Client-Portal Protocol Specification

Version: 1.0 draft

June 2017

|  |  |  |
| --- | --- | --- |
| date | contributor | revision |
| 12-06-2017 | Xiande | draft |
| 28-06-2017 | Xiande | 1.0.0 reviewed |
|  |  |  |

# Overview

This documentation covers all signaling messages between Client and Portal component, including signaling messages transported through Socket.io connections and RESTful interfaces. All the signaling interaction at the Client side should be encapsulated in the Client SDK, and Client SDK should provide an API suite for client side application integration.

# Definitions

**Portal:** The MCU component listening at the HTTP server or Socket.io server port, accepting Signaling connections initiated by Clients, receive/send signaling messages from/to Clients.

**Client:** The program running on end-user’s device being built on Client SDK which interacts with MCU Server by messages defined in this documentation. Client can be browsers including Chrome, Firefox and Edge, Android, iOS and Windows Apps.

**Signaling Connection:** The signaling channel between Client and Portal established by Client to send, receive and fetch signaling messages.

**Room:** The logical entity of a conference in which all conference members (participants) exchange their video/audio streams, and mixed streams are generated. Every room must be assigned with a unique identification (**RoomID**) in the MCU scope.

**User:** The service account defined in the third-party integration application. The basic information such as the user identification and user name should be passed into MCU system for presence purpose, and the role of the user must be specified when asking for a token.

**Participant:** The logical entity of a user when participating in a conference with a valid token. Every participant must be assigned with a unique identification (**ParticipantID**) in the room scope, and must be assigned a set of operation permissions according to its role.

**Token:** The only credential when a particular participant request to join a particular room. It must contain a unique identification, the hostname of the target portal it can be used to connect, and the checksum to verify its validity.

**Stream:** The object represents an audio and/or video media source which can be consumed (say, be subscribed, recorded, pushed to a live stream server) within a room. A stream can be either a participant published one which is called a **Forward Stream**, or a room generated one which is called a **Mixed Stream**. A stream object must be assigned with a unique identification (**StreamID**) in the room scope, Participants can select a particular stream to subscribe according to StreamID. A stream must also contain the audio information of the codec/sample rate/channel number, and video information of codec (for Forward Stream) and codec/resolution/frame rate/key frame interval/quality level (for Mixed Stream). Participants can determine whether the stream fulfill their expectation based on such information.

**Subscription:** The activity a participant consuming a stream, such as subscribing a stream via a WebRTC PeerConnection, recording a stream to a MCU storage, pushing a stream to a live stream server. A unique identification (**SubscriptionID**) in the room scope must be assigned to the subscription once its request is accepted by MCU. Participants will use the identification to update/cancel/stop a specific subscription.

**Session:** The entity in which a real-time audio/video communication takes place. Typically, participants establish WebRTC sessions between a WebRTC client and MCU (accurately, the webrtc-agent) to publish or subscribe streams. Since a stream ID will be assigned when publishing a stream into a room and a subscription ID will be assigned when subscribing a stream (or a pair of stream if audio and video come from different source), the stream ID and subscription ID in these two cases are re-used to identify the corresponding session, and MCU must guarantee that the stream IDs and subscription IDs will not conflict within a room.

# Socket.io Signaling

## Format

### Client -> Portal

Given that Client has connected Portal successfully and the socket object is ready to send/receive data to/from Portal, Client must send all signaling message to Portal by calling the client socket object’s ***emit*** method in the following format (javascript code for example, same for rest of this documentation):

clientSocket**.**emit**(**"signaling"**,**

RequestName**,**

RequestData**,**

**function(**ResponseStatus**,** ResponseData**)** **{**

//Handle the status and response here.

