

# Video Conference

## Example of video conference

This example can be used to organize video conference for three participants on Web Call Server. Each participant can publish a WebRTC stream

On the screenshot below the participant is connected, publishing a stream and playing streams from the other two participants.


# Conference

**WCS URL**

**Login**


☐ **Record**

ESTABLISHED




Bob

Unmuted



Cindy

Unmuted



PUBLISHING

9:24 chat - room is empty  
 9:25 Bob - joined  
 9:25 Cindy - joined

**Invite**

<http://localhost:8081/client2/examples/demo/streaming/conference/conference.html?roomName=room-955abd>

Three videos are played on the page

- video from the camera of this participant (Alice) - the lower one
- videos from the other participants (Bob and Cindy)

## Code of the example

The path to the source code of the example on WCS server is:

/usr/local/FlashphonerWebCallServer/client/examples/demo/streaming/conference

- conference.css - file with styles
- conference.html - page of video conference participant
- conference.js - script providing functionality for video conference

This example can be tested using the following address:

<https://host:8888/client/examples/demo/streaming/conference/conference.html>

Here host is the address of the WCS server.

## Analyzing the code

To analyze the code, let's take the version of file `conference.js` with hash `90771d4`, which is available [here](#) and can be downloaded with corresponding build `2.0.218`.

Script for video conference uses RoomApi designed for video chats, video conferences, webinars and other applications that involve presence of users in one virtual "room". To use RoomApi, the script `flashphoner-room-api.js` should be included

```
<script type="text/javascript" src="../../../../flashphoner-room-api.js">
</script>
```

In this case, `RoomApi.sdk` object should be used to access Flashphoner methods

```
var Flashphoner = RoomApi.sdk;
...
Flashphoner.init();
```

Unlike direct connection to server with method `createSession()`, method `RoomApi.connect()` is used when a user connects to a conference.

Method `Session.join()` is used to join to a new "room". When joining, object `Room` is created for work with the "room", and object `Participant` - for work with the participant.

All events occurring in the "room" (a user joined/left, or sent a message), are sent to all users connected to the "room". For example, in the following code, a user joins to a "room" and gets the list of already connected users.

```
connection.join({name: getRoomName()}).on(ROOM_EVENT.STATE, function(room){
    var participants = room.getParticipants();
    ...
});
```

Here user receives data of another participant which have just joined:

```
.on(ROOM_EVENT.JOINED, function(participant){
    installParticipant(participant);
    addMessage(participant.name(), "joined");
    ...
});
```

## 1. Initialization of the API

`Flashphoner.init()` [code](#)

```
Flashphoner.init();
```

## 2. Camera and microphone access request

`Flashphoner.getMediaAccess()` [code](#)

```
Flashphoner.getMediaAccess(null, localDisplay).then(function() {
    createConnection(url, username);
}).catch(function(error) {
    console.error("User not allowed media access: "+error);
    $("#failedInfo").text("User not allowed media access. Refresh the page");
    onLeft();
});
```

## 3. Connection to server

`RoomApi.connect()` [code](#)

```
function createConnection(url, username) {
    connection = RoomApi.connect({urlServer: url, username:
    username}).on(SESSION_STATUS.FAILED, function(session){
        ...
    });
}
```

## 4. Receiving the event confirming successful connection

`ConnectionStatusEvent ESTABLISHED` [code](#)

```
connection = RoomApi.connect({urlServer: url, username:
    username}).on(SESSION_STATUS.FAILED, function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(session) {
    ...
}).on(SESSION_STATUS.ESTABLISHED, function(session) {
    setStatus('#status', session.status());
    joinRoom();
});
```

---

## 5. Joining a conference

`Session.join()` [code](#)

To join, name of the conference room is passed to the method. (The name can be specified as parameter in the URL of the client page; otherwise, random name will be generated.)

```
connection.join({name: getRoomName(), record:
isRecord()}).on(ROOM_EVENT.STATE, function(room){
    ...
});
```

## 6. Receiving the event describing chat room state

`RoomStatusEvent STATE` [code](#)

On this event:

