# AhoyConference Server WebSocket API (Version 1.0)

The AhoyConference server (ahoyconferenced) is a multipoint control unit (MCU) for WebRTC video conferencing. Clients connect to it using a JSON message based API transported over the (secure) WebSocket protocol. The JSON message based protocol is described in this document.

All communication is initiated by the client connecting to the WebSocket port using the WebSocket subprotocol “conference-protocol”. The server will automatically remove the client from an active conference when the WebSocket connection has been disconnected. Every JSON message contains a “messageType” property (identifying the message type) and a “transactionID” property (for matching request and response). JSON response messages contain a numerical “status” and a descriptive “reason” property which are similar (but not identical) to HTTP responses.

## 1. Client to server requests

### 1.1. messageType “CONFERENCE_CREATE_request”

This request creates a new conference room. Ahoyconferenced does not have user based authentication the privilege level of a client is determined by the password it supplies. Three different passwords can be set to distinguish between an audience member (“listenerPassword”, can only receive audio/video and can send chat messages), a speaker (“password”, can send audio/video/chat) and a moderator (“moderatorPassword”, can send audio/video/chat, can kick other clients and lock/unlock the conference room).

Parameters:

| | |
|---|---|
| “conferenceID”: | unique identifier for the conference (optional, will be set by server) |
| “conferenceName”: | a name for the conference room (optional) |
| “description”: | a short description (optional) |
| “password”: | speaker password (optional) |
| “listenerPassword”: | audience password (optional) |
| “moderatorPassword”: | moderator password (optional) |
| “maxMember”: | limit the conference room to “maxMember” participants (optional) |
| “maxVideoBitrate”: | limit for the video bitrate (optional) |
| “timeLimit”: | limit the conference to “timeLimit” seconds after the first participant has joined the conference (optional) |

Example request:

```
{       “messageType”:      “CONFERENCE_CREATE_request”,
        “conferenceID”:     “ my-unique-id-1”,
        “transactionID”:    “id-0.5881619271822274”
}
```

“status” / “reason” response codes:

200 / OK

Example response:

```
{
        "messageType":      "CONFERENCE_CREATE_response",
        "status":           200,
        "reason":"          OK",
        "conferenceID":     "my-unique-id-1",
        "conferenceName":   "my-unique-id-1",
        "description":      "",
        "transactionID":    "id-0.5881619271822274"
}
```

## 1.2. messageType “CONFERENCE_JOIN_request”

To join an existing conference a client will send this request.

Parameters:

“conferenceID”: id of the conference room to join

“name”: a name describing the client, will be shown to other conference participants

“password”: password (optional)

Example request:

```
{       “messageType”:  “CONFERENCE_JOIN_request”,
        “conferenceID”: “my-unique-id-1”
        "transactionID": "id-0.8157990577165037"
}
```

“status” / “reason” response codes:

| | |
|---|---|
| 200 / OK | |
| 403 / forbidden | wrong password |
| 404 / not found | the conference ID does not exist |
| 470 / locked | a moderator has locked the conference room |
| 486 / conference full | the conference has a limit on the number of participants which has been reached |

Example response:

```
{
    "messageType":      "CONFERENCE_JOIN_response",
    "status":           200,
    "reason":           "OK",
    "remainingSeconds": -1,     // greather than 0 if a time limit has been set
    "locked":           false,
    "conferenceID":     "my-unique-id-1",
    "conferenceName":   "my-unique-id-1",
    "description":      "",
    "speakerID":        "8e6ba9db9deb96bb23c70788f1b9a7634245aecf1160787ca571ab12dde07b78",
    "memberID":         "8e6ba9db9deb96bb23c70788f1b9a7634245aecf1160787ca571ab12dde07b78",
    "moderator":        true,   // the user is a moderator
    "speaker":          true,   // the user is allowed to publish audio/video
    "webrtc":           true,
    "members":          [],     // Array of participants, see “CONFERENCE_JOIN_indication” for details
    "transactionID":    "id-0.8157990577165037"
}
```

## 1.3. messageType “MEDIA_SHARE_request”

If a member has the “speaker” property set to true then it can share its audio and/or video by sending a MEDIA_SHARE_request.

The server will reply with a MEDIA_SHARE_response and will then send a MEDIA_RECEIVE_request to the client. The client should be able to handle consecutive MEDIA_RECEIVE_request messages (without having sent another MEDIA_SHARE_request message).

Parameters:

| | |
|---|---|
| “audio”: | share the microphone |
| “video”: | share the camera |
| “maxVideoBitrate”: | when sharing the camera limit the video bitrate (optional, default 500 kbit/s) |

Example request:

```
{
  "messageType":        "MEDIA_SHARE_request",
  "audio":              true,
  "video":              true,
  "maxVideoBitrate":     500,
  "transactionID":      "0672B4A8-E067-4747-821D-BA43E4165AE8"
}
```

“status” / “reason” response codes:

200 / OK

403 / forbidden

Example response:

```
{
      "messageType":      "MEDIA_SHARE_response",
      "status":           200,
      "reason":           "OK",
      "transactionID":    "0672B4A8-E067-4747-821D-BA43E4165AE8"
}
```

## 1.4. messageType “MEDIA_request”

Ahoyconferenced does not mix audio/video streams, that's why a client requests each other's audio/video streams individually. This offers great flexibility on the client side and maintains the best possible quality. To receive audio/video streams the client will send a MEDIA_request containing an array of the member IDs and requested media types.