**}**

**);**

Where

1. **RequestName** must be with type of string, and with value defined in this specification;
2. **RequestData** must be with type of javascript object(or equivalent in other programming language), and with format defined in this specification;
3. **ResponseStatus** must be with value either “success” or “error”;
4. **ResponseData** must be with type of javascript object or undefined if **ResponseStatus** equals to “success”, and with type of ErrorDescription object defined as following if **ResponseStatus** equals to “error”.

object**(**ErrorDesciption**)::**

**{**

code**:** number**(**ErrorCode**),**

description**:** string**(**Reason**)**

**}**

### Portal -> Client

Given that Portal has accepted Clients connecting and the socket object is ready to send/receive data to/from Client, Portal must send all signaling messages to Client by calling the server socket object’s ***emit*** method in the following format:

serverSocket**.**emit**(**"signaling"**,**

NotificationName**,**

NotificationData

**);**

Where

1. **NotificationName** must be with type of string, and with value defined in this specification;
2. **NotificationData** must be with type of object, and with format defined in this specification.

## Connection Maintenance

### Client Connects

Portal should be able to listen at both a secure and an insecure socket.io server port to accept Clients’ connecting requests. If the secure socket.io server is enabled, the SSL certificate and private key store path must be correctly specified by configuration item portal.keystorePath in portal.toml.

Socket.io server will pop a ‘connection’ event indicating a new client’s successfully connecting.

### Client Keeps Alive

Socket.io server has its own underlying connection keep-alive mechanism, so no application level heart-beat is needed. But Client needs to refresh the reconnection ticket periodically before the current one expires if it wants to reconnect to Portal in case the network break or switch happens. The refreshing request must keep to the following format:

**RequestName**: “refresh”

**RequestData:** The RefreshTicketInfo object with following definition:

object**(**RefreshTicketInfo**)::**

**{**

id**:** string**(**ParticipantId**)** /\*The participant id returned at successful login, see [3.3.1](#_Participant_Joins_a)\*/

**}**

**ResponseData:** A refreshed ReconnectionTicket object if ResponseStatus is “success”.

### Client Disconnects

The connected socket.io object at server side will be notified with a ‘disconnect’ event. The waiting for reconnecting timer (**Timer100**) will be started at receiving the ‘disconnect’ event if the following conditions are fulfilled:

1. The connection is initiated from a mobile client;
2. No participant leaving signaling has been received.

If Client has not reconnected successfully before **Timer100** expires, Portal will not wait for and accept the Client’s reconnecting any longer. Otherwise, if Client reconnects successfully before **Timer100** expires, Portal will resume all the signaling activities with Client and kill **Timer100**.

### Client Reconnects

**RequestName**: “relogin”

**RequestData:** The ReconnectionInfo object with following definition:

object**(**ReconnectionInfo**)::**

**{**

id**:** string**(**ParticipantId**),** /\*The participant id returned at successful login, see [3.3.1](#_Participant_Joins_a)\*/

ticket**:** object**(**ReconnectionTicket**)** /\*The reconnectionTicket returned at successful login, see [3.3.1](#_3.3.1_User_Joins)\*/

**}**

**ResponseData:** A refreshed ReconnectionTicket object if ResponseStatus is “success”.

## Conferencing

### Participant Joins a Room

**RequestName**: “login”

**RequestData:** The LoginInfo object with following definition:

object**(**LoginInfo**)::**

**{**

token**:** string**(**Base64EncodedToken**),**

userAgent**:** object**(**ClientInfo**),**

user**:** object**(**UserInfo**),**

protocol**:** string**(**ProtocolVersion**)** //e.g. “1.0.0”

