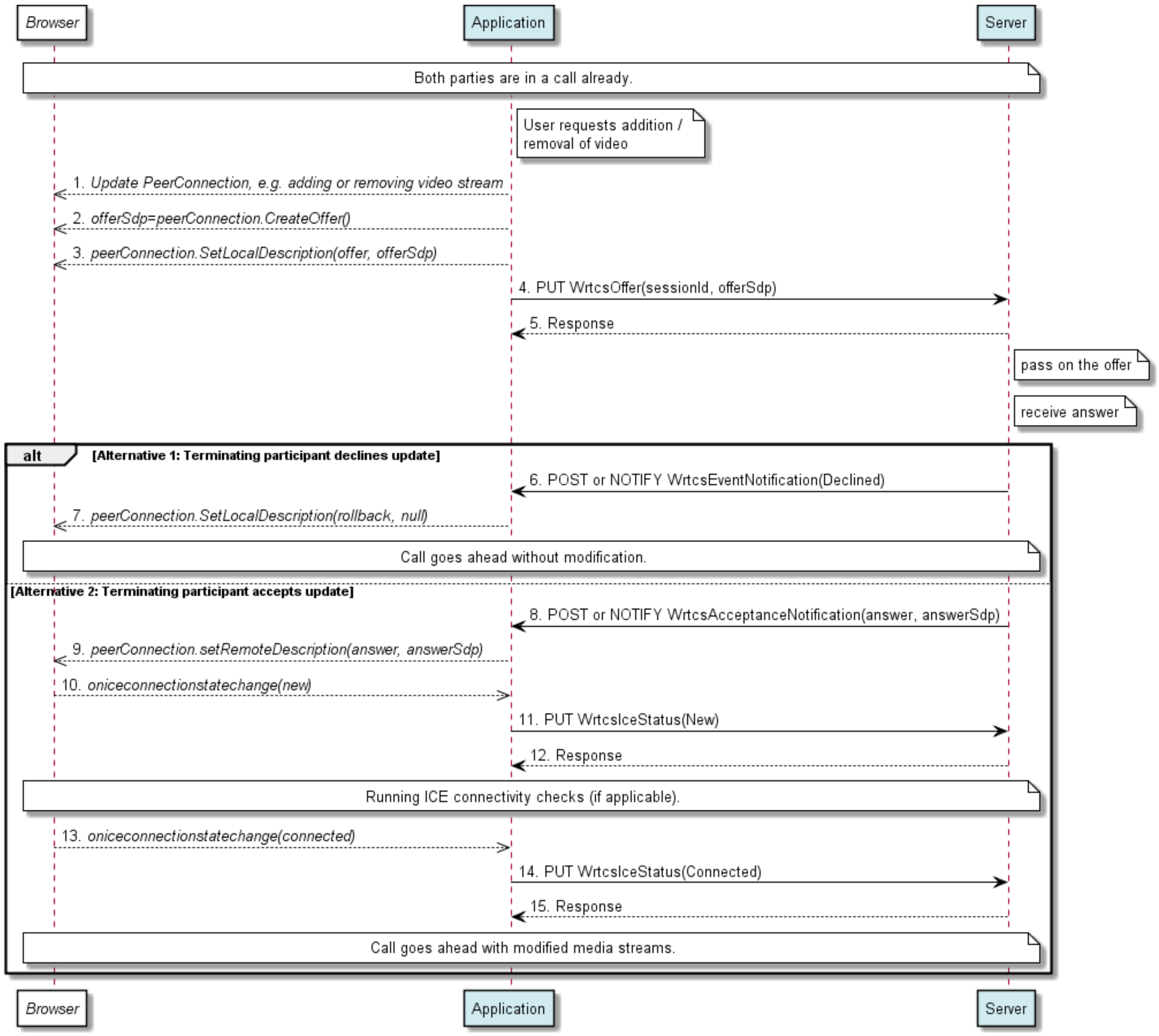
Browser
Application
Server
Both parties are in a call already.
User requests addition / removal of video
1. Update PeerConnection, e.g. adding or removing video stream
2. offerSdp=peerConnection.CreateOffer()
3. peerConnection.SetLocalDescription(offer, offerSdp)
4. PUT WrtcsOffer(sessionId, offerSdp)
5. Response
pass on the offer
receive answer
alt
[Alternative 1: Terminating participant declines update]
6. POST or NOTIFY WrtcsEventNotification(Declined)
7. peerConnection.SetLocalDescription(rollback, null)
Call goes ahead without modification.
[Alternative 2: Terminating participant accepts update]
8. POST or NOTIFY WrtcsAcceptanceNotification(answer, answerSdp)
9. peerConnection.setRemoteDescription(answer, answerSdp)
10. oniceconnectionstatechange(new)
11. PUT WrtcsIceStatus(New)
12. Response
Running ICE connectivity checks (if applicable).
13. oniceconnectionstatechange(connected)
14. PUT WrtcsIceStatus(Connected)
15. Response
Call goes ahead with modified media streams.
Browser
Application
Server

Figure 13: Signaling flow of a WebRTC session modification – Update Originator

Outline of the flows:

1. After the user has requested from the application to update the session (e.g. to add or remove video), the application updates the PeerConnection object accordingly.
2. The application requests an offer from the updated PeerConnection, reflecting the update.
3. The application installs this offer as the new local offer in the PeerConnection object.
4. Using the PUT method, the application updates the resource representing the update offer in the session with the new offer
5. The server returns a response. After that, the server passes on the offer to the Update Recipient via the network, and waits for an answer. When the answer eventually arrives, it can contain one of the following responses: a rejection of the update offer or an acceptance of the update offer.

Alternative flow 1: Update Recipient declines the update request

6. The server sends to the application a WrtcsEventNotification with the eventType set to “Declined”.
7. The application rolls back the session state to the state before the offer was sent. After that, the call goes ahead without modification.

Alternative flow 2: Update Recipient accepts the update request

8. The server sends to the application a WrtcsAcceptanceNotification including the answer.
9. The application installs the answer as the remote description in the PeerConnection object.
10. In case a stream was added, the previous step triggers a restart of the ICE procedures. The browser reports this change by sending an “oniceconnectionstatechange” message to the application.
11. The application reports the change of the ICE status to the server by updating the ICE status resource, using the PUT method.
12. The server returns a response.
13. Eventually, the ICE connectivity checks run and succeed. The browser reports the successful ICE run by sending an “oniceconnectionstatechange” message to the application.
14. The application reports the change of the ICE status to the server by updating the ICE status resource, using the PUT method.
15. The server returns a response. As the media path is established now for the added streams as well, the call goes ahead with the modified data streams.

End of alternatives.

5.3.12 Signaling flow of a WebRTC session modification – Update Recipient

The figure below shows a scenario where a WebRTC session is modified (e.g. to add or remove a video stream). The flow is shown from the Update Recipient’s point of view.

The resources:

- To accept a session modification request, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/answer**
- To reject a session update request, delete the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/update**

– To indicate to the server changes of the ICE status, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status**

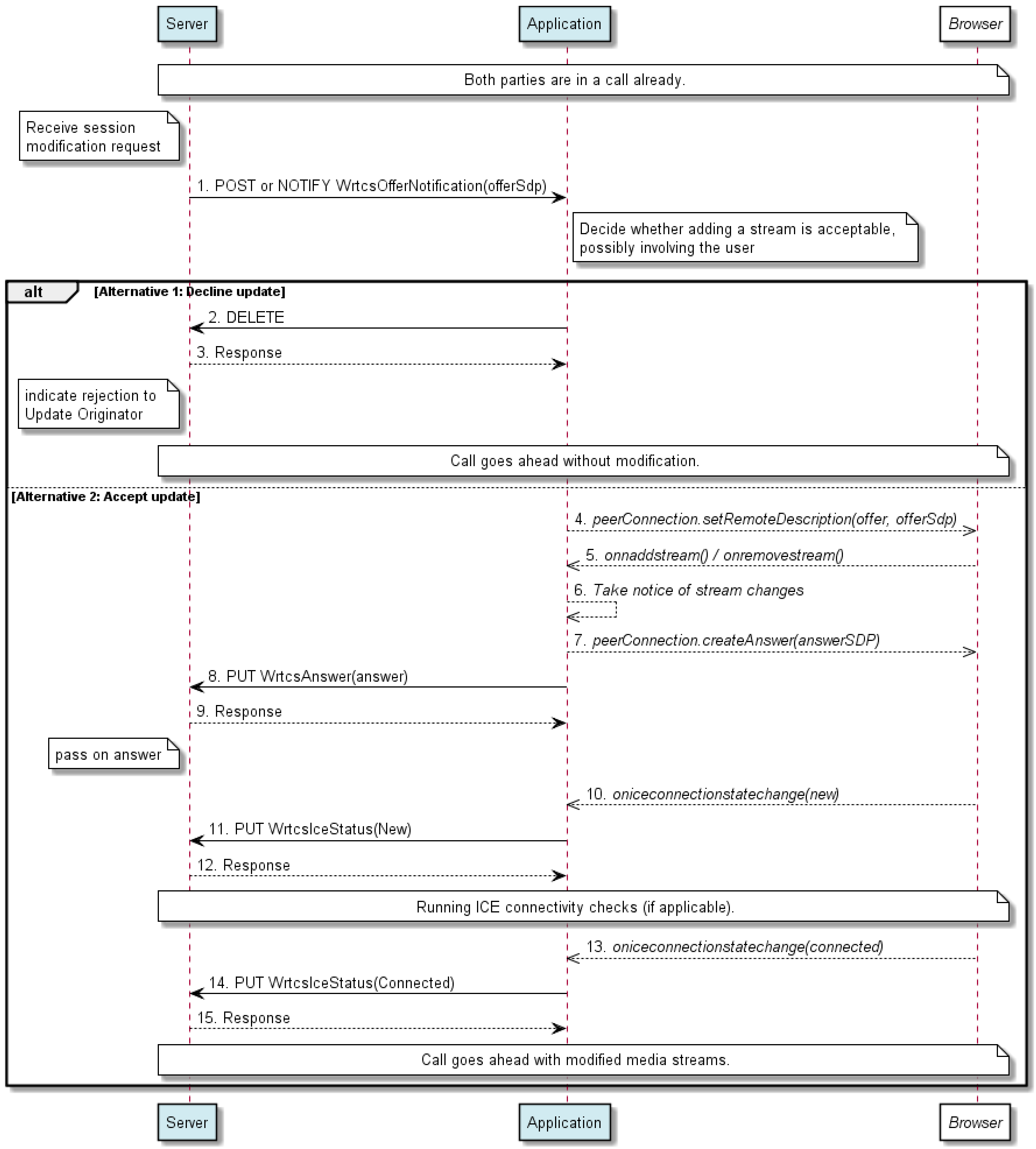
Server
Application
Browser
Both parties are in a call already.
Receive session modification request
1. POST or NOTIFY WrtcsOfferNotification(offerSdp)
Decide whether adding a stream is acceptable, possibly involving the user
alt
[Alternative 1: Decline update]
2. DELETE
3. Response
indicate rejection to Update Originator
Call goes ahead without modification.
[Alternative 2: Accept update]
4. peerConnection.setRemoteDescription(offer, offerSdp)
5. onnaddstream() / onremovestream()
6. Take notice of stream changes
7. peerConnection.createAnswer(answerSDP)
8. PUT WrtcsAnswer(answer)
9. Response
pass on answer
10. oniceconnectionstatechange(new)
11. PUT WrtcsIceStatus(New)
12. Response
Running ICE connectivity checks (if applicable).
13. oniceconnectionstatechange(connected)
14. PUT WrtcsIceStatus(Connected)
15. Response
Call goes ahead with modified media streams.

Figure 14: Signaling flow of a WebRTC session modification – Update Recipient

Outline of the flows:

1. The server sends to the application a WrtcsOfferNotification containing an update offer. The application decides based on that offer whether or not to accept it. Such decision might or might not include a dialog with the user.

Alternative flow 1: Decline update

2. If the application has decided to reject the update offer, it deletes the resource representing the update offer, using the DELETE method.

3. The server returns a response. After that, the call goes ahead without modification.

Alternative flow 2: Accept update

4. On the other hand, if the application has decided to accept the update offer, it installs the received update offer in the PeerConnection object as remote offer.

5. The PeerConnection object informs the application of the addition and/or removal of media streams.

6. The application takes notice of the changes and adapts its internal state. Details of how this is done are out of scope of this specification.

7. The application asks the PeerConnection object to create an answer.

8. The application updates the answer resource using the PUT method, replacing it with the new answer returned by the PeerConnection object.

9. The server returns a response. Also, the server takes care of sending the answer back to the Update Originator via the network infrastructure.

10. In case a stream was added, the previous step triggers a restart of the ICE procedures. The browser reports this change by sending an “oniceconnectionstatechange” message to the application.

11. The application reports the change of the ICE status to the server by updating the ICE status resource, using the PUT method.

12. The server returns a response.

13. Eventually, the ICE connectivity checks run and succeed. The browser reports the successful ICE run by sending an “oniceconnectionstatechange” message to the application.

14. The application reports the change of the ICE status to the server by updating the ICE status resource, using the PUT method.

15. The server returns a response. As the media path is established now for the added streams as well, the call goes ahead with the modified data streams.

End of alternatives.

5.3.13 Resolving an offer conflict

The figure below shows a scenario where both parties in a call have sent an offer concurrently, i.e. an offer conflict occurs. The reason for this may be that one of the clients has erroneously sent a new offer before a previous one was accepted/rejected, or that a network-internal race condition has led to that state. In any case, the server that detects the problem will decline the second offer as depicted below.

There are no resources defined in this section, as the application can only react locally to the cancellation notification.

The resources:

- To create an update offer, update the resource **http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/update**

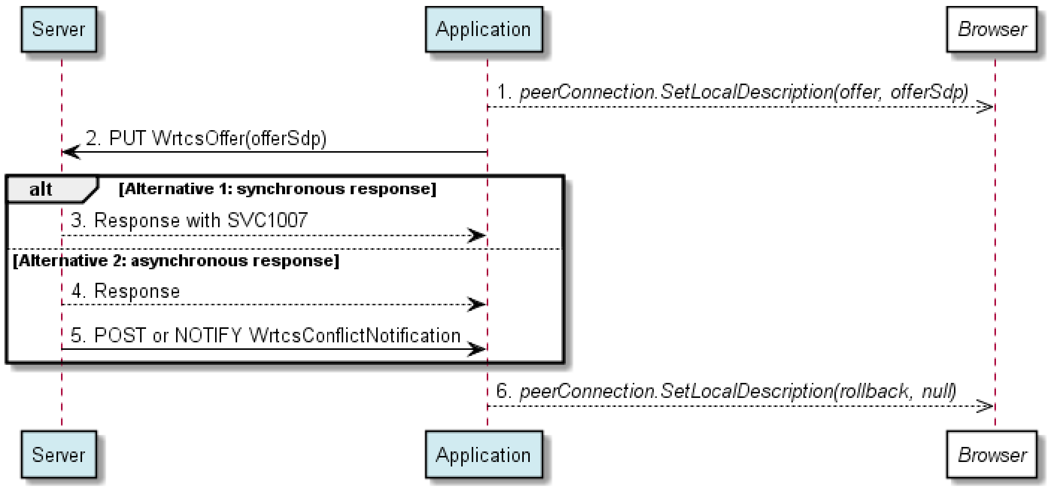
Server
Application
Browser
1. peerConnection.SetLocalDescription(offer, offerSdp)
2. PUT WrtcsOffer(offerSdp)
alt
[Alternative 1: synchronous response]
3. Response with SVC1007
[Alternative 2: asynchronous response]
4. Response
5. POST or NOTIFY WrtcsConflictNotification
6. peerConnection.SetLocalDescription(rollback, null)
Server
Application
Browser

Figure 15: Resolving an offer conflict

Outline of the flows:

1. The Update Originator’s application creates an update offer and installs this offer as the new local offer in the PeerConnection object.

2. The Update Originator’s application sends the update offer by modifying the “offer” resource in the session using the PUT method.

Alternative 1: Synchronous case

3. The server immediately detects a conflict and returns SVC1007 exception immediately with the HTTP response.

Alternative 2: Asynchronous case

4. The server returns a “success” response

5. The server later detects a conflict and sends a WrtcsConflictNotification to the application.

6. The application rolls back the session state to the state before the offer was sent.

6. Detailed specification of the resources

The following applies to all resources defined in this specification regardless of the representation format (i.e. XML, JSON):

- Reserved characters in URL variables (parts of a URL denoted below by a name in curly brackets) MUST be percent-encoded according to [RFC3986]. Note that this always applies, no matter whether the URL is used as a Request URL or inside the representation of a resource (such as in “resourceURL” and “link” elements).
- If a user identifier (e.g. address, participantAddress, etc.) of type anyURI is in the form of an MSISDN, it MUST be defined as a global number according to [RFC3966] (e.g. tel:+19585550100). The use of characters other than digits and the leading “+” sign SHOULD be avoided in order to ensure uniqueness of the resource URL. This applies regardless of whether the user identifier appears in a URL variable or in a parameter in the body of an HTTP message.
- If an equipment identifier of type anyURI is in the form of a SIP URI, it MUST be defined according to [RFC3261].
- If a user identifier (e.g. address, userId, etc) of type anyURI is in the form of an Anonymous Customer Reference (ACR), it MUST be defined according to Appendix H of [REST_NetAPI_ACR].
    - The ACR ‘auth’ is a supported reserved keyword, and MUST NOT be assigned as an ACR to any particular end user. See G.1.2 for details regarding the use of this reserved keyword.
- For requests and responses that have a body, the following applies: in the requests received, the server SHALL support JSON and XML encoding of the parameters in the body. The server SHALL return either JSON or XML encoded parameters in the response body, according to the result of the content type negotiation as specified in [REST_NetAPI_Common]. In notifications to the Client, the server SHALL use either XML or JSON encoding, depending on which format the client has specified in the related subscription. The generation and handling of the JSON representations SHALL follow the rules for JSON encoding in HTTP Requests/Responses as specified in [REST_NetAPI_Common].

Note 1: Offers and answers in the examples below contain Session Description Protocol (SDP) instances [RFC3264]. These instances can be treated by the application as opaque blobs that need to be extracted from and passed to the web browser in order to allow media communication (see section 5.3.2). Compared to SDP instances in a real-world WebRTC deployment, the instances in this specification are simplified, in particular, the details of ICE usage, media stream identification and RTP multiplexing signaling have been omitted.

Note 2: The examples illustrate an application called “Alice’s application” that uses the RESTful WebRTC Signaling API. Two sessions are modelled – one session with “Bob” where Alice is the Originator, and one session with “Caesar” where Alice is the Terminating Participant. All interactions with the API are depicted from Alice’s point of view. Bob’s and Caesar’s connectivity details are hidden from Alice’s application by the API.

6.1 Resource: All subscriptions to WebRTC signaling notifications

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/subscriptions

This resource is used to manage subscriptions to event notifications related to WebRTC Signaling events. Note that there is one subscription per client instance.

This resource can be used in conjunction with a Client-side Notification URL, or in conjunction with a Server-side Notification URL. In this latter case, the application MUST first create a Notification Channel (see [REST_NetAPI_NotificationChannel]) before creating a subscription.

6.1.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.1.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.1.3 GET

This operation is used for reading the list of active notification subscriptions.

6.1.3.1 Example: Reading all active subscriptions (Informative)

Alice’s application reads all active subscriptions.

6.1.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Accept: application/xml
Host: example.com
```

6.1.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSubscriptionList xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <wrtcsNotificationSubscription>
      <callbackReference>
         <notifyURL>http://application-alice.example.com/webrtcsignaling/notifications/77777</notifyURL>
         <callbackData>abcd</callbackData>
      </callbackReference>
      <duration>7037</duration>
      <clientCorrelator>12345</clientCorrelator>
      <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001</resourceURL>
   </wrtcsNotificationSubscription>
   <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions</resourceURL>
</wrtcs:wrtcsSubscriptionList>
```

6.1.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, POST’ field in the response as per section 14.7 of [RFC2616].

6.1.5 POST

This operation is used to create a new subscription for notifications related to WebRTC Signaling events.

The notifyURL in the callbackReference either contains the Client-side Notification URL (as defined by the client) or the Server-side Notification URL (as obtained during the creation of the Notification Channel [REST_NetAPI_NotificationChannel]).

6.1.5.1 Example: Creating a new subscription, response with copy of created resource (Informative)

Alice’s application creates a subscription.

6.1.5.1.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsNotificationSubscription xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <callbackReference>
    <notifyURL>http://application-alice.example.com/webrtcsignaling/notifications/77777</notifyURL>
    <callbackData>abcd</callbackData>
  </callbackReference>
  <duration>7200</duration>
  <clientCorrelator>12345</clientCorrelator>
</wrtcs:wrtcsNotificationSubscription>
```

6.1.5.1.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsNotificationSubscription xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <callbackReference>
    <notifyURL>http://application-alice.example.com/webrtcsignaling/notifications/77777</notifyURL>
    <callbackData>abcd</callbackData>
  </callbackReference>
  <duration>7200</duration>
  <clientCorrelator>12345</clientCorrelator>
  <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001</resourceURL>
</wrtcs:wrtcsNotificationSubscription>
```

6.1.5.2 Example: Creating a new subscription, response with location of created resource (Informative)

Alice's application creates a subscription.