After replying with a MEDIA_response the server will send a SDP_request to the client for each requested member media stream. The SDP_request will use the memberID as the transactionID, so the client can easily map the audio/video stream to the corresponding member.

Parameters:

“members”: Array of media request objects

- “memberID”: unique ID of the participant from which audio/video is being requested
- “audio”: request audio
- “video”: request video

Example request:

```
{
    "messageType":  "MEDIA_request",
    "members":      [
        {
            "memberID":     "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf",
            "audio":        true,
            "video":        true
        }
    ],
    "transactionID":"  WEqP3o9MgR7l9WU3XHMkEevxvwRoEfco"
}
```

“status” / “reason” response codes:

200 / OK

Example response:

```
{
    "messageType":  "MEDIA_response",
    "status":       200,
    "reason":       "OK",
    "transactionID":"WEqP3o9MgR7l9WU3XHMkEevxvwRoEfco"
}
```

## 1.5. messageType “CONFERENCE_LEAVE_request”

To leave a conference the client will send a CONFERENCE_LEAVE_request. Alternatively the client can also just disconnect the WebSocket connection. The server will send a CONFERENCE_LEAVE_indication to the remaining participants.

Parameters: none

Example request:

```
{
    "messageType":      "CONFERENCE_LEAVE_request",
    "transactionID":    "DFCBBD79-A6A6-4E81-BB7D-0CC2369A37D1"
}
```

“status” / “reason” response codes:

200 / OK

Example response:

```
{
    "messageType":  "CONFERENCE_LEAVE_response",
    "status":       200,
    "reason":       "OK",
    "transactionID":    "DFCBBD79-A6A6-4E81-BB7D-0CC2369A37D1"
}
```

## 1.6. messageType “CONFERENCE_KICK_request”

A moderator can remove a member from the conference by sending a CONFERENCE_KICK_request.

Parameters:

“memberID”: ID of the conference participant to remove

Example request:

```
{
    "messageType":      "CONFERENCE_KICK_request",
    "memberID":         "7803c009c170ef38589dcbaff6c279b31baf8cb9964433422cc50f152517319c",
    "transactionID":    "Elv1mqEsSlyTTaH7T427WCoK7ECdpKNx"
}
```

“status” / “reason” response codes:

200 / OK

Example response:

```
{
    "messageType":      "CONFERENCE_KICK_response",
    "status":           200,
    "reason":           "OK",
    "transactionID":    "Elv1mqEsSlyTTaH7T427WCoK7ECdpKNx"
}
```

## 1.7. messageType “CHAT_MESSAGE_request”

Send a chat message to the conference.

Parameters:

“message”: Object describing the chat message

“text”: Message text which is forwarded transparently. Different types of messages (e.g. control and chat messages) can be transported by JSON-encoding them into the “text” property.

Example request:

```
{
    “messageType": "CHAT_MESSAGE_request",
    "message": {
        "text": "{\"chat\":\"test\"}"
    },
    "transactionID": "TP0FcTOx4zm8MMkfOcdgDs3ggAODxKyr"
}
```

“status” / “reason” response codes:

200 / OK

Example response:

```
{
    "messageType": "CHAT_MESSAGE_response",
    "status": 200,
    "reason": "OK",
    "transactionID": "TP0FcTOx4zm8MMkfOcdgDs3ggAODxKyr"
}
```

## 1.8. messageType “CONFERENCE_DESCRIPTION_request”

A moderator can change the description of the conference. A CONFERENCE_DESCRIPTION_indication will be sent to every participant;

Parameters:

“description”: New description of the conference

Example request:

```
{
    "messageType":    "CONFERENCE_DESCRIPTION_request",
    "description":    "Annual shareholder conference",
    "transactionID":  "D515798D-B714-4DE8-8C81-AC1826800C68"
}
```

“status” / “reason” response codes:

200 / OK

403 / Forbidden

Example response:

```
{
    "messageType":    "CONFERENCE_DESCRIPTION_response",
    "status":         200,
    "reason":         "OK",
    "transactionID":  "D515798D-B714-4DE8-8C81-AC1826800C68"
}
```

## 2. Server events / indications

### 2.1. messageType “CONFERENCE_JOIN_indication”

A new participant has joined the conference.