**}**

object**(**ClientInfo**)::**

**{**

sdk**:** **{**

type**:** string**(**SDKName**),**

version**:** string**(**SDKVersion**)**

**},**

runtime**:** **{**

name**:** string**(**RuntimeName**),**

version**:** string**(**RuntimeVersion**)**

**},**

os**:** **{**

name**:** string**(**OSName**),**

version**:** string**(**OSVersion**)**

**}**

**}**

object**(**UserInfo**)::**

**{**

id**:** string**(**UserId**),**

name**:** string**(**UserName**)**

**}**

**ResponseData:** The LoginResult object with following definition if ResponseStatus is “success”:

object**(**LoginResult**)::**

**{**

id**:** string**(**ParticipantId**),**

role**:** string**(**ParticipantRole**),**

permission**:** object**(**Permission**),**

room**:** object**(**RoomInfo**),**

reconnectionTicket**:** undefined/\*when reconnection is not promised\*/

**|** object**(**ReconnectionTicket**)**/\*when reconnection is promised\*/

**}**

object**(**Permission**)::**

**{**

publish**:** **[**"webrtc" **|** "streaming"**]** **|** **false,**

subscribe**:** **[**"webrtc" **|** "recording" **|** "streaming"**]** **|** **false,**

**text:** "to-all" **|** "to-peer" **|** **false,**

manage**:** **true** **|** **false**

**}**

object**(**RoomInfo**)::**

**{**

id**:** string**(**RoomId**),**

views**:** **[**string**(**ViewLabel**)],**

streams**:** **[**object**(**StreamInfo**)],**

polycode**:** object**(**Polycode**),**

participants**:** **[**object**(**ParticipantInfo**)]**

**}**

object**(**StreamInfo**)::**

**{**

id**:** string**(**StreamID**),**

type**:** "forward" **|** "mixed"**,**

media**:** object**(**MediaInfo**),**

info**:** object**(**PublicationInfo**)**/\*If type equals "forward"\*/

**|** object**(**ViewInfo**)**/\*If type equals "mixed"\*/

**}**

object**(**MediaInfo**)::**

**{**

audio**:** object**(**AudioInfo**)** **|** **false,**

video**:** object**(**VideoInfo**)** **|** **false**

**}**

object**(**AudioInfo**)::**

**{**

format**:** object**(**AudioFormat**)**

**}**

object**(**AudioFormat**)::**

**{**

codec**:** "pcmu" **|** "pcma" **|** "opus" **|** "g722" **|** "iSAC" **|** "iLBC" **|** "aac" **|** "ac3" **|** "nellymoser"**,**

sampleRate**:** number**(**SampleRate**),**

channelNum**:** number**(**ChannelNumber**)**

**}**

object**(**VideoInfo**)::**

**{**

format**:** object**(**VideoFormat**),**

parameters**:** object**(**VideoParameters**)**

**}**

object**(**VideoFormat**)::**

**{**

codec**:** "h264" **|** "h265" **|** "vp8" **|** "vp9"**,**

profile**:** "baseline" **|** "constrained-baseline" **|** "main" **|** "high" //If codec equals "h264".

**|** undefined //If codec does NOT equal "h264"

**}**

object**(**VideoParameters**)::**

**{**

resolution**:** object**(**Resolution**)** **|** "dynamic"**,**

framerate**:** number**(**FrameRateFPS**)** **|** "dyanamic"**,**

bitrateLevel**:** string**(**BitrateLevel**)** **|** "dyanamic"**,**

keyFrameInterval**:** number**(**KeyFrameIntervalSecond**)** **|** "dynamic"

**}**

object**(**Resolution**)::**

**{**

width**:** number**(**WidthPX**),**

height**:** number**(**HeightPX**)**

**}**

object**(**PublicationInfo**):**

**{**

owner**:** string**(**ParticipantId**),**

type**:** "webrtc" **|** "streaming" **|** "sip"**,**

audio**:** **{**

source**:** "mic"

**|** "screen-cast"

**|** "raw-file"

**|** "encoded-file"

**|** "streaming"

**|** undefined**,**

isActive**:** **true** **|** **false**

**}**

**|** **false,**

video**:** **{**

source**:** "camera"