Besides showing subscription creation, this example illustrates a technique to return only a reference to the created resource, rather than a copy of it (defined in [REST_NetAPI_Common] as an alternative way of resource creation responses).

6.1.5.2.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsNotificationSubscription xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackReference>
      <notifyURL>http://application-alice.example.com/webrtcsignaling/notifications/77777</notifyURL>
      <callbackData>abcd</callbackData>
   </callbackReference>
   <duration>7200</duration>
   <clientCorrelator>12345</clientCorrelator>
</wrtcs:wrtcsNotificationSubscription>
```

6.1.5.2.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<common:resourceReference xmlns:common="urn:oma:xml:rest:netapi:common:1">
   <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001</resourceURL>
</common:resourceReference>
```

6.1.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, POST' field in the response as per section 14.7 of [RFC2616].

6.2 Resource: Individual subscription to WebRTC signaling notifications

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion/{userId}/subscriptions/{subscriptionId}

This resource represents an individual subscription to notifications related to WebRTC Signaling events.

This resource can be used in conjunction with a Client-side Notification URL, or in conjunction with a Server-side Notification URL. In this latter case, the application MUST first create a Notification Channel (see [REST_NetAPI_NotificationChannel]) before creating a subscription.

6.2.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
subscriptionId	Identifier of the subscription

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.2.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.2.3 GET

This operation is used for reading an individual subscription.

6.2.3.1 Example: Reading an individual subscription (Informative)

Alice's application reads a subscription.

6.2.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001 HTTP/1.1
Accept: application/xml
Host: example.com
```

6.2.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsNotificationSubscription xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackReference>
      <notifyURL>http://application-alice.example.com/webrtcsignaling/notifications/77777</notifyURL>
      <callbackData>abcd</callbackData>
   </callbackReference>
   <duration>7200</duration>
   <clientCorrelator>12345</clientCorrelator>
   <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001</resourceURL>
</wrtcs:wrtcsNotificationSubscription>
```

6.2.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, DELETE' field in the response as per section 14.7 of [RFC2616].

6.2.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, DELETE' field in the response as per section 14.7 of [RFC2616].

6.2.6 DELETE

This operation is used to cancel a subscription and to stop corresponding notifications.

6.2.6.1 Example: Cancelling a subscription (Informative)

Alice's application cancels a subscription.

6.2.6.1.1 Request

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001 HTTP/1.1
Accept: application/xml
Host: example.com
```

6.2.6.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.3 Resource: All WebRTC sessions

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions

This resource contains information about all WebRTC sessions available to a particular client instance.

6.3.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.3.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.3.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.3.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.3.5 POST

This operation is used to create a new WebRTC session with the user represented by {userId} as Originator.

Apart from illustrating the creation of different types of sessions (audio-only and audio+video), this section also illustrates the different user identity options and response options after resource creation. In fact, these three dimensions are orthogonal.

6.3.5.1 Example: Creating a new WebRTC session – audio only, using tel URI (Informative)

Alice's application creates a new audio session, identifying Alice by means of a tel: URI.

Besides illustrating the creation of an audio-only WebRTC session, this example illustrates how a tel URI can be used to identify the user.

6.3.5.1.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: example.com
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <originatorAddress>tel:+19585550100</originatorAddress>
   <originatorName>Alice</originatorName>
   <tParticipantAddress>tel:+19585550101</tParticipantAddress>
   <tParticipantName>Bob</tParticipantName>
   <offer>
      <sdp>
         <![CDATA[v=0
o=alice 8946567654657144810 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001
]]>
        </sdp>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
</wrtcs:wrtcsSession>
```

6.3.5.1.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <originatorAddress>tel:+19585550100</originatorAddress>
    <originatorName>Alice</originatorName>
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <status>Initiated</status>
    <offer>
        <type>Local</type>
        <sdp>
            <![CDATA[v=0
o=alice 8946567654657144810 0 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001
            ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
    <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001</resourceURL>
</wrtcs:wrtcsSession>
```

6.3.5.2 Example: Creating a new WebRTC session – audio only, using SIP URI and encoding the SDP with base64 (Informative)

Alice's application creates a new audio session, identifying Alice by means of a SIP URI.

Besides illustrating the creation of an audio-only WebRTC session, this example illustrates how a SIP URI can be used to identify the user, and how an SDP can be base64-encoded to be robust against the case that it contains characters which break the structure of XML or JSON.

6.3.5.2.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: example.com
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <offer>
        <sdpBase64>
ICAgICAgICAgICAgdj0wDQogICAgICAgICAgICBvPWFsaWNlc19icm93c2VyIDI4OTA4NDQ1MjYg
Mjg5MDg0MjgwNyBJTiBJUDQgMTAuMC4xLjENCiAgICAgICAgICAgIHM9DQogICAgICAgICAgICB0
PTAgMA0KICAgICAgICAgICAgYz1JTiBJUDQgMTkyLjAuMi4zMA0KICAgICAgICAgICAgYT1pY2Ut
cHdkOmFzZDg4ZmdwZGQ3Nzd1empZaGFnWmcNCiAgICAgICAgICAgIGE9aWNlLXVmcmFnOjhoaFkN
CiAgICAgICAgICAgIGE9ZmluZ2VycHJpbnQ6c2hhLTEgOTk6NDE6NDk6ODM6NGE6OTc6MGU6MWY6
ZWY6NmQ6Zjc6Yzk6Yzc6NzA6OWQ6MWY6NjY6Nzk6YTg6MDcNCiAgICAgICAgICAgIA0KICAgICAg
ICAgICAgbT1hdWRpbyAxMDAwMCBSVFAvU0FWUEYgMCA5Ng0KICAgICAgICAgICAgYT1ydHBtYXA6
MCBQQ01VLzgwMDANCiAgICAgICAgICAgIGE9cnRwbWFwOjk2IG9wdXMvNDgwMDANCiAgICAgICAg
ICAgIGE9c2VuZHJlY3YNCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjEgMSBVRFAgMjEzMDcwNjQz
MSAxMC4wLjEuMSA4MDAwIHR5cCBob3N0DQogICAgICAgICAgICBhPWNhbmRpZGF0ZToxIDIgVURQ
IDIxMzA3MDY0MzAgMTAuMC4xLjEgODAwMSB0eXAgaG9zdA0KICAgICAgICAgICAgYT1jYW5kaWRh
```

```
dGU6MiAxIFVEUCAxNjk0NDk4ODE1IDE5Mi4wLjIuMzAgMTAwMDAgdHlwIHNyZmx4IHJhZGRyIDEw
LjAuMS4xIHJwb3J0IDgwMDANCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjIgMiBVRFAgMTY5NDQ5
ODgxNCAxOTIuMC4yLjMwIDEwMDAxIHR5cCBzcmZseCByYWRkciAxMC4wLjEuMSBycG9ydCA4MDAx
DQo=
        </sdpBase64>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
</wrtcs:wrtcsSession>
```

6.3.5.2.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions/sess001

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <originatorAddress>sip:alice@example.com</originatorAddress>
    <originatorName>Alice</originatorName>
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <status>Initiated</status>
    <offer>
        <type>Local</type>
        <sdpBase64>
ICAgICAgICAgICAgdj0wDQogICAgICAgICAgICBvPWFsaWNlc19icm93c2VyIDI4OTA4NDQ1MjYg
Mjg5MDg0MjgwNyBJTiBJUDQgMTAuMC4xLjENCiAgICAgICAgICAgIHM9DQogICAgICAgICAgICB0
PTAgMA0KICAgICAgICAgICAgYz1JTiBJUDQgMTkyLjAuMi4zMA0KICAgICAgICAgICAgYT1pY2Ut
cHdkOmFzZDg4ZmdwZGQ3Nzd1empZaGFnWmcNCiAgICAgICAgICAgIGE9aWNlLXVmcmFnOjhoaFkN
CiAgICAgICAgICAgIGE9ZmluZ2VycHJpbnQ6c2hhLTEgOTk6NDE6NDk6ODM6NGE6OTc6MGU6MWY6
ZWY6NmQ6Zjc6Yzk6Yzc6NzA6OWQ6MWY6NjY6Nzk6YTg6MDcNCiAgICAgICAgICAgIA0KICAgICAg
ICAgICAgbT1hdWRpbyAxMDAwMCBSVFAvU0FWUEYgMCA5Ng0KICAgICAgICAgICAgYT1ydHBtYXA6
MCBQQ01VLzgwMDANCiAgICAgICAgICAgIGE9cnRwbWFwOjk2IG9wdXMvNDgwMDANCiAgICAgICAg
ICAgIGE9c2VuZHJlY3YNCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjEgMSBVRFAgMjEzMDcwNjQz
MSAxMC4wLjEuMSA4MDAwIHR5cCBob3N0DQogICAgICAgICAgICBhPWNhbmRpZGF0ZToxIDIgVURQ
IDIxMzA3MDY0MzAgMTAuMC4xLjEgODAwMSB0eXAgaG9zdA0KICAgICAgICAgICAgYT1jYW5kaWRh
dGU6MiAxIFVEUCAxNjk0NDk4ODE1IDE5Mi4wLjIuMzAgMTAwMDAgdHlwIHNyZmx4IHJhZGRyIDEw
LjAuMS4xIHJwb3J0IDgwMDANCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjIgMiBVRFAgMTY5NDQ5
ODgxNCAxOTIuMC4yLjMwIDEwMDAxIHR5cCBzcmZseCByYWRkciAxMC4wLjEuMSBycG9ydCA4MDAx
DQo=
        </sdpBase64>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
    <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions/sess001</resourceURL>
</wrtcs:wrtcsSession>
```

6.3.5.3 Example: Creating a new WebRTC session – audio and video, using ACR (Informative)

Alice's application creates a new session with audio and video, identifying Alice by means of an ACR.

Besides illustrating the creation of a WebRTC session which contains audio and video, this example illustrates how an ACR (Anonymous Customer Reference) can be used to identify the user.

6.3.5.3.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: example.com
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <offer>
        <sdp>
            <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
      ]]>
    </sdp>
  </offer>
  <clientCorrelator>4567</clientCorrelator>
</wrtcs:wrtcsSession>
```

6.3.5.3.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <originatorAddress>acr:pseudonym123</originatorAddress>
  <tParticipantAddress>tel:+19585550101</tParticipantAddress>
  <tParticipantName>Bob</tParticipantName>
  <status>Initiated</status>
  <offer>
    <type>Local</type>
    <sdp>
      <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001
m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101          ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
        <mediaIndicator>
            <type>Video</type>
            <entryIdx>1</entryIdx>
            <entryId>2</entryId>
            <streamId>stream1</streamId>
            <trackId>track2</trackId>
            <payload>
                <payloadType>97</payloadType>
                <encoding>H264</encoding>
                <formatParams>profile-level-id=4d0028;packetization-mode=1</formatParams>
            </payload>
            <payload>
                <payloadType>98</payloadType>
                <encoding>VP8</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
    <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001</resourceURL>
</wrtcs:wrtcsSession>
```

6.3.5.4 Example: Creating a new WebRTC session – audio and video, using acr:auth (Informative)

Alice’s application creates a new session with audio and video, identifying Alice by means of acr:auth.

Besides illustrating the creation of a WebRTC session which contains audio and video, this example illustrates how an OAuth 2.0 bearer token can be used to identify the user. In the request URL, the string “acr:auth” indicates that the user identity can be obtained by evaluating the access token.

6.3.5.4.1 Request

```
POST /exampleAPI/webrtcsignaling/v1/acr%3Aauth/sessions HTTP/1.1
Authorization: Bearer mF_9.B5f-4.1JqM
Accept: application/xml
Content-Type: application/xml
Host: example.com
Content-Length: nnnn

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <tParticipantAddress>tel:+19585550101</tParticipantAddress>
  <tParticipantName>Bob</tParticipantName>
  <offer>
    <sdp>
      <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
      ]]>
    </sdp>
  </offer>
  <clientCorrelator>4567</clientCorrelator>
</wrtcs:wrtcsSession>
```

6.3.5.4.2 Response

```
HTTP/1.1 201 Created
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Aauth/sessions/sess001

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <originatorAddress>acr%3Aauth</originatorAddress>
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <status>Initiated</status>
    <offer>
        <type>Local</type>
        <sdp>
            <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
            ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
        <mediaIndicator>
            <type>Video</type>
            <entryIdx>1</entryIdx>
            <entryId>2</entryId>
            <streamId>stream1</streamId>
            <trackId>track2</trackId>
            <payload>
                <payloadType>97</payloadType>
                <encoding>H264</encoding>
                <formatParams>profile-level-id=4d0028;packetization-mode=1</formatParams>
            </payload>
            <payload>
                <payloadType>98</payloadType>
                <encoding>VP8</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
    <clientCorrelator>4567</clientCorrelator>
    <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001</resourceURL>
</wrtcs:wrtcsSession>
```

6.3.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.4 Resource: Individual WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}

This resource represents a WebRTC session.

6.4.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.4.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.4.3 GET

This operation is used to retrieve information about a WebRTC session.

6.4.3.1 Example: Retrieving WebRTC session information (Informative)

Alice’s application reads the session information, which includes an offer, an answer and a pending update offer.

6.4.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001 HTTP/1.1
Accept: application/xml
Host: example.com
```

6.4.3.1.2 Response

The body of this response illustrates a session with an offer, the associated answer and a pending update.

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSession xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <originatorAddress>tel:+19585550100</originatorAddress>
    <originatorName>Alice</originatorName>
    <tParticipantAddress>tel:+19585550101</tParticipantAddress>
    <tParticipantName>Bob</tParticipantName>
    <status>Connected</status>
    <offer>
        <type>Local</type>
        <sdp>
```

```
            <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001
            ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
    <answer>
        <type>Remote</type>
        <isProvisional>false</isProvisional>
        <sdp>
            <![CDATA[v=0
o=bob 98746513249823567101 0 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=ice-pwd:YH75Fviy6338Vbrhrlp8Yh
a=ice-ufrag:9uB6
a=ice-lite
```

```
m=audio 20000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10122
a=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host      ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </answer>
    <update>
        <type>Local</type>
        <sdp>
            <![CDATA[v=0
o=alice 8946567654657144810 0 1 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
```

```
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101          ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
        <mediaIndicator>
            <type>Video</type>
            <entryIdx>1</entryIdx>
            <entryId>2</entryId>
            <streamId>stream1</streamId>
            <trackId>track2</trackId>
            <payload>
                <payloadType>97</payloadType>
                <encoding>H264</encoding>
            </payload>
            <payload>
                <payloadType>98</payloadType>
                <encoding>VP8</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </update>
    <clientCorrelator>4567</clientCorrelator>
    <resourceURL>http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001</resourceURL>
</wrtcs:wrtcsSession>
```

6.4.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, DELETE’ field in the response as per section 14.7 of [RFC2616].

6.4.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, DELETE’ field in the response as per section 14.7 of [RFC2616].

6.4.6 DELETE

This operation is used by any Participant to terminate a WebRTC session, by the Terminating Participant to decline a WebRTC session invitation, or by the Originator to cancel a WebRTC session invitation before it has been accepted.

Upon acting on the DELETE request, the server removes the resource representing the session immediately.

6.4.6.1 Example: Cancelling or terminating a WebRTC session, or declining a WebRTC session invitation (Informative)

Alice's application deletes the created session. This cancels the invitation sent to Bob if Bob has not yet accepted, or terminates the session otherwise.

6.4.6.1.1 Request

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001 HTTP/1.1
Accept: application/xml
Host: example.com
```

6.4.6.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.5 Resource: Status of a WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/status

This resource represents the status of a WebRTC session.

6.5.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.5.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.5.3 GET

This operation is used to read the status of a WebRTC session.

6.5.3.1 Example: Reading the status of a WebRTC session (Informative)

Alice's application reads the status of the session.

6.5.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/status HTTP/1.1
Accept: application/xml
Host: example.com
```

6.5.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Ringing</status>
</wrtcs:wrtcsSessionStatus>
```

6.5.4 PUT

This operation is used to update the status of a WebRTC session, in order to accept a WebRTC session invitation, or to indicate that the Terminating Participant is being alerted (“Ringing”).200 OK and 204 No Content are valid success responses.

6.5.4.1 Example: Accepting a WebRTC session invitation (Informative)

Alice’s application accepts a session invitation.

6.5.4.1.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/status HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Connected</status>
</wrtcs:wrtcsSessionStatus>
```

6.5.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Connected</status>
</wrtcs:wrtcsSessionStatus>
```

6.5.4.2 Example: Indicating the alerting of the Terminating Participant (“Ringing”) (Informative)

Alice’s application indicates that it is alerting the user.

6.5.4.2.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/status HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Ringing</status>
</wrtcs:wrtcsSessionStatus>
```

6.5.4.2.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Ringing</status>
</wrtcs:wrtcsSessionStatus>
```

6.5.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.5.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.6 Resource: Initial or most recent offer in a WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/offer

This resource represents the initial or most recent offer in a WebRTC session. In case it represents the initial offer in the session, the offer may be answered or still unanswered, depending on whether or not the sibling resource “answer” exists. In case it does not represent the initial offer, the “answer” sibling always exists, i.e. the contents of this resource always represents the most recent answered offer.

Note that an additional sibling resource “update” may exist, which represents an update to the offer represented by the resource defined in the current section. More details can be found in section 6.8.

6.6.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.6.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.6.3 GET

This operation is used to read the initial or most recent offer in a WebRTC session.

6.6.3.1 Example: Reading initial or most recent offer in a WebRTC session (Informative)

Alice's application reads the initial or most recent offer of the session.

6.6.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/offer HTTP/1.1
Accept: application/xml
Host: example.com
```

6.6.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Local</type>
    <sdp>
        <![CDATA[v=0
o=alice 89465676546571448100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001     ]]>
  </sdp>
  <mediaIndicator>
    <type>Audio</type>
    <entryIdx>0</entryIdx>
    <entryId>1</entryId>
    <streamId>stream1</streamId>
    <trackId>track1</trackId>
    <payload>
      <payloadType>0</payloadType>
      <encoding>PCMU</encoding>
    </payload>
    <payload>
      <payloadType>96</payloadType>
      <encoding>opus</encoding>
    </payload>
    <direction>SendRecv</direction>
  </mediaIndicator>
</wrtcs:wrtcsOffer>
```

6.6.4 PUT

This operation is used to provide an offer to an offerless session invitation.

200 OK and 204 No Content are valid success responses.

6.6.4.1 Example: Providing an offer to an offerless session invitation(Informative)

Alice’s application provides an offer to an offerless session invitation.

6.6.4.1.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/offer HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <sdp>
```

```
        <![CDATA[v=0
o=alice 78643246856870134100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10200 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10044
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr 10.0.1.1 rport 9000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001
        ]]>
    </sdp>
</wrtcs:wrtcsOffer>
```

6.6.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Local</type>
    <sdp>
        <![CDATA[v=0
o=alice 78643246856870134100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07

m=audio 10200 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10044
a=rtcp-mux
a=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr 10.0.1.1 rport 9000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001
      ]]>
  </sdp>
  <mediaIndicator>
    <type>Audio</type>
    <entryIdx>0</entryIdx>
    <entryId>1</entryId>
    <streamId>stream1</streamId>
    <trackId>track1</trackId>
    <payload>
      <payloadType>0</payloadType>
      <encoding>PCMU</encoding>
    </payload>
    <payload>
      <payloadType>96</payloadType>
      <encoding>opus</encoding>
    </payload>
    <direction>SendRecv</direction>
  </mediaIndicator>
</wrtcs:wrtcsOffer>
```

6.6.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.6.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.7 Resource: Most recent answer in a WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion/{userId}/sessions/{sessionId}/answer

This resource represents the most recent answer in a WebRTC session. This resource does not exist in the initial stages in the lifecycle of a session when an answer has not yet been received.

6.7.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.7.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.7.3 GET

This operation is used to read the most recent answer in a WebRTC session.

6.7.3.1 Example: Reading most recent answer in a WebRTC session(Informative)

Alice’s application reads the most recent answer in a session.

6.7.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/answer HTTP/1.1
Accept: application/xml
Host: example.com
```

6.7.3.1.2 Response

The answer in the example below corresponds e.g. to the offer in section 6.3.5.1.

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAnswer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Remote</type>
    <isProvisional>false</isProvisional>
    <sdp>
        <![CDATA[v=0
o=bob 98746513249823567101 0 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=ice-pwd:YH75Fviy6338Vbrhrlp8Yh
a=ice-ufrag:9uB6
a=ice-lite

m=audio 20000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10122
a=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host        ]]>
    </sdp>
    <mediaIndicator>
        <type>Audio</type>
        <entryIdx>0</entryIdx>
        <entryId>1</entryId>
        <streamId>stream1</streamId>
        <trackId>track1</trackId>
        <payload>
            <payloadType>0</payloadType>
            <encoding>PCMU</encoding>
        </payload>
        <direction>SendRecv</direction>
    </mediaIndicator>
</wrtcs:wrtcsAnswer>
```

6.7.4 PUT

This operation is used to provide an answer to an offer, such as a session invitation (initial offer) or session modification (update offer).