- the length of the array of `Participant` objects returned by method `Room.getParticipants()` is determined to get the number of already connected participants
- if the maximum allowed number of participants had already been reached, the user leaves the "room" using `Room.leave()`
- otherwise, the user starts publishing video stream

```
connection.join({name: getRoomName(), record:
isRecord()}).on(ROOM_EVENT.STATE, function(room){
    var participants = room.getParticipants();
    console.log("Current number of participants in the room: " +
participants.length);
    if (participants.length >= _participants) {
        console.warn("Current room is full");
        $("#failedInfo").text("Current room is full.");
        room.leave().then(onLeft, onLeft);
        return false;
    }
    setInviteAddress(room.name());
    if (participants.length > 0) {
        var chatState = "participants: ";
        for (var i = 0; i < participants.length; i++) {
            installParticipant(participants[i]);
            chatState += participants[i].name();
            if (i != participants.length - 1) {
                chatState += ",";
            }
        }
        addMessage("chat", chatState);
    } else {
        addMessage("chat", " room is empty");
    }
});
```

```

    }
    publishLocalMedia(room);
    onJoined(room);
  }).on(ROOM_EVENT.JOINED, function(participant){
    ...
  }).on(ROOM_EVENT.LEFT, function(participant){
    ...
  }).on(ROOM_EVENT.PUBLISHED, function(participant){
    ...
  }).on(ROOM_EVENT.FAILED, function(room, info){
    ...
  }).on(ROOM_EVENT.MESSAGE, function(message){
    ...
  });
});

```

## 7. Low Power Mode checking before publishing on mobile device

`Flashphoner.playFirstVideo()` [code](#)

```

if (Browser.isSafariWebRTC()) {
  var display = document.getElementById("localDisplay");
  Flashphoner.playFirstVideo(display, true, PRELOADER_URL).then(function()
  {
    publishLocalMedia(room);
  }).catch(function (error) {
    console.log("Can't automatically publish local stream, use Publish
button");
    for (var i = 0; i < display.children.length; i++) {
      if (display.children[i]) {
        console.log("remove cached instance id " +
display.children[i].id);
        display.removeChild(display.children[i]);
      }
    }
    onMediaStopped(room);
  });
}

```

## 8. Video streaming

`Room.publish()` [code](#)

`div` element to display video from camera is passed to the `Room.publish()` method

```

room.publish({
  display: display,
  constraints: constraints,
  record: false,
  receiveVideo: false,
  receiveAudio: false
}).on(STREAM_STATUS.FAILED, function (stream) {
  console.warn("Local stream failed!");
  setStatus("#localStatus", stream.status());
});

```

```

        onMediaStopped(room);
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        setStatus("#localStatus", stream.status());
        onMediaPublished(stream);
    }).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
        setStatus("#localStatus", stream.status());
        onMediaStopped(room);
    });

```

## 9. Receiving the event notifying that other participant joined to the room

RoomStatusEvent JOINED [code](#)

```

connection.join({name: getRoomName(), record:
isRecord()}).on(ROOM_EVENT.STATE, function(room){
    ...
}).on(ROOM_EVENT.JOINED, function(participant){
    installParticipant(participant);
    addMessage(participant.name(), "joined");
}).on(ROOM_EVENT.LEFT, function(participant){
    ...
}).on(ROOM_EVENT.PUBLISHED, function(participant){
    ...
}).on(ROOM_EVENT.FAILED, function(room, info){
    ...
}).on(ROOM_EVENT.MESSAGE, function(message){
    ...
});

```

## 10. Receiving the event notifying that other participant published video stream

RoomStatusEvent PUBLISHED [code](#)

```

connection.join({name: getRoomName(), record:
isRecord()}).on(ROOM_EVENT.STATE, function(room){
    ...
}).on(ROOM_EVENT.JOINED, function(participant){
    ...
}).on(ROOM_EVENT.LEFT, function(participant){
    ...
}).on(ROOM_EVENT.PUBLISHED, function(participant){
    playParticipantsStream(participant);
}).on(ROOM_EVENT.FAILED, function(room, info){
    ...
}).on(ROOM_EVENT.MESSAGE, function(message){
    ...
});

```

## 11. Low Power Mode checking before playback on mobile device

`Flashphoner.playFirstVideo()` [code](#)

```
if (Browser.isSafariWebRTC()) {
    Flashphoner.playFirstVideo(pDisplay, false,
    PRELOADER_URL).then(function() {
        playStream(participant, pDisplay);
    }).catch(function (error) {
        // Low Power Mode detected, user action is needed to start playback
        in this mode #WCS-2639
        console.log("Can't automatically play participant" +
        participant.name() + " stream, use Play button");
        for (var i = 0; i < pDisplay.children.length; i++) {
            if (pDisplay.children[i]) {
                console.log("remove cached instance id " +
                pDisplay.children[i].id);
                pDisplay.removeChild(pDisplay.children[i]);
            }
        }
        onParticipantStopped(participant);
    });
}
```