**|** "screen-cast"

**|** "raw-file"

**|** "encoded-file"

**|** "streaming"

**|** undefined**,**

isActive**:** **true** **|** **false**

**}**

**|** **false**

attributes**:** object**(**ClientDefinedAttributes**)**

**}**

object**(**ViewInfo**):**

**{**

label**:** string**(**ViewLable**),**

layout**:** **[**object**(**SubStream2RegionIdMapping**)]**

**}**

object**(**SubStream2RegionIdMapping**)::**

**{**

stream**:** string**(**StreamId**),**

region**:** object**(**Region**)**

**}**

object**(**Region**)::**

**{**

id**:** string**(**RegionId**),**

shape**:** "rectangle" **|** "circle"**,**

**area:** object**(**Rectangle**)**/\*If shape equals "rectangle"\*/

**|** object**(**Circle**)**/\*If shape equals "circle"\*/

**}**

object**(**Rectangle**)::**

**{**

left**:** number**(**LeftRatio**),**

**top:** number**(**TopRatio**),**

width**:** number**(**WidthRatio**),**

height**:** number**(**HeightRatio**)**

**}**

object**(**Circle**)::**

**{**

centerW**:** number**(**CenterWRatio**),**

cenerH**:** number**(**CenterHRatio**),**

radius**:** number**(**RadiusWRatio**)**

**}**

object**(**Polycode**)::**

**{**

forward**:** object**(**PolycodeOptions**)** **|** **false,**

mixed**:** object**(**PolycodeOptions**)** **|** **falses**

**}**

object**(**PolycodeOptions**)::**

**{**

audio**:** object**(**AudioPolycode**)** **|** **false,**

video**:** object**(**VideoPolycode**)** **|** **false**

**}**

object**(**AudioPolycode**)::**

**{**

formats**:** **[**object**(**AudioFormat**)]**

**}**

object**(**VideoPolycode**)::**

**{**

formats**:** **[**object**(**VideoFormat**)],**

parameterses**:** object**(**VideoParametersSpecifiables**)**

**}**

object**(**VideoParametersSpecifiables**)::**

**{**

resolution**:** **[**object**(**Resolution**)],**

framerate**:** **[**5 **|** 15 **|** 24 **|** 30 **|** 48 **|** 60**],**

bitrateLevel**:** **[**"1.6x" **|** "1.4x" **|** "1.2x" **|** "1.0x" **|** "0.8x" **|** "0.6x" **|** "0.4x"**],**

keyFrameInterval**:** **[**1 **|** 2 **|** 5 **|** 15 **|** 30 **|** 60 **|** 100 **]**

**}**

object**(**ParticipantInfo**)::**

**{**

id**:** string**(**ParticilantId**),**

role**:** string**(**ParticipantRole**),**

user**:** object**(**UserInfo**)** /\*object(UserInfo) carried while login-ing\*/

**}**

object**(**ReconnectionTicket**)::**

**{**

credential**:** string**(**Credential**),**

notBefore**:** number**(**ValidTimeBegin**),**

notAfter**:** number**(**ValidTimeEnd**)**

**}**

### Participant Leaves a Room

**RequestName**: “logout”

**RequestData:** The LogoutInfo object with following definition:

object**(**LogoutInfo**)::**

**{**

id**:** string**(**ParticipantId**)**

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Is Notified on Other Participant’s Action

**NotificationName:** “participant”

**NotificationData:** The ParticipantAction object with following definition:

object**(**ParticipantAction**)::**

**{**

action**:** "join" **|** "leave"**,**

data**:** object**(**ParticipantInfo**)**/\*If action equals “join”\*/

**|** string**(**ParticipantId**)**/\*If status equals "leave"\*/

**}**

### Participant Sends a Text Message

**RequestName**: “text”

**RequestData:** The TextSendMessage object with following definition:

object**(**TextSendMessage**)::**

**{**

to**:** string**(**ParticipantId**)** **|** "all"**,**

message**:** string**(**Message**)** /\*The message length must not greater than 2048\*/

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Receives Text Message from another Participant

