Working with chat rooms

Overview

Web Call Server allows embedding of a video chat to your project, that will work on most of modern browsers without installing third-party software as well as on mobile devices.

	Chrome	Firefox	Safari	Edge
Windows		\checkmark	×	
Mac OS				
Android			×	
iOS				

Supported platforms and browsers

Supported codecs

- Video: H.264, VP8
- Audio: Opus, G.711

Functions

- Video chat
- Text chat
- Video conference
- Video conference with screen sharing

Operation flowchart



- 1. The browser of the participant 1 connects to the server via Websocket and sends the join command.
- 2. The browser of the participant 1 can send a stream via WebRTC to publish it in the chat room and receive streams published in the room.
- 3. The browser of the participant 2 connects to the server using Flash and sends the join command.
- 4. The browser of the participant 2 can send a stream via RTMP to publish it in the chat room and receive streams published in the room.

Quick testing

Video conference testing

- 1. For the test we use:
- 2. the demo server at demo.flashphoner.com
- 3. the Conference web application to arrange a video conference.
- 4. Open the Conference web application. In the Login field enter any arbitrary user name, for example test:

	Confer	ence
WCS URL	wss://p11.flashphoner.com:84-	
Login	test	Join
	NONE	NONE

5. Click the Join button. A connection with the server is established, and you should see the corresponding **ESTABLISHED** label. The chat room is automatically created:

WCS URL	wss://p11.flashphoner.com:84-			
Login				
Login	test	Leave	ESTABLISHED	
	1			
	NONE	9	NONE	

In the bottom of the screen, an image from the web camera, a text chat and a link to invite users to the room are shown:

PUBLISHING	
Mute A Mute V Stop	
10:29 chat - room is empty 10:30 test - test message	*
	-
Invite	Send
https://p11.flashphoner.com:8888/client2/examples/demo/streaming/conference/conference.html? roomName=room-72be4a	

6. Copy the link to the chat room and open it in a new tab of the browser. Enter a user name other than the name of the chat room creator, for example, test2, and click the Join button. The page will display an image from the web camera of the test participant



	Confe	rence
WCS URL	wss://p11.flashphoner.com:8443	
Login	test2	Leave
	test	NONE

7. In the text chat window of the test2 participant enter a message and click Send:

	PUBLISHING	
	Mute A Mute V Stop	
10:31 chat - participants: test 10:32 test2 - this is a test, do you see me?		*
		-
		Send

8. On the browser tab of the test participant enter an answer:

PUBLISHING	
Mute A Mute V	
10:29 cnat - room is empty 10:30 test - test message 10:31 test2 - joined 10:32 test2 - this is a test, do you see me? 10:34 test - "zee val zee val", Rabbit Eggs shout	Ì
	Send

9. Make sure the answer is received:

	PUBLI	SHING	- de		
	Mute A	Mute V			
	St	top			
10:31 chat - participants: test 10:32 test2 - this is a test, do you see me?					*
10:34 test - "zee ya! zee ya!", Rabbit Eggs shout					
					*
					Send

10. To leave the chat room, click the Leave button

Video chat testing

- 1. For the test we use:
- 2. the demo server at demo.flashphoner.com

- 3. the Two Way Video Chat web application to arrange a video chat
- 4. Open the Two Way Video Chat web application. In the Login field enter any arbitrary user name, for example test:

	Two Way	Video Ch	nat	
WCS URL	wss://p11.flashphone			
Login	test		Join	

5. Click the Join button. A connection is established to the server, and the corresponding **ESTABLISHED** label is shown. The chat room is automatically created:

Login	test		Leave	
		EST	ABLISHED	
	A sector			
		NONE		

A
-
Send

Below the screen, a text chat and a link to invite other users to the room are shown:

6. Copy the link to the chat room and open it in a new tab of the browser. Enter a user name other than that of the creator of the room, for example, test2, and click the Join button. The page will display a large image from the web camera of the test user and a





7. In the text chat box, enter a message and click the Send button:

Stop PUBLISHING	
10:42 chat - participants: test	

8. On the tab of the test user enter an answer:

and the second second second	
Stop PUBLISHING	
10:40 chat - room is empt	/
10:42 test2 - juilleu	do vou see me?
10.40 (63(2 - (113)))) 0.50	
"zee ya! zee ya!", Rabbit	∃ggs shout∣

9. Make sure the answer is received:

Stop PUBLISHING	
10:42 chat - participants: test	
10:43 test2 - this ih a test, do you see me?	
10:44 test - "zee ya! zee ya!", Rabbit Eggs shout	
10:44 test - "Zee ya! zee ya!", Rabbit Eggs shout	
10:44 test - "Zee ya! zee ya!", Rabbit Eggs shout	
10:44 test - "zee yal zee yal", Rabbit Eggs shout	,
10:44 test - "zee yal zee yal", Rabbit Eggs shout	,
10:44 test - "zee ya! zee ya!", Rabbit Eggs shout	