200 OK and 204 No Content are valid success responses.

6.7.4.1 Example: Providing an answer to an offer (Informative)

Alice's application provides an answer to Caesar's offer.

6.7.4.1.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/answer HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAnswer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <isProvisional>false</isProvisional>
    <sdp>
        <![CDATA[v=0
o=alice 78643246856870134100 0 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10200 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10044
a=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr 10.0.1.1 rport 9000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001

    ]]>
  </sdp>
</wrtcs:wrtcsAnswer>
```

6.7.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAnswer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <type>Local</type>
  <isProvisional>false</isProvisional>
  <sdp>
    <![CDATA[v=0
o=alice 786432468568701341000 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10200 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10044
a=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr 10.0.1.1 rport 9000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001
    ]]>
  </sdp>
  <mediaIndicator>
    <type>Audio</type>
    <entryIdx>0</entryIdx>
    <entryId>1</entryId>
    <streamId>stream1</streamId>
    <trackId>track1</trackId>
    <payload>
      <payloadType>0</payloadType>
      <encoding>PCMU</encoding>
    </payload>
    <direction>SendRecv</direction>
  </mediaIndicator>
</wrtcs:wrtcsAnswer>
```

6.7.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, PUT' field in the response as per section 14.7 of [RFC2616].

6.7.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: GET, PUT' field in the response as per section 14.7 of [RFC2616].

6.8 Resource: Update offer in a WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion/{userId}/sessions/{sessionId}/update

This resource represents the most recent unanswered update offer in the WebRTC session. The content of this resource is moved to the sibling "offer" resource once an answer has been received for this offer.

6.8.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.8.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.8.3 GET

This operation is used to read the update offer in a WebRTC session.

6.8.3.1 Example: Reading the update offer in a WebRTC session (Informative)

Alice's application reads the update offer in the session.

6.8.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Accept: application/xml
Host: example.com
```

6.8.3.1.2 Response

The update offer in this example updates an audio-only session (e.g. section 6.3.5.1) with video.

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Local</type>
    <sdp>
        <![CDATA[v=0
o=alice 89465676546571448100 1 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
        ]]>
    </sdp>
    <mediaIndicator>
        <type>Audio</type>
        <entryIdx>0</entryIdx>
        <entryId>1</entryId>
        <streamId>stream1</streamId>
        <trackId>track1</trackId>
        <payload>
            <payloadType>0</payloadType>
            <encoding>PCMU</encoding>
        </payload>
        <direction>SendRecv</direction>
    </mediaIndicator>
    <mediaIndicator>
        <type>Video</type>
        <entryIdx>1</entryIdx>
        <entryId>2</entryId>
        <streamId>stream1</streamId>
        <trackId>track2</trackId>
        <payload>
            <payloadType>97</payloadType>
            <encoding>H264</encoding>
        </payload>
        <payload>
            <payloadType>98</payloadType>
            <encoding>VP8</encoding>
        </payload>
        <direction>SendRecv</direction>
    </mediaIndicator>
</wrtcs:wrtcsOffer>
```

6.8.4 PUT

This operation is used to provide an update offer in a WebRTC session.

200 OK and 204 No Content are valid success responses.

6.8.4.1 Example: Initiating an update offer in a WebRTC session to upgrade from audio-only to audio+video (Informative)

Alice’s application initiates an update offer towards Bob’s application to add video to the session.

Note that the “upgrade” semantics is only visible in the SDP which in a typical WebRTC deployment has been emitted by the browser and is merely passed to the involved network elements using this API. Hence, active control of the streams involved in a session is achieved using the WebRTC APIs in the browser, not this API.

6.8.4.1.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <sdp>
        <![CDATA[v=0
o=alice 8946567654657144810 0 1 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
        ]]>
    </sdp>
</wrtcs:wrtcsOffer>
```

6.8.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Local</type>
    <sdp>
        <![CDATA[v=0
o=alice 89465676546571448100 1 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 10100 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
a=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr 10.0.1.1 rport 8100
a=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101
    ]]>
  </sdp>
  <mediaIndicator>
    <type>Audio</type>
    <entryIdx>0</entryIdx>
    <entryId>1</entryId>
    <streamId>stream1</streamId>
    <trackId>track1</trackId>
    <payload>
      <payloadType>0</payloadType>
      <encoding>PCMU</encoding>
    </payload>
    <direction>SendRecv</direction>
  </mediaIndicator>
  <mediaIndicator>
    <type>Video</type>
    <entryIdx>1</entryIdx>
    <entryId>2</entryId>
    <streamId>stream1</streamId>
    <trackId>track2</trackId>
    <payload>
      <payloadType>97</payloadType>
      <encoding>H264</encoding>
    </payload>
    <payload>
      <payloadType>98</payloadType>
      <encoding>VP8</encoding>
    </payload>
    <direction>SendRecv</direction>
  </mediaIndicator>
</wrtcs:wrtcsOffer>
```

6.8.4.2 Example: Initiating an update offer in a WebRTC session to downgrade from audio+video to audio-only (Informative)

Alice’s application initiates an update offer towards Bob’s application to remove video from the session.

Note that the “downgrade” semantics is only visible in the SDP which in a typical WebRTC deployment has been emitted by the browser and is merely passed to the involved network elements using this API. Hence, active control of the streams involved in a session is achieved using the WebRTC APIs in the browser, not this API.

6.8.4.2.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <sdp>
        <![CDATA[v=0
o=alice 89465676546571448100 2 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 0 RTP/SAVPF 97
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
        ]]>
    </sdp>
</wrtcs:wrtcsOffer>
```

6.8.4.2.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOffer xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <type>Local</type>
    <sdp>
        <![CDATA[v=0
o=alice 89465676546571448100 2 IN IP4 10.0.1.1
s=
t=0 0
c=IN IP4 192.0.2.30
a=msid-semantic:WMS
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY

m=audio 10000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10022
a=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000
a=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001

m=video 0 RTP/SAVPF 97
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=mid:2
a=msid:stream1 track2
a=ssrc:10033
        ]]>
    </sdp>
    <mediaIndicator>
        <type>Audio</type>
        <entryIdx>0</entryIdx>
        <entryId>1</entryId>
        <streamId>stream1</streamId>
        <trackId>track1</trackId>
        <payload>
            <payloadType>0</payloadType>
            <encoding>PCMU</encoding>
        </payload>
        <direction>SendRecv</direction>
    </mediaIndicator>
</wrtcs:wrtcsOffer>
```

6.8.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT, DELETE’ field in the response as per section 14.7 of [RFC2616].

6.8.6 DELETE

This operation is used by the Update Originator to cancel an update offer, and by the Update Recipient to decline an update offer.

6.8.6.1 Example: Cancelling or declining an update (Informative)

Alice’s application declines an update offer.

6.8.6.1.1 Request

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/update HTTP/1.1
Accept: application/xml
Host: example.com
```

6.8.6.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.9 Resource: ICE status of a WebRTC session

The resource used is:

http://{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions/{sessionId}/ice/status

This resource represents the status of the ICE connectivity checks for the WebRTC session.

6.9.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
userId	Identifier of the user on whose behalf the application acts Examples: tel:+19585550100, acr:pseudonym123, sip:alice@example.com
sessionId	Identifier of the WebRTC session

See section 6 for a statement on the escaping of reserved characters in URL variables.

6.9.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.9.3 GET

This operation is used to retrieve the ICE status of the WebRTC session.

6.9.3.1 Example: Reading the ICE status of a WebRTC session (Informative)

Alice's application reads the ICE status of the session.

6.9.3.1.1 Request

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/ice/status HTTP/1.1
Accept: application/xml
Host: example.com
```

6.9.3.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsIceStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>New</status>
</wrtcs:wrtcsIceStatus>
```

6.9.4 PUT

This operation is used for updating the ICE status of the WebRTC session.

200 OK and 204 No Content are valid success responses.

6.9.4.1 Example: Updating the ICE status of a WebRTC session (Informative)

Alice’s application updates the ICE status of the session.

6.9.4.1.1 Request

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/ice/status HTTP/1.1
Content-Type: application/xml
Content-Length: nnnn
Accept: application/xml
Host: example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsIceStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Connected</status>
</wrtcs:wrtcsIceStatus>
```

6.9.4.1.2 Response

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsIceStatus xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <status>Connected</status>
</wrtcs:wrtcsIceStatus>
```

6.9.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.9.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET, PUT’ field in the response as per section 14.7 of [RFC2616].

6.10 Resource: Client notification about WebRTC signaling events

This resource is a callback URL provided by the client for notifications about WebRTC session event notifications.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.10.5.

The following table applies to notifications related to WebRTC signaling events:

EventType	**Notification Root Element Type**	**Notification sent to**	**Response to Notification**	**Link rel**	**Link href** Base URL: //{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions
Cancelled	WrtcsEventNotification	Participant, Update Recipient	n/a	WrtcsSession	/{sessionId}
SessionEnded	WrtcsEventNotification	Participants	n/a	WrtcsSession	/{sessionId}
Declined	WrtcsEventNotification	Originator, Update Originator	n/a	WrtcsSession	/{sessionId}
NoAnswer	WrtcsEventNotification	Originator	n/a	WrtcsSession	{sessionId}
NotReachable	WrtcsEventNotification	Originator	n/a	WrtcsSession	/{sessionId}
Ringing	WrtcsEventNotification	Originator	n/a	WrtcsSession	/{sessionId}
Busy	WrtcsEventNotification	Originator	n/a	WrtcsSession	/{sessionId}

If the event type is one of Cancelled, SessionEnded, Declined, NoAnswer, NotReachable and Busy, the underlying session changes its status to “Closed”. Resources representing closed sessions can be removed from the server immediately, or after a time period defined by service provider policies.

6.10.1 Request URL variables

Client provided if any.

6.10.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.10.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.10.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.10.5 POST

This operation is used to notify the client about WebRTC signaling events.

6.10.5.1 Example: Notify a client about the "Ringing" event (Informative)

Alice's application is informed that Bob is being alerted.

6.10.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsEventNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsSession"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001"/>
   <link rel="WrtcsNotificationSubscription"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
   <eventType>Ringing</eventType>
   <eventDescription>The called party is being alerted.</eventDescription>
</wrtcs:wrtcsEventNotification>
```

6.10.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.10.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.11 Resource: Client notification about WebRTC session invitation

This resource is a callback URL provided by the client for notifications about WebRTC session invitations.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.11.5.

The following table applies to WebRTC session invitation notifications:

EventType	Notification Root Element Type	Notification sent to	Response to Notification	Link rel	Link href Base URL: //{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions
n/a	WrtcsSessionInvitationNotification	Terminating Participant	accept (6.5.4) decline (6.4.6)	WrtcsSession	/{sessionId}

The resource URL of the resource representing the underlying WebRTC session is passed in the "href" attribute of the "link" element with rel="WrtcsSession".

To accept the session invitation request, the application of the Terminating Participant MUST update the status of the session as defined in section 6.5.4. The status is represented by the child "/status" of the resource representing the WebRTC session.

To decline the session invitation request, the application of the Terminating Participant MUST destroy the resource representing the underlying WebRTC session as defined in section 6.4.6.

6.11.1 Request URL variables

Client provided if any.

6.11.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.11.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.11.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.11.5 POST

This operation is used to notify the client about WebRTC session invitations.

6.11.5.1 Example: Notify a client about a WebRTC session invitation (Informative)

Alice’s application is informed that Caesar invites Alice to a WebRTC session.

6.11.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionInvitationNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <callbackData>abcd</callbackData>
  <link rel="WrtcsSession"
    href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002"/>
  <link rel="WrtcsNotificationSubscription"
    href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
  <originatorAddress>tel:+19585550102</originatorAddress>
  <originatorName>Caesar</originatorName>
  <tParticipantAddress>tel:+19585550100</tParticipantAddress>
  <tParticipantName>Alice</tParticipantName>
  <offer>
    <type>Remote</type>
    <sdp>
      <![CDATA[v=0
o=caesar 86765415341651786102 0 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be
a=ice-pwd:Ld0K23q46KJGu7643dcIUT
a=ice-ufrag:3yXa
a=ice-lite

m=audio 30000 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10144
a=candidate:1 1 UDP 2130706431 192.0.2.1 30000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host
      ]]>
    </sdp>
    <mediaIndicator>
      <type>Audio</type>
      <entryIdx>0</entryIdx>
      <entryId>1</entryId>
      <streamId>stream1</streamId>
      <trackId>track1</trackId>
```

```
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <payload>
                <payloadType>96</payloadType>
                <encoding>opus</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
</wrtcs:wrtcsSessionInvitationNotification>
```

6.11.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.11.5.2 Example: Notify a client about a WebRTC session invitation without offer (aka offerless invite) (Informative)

Alice's application is informed that Alice is invited to a WebRTC session, and that it is solicited to provide an offer.

6.11.5.2.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSessionInvitationNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <callbackData>abcd</callbackData>
    <link rel="WrtcsSession"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002"/>
    <link rel="WrtcsNotificationSubscription"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
    <originatorAddress>tel:+19585550102</originatorAddress>
    <originatorName>Caesar</originatorName>
    <tParticipantAddress>tel:+19585550100</tParticipantAddress>
    <tParticipantName>Alice</tParticipantName>
</wrtcs:wrtcsSessionInvitationNotification>
```

6.11.5.2.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.11.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.12 Resource: Client notification about session invitation acceptance or session update acceptance

This resource is a callback URL provided by the client for notifications about the acceptance of session invitations or session updates.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.12.5.

To WebRTC session invitation / update acceptance notifications, the following table applies:

EventType	**Notification Root Element Type**	**Notification sent to**	**Response to Notification**	**Link rel**	**Link href** Base URL: //{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions
n/a	WrtcsAcceptanceNotification	Originator	n/a	WrtcsSession	/{sessionId}

The resource URL of the resource representing the underlying WebRTC session is passed in the “href” attribute of the “link” element with rel=”WrtcsSession”.

The accepted offer can be found in the “offer” child element of the session referenced by the above link.

The notification includes an “answer” child element if an answer was provided by the underlying network as part of declaring acceptance. Note that an answer can also be sent earlier than declaring acceptance; in such a case the notification does not include an “answer” child element. The “answer” child MUST also be available in the session resource referenced from the notification, regardless of whether or not it has been embedded in the notification.

6.12.1 Request URL variables

Client provided if any.

6.12.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.12.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: POST’ field in the response as per section 14.7 of [RFC2616].

6.12.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: POST’ field in the response as per section 14.7 of [RFC2616].

6.12.5 POST

This operation is used to notify the client about WebRTC session invitation / session update acceptance.

6.12.5.1 Example: Notify a client about session invitation acceptance / update acceptance, including answer (Informative)

Alice’s application is informed that Bob has accepted the session invitation, and receives an answer SDP.

6.12.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAcceptanceNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsSession"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001"/>
   <link rel="WrtcsNotificationSubscription"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
   <answer>
      <type>Remote</type>
      <isProvisional>false</isProvisional>
      <sdp>
         <![CDATA[v=0
o=bob 98746513249823567101 0 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=ice-pwd:YH75Fviy6338Vbrhrlp8Yh
a=ice-ufrag:9uB6
a=ice-lite

m=audio 20000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10122
a=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host         ]]>
      </sdp>
      <mediaIndicator>
         <type>Audio</type>
         <entryIdx>0</entryIdx>
         <entryId>1</entryId>
         <streamId>stream1</streamId>
         <trackId>track1</trackId>
         <payload>
            <payloadType>0</payloadType>
            <encoding>PCMU</encoding>
         </payload>
```

```
            <direction>SendRecv</direction>
        </mediaIndicator>
    </answer>
</wrtcs:wrtcsAcceptanceNotification>
```

6.12.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.12.5.2 Example: Notify a client about session invitation acceptance / update acceptance, without answer (Informative)

Alice's application is informed that Bob has accepted the session invitation. The answer SDP has been received in a previous notification.

6.12.5.2.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAcceptanceNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <callbackData>abcd</callbackData>
    <link rel="WrtcsSession"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001"/>
    <link rel="WrtcsNotificationSubscription"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
</wrtcs:wrtcsAcceptanceNotification>
```

6.12.5.2.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.12.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.13 Resource: Client notification about update offer in a WebRTC session

This resource is a callback URL provided by the client for notifications about update offers in a WebRTC session.

The RESTful WebRTC API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.13.5.

The following table applies to WebRTC update offer notifications:

EventType	Notification Root Element Type	Notification sent to	Response to Notification	Link rel	Link href Base URL: //{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions
n/a	WrtcsOfferNotification	Update Recipient	accept (6.7.4) decline (6.8.6)	WrtcsSession	/{sessionId}

The resource URL of the resource representing the underlying WebRTC session is passed in the "href" attribute of the "link" element with rel="WrtcsSession".

The application MUST either accept or decline the offer contained in the notification.

- To accept the offer, the application MUST create an answer, and update the "answer" object of the session as defined in section 6.7.4. The "answer" object of the session is represented by the child "answer" of the resource representing the WebRTC session.
- To decline the offer, the application MUST destroy the resource representing the "update offer" object in the underlying WebRTC session as defined in section 6.8.6. The "update offer" object of the session is represented by the child "update" of the resource representing the WebRTC session.

6.13.1 Request URL variables

Client provided if any.

6.13.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.13.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.13.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.13.5 POST

This operation is used to notify the client about an update offer in a WebRTC session.

6.13.5.1 Example: Notify a client about an update offer in a WebRTC session, adding video (Informative)

Alice's application receives an update offer that adds video to an audio-only session.

6.13.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOfferNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsSession"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002"/>
   <link rel="WrtcsNotificationSubscription"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
   <offer>
      <type>Remote</type>
      <sdp>
         <![CDATA[v=0
o=caesar 86765415341651786102 1 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be
a=ice-pwd:Ld0K23q46KJGu7643dcIUT
a=ice-ufrag:3yXa
a=ice-lite

m=audio 30000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10144
a=candidate:1 1 UDP 2130706431 192.0.2.1 30000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host

m=video 30300 RTP/SAVPF 97 98
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10155
a=candidate:1 1 UDP 2130706431 10.0.1.1 9100 typ host
a=candidate:1 2 UDP 2130706430 10.0.1.1 9101 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.30 30300 typ srflx raddr 10.0.1.1 rport 9100
a=candidate:2 2 UDP 1694498814 192.0.2.30 30301 typ srflx raddr 10.0.1.1 rport 9101
         ]]>
      </sdp>
      <mediaIndicator>
         <type>Audio</type>
         <entryIdx>0</entryIdx>
         <entryId>1</entryId>
         <streamId>stream1</streamId>
         <trackId>track1</trackId>
```

```
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
        <mediaIndicator>
            <type>Video</type>
            <entryIdx>1</entryIdx>
            <entryId>2</entryId>
            <streamId>stream1</streamId>
            <trackId>track2</trackId>
            <payload>
                <payloadType>97</payloadType>
                <encoding>H264</encoding>
            </payload>
            <payload>
                <payloadType>98</payloadType>
                <encoding>VP8</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
</wrtcs:wrtcsOfferNotification>
```

6.13.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.13.5.2 Example: Notify a client about an update offer in a WebRTC session, removing video (Informative)

Alice’s application receives an update offer that removes video from an audio/video session.

6.13.5.2.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsOfferNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
    <callbackData>abcd</callbackData>
    <link rel="WrtcsSession"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002"/>
    <link rel="WrtcsNotificationSubscription"
        href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
    <offer>
        <sdp>
            <![CDATA[v=0
o=caesar 86765415341651786102 2 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be
a=ice-pwd:Ld0K23q46KJGu7643dcIUT
a=ice-ufrag:3yXa
a=ice-lite

m=audio 30000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10144
a=candidate:1 1 UDP 2130706431 192.0.2.1 30000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host

m=video 0 RTP/SAVPF 97
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=mid:2
a=msid:stream1 track2
a=ssrc:10155          ]]>
        </sdp>
        <mediaIndicator>
            <type>Audio</type>
            <entryIdx>0</entryIdx>
            <entryId>1</entryId>
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </offer>
</wrtcs:wrtcsOfferNotification>
```

6.13.5.2.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.13.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.14 Resource: Client notification about answer in a WebRTC session

This resource is a callback URL provided by the client for notifications about answers in a WebRTC session.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.13.5.