**NotificationName:** “text”

**NotificationData:** The TextReceiveMessage object with following definition:

object**(**TextReceiveMessage**)::**

**{**

from**:** string**(**ParticipantId**),**

message**:** string**(**Message**)**

**}**

### Participant Starts Publishing a Stream to Room

**RequestName**: “publish”

**RequestData:** The PublicationInfo object with following definition:

object**(**PublicationInfo**)::**

**{**

type**:** "webrtc" **|** "streaming"**,**

connection**:** undefined/\*If type equals "webrtc"\*/

**|** object**(**StreamingInConnectionOptions**)**/\*If type equals "streaming"\*/**,**

media**:** object**(**WebRTCMediaOptions**)**/\*If type equals "webrtc"\*/

**|** object**(**StreamingInMediaOptions**)**/\*If type equals "streaming"\*/**,**

attributes**:** object**(**ClientDefinedAttributes**)**

**}**

object**(**WebRTCMediaOptions**)::**

**{**

audio**:** **{**

source**:** "mic" **|** "screen-cast" **|** "raw-file" **|** "encoded-file"

**}**

**|** **false,**

video**:** **{**

source**:** "camera"**|** "screen-cast" **|** "raw-file" **|** "encoded-file"

**}**

**|** **false**

**}**

object**(**StreamingInConnectionOptions**):://Should categorize protocols.**

**{**

url**:** string**(**URLOfSourceStreaming**),**

transportProtocol**:** "tcp" **|** "udp"**,** //optional, default: "tcp"

bufferSize**:** number**(**BufferSizeBytes**)** //optional, default: 8192 bytes

**}**

object**(**StreamingInMediaOptions**)::**

**{**

audio**:** "auto" **|** **true** **|** **false,**

video**:** "auto" **|** **true** **|** **false,**

**}**

**ResponseData:** The PublicationResult object with following definition if ResponseStatus is “success”:

object**(**PublicationResult**)::**

**{**

id**:** string**(**StreamId**)**

**}**

### Participant Stops Publishing a Stream to Room

**RequestName**: “unpublish”

**RequestData:** The UnpublicationInfo object with following definition:

object**(**UnpublicationInfo**)::**

**{**

id**:** string**(**StreamId**)**

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Is Notified on Streams Update in Room

**NotificationName:** “stream”

**NotificationData:** The StreamUpdateMessage object with following definition.

object**(**StreamUpdateMessage**)::**

**{**

id**:** string**(**StreamId**),**

**status:** "add" **|** "update" **|** "remove"**,**

data**:** object**(**StreamInfo**)**/\*If status equals "add"\*/

**|** object**(**ForwardStreamUpdate**)**/\*If status equals "update" and id refers to a forward stream\*/

**|** object**(**MixedStreamUpdate**)**/\*If status equals "update" and id refers to a mixed stream\*/

**|** undefined/\*If status equals "remove"\*/

**}**

object**(**ForwardStreamUpdate**)::**

**{**

audio**:** **{**

isActive**:** **true** **|** **false**

**}**

**|** undefined**,**

video**:** **{**

isActive**:** **true** **|** **false**

**}**

**|** undefined

**}**

object**(**MixedStreamUpdate**)::**

**{**

layout**:** **[**object**(**SubStream2RegionIdMapping**)]**

**}**

### Participant Controls a Stream

**RequestName**: “stream-control”

**RequestData:** The StreamControlInfo object with following definition:

object**(**StreamControlInfo**)::**

**{**

id**:** string**(**StreamId**),** //Must refer to a forward stream.

operation**:** "mix" **|** "unmix" **|** "set-region" **|** "get-region" **|** "hold" **|** "resume"**,**

data**:** string**(**ViewLabel**)**/\*If operation equals "mix", "unmix" or "get-region\*/

**|** object**(**RegionSetting**)**/\*If operation equals "set-region"\*/

**|** **(**"audio" **|** "video" **|** "av"**)**/\*If operation equals "hold" or "resume"\*/

**}**

object**(**RegionSetting**)::**

**{**

view**:** string**(**ViewLabel**),**

region**:** string**(**RegionId**)**

**}**

**ResponseData:** undefined in case operation is not “get-region”, and object RegionInfo defined below in case operation is “get-region” if ResponseStatus is “success”.