## 12. Playback of video stream

`Participant.play()` [code](#)

The following parameters are passed to the method:

- `display` - `div` element to display remote video;
- `options.unmutePlayOnStart` - enables (by default) or disables (for example, in Android Edge) automatic audio unmuting while starting playback;
- `options.constraints.audio.deviceId` - audio output device Id (the example uses default audio device)

A user must click a button if automatic audio playback is disabled

```
var options = {
    unmutePlayOnStart: true,
    constraints: {
        audio: {
            deviceId: 'default'
        }
    }
};
// Leave participant stream muted in Android Edge browser #WCS-3445
if (Browser.isChromiumEdge() && Browser.isAndroid()) {
    options.unmutePlayOnStart = false;
}
participant.getStreams()[0].play(display, options).on(STREAM_STATUS.PLAYING,
function (playingStream) {
    var video = document.getElementById(playingStream.id())
    video.addEventListener('resize', function (event) {
```



```

        resizeVideo(event.target);
    });
    // Set up participant Stop/Play button
    if (playBtn) {
        $(playBtn).text("Stop").off('click').click(function() {
            $(this).prop('disabled', true);
            playingStream.stop();
        }).prop('disabled', false);
    }
    // Set up participant audio toggle button #WCS-3445
    if (audioBtn) {
        $(audioBtn).text("Audio").off('click').click(function() {
            if (playingStream.isRemoteAudioMuted()) {
                playingStream.unmuteRemoteAudio();
            } else {
                playingStream.muteRemoteAudio();
            }
        }).prop('disabled', false);
    }
    // Start participant audio state checking timer #WCS-3445
    participantState.startMutedCheck(playingStream);
}).on(STREAM_STATUS.STOPPED, function () {
    onParticipantStopped(participant);
}).on(STREAM_STATUS.FAILED, function () {
    onParticipantStopped(participant);
});

```

### 13. Stop of streaming.

`Stream.stop()` [code](#)

```

function onMediaPublished(stream) {
    $("#localStopBtn").text("Stop").off('click').click(function(){
        $(this).prop('disabled', true);
        stream.stop();
    }).prop('disabled', false);
    ...
}

```

### 14. Receiving the event confirming successful streaming stop

`StreamStatusEvent UNPUBLISHED` [code](#)

```

room.publish({
    display: display,
    constraints: constraints,
    record: false,
    receiveVideo: false,
    receiveAudio: false
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    ...
}

```

```
}).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
    setStatus("#localStatus", stream.status());
    onMediaStopped(room);
});
```

## 15. Leaving conference room

`Room.leave()` [code](#)

```
function onJoined(room) {
    $("#joinBtn").text("Leave").off('click').click(function(){
        $(this).prop('disabled', true);
        room.leave().then(onLeft, onLeft);
    }).prop('disabled', false);
    ...
}
```

## 16. Mute/unmute audio and video of the published stream

`Stream.isAudioMuted()`, `Stream.isVideoMuted()`, `Stream.muteAudio()`,  
`Stream.unmuteAudio()`, `Stream.muteVideo()`, `Stream.unmuteVideo()` [code](#)

```
function onMediaPublished(stream) {
    ...
    $("#localAudioToggle").text("Mute A").off('click').click(function(){
        if (stream.isAudioMuted()) {
            $(this).text("Mute A");
            stream.unmuteAudio();
        } else {
            $(this).text("Unmute A");
            stream.muteAudio();
        }
    }).prop('disabled', false);
    $("#localVideoToggle").text("Mute V").off('click').click(function() {
        if (stream.isVideoMuted()) {
            $(this).text("Mute V");
            stream.unmuteVideo();
        } else {
            $(this).text("Unmute V");
            stream.muteVideo();
        }
    }).prop('disabled', false);
}
```

## 17. Sending text message.

`Participant.sendMessage()` [code](#)

When `Send` button is clicked,

- method `room.getParticipants()` is used to get the array of connected participants
- the message is sent to each participant

```
function onJoined(room) {  
    ...  
    $('#sendMessageBtn').off('click').click(function(){  
        var message = field('message');  
        addMessage(connection.username(), message);  
        $('#message').val("");  
        //broadcast message  
        var participants = room.getParticipants();  
        for (var i = 0; i < participants.length; i++) {  
            participants[i].sendMessage(message);  
        }  
    }).prop('disabled', false);  
    $('#failedInfo').text("");  
}
```