The following table applies to WebRTC answer notifications:

EventType	**Notification Root Element Type**	**Notification sent to**	**Response to Notification**	**Link rel**	**Link href** Base URL: //{serverRoot}/webrtcsignaling /{apiVersion}/{userId}/session s
n/a	WrtcsAnswerNotifi cation	Originator, Terminating Participant, Update Originator	n/a	WrtcsSession	/{sessionId}

The resource URL of the resource representing the underlying WebRTC session is passed in the “href” attribute of the “link” element with rel=”WrtcsSession”.

Depending on the actual flow, this notification may be sent to Originator (answer to a session invitation), Terminating Participant (answer to an offer in an offerless session invitation) or Update Originator (answer to an update offer).

The application needs to take notice of the state change of the session signaled by the answer. If the application runs in a web browser supporting WebRTC [W3C_WebRTC], this usually means to install the answer in the PeerConnection object representing the session.

6.14.1 Request URL variables

Client provided if any.

6.14.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.14.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: POST’ field in the response as per section 14.7 of [RFC2616].

6.14.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: POST’ field in the response as per section 14.7 of [RFC2616].

6.14.5 POST

This operation is used to notify the client about an update offer in a WebRTC session.

6.14.5.1 Example: Notify a client about an answer in a WebRTC session(Informative)

Alice's application receives an answer from Bob.

6.14.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsAnswerNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsSession"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001"/>
   <link rel="WrtcsNotificationSubscription"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
   <answer>
      <type>Remote</type>
      <isProvisional>false</isProvisional>
      <sdp>
         <![CDATA[v=0
o=bob 98746513249823567101 0 IN IP4 192.0.2.1
s=
t=0 0
c=IN IP4 192.0.2.1
a=msid-semantic:WMS
a=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=ice-pwd:YH75Fviy6338Vbrhrlp8Yh
a=ice-ufrag:9uB6
a=ice-lite

m=audio 20000 RTP/SAVPF 0
a=rtpmap:0 PCMU/8000
a=sendrecv
a=mid:1
a=msid:stream1 track1
a=ssrc:10122
a=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host

m=video 20200 RTP/SAVPF 97
a=rtpmap:97 H264/90000
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1
a=sendrecv
a=mid:2
a=msid:stream1 track2
a=ssrc:10133
a=candidate:1 1 UDP 2130706431 192.0.2.1 20200 typ host
a=candidate:1 2 UDP 2130706430 192.0.2.1 20201 typ host
         ]]>
      </sdp>
      <mediaIndicator>
         <type>Audio</type>
         <entryIdx>0</entryIdx>
         <entryId>1</entryId>
```

```
            <streamId>stream1</streamId>
            <trackId>track1</trackId>
            <payload>
                <payloadType>0</payloadType>
                <encoding>PCMU</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
        <mediaIndicator>
            <type>Video</type>
            <entryIdx>1</entryIdx>
            <entryId>2</entryId>
            <streamId>stream1</streamId>
            <trackId>track2</trackId>
            <payload>
                <payloadType>97</payloadType>
                <encoding>H264</encoding>
            </payload>
            <direction>SendRecv</direction>
        </mediaIndicator>
    </answer>
</wrtcs:wrtcsAnswerNotification>
```

6.14.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.14.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.15 Resource: Client notification about subscription cancellation

This resource is a callback URL provided by the client for notifications about subscription cancellation.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.15.5.

The notification is sent by the server to the user to whom the cancelled subscription belongs.

The following table applies to WebRTC subscription cancellation notifications:

EventType	Notification Root Element Type	Notification sent to	Response to Notification	Link rel	Link href Base URL: //{serverRoot}/webrtcsignaling /{apiVersion}/{userId}
n/a	WrtcsSubscription CancellationNotific ation	subscriber	n/a	WrtcsNotificationSubscriptio n	/subscriptions/{subscriptionId}

6.15.1 Request URL variables

Client provided if any.

6.15.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.15.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.15.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.15.5 POST

This operation is used to notify the client about a cancelled subscription, e.g. due to expiry or an error.

6.15.5.1 Example: Notify a client about subscription cancellation due to expiry (Informative)

Alice's application is informed about subscription expiry.

6.15.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSubscriptionCancellationNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsNotificationSubscription"
      href="http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
</wrtcs:wrtcsSubscriptionCancellationNotification>
```

6.15.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.15.5.2 Example: Notify a client about subscription cancellation due to an error (Informative)

Alice's application is informed about subscription expiry due to an error.

6.15.5.2.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsSubscriptionCancellationNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
  <callbackData>abcd</callbackData>
  <link rel="WrtcsNotificationSubscription"
    href="http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
  <reason>
    <messageId>SVC2001</messageId>
    <text>No server resources available to process the request </text>
  </reason>
</wrtcs:wrtcsSubscriptionCancellationNotification>
```

6.15.5.2.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.15.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.16 Resource: Client notification about conflicts

This resource is a callback URL provided by the client for notifications about conflicts.

The RESTful WebRTC Signaling API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel to obtain the notifications, the client needs to use the mechanisms described in [REST_NetAPI_NotificationChannel], instead of the mechanism described in section 6.15.5.

The following table applies to WebRTC conflict notifications:

EventType	Notification Root Element Type	Notification sent to	Response to Notification	Link rel	Link href Base URL: //{serverRoot}/webrtcsignaling/{apiVersion}/{userId}/sessions
n/a	WrtcsConflictNotification	Depends	n/a	WrtcsSession WrtcsOffer	/{sessionId} /{sessionId}/offer or /{sessionId}/update

The resource URL of the resource representing the underlying WebRTC session is passed in the "href" attribute of the "link" element with rel="WrtcsSession".

A reference to the initial offer or update offer that needs to be rolled back to resolve the conflict is passed in the "href" attribute of the "link" element with rel="WrtcsOffer".

6.16.1 Request URL variables

Client provided if any.

6.16.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to the RESTful Network API for WebRTC Signaling, see section 7.

6.16.3 GET

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.16.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: POST' field in the response as per section 14.7 of [RFC2616].

6.16.5 POST

This operation is used to notify the client about a conflict that violates the offer-answer sequence rules. Such conflict is typically resolved by rolling back the offer that caused the conflict, which is referenced via a link from the notification.

6.16.5.1 Example: Notify a client about a conflict (Informative)

Alice's application is informed about a conflict regarding the update offer.

6.16.5.1.1 Request

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

<?xml version="1.0" encoding="UTF-8"?>
<wrtcs:wrtcsConflictNotification xmlns:wrtcs="urn:oma:xml:rest:netapi:webrtcsignaling:1">
   <callbackData>abcd</callbackData>
   <link rel="WrtcsSession"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001"/>
   <link rel="WrtcsOffer"
      href="http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update"/>
   <link rel="WrtcsNotificationSubscription"
      href=" http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"/>
   <reason>
      <messageId>SVC1007</messageId>
      <text>Offer rejected due to conflict.</text>
   </reason>
</wrtcs:wrtcsConflictNotification>
```

6.16.5.1.2 Response

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

6.16.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: POST’ field in the response as per section 14.7 of [RFC2616].

7. Fault definitions

7.1 Service Exceptions

For common Service Exceptions refer to [REST_NetAPI_Common]. The following additional Service Exception codes are defined for the RESTful WebRTC Signaling API.

7.1.1 SVC1007: Offer rejected due to conflict

Name	Description
MessageID	SVC1007
Text	Offer rejected due to conflict
Variables	None
HTTP status code(s)	403 Forbidden

The offer-answer model mandates that there is at most one unanswered offer at any point in time during a session. The exception above is thrown if this constraint is violated by the client (e.g. by sending another offer while the answer to the previous offer is still pending), or if a race condition in the network has led to a violation of that constraint.

7.2 Policy Exceptions

For common Policy Exceptions refer to [REST_NetAPI_Common]. There are no additional Policy Exception codes defined for the RESTful WebRTC Signaling API.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-REST_NetAPI_WebRTCSignaling-V1_0-20170131-A	31 Jan 2017	Status changed to Approved by TP TP Ref # OMA-TP-2017-0004-INP_REST_NetAPI_WebRTCSignaling-V1_0_ERP_for_Final_Approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for REST.WRTCSIG Server

Item	Function	Reference	Requirement
REST-WRTCSIG-SUPPORT-S-001-M	Support for the RESTful WebRTC Signaling API	5,6	
REST- WRTCSIG-SUPPORT-S-002-M	Support for the XML request & response format	6	
REST- WRTCSIG-SUPPORT-S-003-M	Support for the JSON request & response format	6	

B.1.1 SCR for REST.WRTCSIG.Subscriptions Server

Item	Function	Reference	Requirement
REST-WRTCSIG-SUBSCR-S-001-M	Support for subscriptions to notifications regarding WebRTC Signaling events	6.1	
REST-WRTCSIG-SUBSCR-S-002-O	Read the list of active subscriptions – GET	6.1.3	
REST-WRTCSIG-SUBSCR-S-003-M	Create new subscription – POST	6.1.5	

B.1.2 SCR for REST.WRTCSIG.IndSubscription Server

Item	Function	Reference	Requirement
REST-WRTCSIG-INDSUBSCR-S-001-M	Support for accessing an individual subscription to notifications regarding WebRTC Signaling events	6.2	
REST-WRTCSIG-INDSUBSCR-S-002-O	Read an individual subscription – GET	6.2.3	
REST-WRTCSIG-INDSUBSCR-S-003-M	Cancel subscription and stop corresponding notifications – DELETE	6.2.6	

B.1.3 SCR for REST.WRTCSIG.Sessions Server

Item	Function	Reference	Requirement
REST-WRTCSIG-SESS-S-001-M	Support for WebRTC sessions	6.3	
REST-WRTCSIG-SESS-S-002-M	Create a new WebRTC session – POST	6.3.5	

B.1.4 SCR for REST.WRTCSIG.IndSession Server

Item	Function	Reference	Requirement
REST-WRTCSIG-INDSESS-S-001-M	Support for individual WebRTC sessions	6.4	
REST-WRTCSIG-INDSESS-S-002-O	Retrieve information about an individual	6.4.3	

Item	Function	Reference	Requirement
	WebRTC session – GET		
REST-WRTCSIG-INDSESS-S-003-M	Terminate individual WebRTC session – DELETE	6.4.6	
REST-WRTCSIG-INDSESS-S-004-M	Cancel WebRTC session invitation – DELETE	6.4.6	
REST-WRTCSIG-INDSESS-S-005-M	Decline WebRTC session invitation _DELETE	6.4.6	

B.1.5 SCR for REST.WRTCSIG.IndSession.Status Server

Item	Function	Reference	Requirement
REST-WRTCSIG-STATUS-S-001-M	Support for WebRTC session status	6.5	
REST-WRTCSIG-STATUS-S-002-M	Retrieve WebRTC session status – GET	6.5.3	
REST-WRTCSIG-STATUS-S-003-M	Update WebRTC session status – PUT	6.5.4	

B.1.6 SCR for REST.WRTCSIG.IndSession.Offer Server

Item	Function	Reference	Requirement
REST-WRTCSIG-OFFER-S-001-M	Support for offer in a WebRTC session	6.6	
REST-WRTCSIG-OFFER-S-002-M	Retrieve offer – GET	6.6.3	
REST-WRTCSIG-OFFER-S-003-M	Provide offer – PUT	6.6.4	

B.1.7 SCR for REST.WRTCSIG.IndSession.Answer Server

Item	Function	Reference	Requirement
REST-WRTCSIG-ANSWER-S-001-M	Support for answer in a WebRTC session	6.7	
REST-WRTCSIG-ANSWER -S-002-M	Retrieve answer – GET	6.7.3	
REST-WRTCSIG-ANSWER-S-003-M	Provide answer – PUT	6.7.4	

B.1.8 SCR for REST.WRTCSIG.IndSession.Update Server

Item	Function	Reference	Requirement
REST-WRTCSIG-UPDATE-S-001-M	Support for update offer in a WebRTC session	6.8	
REST-WRTCSIG-UPDATE-S-002-M	Retrieve update offer – GET	6.8.3	
REST-WRTCSIG-UPDATE-S-003-M	Initiate update offer – PUT	6.8.4	
REST-WRTCSIG-UPDATE-S-004-M	Cancel / Decline update offer – DELETE	6.8.6	

B.1.9 SCR for REST.WRTCSIG.IndSession.IceStatus Server

Item	Function	Reference	Requirement
REST-WRTCSIG-	Support for ICE status in	6.9	

Item	Function	Reference	Requirement
ICESTAT-S-001-M	a WebRTC session		
REST-WRTCSIG-ICESTAT-S-002-M	Retrieve ICE status – GET	6.9.3	
REST-WRTCSIG-ICESTAT-S-003-M	Update ICE status– PUT	6.9.4	

B.1.10 SCR for REST.WRTCSIG.Notifications.Event Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-EVENT-S-001-M	Support for notifications about WebRTC signaling events	6.10	
REST-WRTCSIG-NOTIF-EVENT-S-002-M	Notification about WebRTC signaling event – POST	6.10.5	

B.1.11 SCR for REST.WRTCSIG.Notifications.Invite Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-INVITE-S-001-M	Support for notifications about WebRTC session invitations	6.11	
REST-WRTCSIG-NOTIF-INVITE-S-002-M	Notification about WebRTC session invitation – POST	6.11.5	

B.1.12 SCR for REST.WRTCSIG.Notifications.Acceptance Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-ACCEPT-S-001-M	Support for notifications about acceptance of session invitations or updates	6.12	
REST-WRTCSIG-NOTIF-ACCEPT-S-002-M	Notification about acceptance of session invitation or update – POST	6.12.5	

B.1.13 SCR for REST.WRTCSIG.Notifications.Offer Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-OFFER-S-001-M	Support for notifications about update offer in a WebRTC session	6.13	
REST-WRTCSIG-NOTIF-OFFER-S-002-M	Notification about update offer offer in a WebRTC session – POST	6.13.5	

B.1.14 SCR for REST.WRTCSIG.Notifications.Answer Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-ANSWER-S-001-M	Support for notifications about answer in a WebRTC session	6.14	
REST-WRTCSIG-NOTIF-ANSWER-S-002-M	Notification about answer in a WebRTC session – POST	6.14.5	

B.1.15 SCR for REST.WRTCSIG.Notifications.SubscriptionCancellation Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-SUBCXL-S-001-M	Support for notifications about subscription cancellation	6.15	
REST-WRTCSIG-NOTIF-SUBCXL-S-002-M	Notification about subscription cancellation – POST	6.15.5	

B.1.16 SCR for REST.WRTCSIG.Notifications.Conflict Server

Item	Function	Reference	Requirement
REST-WRTCSIG-NOTIF-CONFLICT-S-001-M	Support for notifications about conflicts	6.16	
REST-WRTCSIG-NOTIF-CONFLICT-S-002-M	Notification about conflict – POST	6.16.5	

Appendix C. Application/x-www-form-urlencoded Request Format for POST Operations (Normative)

This specification does not define any API request based on application/x-www-form-urlencoded MIME type.

Appendix D. JSON examples (Informative)

JSON (JavaScript Object Notation) is a Light-weight, text-based, language-independent data interchange format. It provides a simple means to represent basic name-value pairs, arrays and objects. JSON is relatively trivial to parse and evaluate using standard JavaScript libraries, and hence is suited for REST invocations from browsers or other processors with JavaScript engines. Further information on JSON can be found at [RFC4627].

The following examples show the request and response for various operations using the JSON data format. The examples follow the XML to JSON serialization rules in [REST_NetAPI_Common]. A JSON response can be obtained by using the content type negotiation mechanism specified in [REST_NetAPI_Common].

For full details on the operations themselves please refer to the section number indicated.

D.1 Reading all active subscriptions (section 6.1.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsSubscriptionList": {
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions",
    "wrtcsNotificationSubscription": {
        "callbackReference": {
            "callbackData": "abcd",
            "notifyURL": "http://application-alice.example.com/webrtcsignaling/notifications/77777"
        },
        "clientCorrelator": "12345",
        "duration": "7037",
        "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"
    }
}}
```

D.2 Creating a new subscription, response with copy of created resource (section 6.1.5.1)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsNotificationSubscription": {
    "callbackReference": {
        "callbackData": "abcd",
        "notifyURL": "http://application-alice.example.com/webrtcsignaling/notifications/77777"
    },
    "clientCorrelator": "12345",
    "duration": "7200"
}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsNotificationSubscription": {
    "callbackReference": {
        "callbackData": "abcd",
        "notifyURL": "http://application-alice.example.com/webrtcsignaling/notifications/77777"
    },
    "clientCorrelator": "12345",
    "duration": "7200",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"
}}
```

D.3 Creating a new subscription, response with location of created resource (section 6.1.5.2)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsNotificationSubscription": {
    "callbackReference": {
        "callbackData": "abcd",
        "notifyURL": "http://application-alice.example.com/webrtcsignaling/notifications/77777"
    },
    "clientCorrelator": "12345",
    "duration": "7200"
}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"resourceReference": {"resourceURL":
```

```
"http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"}}
```

D.4 Reading an individual subscription (section 6.2.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001 HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsNotificationSubscription": {
   "callbackReference": {
      "callbackData": "abcd",
      "notifyURL": "http://application-alice.example.com/webrtcsignaling/notifications/77777"
   },
   "clientCorrelator": "12345",
   "duration": "7200",
   "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001"
}}
```

D.5 Cancelling a subscription (section 6.2.6.1)

Request:

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001 HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.6 Creating a new WebRTC session – audio only, using tel URI (section 6.3.5.1)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: example.com
Content-Length: nnnn

{"wrtcsSession": {
    "clientCorrelator": "4567",
    "offer": {"sdp": "v=0\no=alice 894656765465714481000 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001"},
    "originatorAddress": "tel:+19585550100",
    "originatorName": "Alice",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001

{"wrtcsSession": {
    "clientCorrelator": "4567",
    "offer": {
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": [
                {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                {
                    "encoding": "opus",
                    "payloadType": "96"
                }
            ],
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=alice 894656765465714481000 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001",
        "type": "Local"
    },
    "originatorAddress": "tel:+19585550100",
    "originatorName": "Alice",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
    "status": "Initiated",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

D.7 Creating a new WebRTC session – audio only, using SIP URI and encoding the SDP with base64 (section 6.3.5.2)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: example.com
Content-Length: nnnn

{"wrtcsSession": {
   "clientCorrelator": "4567",
   "offer": {"sdpBase64":
"ICAgICAgICAgICAgdj0wDQogICAgICAgICAgICBvPWFsaWNlc19icm93c2VyIDI4OTA4NDQ1MjYg\nMjg5MDg0MjgwNyBJTiBJUDQgMTA
uMC4xLjENCiAgICAgICAgICAgIHM9DQogICAgICAgICAgICB0\nPTAgMA0KICAgICAgICAgICAgYz1JTiBJUDQgMTkyLjAuMi4zMA0KICA
gICAgICAgICAgYT1pY2Ut\ncHdkOmFzZDg4ZmdwZGQ3Nzd1empZaGFnWmcNCiAgICAgICAgICAgIGE9aWNlLXVmcmFnOjhoaFkN\nCi
AgICAgICAgICAgIGE9ZmluZ2VycHJpbnQ6c2hhLTEgOTk6NDE6NDk6ODM6NGE6OTc6MGU6MWY6\nZWY6NmQ6Zjc6Yzk6Yzc6NzA6
OWQ6MWY6NjY6Nzk6YTg6MDcNCiAgICAgICAgICAgIA0KICAgICAg\nICAgICAgbT1hdWRpbyAxMDAwMCBSVFAvU0FWUEYgMCA5Ng
0KICAgICAgICAgICAgYT1ydHBtYXA6\nMCBQQ01VLzgwMDANCiAgICAgICAgICAgIGE9cnRwbWFwOjk2IG9wdXMvNDgwMDANCiAgIC
AgICAg\nICAgIGE9c2VuZHJlY3YNCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjEgMSBVRFAgMjEzMDcwNjQz\nMSAxMC4wLjEuMSA4MD
AwIHR5cCBob3N0DQogICAgICAgICAgICBhPWNhbmRpZGF0ZToxIDIgVURQ\nIDIxMzA3MDY0MzAgMTAuMC4xLjEgODAwMSB0eXAg
aG9zdA0KICAgICAgICAgICAgYT1jYW5kaWRh\ndGU6MiAxIFVEUCAxNjk0NDk4ODE1IDE5Mi4wLjIuMzAgMTAwMDAgdHlwIHNyZmx4IH
JhZGRyIDEw\nLjAuMS4xIHJwb3J0IDgwMDANCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjIgMiBVRFAgMTY5NDQ5\nODgxNCAxOTIuMC
4yLjMwIDEwMDAxIHR5cCBzcmZseCByYWRkciAxMC4wLjEuMSBycG9ydCA4MDAx\nDQo="},
   "tParticipantAddress": "tel:+19585550101",
   "tParticipantName": "Bob"