object**(**RegionInfo**)::**

**{**

region**:** string**(**RegionId**)**

**}**

### Participant Starts a Subscription

**RequestName**: “subscribe”

**RequestData:** The SubscriptionInfo object with following definition:

object**(**SubscriptionInfo**)::**

**{**

type**:** "webrtc" **|** "streaming" **|** "recording"**,**

connection**:** undefined/\*If type equals "webrtc"\*/

**|** object**(**StreamingOutConnectionOptions**)**/\*If type equals "streaming"\*/

**|** object**(**RecordingStorageOptions**)**/\*If type equals "recording"\*/**,**

media**:** object**(**MediaSubOptions**)**

**}**

object**(**StreamingOutConnectionOptions**)::**

**{**

url**:** string**(**TargetStreamingURL**)**

**}**

object**(**RecordingStorageOptions**)::**

**{**

container**:** "mkv" **|** "mp4" **|** "ts"/\*Must be compatible with media.audo.codec and media.video.codec\*/ //optional, default: determined by media.video.codec.

**}**

object**(**MediaSubOptions**)::**

**{**

audio**:** object**(**AudioSubOptions**)** **|** **false,**

video**:** object**(**VideoSubOptions**)** **|** **false**

**}**

object**(**AudioSubOptions**)::**

**{**

**source:** string**(**StreamId**),**

spec**:** object**(**AudioSubSpecification**)**/\*If type does NOT equal "webrtc", and audio transcoding is needed or non-default mixed audio is wanted\*/

**|** undefined/\*If type equals "webrtc", or original forward audio or default mixed audio is wanted\*/

**}**

object**(**AudioSubSpecification**)::**

**{**

codec**:** string**(**WantedAudioCodec**),**

sampleRate**:** number**(**WantedSampleRate**)** **|** undefined**,**

channelNum**:** number**(**WantedChannelNum**)** **|** undefined

**}**

object**(**VideoSubOptions**)::**

**{**

**source:** string**(**StreamId**),**

spec**:** object**(**VideoSubSpecification**)**/\*If video transcoding is needed or non-default mixed video is wanted\*/

**|** undefined/\*If original forward video or default mixed video is wanted\*/

**}**

object**(**VideoSubSpecification**)::**

**{**

codec**:** string**(**WantedVideoCodec**),**//Will be ignored if type equals "webrtc"

profile**:** string**(**WantedVideoProfile**)** /\*Will be ignored if type equals "webrtc" or codec does NOT equal "h264"\*/ **|** undefined**,**

resolution**:** object**(**Resolution**)** **|** undefined**,**

framerate**:** number**(**WantedFrameRateFPS**)** **|** undefined**,**

bitrateLevel**:** string**(**WantedBitrateLevel**)** **|** undefined**,**

keyFrameInterval**:** number**(**WantedKeyFrameIntervalSecond**)** **|** undefined

**}**

**ResponseData:** The SubscriptionResult object with following definition if ResponseStatus is “success”:

object**(**SubscriptionResult**)::**

**{**

id**:** string**(**SubscriptionId**)**

**}**

### Participant Stops a Self-Initiated Subscription

**RequestName**: “unsubscribe”

**RequestData:** The UnsubscriptionInfo object with following definition:

object**(**UnsubscriptionInfo**)::**

**{**

id**:** string**(**SubscriptionId**)**

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Controls a Self-Initiated Subscription