Parameters:

“member”: Object describing the participant

- “name”: Name of the participant
- “memberID”: Unique ID of the participant
- “moderator”: flag indicating if the participant is a moderator
- “speaker”: flag indicating if the participant is allowed to share audio/video streams
- “webrtc”: flag indicating if the participant is a WebRTC client or a SIP client (without video support)
- “audio”: is an audio stream available and if yes is it currently muted?
- “video”: is a video stream available and if yes is it currently muted?
  A WebRTC client has to allow access to the mic/camera. That's why a WebRTC participant will always join the conference with “audio.available” and “video.available” set to false.
  After allowing mic/camera access a MEDIA_indication will be sent.
- “ip”: ip address of the participant

Example indication:

```
{
    "messageType":  "CONFERENCE_JOIN_indication",
    "member":    {
        "name": "K-P Junghanns",
        "memberID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf",
        "moderator":   false,
        "speaker":     true,
        "webrtc":      true,
        "audio":      {
            "available":   false,
            "muted":       true
        },
        "video":      {
            "available":   false,
            "muted":       true
        },
        "ip":   "127.0.0.1"
    }
}
```

## 2.2. messageType “CONFERENCE_LEAVE_indication”

A participant has left the conference.

Parameters:

“member”: Object describing the participant

“name”: Name of the participant

“memberID”: Unique ID of the participant

“moderator”: flag indicating if the participant is a moderator

Example indication:

```
{
    "messageType": "CONFERENCE_LEAVE_indication",
    "member":    {
        "name": "K-P Junghanns",
        "memberID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf",
        "moderator":    false
    }
}
```

## 2.3. messageType “CONFERENCE_KICK_indication”

The (local) user has been kicked from the conference. The server will also disconnect the WebSocket.

Parameters:

“timeout”: flag indicating if the user was kicked because a conference time limit was reached

Example indication:

```
{
    "messageType":  "CONFERENCE_KICK_indication",
    "timeout":    0
}
```

## 2.4. messageType “MEDIA_indication”

The media status of a participant has changed.

Parameters:

“member”: Object describing the participant

- “name”: Name of the participant
- “memberID”: Unique ID of the participant

“audio”: Object describing the state of the audio stream

- “available”: flag indicating if a stream is available
- “muted”: flag indicating if the stream is muted
- “speaking”: flag indicating if the member is currently speaking (server side speaker detection)

“video”: Object describing the state of the video stream

- “available”: flag indicating if a stream is available
- “muted”: flag indicating if the stream is muted

Example indication:

```
{
    "messageType":  "MEDIA_indication",
    "member":    {
        "name": "K-P Junghanns",
        "memberID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf"
    },
    "audio":    {
        "available":    true,
        "muted":    false,
        "speaking":    false
    },
    "video":    {
        "available":    true,
        "muted":    false
    }
}
```

## 2.5. messageType “CHAT_MESSAGE_indication”

A participant has sent a chat message.

Parameters:

“from”: Object describing the participant who sent the message

“message”: Object describing the text message

“text”: Message text which was forwarded transparently. Different types of messages (e.g. control and chat messages) can be transported by JSON-encoding them into the “text” property.

Example indication:

```
{
    "messageType":  "CHAT_MESSAGE_indication",
    "from": {
        "name": "           Somebody",
        "memberID":         "c6426f3c59b1c63cdcd9b1c3779f59acf09ac1790c4e547383ada3ea66c8428b"
    },
    "message":    {
        "text":             "{\"chat\":\"test\"}"
    }
}
```

## 2.6. messageType “CONFERENCE_DESCRIPTION_indication”

A moderator has changed the description of the conferenfce.

Parameters:

“description”: New description of the conference

Example indication:

```
{
        "messageType":     "CONFERENCE_DESCRIPTION_indication",
        "description":     "Annual shareholder conference"
}
```

## 3. Server to client requests

### 3.1. messageType “MEDIA_RECEIVE_request”

When a participant has requested to publish its audio/video streams the server will send a MEDIA_RECEIVE_request.

The client will create a new WebRTC PeerConnection, feed the supplied SDP offer to, generate an SDP answer and return it in a MEDIA_RECEIVE_response message.

The client should be able to handle consecutive MEDIA_RECEIVE_request messages (without having sent another MEDIA_SHARE_request message).

Parameters:

“sdp”: server generated SDP offer

Example request:

```
{
    "messageType":      "MEDIA_RECEIVE_request",
    "sdp":              “.....”,
    "transactionID":    "f59bd812c760ca11df589fc823e1c35d32079881e75a96886096adc0b7d7633d"
}
```

Example response:

```
{
    "messageType":      "MEDIA_RECEIVE_response",
    "status":200,
    "reason":"OK",
    "sdp":              "…....",
     "transactionID":   "f59bd812c760ca11df589fc823e1c35d32079881e75a96886096adc0b7d7633d"
 }
```

## 3.2. messageType “SDP_request”

After a client has requested to receive the media streams of one or more other participants the server will send a SDP_request for each of the requested media streams.

The client will create a new WebRTC PeerConnection, feed the supplied SDP offer to it, generate an SDP answer and return it in a SDP_response message.

Parameters:

“sdp”: server generated SDP offer

“member”: Object describing the participant

Example request:

```
{
    "messageType":  "SDP_request",
    "sdp":  "….",
    "member":    {
        "name": "K-P Junghanns",
        "memberID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf"
    },
    "transactionID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf"
}
```

Example response:

```
{
        "messageType":    "SDP_response",
        "status":  200,
        "reason":  "OK",
        "sdp":    "...",
        “transactionID":    "5925d22e17723e76bd240af5b3cfa057218391cd334e700e8e95d897b9c8a5cf"
}
```