}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions/sess001

{"wrtcsSession": {
   "clientCorrelator": "4567",
   "offer": {
      "mediaIndicator": {
         "direction": "SendRecv",
         "entryId": "1",
         "entryIdx": "0",
         "payload": [
            {
               "encoding": "PCMU",
               "payloadType": "0"
            },
            {
               "encoding": "opus",
               "payloadType": "96"
            }
         ],
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdpBase64":
"ICAgICAgICAgICAgdj0wDQogICAgICAgICAgICBvPWFsaWNlc19icm93c2VyIDI4OTA4NDQ1MjYg\nMjg5MDg0MjgwNyBJTiBJUDQgMTA
uMC4xLjENCiAgICAgICAgICAgIHM9DQogICAgICAgICAgICB0\nPTAgMA0KICAgICAgICAgICAgYz1JTiBJUDQgMTkyLjAuMi4zMA0KICA
gICAgICAgICAgYT1pY2Ut\ncHdkOmFzZDg4ZmdwZGQ3Nzd1empZaGFnWmcNCiAgICAgICAgICAgIGE9aWNlLXVmcmFnOjhoaFkN\nCi
AgICAgICAgICAgIGE9ZmluZ2VycHJpbnQ6c2hhLTEgOTk6NDE6NDk6ODM6NGE6OTc6MGU6MWY6\nZWY6NmQ6Zjc6Yzk6Yzc6NzA6
OWQ6MWY6NjY6Nzk6YTg6MDcNCiAgICAgICAgICAgIA0KICAgICAg\nICAgICAgbT1hdWRpbyAxMDAwMCBSVFAvU0FWUEYgMCA5Ng
0KICAgICAgICAgICAgYT1ydHBtYXA6\nMCBQQ01VLzgwMDANCiAgICAgICAgICAgIGE9cnRwbWFwOjk2IG9wdXMvNDgwMDANCiAgIC
AgICAg\nICAgIGE9c2VuZHJlY3YNCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjEgMSBVRFAgMjEzMDcwNjQz\nMSAxMC4wLjEuMSA4MD
AwIHR5cCBob3N0DQogICAgICAgICAgICBhPWNhbmRpZGF0ZToxIDIgVURQ\nIDIxMzA3MDY0MzAgMTAuMC4xLjEgODAwMSB0eXAg
aG9zdA0KICAgICAgICAgICAgYT1jYW5kaWRh\ndGU6MiAxIFVEUCAxNjk0NDk4ODE1IDE5Mi4wLjIuMzAgMTAwMDAgdHlwIHNyZmx4IH
JhZGRyIDEw\nLjAuMS4xIHJwb3J0IDgwMDANCiAgICAgICAgICAgIGE9Y2FuZGlkYXRlOjIgMiBVRFAgMTY5NDQ5\nODgxNCAxOTIuMC
4yLjMwIDEwMDAxIHR5cCBzcmZseCByYWRkciAxMC4wLjEuMSBycG9ydCA4MDAx\nDQo=",
        "type": "Local"
    },
    "originatorAddress": "sip:alice@example.com",
    "originatorName": "Alice",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/sip%3Aalice%40example.com/sessions/sess001",
    "status": "Initiated",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"

}}
```

D.8 Creating a new WebRTC session – audio and video, using ACR (section 6.3.5.3)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: example.com
Content-Length: nnnn

{"wrtcsSession": {
    "clientCorrelator": "4567",
    "offer": {"sdp": "v=0\no=alice 89465676546571448100 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n\nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8100 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx
raddr 10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101"},
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001

{"wrtcsSession": {
   "clientCorrelator": "4567",
   "offer": {
      "mediaIndicator": [
         {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": [
               {
                  "encoding": "PCMU",
                  "payloadType": "0"
               },
               {
                  "encoding": "opus",
                  "payloadType": "96"
               }
            ],
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
         },
         {
            "direction": "SendRecv",
            "entryId": "2",
            "entryIdx": "1",
            "payload": [
               {
                  "encoding": "H264",
                  "formatParams": "profile-level-id=4d0028;packetization-mode=1",
                  "payloadType": "97"
               },
               {
                  "encoding": "VP8",
                  "payloadType": "98"
               }
            ],
            "streamId": "stream1",
            "trackId": "track2",
            "type": "Video"
         }
      ],
      "sdp": "v=0\no=alice 894656765465714481000 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n\nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98 
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1 
8100 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx 
raddr 10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101",
        "type": "Local"
    },
    "originatorAddress": "acr:pseudonym123",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001",
    "status": "Initiated",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

D.9 Creating a new WebRTC session – audio and video, using acr:auth (section 6.3.5.4)

Request:

```
POST /exampleAPI/webrtcsignaling/v1/acr%3Aauth/sessions HTTP/1.1
Authorization: Bearer mF_9.B5f-4.1JqM
Accept: application/json
Content-Type: application/json
Host: example.com
Content-Length: nnnn

{"wrtcsSession": {
    "clientCorrelator": "4567",
    "offer": {"sdp": "v=0\no=alice 894656765465714481000 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1 
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96 
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1 
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx 
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n\nm=video 10100 
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98 
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1 
8100 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx 
raddr 10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101"},
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT
Location: http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Aauth/sessions/sess001

{"wrtcsSession": {
    "clientCorrelator": "4567",
    "offer": {
        "mediaIndicator": [
            {
                "direction": "SendRecv",
                "entryId": "1",
                "entryIdx": "0",
                "payload": [
                    {
                        "encoding": "PCMU",
                        "payloadType": "0"
                    },
                    {
                        "encoding": "opus",
                        "payloadType": "96"
                    }
                ],
                "streamId": "stream1",
                "trackId": "track1",
                "type": "Audio"
            },
            {
                "direction": "SendRecv",
                "entryId": "2",
                "entryIdx": "1",
                "payload": [
                    {
                        "encoding": "H264",
                        "formatParams": "profile-level-id=4d0028;packetization-mode=1",
                        "payloadType": "97"
                    },
                    {
                        "encoding": "VP8",
                        "payloadType": "98"
                    }
                ],
                "streamId": "stream1",
                "trackId": "track2",
                "type": "Video"
            }
        ],
        "sdp": "v=0\no=alice 894656765465714448100 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n\nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8100 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx
raddr 10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101",
        "type": "Local"
    },
    "originatorAddress": "acr%3Aauth",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/acr%3Apseudonym123/sessions/sess001",
    "status": "Initiated",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob"
}}
```

D.10 Retrieving WebRTC session information (section 6.4.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001 HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsSession": {
    "answer": {
        "isProvisional": "false",
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=bob 98746513249823567101 0 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03\na=ice-
pwd:YH75Fviy6338Vbrhrlp8Yh\na=ice-ufrag:9uB6\na=ice-lite\n\nm=audio 20000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10122\na=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host ",
        "type": "Remote"
    },
    "clientCorrelator": "4567",
    "offer": {
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": [
                {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                {
                    "encoding": "opus",
                    "payloadType": "96"
                }
            ],
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=alice 894656765465714481000 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96 opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001",
        "type": "Local"
    },
    "originatorAddress": "tel:+19585550100",
    "originatorName": "Alice",
    "resourceURL": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
    "status": "Connected",
    "tParticipantAddress": "tel:+19585550101",
    "tParticipantName": "Bob",
    "update": {
        "mediaIndicator": [
            {
                "direction": "SendRecv",
                "entryId": "1",
                "entryIdx": "0",
                "payload": {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                "streamId": "stream1",
                "trackId": "track1",
                "type": "Audio"
            },
            {
                "direction": "SendRecv",
                "entryId": "2",
                "entryIdx": "1",
                "payload": [
                    {
                        "encoding": "H264",
                        "payloadType": "97"
                    },
                    {
                        "encoding": "VP8",
                        "payloadType": "98"
                    }
                ],
                "streamId": "stream1",
                "trackId": "track2",
                "type": "Video"
            }
        ],
        "sdp": "v=0\no=alice 894656765465714481000 1 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0\na=rtpmap:0 PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 10100 RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98 VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ
```

```
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr
10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101 ",
      "type": "Local"
   }
}}
```

D.11 Cancelling or terminating a WebRTC session, or declining a WebRTC session invitation (section 6.4.6.1)

Request:

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001 HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.12 Reading the status of a WebRTC session (section 6.5.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/status HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsSessionStatus": {"status": "Ringing"}}
```

D.13 Accepting a WebRTC session invitation (section 6.5.4.1)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/status HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsSessionStatus": {"status": "Connected"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsSessionStatus": {"status": "Connected"}}
```

D.14 Indicating the alerting of the Terminating Participant (“Ringing”) (section 6.5.4.2)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/status HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsSessionStatus": {"status": "Ringing"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsSessionStatus": {"status": "Ringing"}}
```

D.15 Reading initial or most recent offer in a WebRTC session (section 6.6.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/offer HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsOffer": {
    "mediaIndicator": {
        "direction": "SendRecv",
        "entryId": "1",
        "entryIdx": "0",
        "payload": [
            {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            {
                "encoding": "opus",
                "payloadType": "96"
            }
        ],
        "streamId": "stream1",
        "trackId": "track1",
        "type": "Audio"
    },
    "sdp": "v=0\no=alice 89465676546571448100 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10000 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
8000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx
raddr 10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001",
    "type": "Local"
}}
```

D.16 Providing an offer to an offerless session invitation (section 6.6.4.1)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/offer HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsOffer": {"sdp": "v=0\no=alice 78643246856870134100 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=ice-pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10200 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10044\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
9000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx
raddr 10.0.1.1 rport 9000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsOffer": {
    "mediaIndicator": {
        "direction": "SendRecv",
        "entryId": "1",
        "entryIdx": "0",
        "payload": [
            {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            {
                "encoding": "opus",
                "payloadType": "96"
            }
        ],
        "streamId": "stream1",
        "trackId": "track1",
        "type": "Audio"
    },
    "sdp": "v=0\no=alice 78643246856870134100 0 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-semantic:WMS\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\na=fingerprint:sha-1
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\n\nm=audio 10200 RTP/SAVPF 0 96\na=rtpmap:0 PCMU/8000\na=rtpmap:96
opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10044\na=rtcp-mux\na=candidate:1 1 UDP 2130706431 10.0.1.1
9000 typ host\na=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx
raddr 10.0.1.1 rport 9000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001",
    "type": "Local"
}}
```

D.17 Reading most recent answer in a WebRTC session (section 6.7.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/answer HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsAnswer": {
    "isProvisional": "false",
    "mediaIndicator": {
        "direction": "SendRecv",
        "entryId": "1",
        "entryIdx": "0",
        "payload": {
            "encoding": "PCMU",
            "payloadType": "0"
        },
        "streamId": "stream1",
        "trackId": "track1",
        "type": "Audio"
    },
    "sdp": "v=0\no=bob 98746513249823567101 0 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03\na=ice-
pwd:YH75Fviy6338Vbrhrlp8Yh\na=ice-ufrag:9uB6\na=ice-lite\n\nm=audio 20000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10122\na=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host",
  "type": "Remote"
}}
```

D.18 Providing an answer to an offer (section 6.7.4.1)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/answer HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsAnswer": {
  "isProvisional": "false",
  "sdp": "v=0\no=alice 786432468568701341000 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n\nm=audio 10200 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10044\na=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr
10.0.1.1 rport 9000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001"
}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsAnswer": {
  "isProvisional": "false",
  "mediaIndicator": {
    "direction": "SendRecv",
    "entryId": "1",
    "entryIdx": "0",
    "payload": {
      "encoding": "PCMU",
      "payloadType": "0"
    },
    "streamId": "stream1",
    "trackId": "track1",
    "type": "Audio"
  },
  "sdp": "v=0\no=alice 786432468568701341000 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n\nm=audio 10200 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10044\na=candidate:1 1 UDP 2130706431 10.0.1.1 9000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 9001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10200 typ srflx raddr
10.0.1.1 rport 9000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10201 typ srflx raddr 10.0.1.1 rport 9001",
  "type": "Local"
}}
```

D.19 Reading the update offer in a WebRTC session (section 6.8.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsOffer": {
    "mediaIndicator": [
        {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        {
            "direction": "SendRecv",
            "entryId": "2",
            "entryIdx": "1",
            "payload": [
                {
                    "encoding": "H264",
                    "payloadType": "97"
                },
                {
                    "encoding": "VP8",
                    "payloadType": "98"
                }
            ],
            "streamId": "stream1",
            "trackId": "track2",
            "type": "Video"
        }
    ],
    "sdp": "v=0\no=alice 89465676546571448100 1 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr
10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr
10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101",
  "type": "Local"
}}
```

D.20 Initiating an update offer in a WebRTC session to upgrade from audio-only to audio+video (section 6.8.4.1)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsOffer": {"sdp": "v=0\no=alice 894656765465714481000 1 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr
10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr
10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsOffer": {
  "mediaIndicator": [
    {
      "direction": "SendRecv",
      "entryId": "1",
      "entryIdx": "0",
      "payload": {
        "encoding": "PCMU",
        "payloadType": "0"
      },
      "streamId": "stream1",
      "trackId": "track1",
      "type": "Audio"
    },
    {
      "direction": "SendRecv",
      "entryId": "2",
      "entryIdx": "1",
            "payload": [
                {
                    "encoding": "H264",
                    "payloadType": "97"
                },
                {
                    "encoding": "VP8",
                    "payloadType": "98"
                }
            ],
            "streamId": "stream1",
            "trackId": "track2",
            "type": "Video"
        }
    ],
    "sdp": "v=0\no=alice 89465676546571448100 1 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr
10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 10100
RTP/SAVPF 97 98\na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10033\na=candidate:1 1 UDP 2130706431 10.0.1.1 8100 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10100 typ srflx raddr
10.0.1.1 rport 8100\na=candidate:2 2 UDP 1694498814 192.0.2.30 10101 typ srflx raddr 10.0.1.1 rport 8101",
    "type": "Local"
}}
```

D.21 Initiating an update offer in a WebRTC session to downgrade from audio+video to audio-only (section 6.8.4.2)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsOffer": {"sdp": "v=0\no=alice 89465676546571448100 2 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr
10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 0
RTP/SAVPF 97 \na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=mid:2\na=msid:stream1
track2\na=ssrc:10033"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsOffer": {
    "mediaIndicator": {
        "direction": "SendRecv",
        "entryId": "1",
        "entryIdx": "0",
        "payload": {
            "encoding": "PCMU",
            "payloadType": "0"
        },
        "streamId": "stream1",
        "trackId": "track1",
        "type": "Audio"
    },
    "sdp": "v=0\no=alice 89465676546571448100 2 IN IP4 10.0.1.1\ns=\nt=0 0\nc=IN IP4 192.0.2.30\na=msid-
semantic:WMS\na=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07\na=ice-
pwd:asd88fgpdd777uzjYhagZg\na=ice-ufrag:8hhY\n        \nm=audio 10000 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10022\na=candidate:1 1 UDP 2130706431 10.0.1.1 8000 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 8001 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 10000 typ srflx raddr
10.0.1.1 rport 8000\na=candidate:2 2 UDP 1694498814 192.0.2.30 10001 typ srflx raddr 10.0.1.1 rport 8001\n        \nm=video 0
RTP/SAVPF 97 \na=rtpmap:97 H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=mid:2\na=msid:stream1
track2\na=ssrc:10033",
    "type": "Local"
}}
```

D.22 Cancelling or declining an update (section 6.8.6.1)

Request:

```
DELETE /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002/update HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.23 Reading the ICE status of a WebRTC session (section 6.9.3.1)

Request:

```
GET /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/ice/status HTTP/1.1
Accept: application/json
Host: example.com
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsIceStatus": {"status": "New"}}
```

D.24 Updating the ICE status of a WebRTC session (section 6.9.4.1)

Request:

```
PUT /exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/ice/status HTTP/1.1
Content-Type: application/json
Content-Length: nnnn
Accept: application/json
Host: example.com

{"wrtcsIceStatus": {"status": "Connected"}}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnnn
Date: Fri, 28 Jun 2013 17:51:59 GMT

{"wrtcsIceStatus": {"status": "Connected"}}
```

D.25 Notify a client about the “Ringing” event (section 6.10.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsEventNotification": {
   "callbackData": "abcd",
   "eventDescription": "The called party is being alerted.",
   "eventType": "Ringing",
   "link": [
      {
         "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
         "rel": "WrtcsSession"
      },
      {
         "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
         "rel": "WrtcsNotificationSubscription"
      }
   ]
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.26 Notify a client about a WebRTC session invitation (section 6.11.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsSessionInvitationNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ],
    "offer": {
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": [
                {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                {
                    "encoding": "opus",
                    "payloadType": "96"
                }
            ],
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=caesar 8676541534165178610 2 0 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be\na=ice-
pwd:Ld0K23q46KJGu7643dcIUT\na=ice-ufrag:3yXa\na=ice-lite\n            \nm=audio 30000 RTP/SAVPF 0 96\na=rtpmap:0
PCMU/8000\na=rtpmap:96 opus/48000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10144\na=candidate:1 1 UDP
2130706431 192.0.2.1 30000 typ host\na=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host",
        "type": "Remote"
    },
    "originatorAddress": "tel:+19585550102",
    "originatorName": "Caesar",
    "tParticipantAddress": "tel:+19585550100",
    "tParticipantName": "Alice"
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.27 Notify a client about a WebRTC session invitation without offer (aka offerless invite) (section 6.11.5.2)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/xml
Content-Type: application/xml
Host: application-alice.example.com

{"wrtcsSessionInvitationNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ],
    "originatorAddress": "tel:+19585550102",
    "originatorName": "Caesar",
    "tParticipantAddress": "tel:+19585550100",
    "tParticipantName": "Alice"
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.28 Notify a client about WebRTC session invitation acceptance / update acceptance, including answer (section 6.12.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com
```

```
{"wrtcsAcceptanceNotification": {
    "answer": {
        "isProvisional": "false",
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=bob 98746513249823567101 0 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03\na=ice-
pwd:YH75Fviy6338Vbrhrlp8Yh\na=ice-ufrag:9uB6\na=ice-lite\n\nm=audio 20000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10122\na=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host ",
        "type": "Remote"
    },
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ]
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.29 Notify a client about WebRTC session invitation acceptance / update acceptance, without answer (section 6.12.5.2)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsAcceptanceNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ]
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.30 Notify a client about an update offer in a WebRTC session, adding video (section 6.13.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsOfferNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ],
    "offer": {
        "mediaIndicator": [
            {
                "direction": "SendRecv",
                "entryId": "1",
                "entryIdx": "0",
                "payload": {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                "streamId": "stream1",
                "trackId": "track1",
                "type": "Audio"
            },
            {
                "direction": "SendRecv",
                "entryId": "2",
                "entryIdx": "1",
                "payload": [
                    {
                        "encoding": "H264",
                        "payloadType": "97"
                    },
                    {
                        "encoding": "VP8",
                        "payloadType": "98"
                    }
                ],
                "streamId": "stream1",
                "trackId": "track2",
                "type": "Video"
            }
        ],
        "sdp": "v=0\no=caesar 86765415341651786102 1 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be\na=ice-
pwd:Ld0K23q46KJGu7643dcIUT\na=ice-ufrag:3yXa\na=ice-lite\n            \nm=audio 30000 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10144\na=candidate:1 1 UDP 2130706431 192.0.2.1 30000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host\n            \nm=video 30300 RTP/SAVPF 97 98\na=rtpmap:97
H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=rtpmap:98
VP8/90000\na=sendrecv\na=mid:2\na=msid:stream1 track2\na=ssrc:10155\na=candidate:1 1 UDP 2130706431 10.0.1.1 9100 typ
host\na=candidate:1 2 UDP 2130706430 10.0.1.1 9101 typ host\na=candidate:2 1 UDP 1694498815 192.0.2.30 30300 typ srflx raddr
10.0.1.1 rport 9100\na=candidate:2 2 UDP 1694498814 192.0.2.30 30301 typ srflx raddr 10.0.1.1 rport 9101",
        "type": "Remote"
    }
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.31 Notify a client about an update offer in a WebRTC session, removing video (section 6.13.5.2)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsOfferNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess002",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ],
    "offer": {
        "mediaIndicator": {
            "direction": "SendRecv",
            "entryId": "1",
            "entryIdx": "0",
            "payload": {
                "encoding": "PCMU",
                "payloadType": "0"
            },
            "streamId": "stream1",
            "trackId": "track1",
            "type": "Audio"
        },
        "sdp": "v=0\no=caesar 86765415341651786102 2 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 88:77:79:13:4f:32:0a:8b:21:ff:f3:a9:43:bc:d9:f3:11:82:71:be\na=ice-
pwd:Ld0K23q46KJGu7643dcIUT\na=ice-ufrag:3yXa\na=ice-lite\n            \nm=audio 30000 RTP/SAVPF 0 \na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10144\na=candidate:1 1 UDP 2130706431 192.0.2.1 30000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 30001 typ host\n\nm=video 0 RTP/SAVPF 97 \na=rtpmap:97 H264/90000\na=fmtp:97
profile-level-id=4d0028;packetization-mode=1\na=mid:2\na=msid:stream1 track2\na=ssrc:10155"
    }
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.32 Notify a client about an answer in a WebRTC session (section 6.14.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsAnswerNotification": {
    "answer": {
        "isProvisional": "false",
        "mediaIndicator": [
            {
                "direction": "SendRecv",
                "entryId": "1",
                "entryIdx": "0",
                "payload": {
                    "encoding": "PCMU",
                    "payloadType": "0"
                },
                "streamId": "stream1",
                "trackId": "track1",
                "type": "Audio"
            },
            {
                "direction": "SendRecv",
                "entryId": "2",
                "entryIdx": "1",
                "payload": {
                    "encoding": "H264",
                    "payloadType": "97"
                },
                "streamId": "stream1",
                "trackId": "track2",
                "type": "Video"
            }
        ],
        "sdp": "v=0\no=bob 98746513249823567101 0 IN IP4 192.0.2.1\ns=\nt=0 0\nc=IN IP4 192.0.2.1\na=msid-
semantic:WMS\na=fingerprint:sha-1 91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03\na=ice-
pwd:YH75Fviy6338Vbrhrlp8Yh\na=ice-ufrag:9uB6\na=ice-lite\n\nm=audio 20000 RTP/SAVPF 0\na=rtpmap:0
PCMU/8000\na=sendrecv\na=mid:1\na=msid:stream1 track1\na=ssrc:10122\na=candidate:1 1 UDP 2130706431 192.0.2.1 20000 typ
host\na=candidate:1 2 UDP 2130706430 192.0.2.1 20001 typ host\n\nm=video 20200 RTP/SAVPF 97\na=rtpmap:97
H264/90000\na=fmtp:97 profile-level-id=4d0028;packetization-mode=1\na=sendrecv\na=mid:2\na=msid:stream1
track2\na=ssrc:10133\na=candidate:1 1 UDP 2130706431 192.0.2.1 20200 typ host\na=candidate:1 2 UDP 2130706430 192.0.2.1 20201
typ host ",
        "type": "Remote"
    },
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
            "rel": "WrtcsSession"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ]
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.33 Notify a client about subscription cancellation due to expiry (section 6.15.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com
```

```
{"wrtcsSubscriptionCancellationNotification": {
    "callbackData": "abcd",
    "link": {
        "href": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
        "rel": "WrtcsNotificationSubscription"
    }
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.34 Example: Notify a client about subscription cancellation due to an error (section 6.15.5.2)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsSubscriptionCancellationNotification": {
    "callbackData": "abcd",
    "link": {
        "href": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
        "rel": "WrtcsNotificationSubscription"
    },
    "reason": {
        "messageId": "SVC2001",
        "text": "No server resources available to process the request "
    }
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

D.35 Notify a client about a conflict (section 6.16.5.1)

Request:

```
POST /webrtcsignaling/notifications/77777 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: application-alice.example.com

{"wrtcsConflictNotification": {
    "callbackData": "abcd",
    "link": [
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001",
            "rel": "WrtcsSession"
        },
        {
            "href": "http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/sessions/sess001/update",
            "rel": "WrtcsOffer"
        },
        {
            "href": " http://example.com/exampleAPI/webrtcsignaling/v1/tel%3A%2B19585550100/subscriptions/sub001",
            "rel": "WrtcsNotificationSubscription"
        }
    ],
    "reason": {
        "messageId": "SVC1007",
        "text": "Offer rejected due to conflict."
    }
}}
```

Response:

```
HTTP/1.1 204 No Content
Date: Fri, 28 Jun 2013 17:51:59 GMT
```

Appendix E. Operations mapping to pre-existing baseline specifications (Informative)

As this specification does not have a baseline specification, this appendix is empty.

Appendix F. Light-weight Resources (Informative)

As this version of the specification does not define any Light-weight Resources, this appendix is empty.

Appendix G. Authorization aspects (Normative)

This appendix specifies how to use the RESTful WebRTC Signaling API in combination with some authorization frameworks.

G.1 Use with OMA Authorization Framework for Network APIs

The RESTful WebRTC Signaling API MAY support the authorization framework defined in [Autho4API_10].

A RESTful WebRTC Signaling API supporting [Autho4API_10]:

- SHALL conform to section D.1 of [REST_NetAPI_Common];
- SHALL conform to this section G.1.

G.1.1 Scope values

G.1.1.1 Definitions

In compliance with [Autho4API_10], an authorization server serving clients requests for getting authorized access to the resources exposed by the RESTful WebRTC Signaling API:

- SHALL support the scope values defined in the table below;
- MAY support scope values not defined in this specification.

Scope value	**Description**	**For one-time access token**
oma_rest_wrtcs.all_{apiVersion}	Provide access to all defined operations on the resources in this version of the API. The {apiVersion} part of this identifier SHALL have the same value as the “apiVersion” URL variable which is defined in section 5.1. This scope value is the union of all other scope values that may be defined for this specification.	No

Table 1: Scope values for RESTful WebRTC Signaling API

G.1.1.2 Downscoping

Not applicable in this version of the specification as there is only one scope value defined.

G.1.1.3 Mapping with resources and methods

The single scope value defined in section G.1.1.1 above maps to all REST resources and methods defined in the subsections of section 6.

G.1.2 Use of ‘acr:auth’

This section specifies the use of ‘acr:auth’ in place of an end user identifier in a resource URL path.

An ‘acr’ URI of the form ‘acr:auth’, where ‘auth’ is a reserved keyword MAY be used to avoid exposing a real end user identifier in the resource URL path.

A client MAY use ‘acr:auth’ in a resource URL in place of a {userId} when the RESTful WebRTC Signaling API is used in combination with [Autho4API_10].

In the case the RESTful WebRTC Signaling API supports [Autho4API_10], the server:

- SHALL accept ‘acr:auth’ as a valid value for the resource URL variable {userId}
- SHALL conform to [REST_NetAPI_Common] section 5.8.1.1 regarding the processing of ‘acr:auth’.

Appendix H. SIP mapping (Informative)

This appendix describes how an implementation can map the REST requests to SIP. Apart from giving some guidance to server developers, this appendix also provides rationale for some of the API designs. As the flows below give implementation examples, the appendix is informative.

The flows in the sections below contain messages and participants that are defined in this specification, as well as those that are not defined in this specification, but that show the interworking with external components and systems. The legend below introduces the graphical styles used to distinguish between these categories.

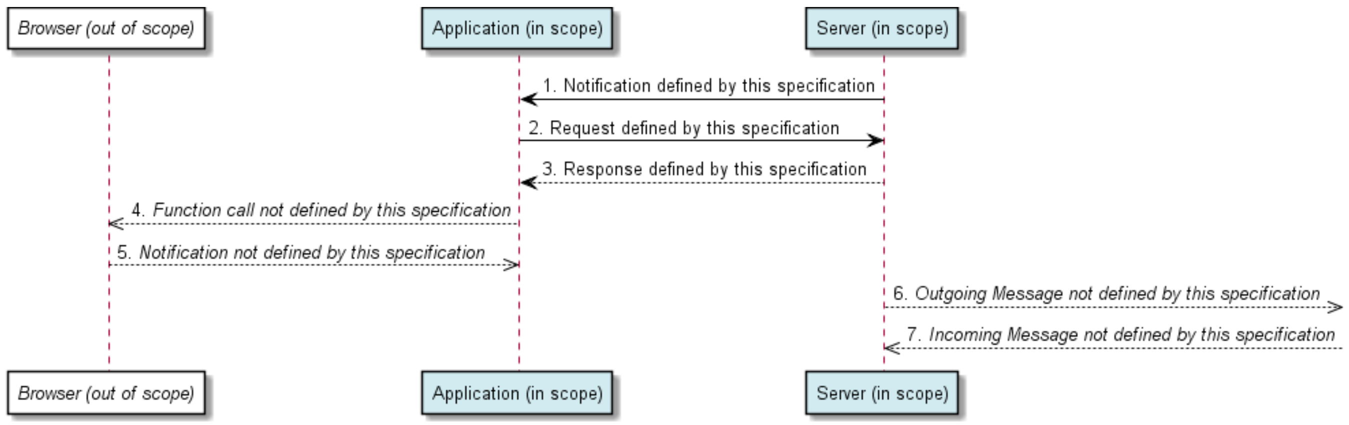
Browser (out of scope)
Application (in scope)
Server (in scope)
1. Notification defined by this specification
2. Request defined by this specification
3. Response defined by this specification
4. Function call not defined by this specification
5. Notification not defined by this specification
6. Outgoing Message not defined by this specification
7. Incoming Message not defined by this specification
Browser (out of scope)
Application (in scope)
Server (in scope)

Figure 16: Legend for the sequence diagrams

H.1 Session set-up with ICE from Originator's point of view

The flows in this section assume that the Originator needs to use ICE in order to set up the connectivity for the media streams. The flows further assume that the media streams are anchored at a media gateway, rather than going peer-to-peer.

As the ICE procedures consume some time and may even fail, it is important that the user is not alerted about an incoming call before the ICE procedures have finished. Different options to achieve this are elaborated in this section. Essentially, there are two basic mechanisms: either to delay the INVITE, or to instruct the Terminating Participant's application not to alert the user until in both cases ICE has finished at the Originator. Sections H.1.1 - H.1.3 provide realizations of the first mechanism whereas section H.1.4 provides a realization of the second.

The section assumes familiarity with the procedures defined in [RFC3261], [RFC3262], [RFC3312] and [RFC5245] and their use with the offer-answer model.

H.1.1 Call set-up with ICE: Delaying the INVITE in the Originator's server without provisional response from Terminating Participant

In this configuration, the Originator's server cannot assume that the Terminating Participant supports preconditions [RFC3312]. To avoid ghost rings [RFC5245], the server therefore synthesizes a provisional answer towards the Originator's application. Note that this approach works with the RESTful WebRTC Signaling API as this API supports provisional answers as defined in JSEP. It would be awkward to use this pattern when SIP is the communication protocol between server and application, as in SIP it is good practice to run offers and answers end to end.

Note that in the flow below, the Terminating Participant's terminal does not have to run the ICE procedures; therefore no provisional response is returned to the Originator's application, but the INVITE is responded to immediately with "Ringing" followed by "OK".

The flow can be mapped to the second alternative in section 5.3.3.

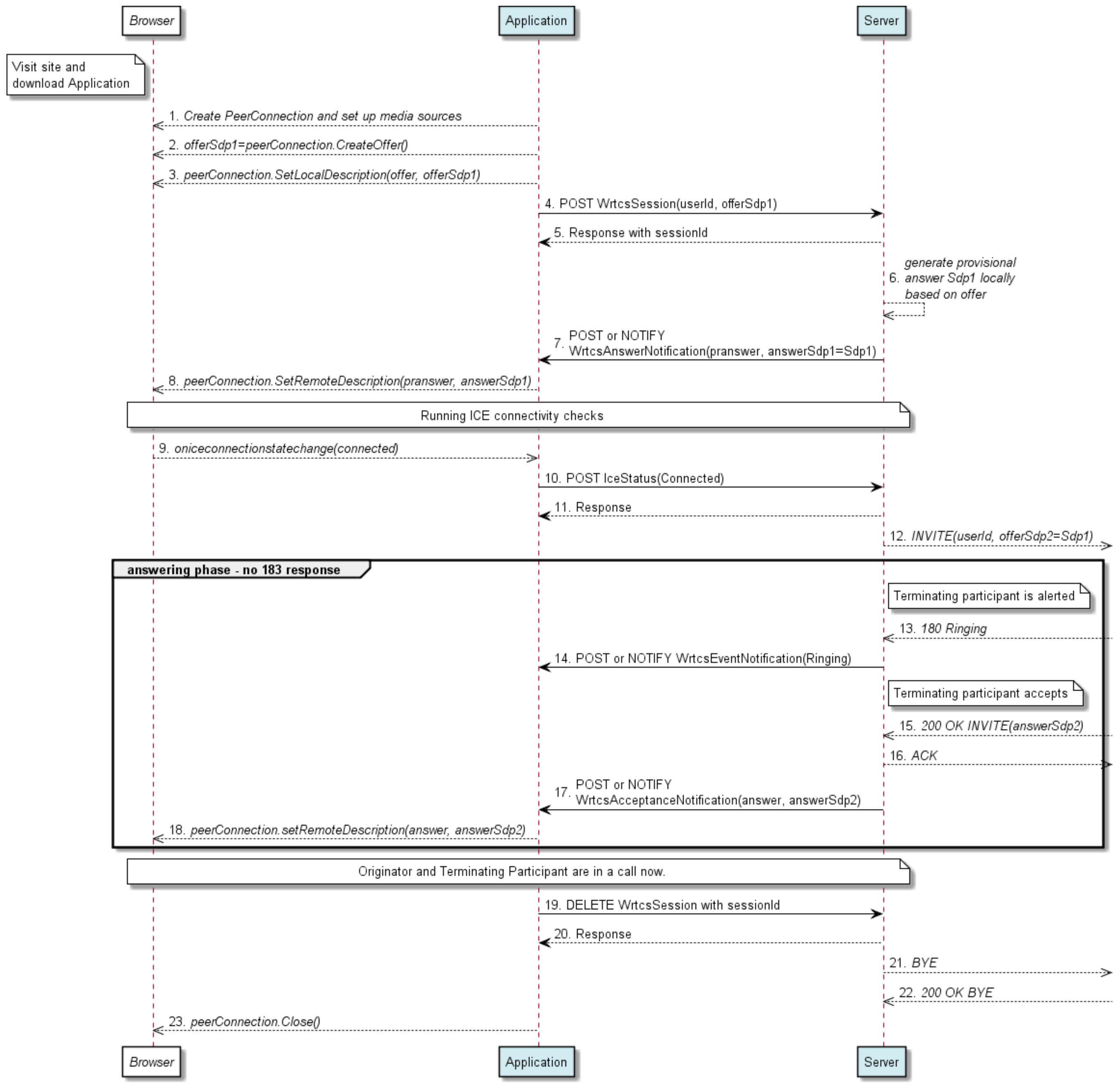
Browser
Application
Server
Visit site and download Application
1. Create PeerConnection and set up media sources
2. offerSdp1=peerConnection.CreateOffer()
3. peerConnection.SetLocalDescription(offer, offerSdp1)
4. POST WrtcsSession(userId, offerSdp1)
5. Response with sessionId
6. generate provisional answer Sdp1 locally based on offer
7. POST or NOTIFY WrtcsAnswerNotification(pranswer, answerSdp1=Sdp1)
8. peerConnection.SetRemoteDescription(pranswer, answerSdp1)
Running ICE connectivity checks
9. oniceconnectionstatechange(connected)
10. POST IceStatus(Connected)
11. Response
12. INVITE(userId, offerSdp2=Sdp1)
answering phase - no 183 response
Terminating participant is alerted
13. 180 Ringing
14. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
15. 200 OK INVITE(answerSdp2)
16. ACK
17. POST or NOTIFY WrtcsAcceptanceNotification(answer, answerSdp2)
18. peerConnection.setRemoteDescription(answer, answerSdp2)
Originator and Terminating Participant are in a call now.
19. DELETE WrtcsSession with sessionId
20. Response
21. BYE
22. 200 OK BYE
23. peerConnection.Close()

Figure 17: Call set-up with ICE: Delaying the INVITE in the server without provisional response from Terminating Participant

H.1.2 Call set-up with ICE: Delaying the INVITE in the Originator's server with provisional response from Terminating Participant, sent reliably