**RequestName**: “subscription-control”

**RequestData:** The SubscriptionControlInfo object with following definition:

object**(**SubscriptionControlInfo**)::**

**{**

id**:** string**(**SubscriptionId**),**

operation**:** "update" **|** "pause" **|** "play"**,**

data**:** object**(**SubscriptionUpdate**)**/\*If operation equals "update"\*/

**|** **(**"audio" **|** "video"**)**/\*If operation equals "pause" or "play"\*/

**}**

object**(**SubscriptionUpdate**)::**

**{**

audio**:** object**(**AudioUpdate**)** **|** undefined**,**

video**:** object**(**VideoUpdate**)** **|** undefined

**}**

object**(**AudioUpdate**)::**

**{**

**source:** string**(**StreamId**)**

**}**

object**(**VideoUpdate**)::**

**{**

**source:** string**(**StreamId**),**

spec**:** object**(**VideoUpdateSpecification**)**/\*If video transcoding is ongoing or mixed stream is being subscribed\*/

**|** undefined/\*If original forward video or default mixed video is wanted\*/

**}**

object**(**VideoUpdateSpecification**)::**

**{**

resolution**:** object**(**Resolution**),**

framerate**:** number**(**WantedFrameRateFPS**),**

bitrateLevel**:** string**(**WantedBitrateLevel**),**

keyFrameInterval**:** number**(**WantedKeyFrameIntervalSecond**)**

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Sends Session Signaling

**RequestName**: “soac”

**RequestData:** The SOACMessage object with following definition:

object**(**SOACMessage**)::**

**{**

id**:** string**(**SessionId**),** /\* StreamId returned in publishing or SubscriptionId returned in subscribing\*/

signaling**:** object**(**OfferAnswer**)** **|** object**(**Candidate)

**}**

object**(**OfferAnser**)::**

**{**

type**:** "offer" **|** "answer"**,**

sdp**:** string**(**SDP**)**

**}**

object**(**Candidate**)::**

**{**

type**:** "candidate"**,**

candidate**:** string**(**Candidate**)**

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

### Participant Receives Session Progress

**NotificationName:** “progress”

**NotificationData:** The SessionProgress object with following definition.

object**(**SessionProgress**)::**

**{**

id**:** string**(**SessionId**),** /\* StreamId returned in publishing or SubscriptionId returned in subscribing\*/

status**:** "soac" **|** "ready" **|** "error"**,**

data**:** object**(**SOAPMessage**)**/\*If status equals “soap”\*/

**|** **(**undefined/\*If status equals “ready” and session is NOT for recording\*/

**|** object**(**RecorderInfo**)**/\*If status equals “ready” and session is for recording\*/ **)**

**|** string**(**Reason**)**/\*If status equals “error”\*/

**}**

object**(**RecorderInfo**)::**

**{**

host**:** string**(**HostnameOrIPOfRecorder**),**

file**:** string**(**FullPathNameOfRecordedFile**)**

**}**

### Participant Sets Permission of another Participant

**RequestName**: “set-permission”

**RequestData:** The SetPermission object with following definition:

object**(**SetPermission**)::**

**{**

id**:** string**(**ParticipantId**),**

operation**:** "publish" **|** "subscribe" **|** "text"**,**

scope**:** **([**"webrtc" **|** "streaming"**])**/\*If operation equals "publish"\*/

**|** **([**"webrtc" **|** "recording" **|** "streaming"**])**/\*If operation equals "subscribe"\*/

**|** **([**"to-peer" **|** "to-all"**])**/\*If operation equals "text"\*/

**}**

**ResponseData:** undefined if ResponseStatus is “success”.

# RESTful Signaling

## Connection Maintenance

### Client Connects

No dedicate signaling message being sent from client is needed. The very first request from a new client must be a client joining request, which is considered as the connecting request as well.

Portal should be able to listen at both an HTTP and an HTTPS server port to receive RESTful request from clients. If the HTTPS server is enabled, the proper SSL certificate and private key store path must be correctly configured by portal.keystorePath item in portal.toml.

### Client Keeps Alive

The participant’s querying request ([3.3](#_3.3_User_queries)) will be considered as the keep alive heart-beat from client.

If three querying request loss are consecutively detected by Portal, Portal will judge that the connection has lost.

### Client Disconnects

### Client Reconnects

## Conferencing

### Participant Joins

### Participant leaves