Similar to the section above, the Originator's server cannot assume that the Terminating Participant supports preconditions [RFC3312]. To avoid ghost rings [RFC5245], the server therefore synthesizes a provisional answer towards the Originator.

However, this flow shows an alternative for the steps in the answer phase: The Terminating Participant is assumed to send a provisional response reliably, e.g. to inform the Originator that it has to run ICE procedures locally and therefore has delayed the ringing.

As the answer is sent in a provisional response reliably, there is no answer in step 8.

The flow can be mapped to the first alternative in section 5.3.3.

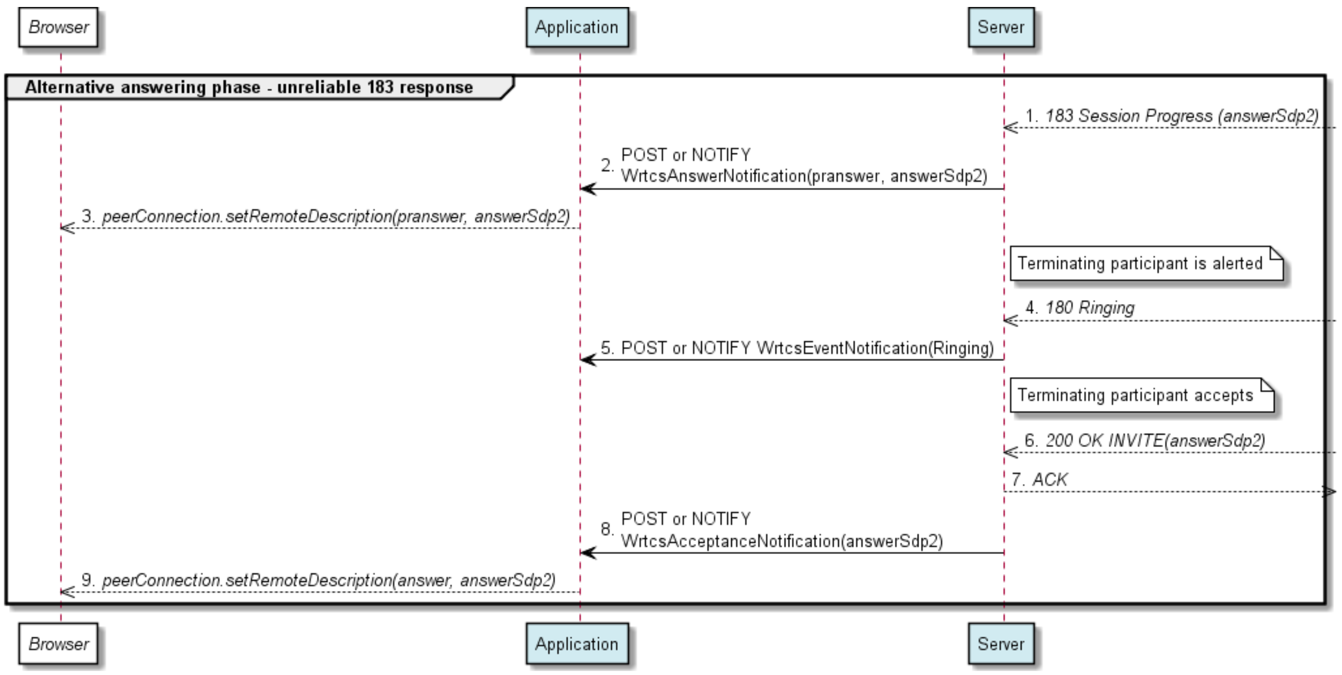
Browser
Application
Server
Alternative answering phase - unreliable 183 response
1. 183 Session Progress (answerSdp2)
2. POST or NOTIFY WrtcsAnswerNotification(pranswer, answerSdp2)
3. peerConnection.setRemoteDescription(pranswer, answerSdp2)
Terminating participant is alerted
4. 180 Ringing
5. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
6. 200 OK INVITE(answerSdp2)
7. ACK
8. POST or NOTIFY WrtcsAcceptanceNotification(answerSdp2)
9. peerConnection.setRemoteDescription(answer, answerSdp2)
Browser
Application
Server

Figure 18: Call set-up with ICE: Delaying the INVITE in the server with provisional response from Terminating Participant, sent reliably

H.1.3 Call set-up with ICE: Delaying the INVITE in the Originator's server with provisional response from Terminating Participant, sent non-reliably

This section is similar to the section above, with the difference that the provisional response is sent unreliably. Therefore, it must be repeated in step 6.

The flow below shows again an alternative for the steps in the answering phase in section H.1.1.

The flow has no direct correspondence in section 5.3.3, but could be realized as a synthesis of alternatives 1 and 2.

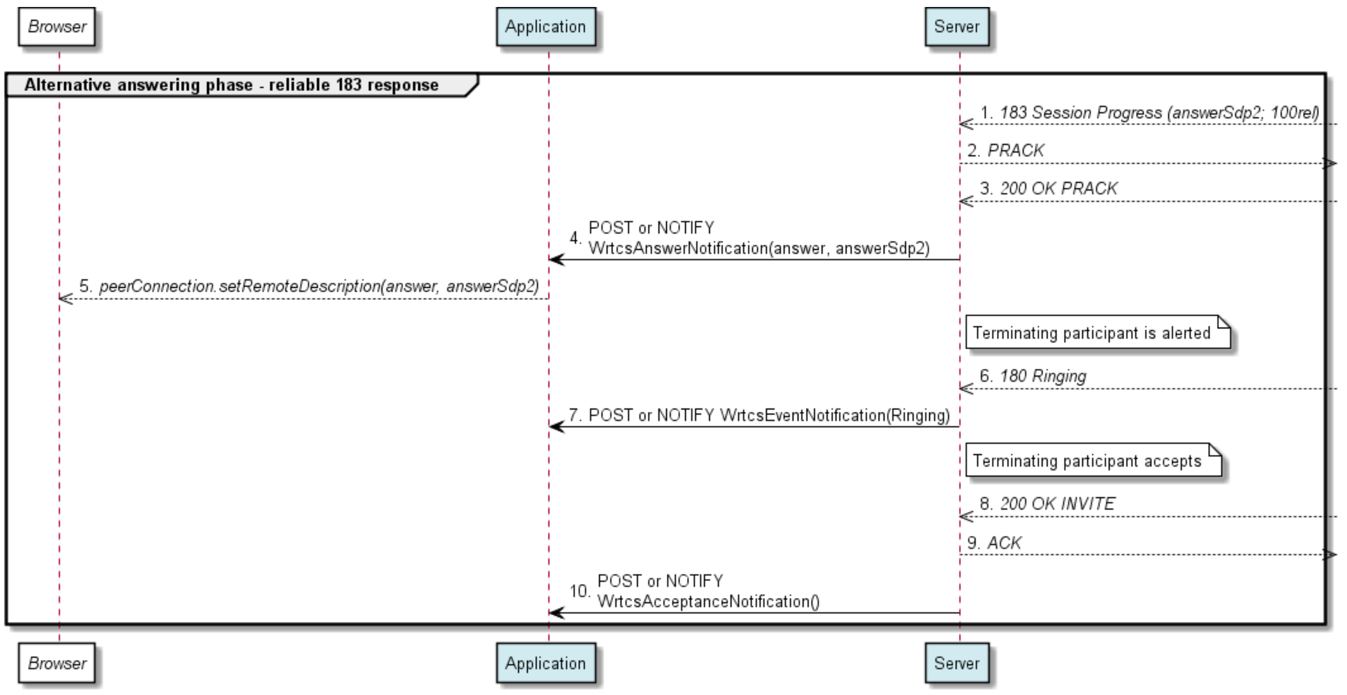
Browser
Application
Server
Alternative answering phase - reliable 183 response
1. 183 Session Progress (answerSdp2; 100rel)
2. PRACK
3. 200 OK PRACK
4. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp2)
5. peerConnection.setRemoteDescription(answer, answerSdp2)
Terminating participant is alerted
6. 180 Ringing
7. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
8. 200 OK INVITE
9. ACK
10. POST or NOTIFY WrtcsAcceptanceNotification()

Figure 19: Call set-up with ICE: Delaying the INVITE in the server with provisional response from Terminating Participant, sent non-reliably

H.1.4 Call set-up with ICE: Originator is using SIP preconditions

The approach in the sections above has a performance penalty when the application of the Terminating Participant also needs to run ICE as part of the call set-up. The reason is that the INVITE will only arrive at the Terminating Participant once the Originator has finished its ICE procedures, thereby delaying the start of the ICE procedures at the Terminating Participant until that time. Note that with the signaling alternative provided here, ICE could run in parallel at both ends.

The idea is to instruct the Terminating Participant to delay alerting the user until certain preconditions (in this case the availability of connectivity) are met at the Originator's side. These preconditions [RFC3312] are declared in the INVITE, and then updated using an offer-answer pair managed by the server.

In this flow, the server is forwarding the answer it has received from the remote peer as a provisional answer to the application (step 10), which allows the server to send another (final) answer in a later step (step 17). Also, note that a provisional answer inhibits the application from sending another offer, which is sometimes of advantage if the server knows that the exchange with the far end has not terminated.

Without the mechanism of provisional answers, the server would have to convert the answer it receives in step 16 into an offer towards the application. This would introduce difficulties in the flow, because the application would be expected to generate an answer to this offer.

Because of such complications, conversion between offer and answer at the server should be avoided as much as possible as they introduce additional states in the server.

From the WebRTC Signaling API point of view, this flow corresponds to alternative 1 in section 5.3.3.

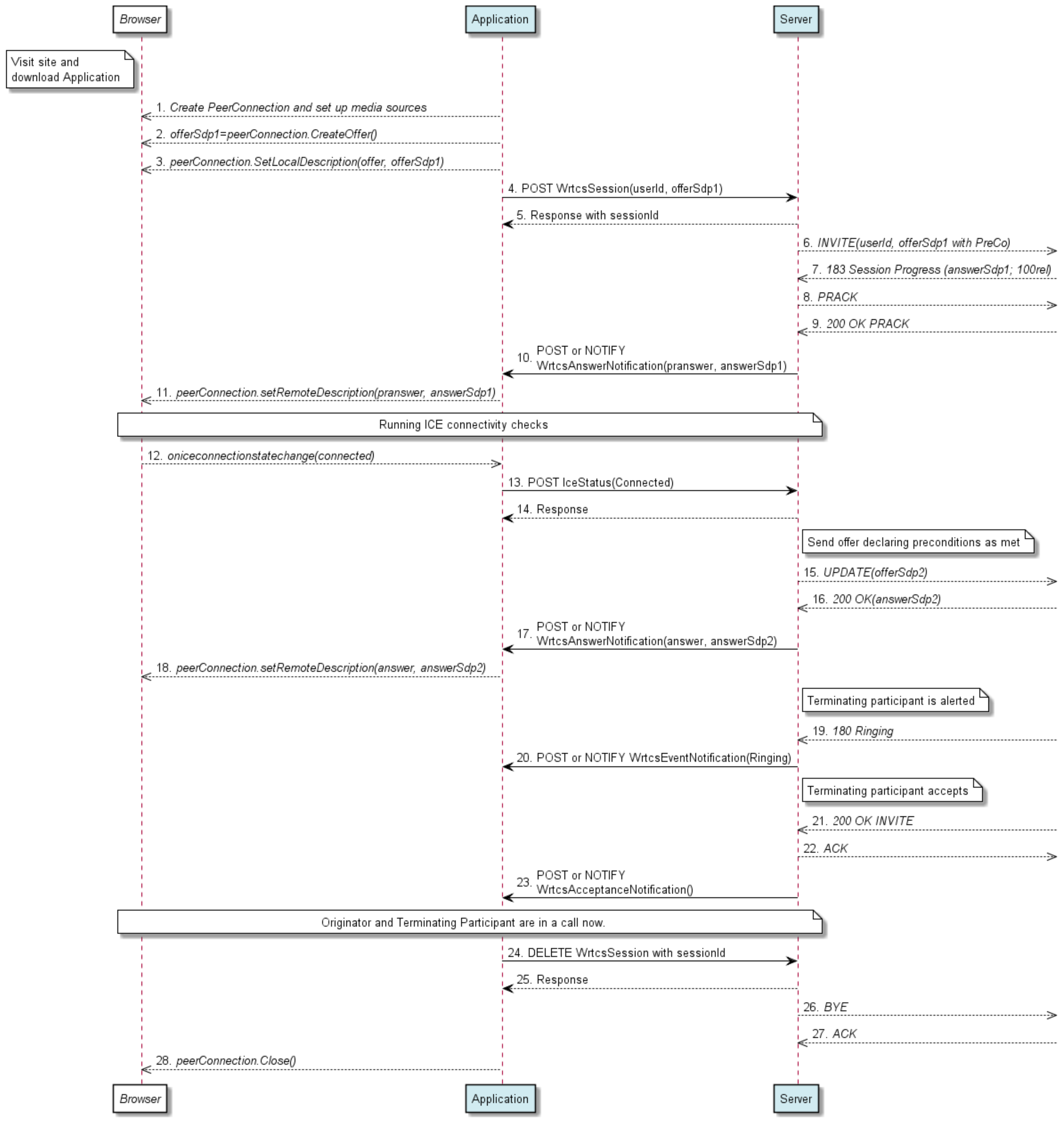
Browser
Application
Server
Visit site and download Application
1. Create PeerConnection and set up media sources
2. offerSdp1=peerConnection.CreateOffer()
3. peerConnection.SetLocalDescription(offer, offerSdp1)
4. POST WrtcsSession(userId, offerSdp1)
5. Response with sessionId
6. INVITE(userId, offerSdp1 with PreCo)
7. 183 Session Progress (answerSdp1; 100rel)
8. PRACK
9. 200 OK PRACK
10. POST or NOTIFY WrtcsAnswerNotification(pranswer, answerSdp1)
11. peerConnection.setRemoteDescription(pranswer, answerSdp1)
Running ICE connectivity checks
12. oniceconnectionstatechange(connected)
13. POST IceStatus(Connected)
14. Response
Send offer declaring preconditions as met
15. UPDATE(offerSdp2)
16. 200 OK(answerSdp2)
17. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp2)
18. peerConnection.setRemoteDescription(answer, answerSdp2)
Terminating participant is alerted
19. 180 Ringing
20. POST or NOTIFY WrtcsEventNotification(Ringing)
Terminating participant accepts
21. 200 OK INVITE
22. ACK
23. POST or NOTIFY WrtcsAcceptanceNotification()
Originator and Terminating Participant are in a call now.
24. DELETE WrtcsSession with sessionId
25. Response
26. BYE
27. ACK
28. peerConnection.Close()
Browser
Application
Server

Figure 20: Call set-up with ICE: Using SIP preconditions

H.2 Session set-up with ICE from Terminating Participant's point of view

When the Terminating Participant's application receives a session Invitation Notification, it will need to successfully run the ICE procedures before it can alert the user. It therefore uses a provisional response (183 Session Progress) to indicate to the Originator that the session setup goes forward silently, and to provide the answer which the Originator needs to possibly set up his own media channels.

H.2.1 Session set-up with ICE from Terminating Participant's point of view without SIP Preconditions

The following flow shows how the Terminating Participant's application responds to a session invitation in case the Originator has not signaled any preconditions.

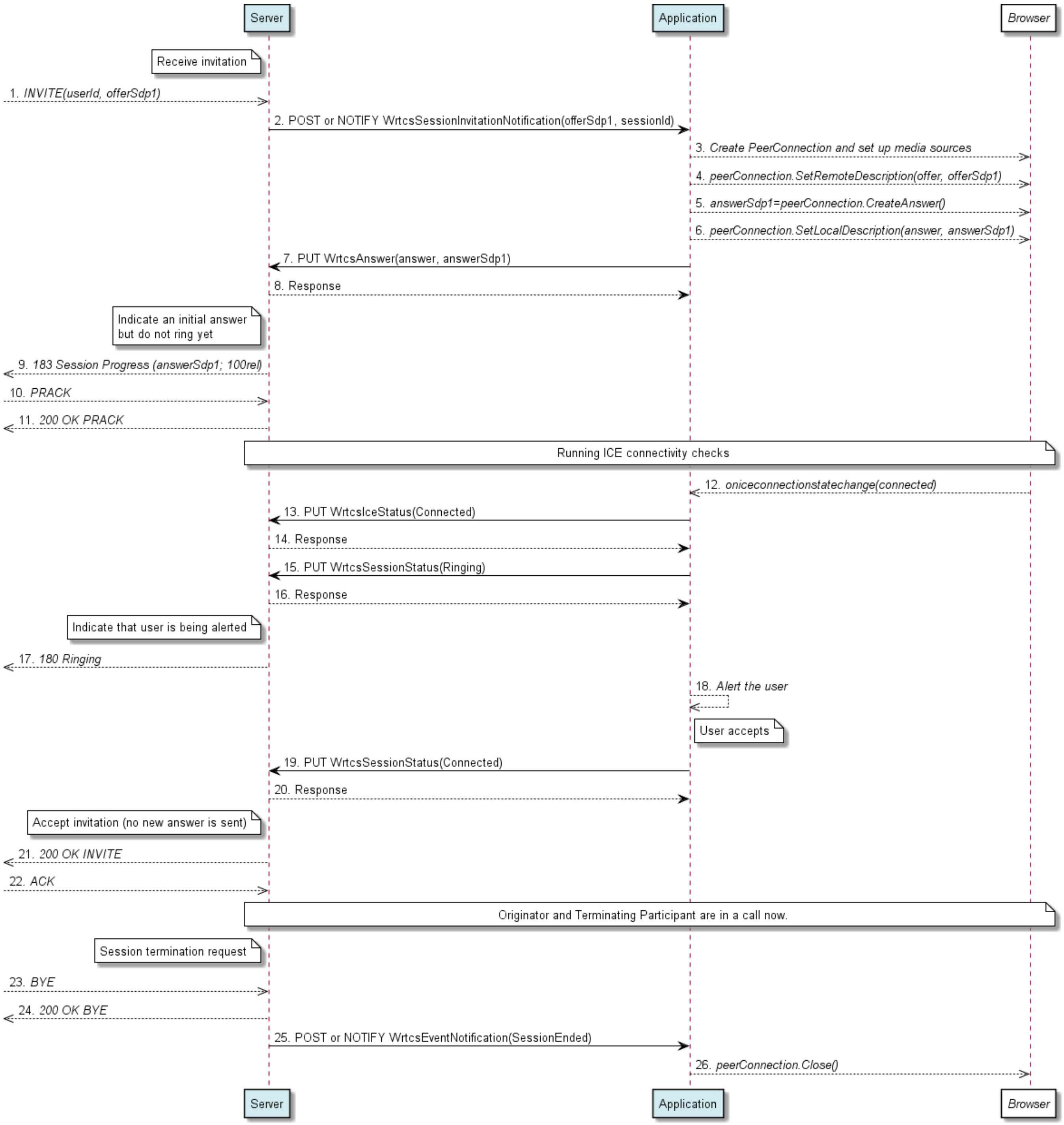
Server
Application
Browser
Receive invitation
1. INVITE(userId, offerSdp1)
2. POST or NOTIFY WrtcsSessionInvitationNotification(offerSdp1, sessionId)
3. Create PeerConnection and set up media sources
4. peerConnection.SetRemoteDescription(offer, offerSdp1)
5. answerSdp1=peerConnection.CreateAnswer()
6. peerConnection.SetLocalDescription(answer, answerSdp1)
7. PUT WrtcsAnswer(answer, answerSdp1)
8. Response
Indicate an initial answer but do not ring yet
9. 183 Session Progress (answerSdp1; 100rel)
10. PRACK
11. 200 OK PRACK
Running ICE connectivity checks
12. oniceconnectionstatechange(connected)
13. PUT WrtcsIceStatus(Connected)
14. Response
15. PUT WrtcsSessionStatus(Ringing)
16. Response
Indicate that user is being alerted
17. 180 Ringing
18. Alert the user
User accepts
19. PUT WrtcsSessionStatus(Connected)
20. Response
Accept invitation (no new answer is sent)
21. 200 OK INVITE
22. ACK
Originator and Terminating Participant are in a call now.
Session termination request
23. BYE
24. 200 OK BYE
25. POST or NOTIFY WrtcsEventNotification(SessionEnded)
26. peerConnection.Close()
Server
Application
Browser

Figure 21: Session set-up with ICE from Terminating Participant's point of view without SIP Preconditions

H.2.2 Session set-up with ICE from Terminating Participant’s point of view using SIP Preconditions

The following flow shows how the Terminating Participant’s application responds to a session invitation in which the Originator has signaled preconditions. With these preconditions, the Originator requests the Terminating Participant’s application to delay the alerting of the user until the Originator reports that the preconditions have been met (e.g. ICE checks have been succeeded, or QoS reservations have been granted).

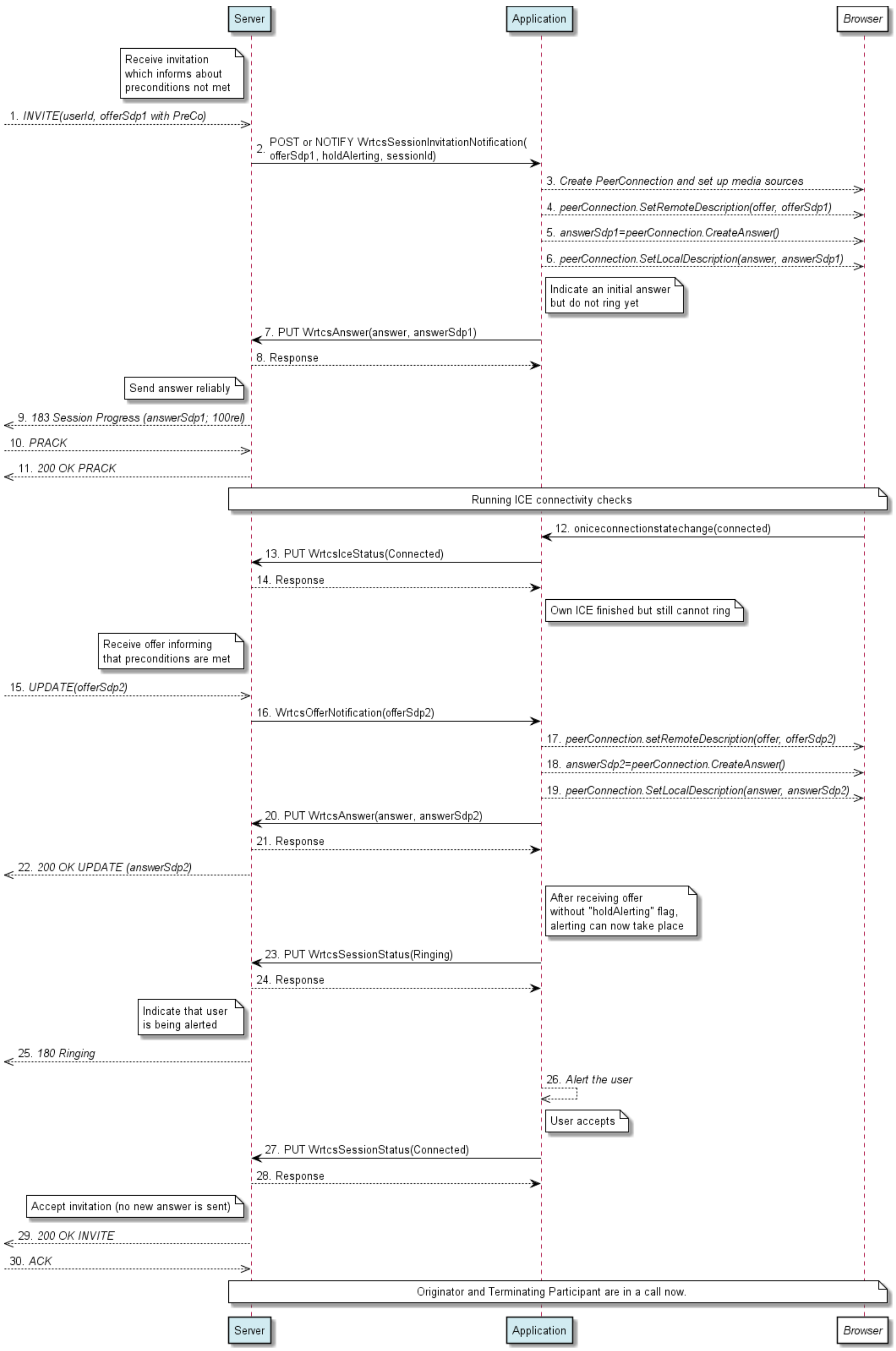
Server
Application
Browser
Receive invitation which informs about preconditions not met
1. INVITE(userId, offerSdp1 with PreCo)
2. POST or NOTIFY WrtcsSessionInvitationNotification(offerSdp1, holdAlerting, sessionId)
3. Create PeerConnection and set up media sources
4. peerConnection.SetRemoteDescription(offer, offerSdp1)
5. answerSdp1=peerConnection.CreateAnswer()
6. peerConnection.SetLocalDescription(answer, answerSdp1)
Indicate an initial answer but do not ring yet
7. PUT WrtcsAnswer(answer, answerSdp1)
8. Response
Send answer reliably
9. 183 Session Progress (answerSdp1; 100rel)
10. PRACK
11. 200 OK PRACK
Running ICE connectivity checks
12. oniceconnectionstatechange(connected)
13. PUT WrtcsIceStatus(Connected)
14. Response
Own ICE finished but still cannot ring
Receive offer informing that preconditions are met
15. UPDATE(offerSdp2)
16. WrtcsOfferNotification(offerSdp2)
17. peerConnection.setRemoteDescription(offer, offerSdp2)
18. answerSdp2=peerConnection.CreateAnswer()
19. peerConnection.SetLocalDescription(answer, answerSdp2)
20. PUT WrtcsAnswer(answer, answerSdp2)
21. Response
22. 200 OK UPDATE (answerSdp2)
After receiving offer without "holdAlerting" flag, alerting can now take place
23. PUT WrtcsSessionStatus(Ringing)
24. Response
Indicate that user is being alerted
25. 180 Ringing
26. Alert the user
User accepts
27. PUT WrtcsSessionStatus(Connected)
28. Response
Accept invitation (no new answer is sent)
29. 200 OK INVITE
30. ACK
Originator and Terminating Participant are in a call now.
Server
Application
Browser

Figure 22 Session set-up with ICE from Terminating Participant's point of view using SIP Preconditions

H.3 Handling of offerless invitations

When the Terminating Participant's application receives a session Invitation Notification without an offer, it needs to respond with an offer which can then be used by the network to invite a second Terminating Participant and connect both of them. Such scenario is called third-party call control (3PCC).

The figures below depict possible mappings of the flow in section 5.3.6 to SIP.

H.3.1 Handling of offerless invitations if reliable provisional responses are supported

This subsection illustrates responding to an offerless invitation if the server supports reliable provisional responses [RFC3262]. It is assumed that this is the case in most deployments.

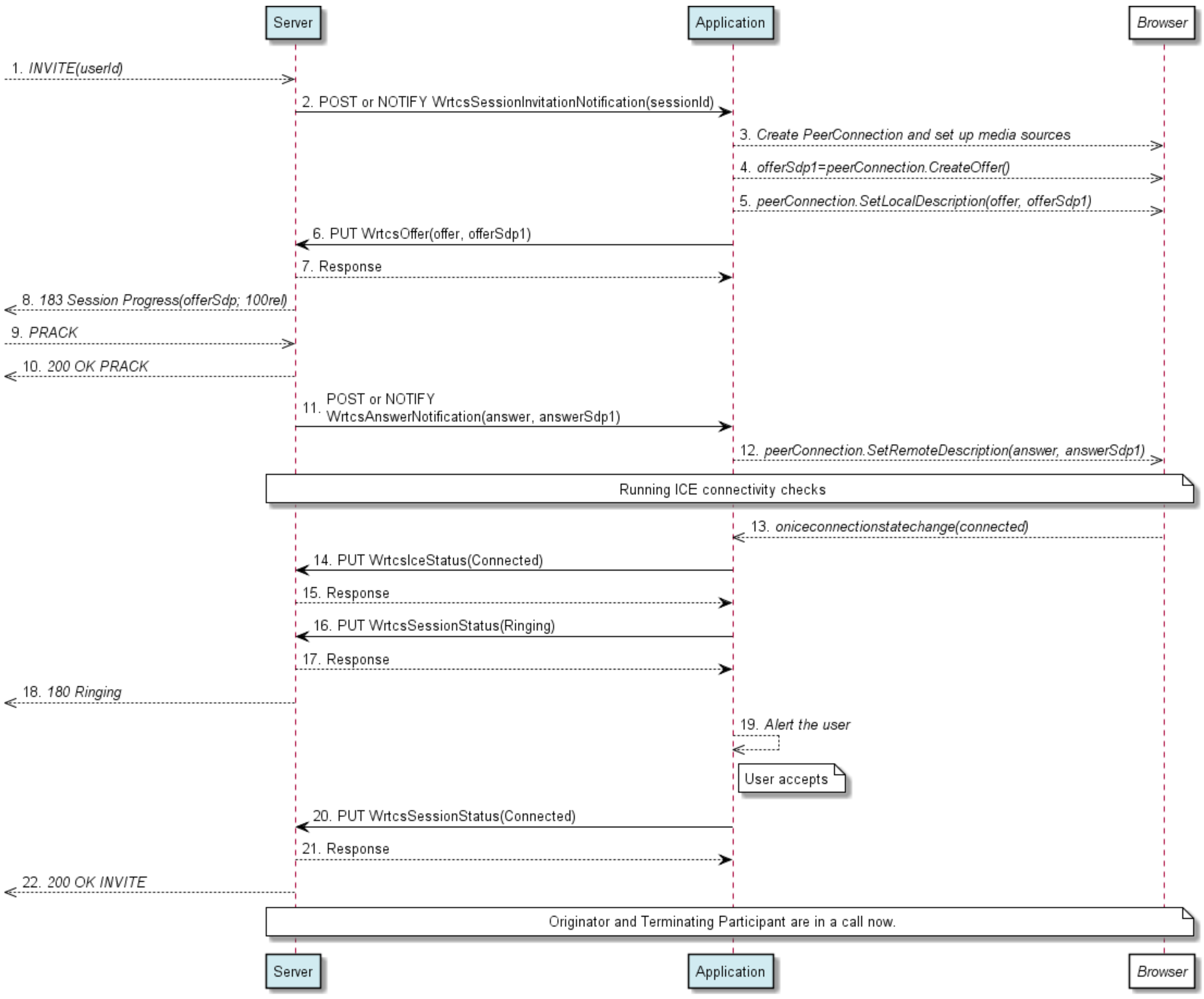
Server
Application
Browser
1. INVITE(userId)
2. POST or NOTIFY WrtcsSessionInvitationNotification(sessionId)
3. Create PeerConnection and set up media sources
4. offerSdp1=peerConnection.CreateOffer()
5. peerConnection.SetLocalDescription(offer, offerSdp1)
6. PUT WrtcsOffer(offer, offerSdp1)
7. Response
8. 183 Session Progress(offerSdp; 100rel)
9. PRACK
10. 200 OK PRACK
11. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp1)
12. peerConnection.SetRemoteDescription(answer, answerSdp1)
Running ICE connectivity checks
13. oniceconnectionstatechange(connected)
14. PUT WrtcsIceStatus(Connected)
15. Response
16. PUT WrtcsSessionStatus(Ringing)
17. Response
18. 180 Ringing
19. Alert the user
User accepts
20. PUT WrtcsSessionStatus(Connected)
21. Response
22. 200 OK INVITE
Originator and Terminating Participant are in a call now.
Server
Application
Browser

Figure 23: Handling of offerless invitations if reliable provisional responses are supported

H.3.2 Handling of offerless invitations if reliable provisional responses are not supported

This scenario is the same as in previous appendix. However reliable provisional responses [RFC3262] are not supported by the server and the SDP negotiation has to be done by 200OK and ACK.

This means, the application cannot wait for a media path o be established (aka ICE completion), but must go ahead immediately with alerting the user and connecting the session on user acceptance, as the offer will only be sent once the session is connected.

This solution can serve as a fallback if the server does not support reliable provisional responses [RFC3262]. Compared to the option in section H.3.1, this option has the disadvantage of longer call setup delays and the possibility of ghost rings if ICE checks fail.

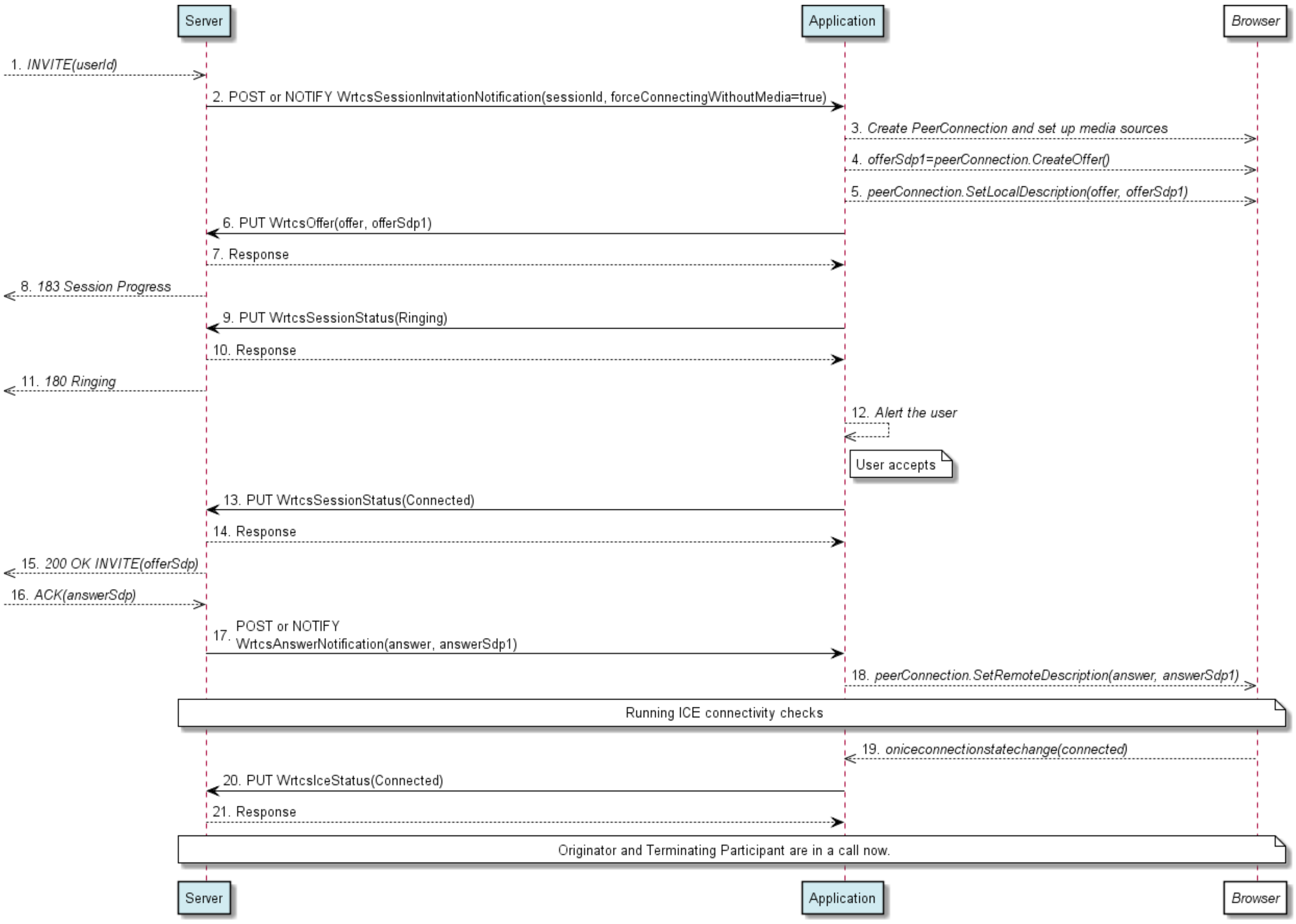
Server
Application
Browser
1. INVITE(userId)
2. POST or NOTIFY WrtcsSessionInvitationNotification(sessionId, forceConnectingWithoutMedia=true)
3. Create PeerConnection and set up media sources
4. offerSdp1=peerConnection.CreateOffer()
5. peerConnection.SetLocalDescription(offer, offerSdp1)
6. PUT WrtcsOffer(offer, offerSdp1)
7. Response
8. 183 Session Progress
9. PUT WrtcsSessionStatus(Ringing)
10. Response
11. 180 Ringing
12. Alert the user
User accepts
13. PUT WrtcsSessionStatus(Connected)
14. Response
15. 200 OK INVITE(offerSdp)
16. ACK(answerSdp)
17. POST or NOTIFY WrtcsAnswerNotification(answer, answerSdp1)
18. peerConnection.SetRemoteDescription(answer, answerSdp1)
Running ICE connectivity checks
19. oniceconnectionstatechange(connected)
20. PUT WrtcsIceStatus(Connected)
21. Response
Originator and Terminating Participant are in a call now.
Server
Application
Browser

Figure 24: Handling of offerless invitations if reliable provisional responses are not supported

H.4 Handling of session updates

Session updates follow the offer-answer model. For elaborations on using the offer-answer with SIP, see [RFC6337].

H.4.1 Handling of session updates by the Update Originator

The following flow shows how the Update Originator's application handles session updates, related to section5.3.9.

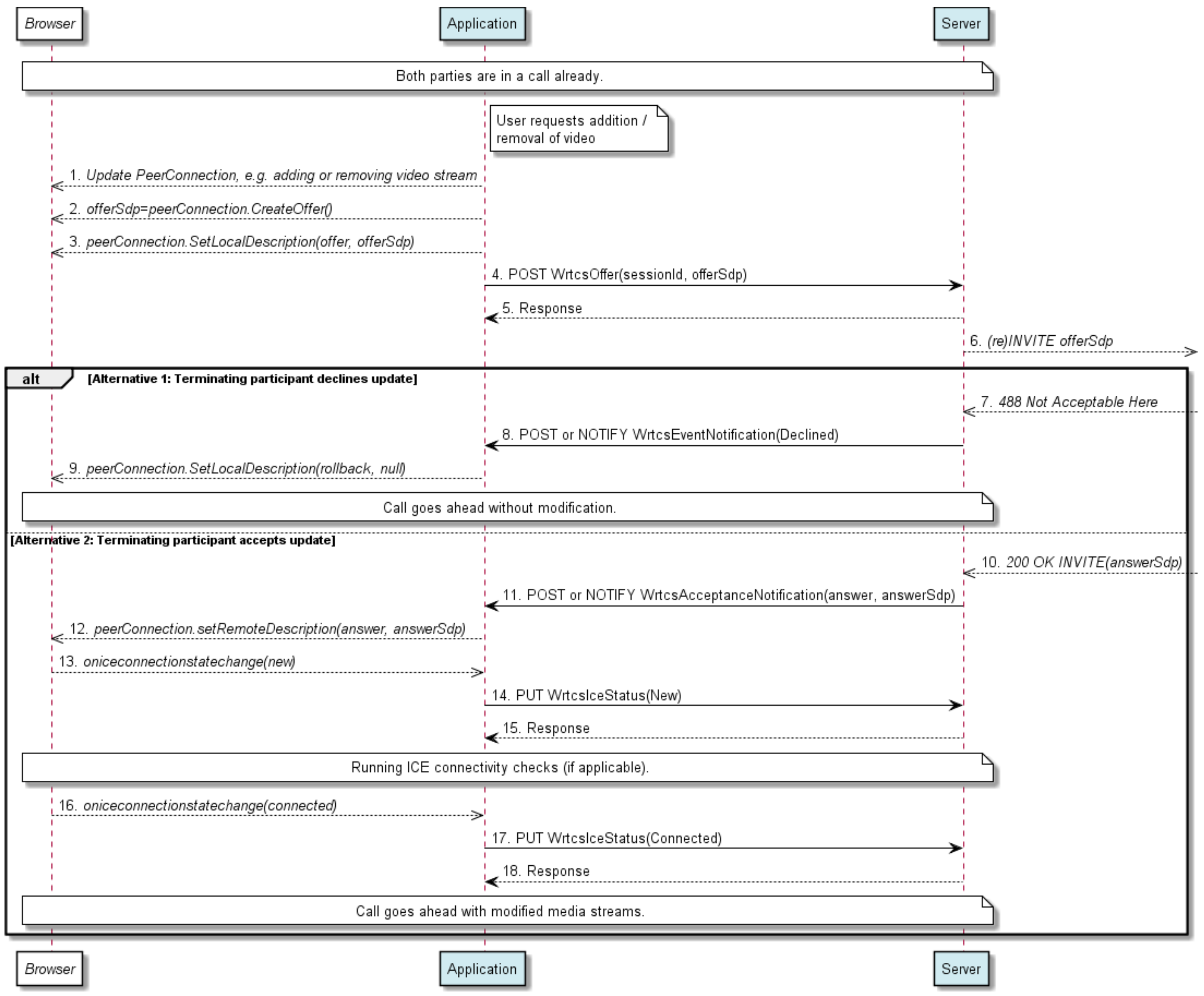
Browser
Application
Server
Both parties are in a call already.
User requests addition / removal of video
1. Update PeerConnection, e.g. adding or removing video stream
2. offerSdp=peerConnection.CreateOffer()
3. peerConnection.SetLocalDescription(offer, offerSdp)
4. POST WrtcsOffer(sessionId, offerSdp)
5. Response
6. (re)INVITE offerSdp
alt [Alternative 1: Terminating participant declines update]
7. 488 Not Acceptable Here
8. POST or NOTIFY WrtcsEventNotification(Declined)
9. peerConnection.SetLocalDescription(rollback, null)
Call goes ahead without modification.
[Alternative 2: Terminating participant accepts update]
10. 200 OK INVITE(answerSdp)
11. POST or NOTIFY WrtcsAcceptanceNotification(answer, answerSdp)
12. peerConnection.setRemoteDescription(answer, answerSdp)
13. oniceconnectionstatechange(new)
14. PUT WrtcsIceStatus(New)
15. Response
Running ICE connectivity checks (if applicable).
16. oniceconnectionstatechange(connected)
17. PUT WrtcsIceStatus(Connected)
18. Response
Call goes ahead with modified media streams.

Figure 25: Handling of session updates by the Update Originator

H.4.2 Handling of session updates by the Update Recipient

The following flow shows how the Update Recipient’s application handles session updates, related to section 5.3.10.

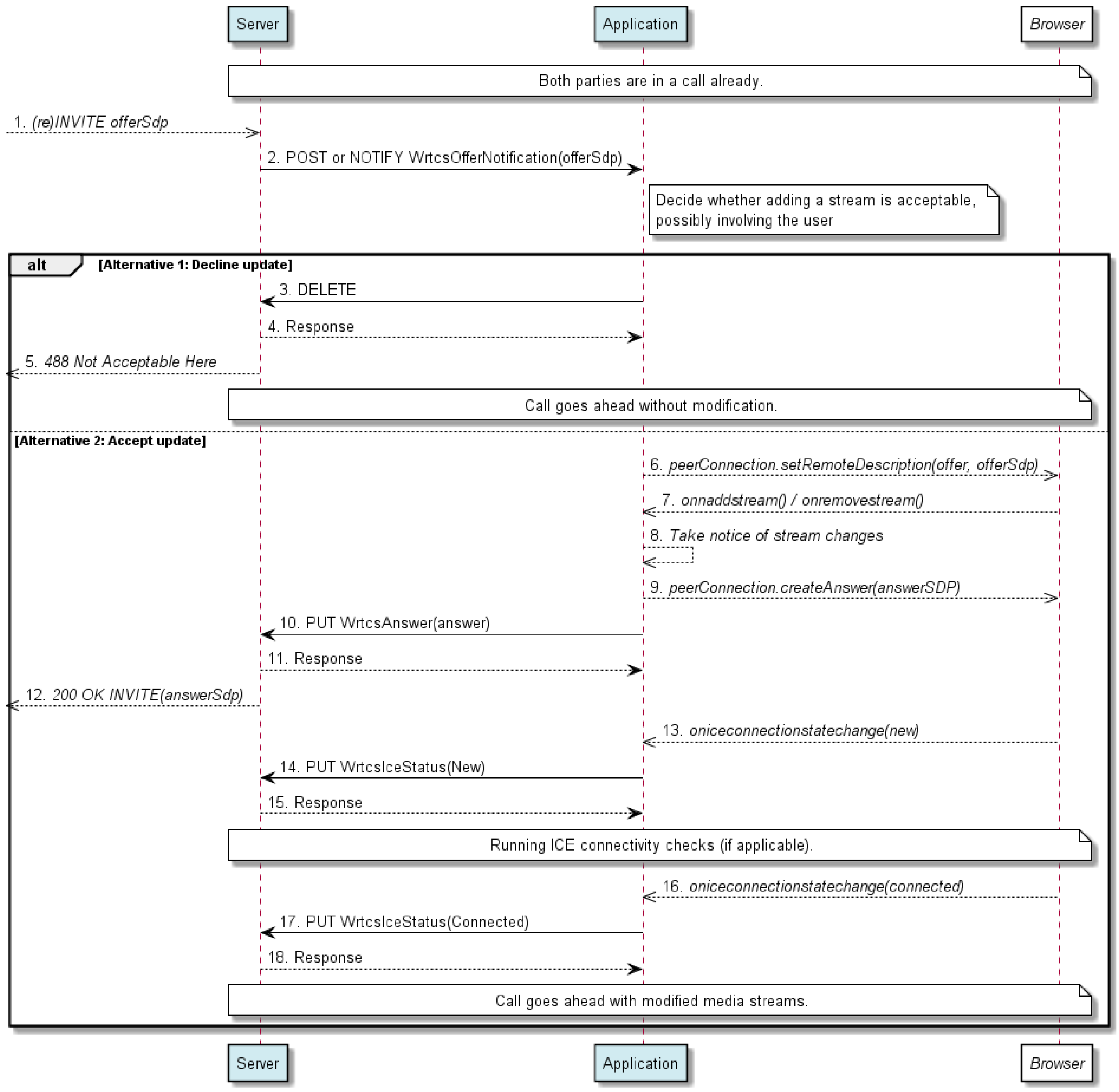
Server
Application
Browser
Both parties are in a call already.
1. (re)INVITE offerSdp
2. POST or NOTIFY WrtcsOfferNotification(offerSdp)
Decide whether adding a stream is acceptable, possibly involving the user
alt
[Alternative 1: Decline update]
3. DELETE
4. Response
5. 488 Not Acceptable Here
Call goes ahead without modification.
[Alternative 2: Accept update]
6. peerConnection.setRemoteDescription(offer, offerSdp)
7. onnaddstream() / onremovestream()
8. Take notice of stream changes
9. peerConnection.createAnswer(answerSDP)
10. PUT WrtcsAnswer(answer)
11. Response
12. 200 OK INVITE(answerSdp)
13. oniceconnectionstatechange(new)
14. PUT WrtcsIceStatus(New)
15. Response
Running ICE connectivity checks (if applicable).
16. oniceconnectionstatechange(connected)
17. PUT WrtcsIceStatus(Connected)
18. Response
Call goes ahead with modified media streams.
Server
Application
Browser

Figure 26: Handling of session updates by the Update Recipient