10. To leave the chat room, click the Leave button

Video conference with screen sharing testing

- 1. For the test we use:
- 2. the demo server at demo.flashphoner.com
- 3. the Two Way Video Chat and Screen Sharing web application to organize a video conference
- 4. the Chrome browser
- 5. Open the Two Way Video Chat and Screen Sharing web application. In the Login field enter any arbitrary user name, for example test. Click the Join button. A connection is established to the server, and the corresponding ESTABLISHED label is displayed. The



chat room is created automatically, and an image from the web camera is shown:

6. Copy the link to the chat room and open it in a new tab of the browser. Enter a user name different from the name of the chat room's creator, for example test2, and click the Join button. The page displays an image from the web camera:



7. Click the **Share** button and allow the browser to gain access to your screen or to the application window:

test	FPS	Width	Height
Mute A Mute V Stop PUBLISHING	30	640	480
10.51 chat - participants: test		Unshare	

8. On the tab of the **test** user you should see the screen or the app window you allowed the browser to access:



9. To leave the chat room, click the Leave button

Call Flow

Below is the call flow when using the Conference example.

conference.html

conference.js



1. Participant 1 establishes a connection to the server RoomApi.connect() code

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

2. Participant 1 receives from the server an event confirming successful connection SESSION_STATUS.ESTABLISHED code



3. Participant 1 joins the chat room

Session.join() code



4. Participant 1 receives from the server an event describing the state of the room **ROOM_EVENT.STATE** code

```
connection.join({name: getRoomName()}).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);
  setInviteAddress(room.name());
  if (participants.length > 0) {
     var chatState = "participants: ";
      for (var i = 0; i < participants.length; i++) {</pre>
          installParticipant(participants[i]);
          chatState += participants[i].name();
          if (i != participants.length - 1) {
              chatState += ",";
          }
      addMessage("chat", chatState);
      addMessage("chat", " room is empty");
  publishLocalMedia(room);
  onJoined(room);
```

5. Participant 1 publishes the media stream

Room.publish() code



6. Participant 1 receives from the server an event confirming successful publishing of the stream

STREAM_STATUS.PUBLISHING code

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

- 7. Participant 1 sends the stream via WebRTC
- 8. Participant 2 establishes a connection to the server
- 9. Participant 2 receives from the server an event confirming successful connection
- 10. Participant 2 enters the chat room
- 11. Participant 2 receives from the server an event describing the state of the room
- 12. Participant 1 receives from the server an event informing that participant 2 has joined ROOM_EVENT.JOINED code

```
connection.join({name: getRoomName()}).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){
```

});

- 13. Participant 2 receives the stream published by participant 1
- 14. Participant 2 publishes the media stream
- 15. Participant 2 receives from the server an event confirming successful publishing of the stream
- 16. Participant 2 sends the stream via WebRTC, participant 1 receives this stream
- 17. Participant 1 leaves the chat room

Room.leave() code



18. Participants of the room receive from the server an event informing that participant 1 has left the room

ROOM_EVENT.LEFT code

```
connection.join({name: getRoomName()}).on(ROOM_EVENT.STATE, function(room)
{
    ...
}).on(ROOM_EVENT.JOINED, function(participant){
    ...
}).on(ROOM_EVENT.LEFT, function(participant){
    //remove participant
    removeParticipant(participant);
    addMessage(participant.name(), "left");
}).on(ROOM_EVENT.PUBLISHED, function(participant){
    ...
}).on(ROOM_EVENT.FAILED, function(room, info){
    ...
}).on(ROOM_EVENT.MESSAGE, function(message){
    ...
});
```

How to record streams published by room participants

Video streams published by room participants may be recorded. To do this, record parameter must be set to true while publishing a stream:



```
constraints: constraints,
record: true,
...
);
```

A stream from any participant is recorded to a separate file. The issue of record files futher processing is that they starts not at the same time.

Stream records synchronization

🛕 Warning

The feature is not supported sinsce build 5.2.142. Use stream mixer or multirecording instead.

To allow streams merging, room streams may be synchronized by the first stream published. To enable this feature set the following parameter in flashphoner.properties file

enable_empty_shift_writer=true

For example, if User1 participant started publishing stream at 00:00:10, and User2 participant did it at 00:00:55, then second participant will get 45 seconds of empty video (black screen and silence) at record beginning. So, stream record files User1.mp4 and User2.mp4 will be same in duration, and they can be merged.

Merging synchronized stream records using ffmpeg

Synchronized stream record files can be merged in chronological order using ffmpeg. To allow this, when stream is created, stream timeshift relative to room creation time is specified on server side. Stream record files written by this way are merged with command (two participants example)

```
ffmpeg -i stream1.mp4 -i stream2.mp4 -filter_complex "[0:v]pad=iw*2:ih[int];
[int][1:v]overlay=W/2:0[vid];[0:a][1:a]amerge[a]" -map [vid] -map "[a]" -ac 2
-strict -2 -c:v libx264 -crf 23 -preset veryfast output.mp4
```

Where

- stream1 first participant stream
- stream2 second participant stream

Room multiple streams recording to one file with subsequent mixing

Since WCS build 5.2.1012 and WebSDK build 2.0.190 it is possible to record all the room streams to one file, with automatic mixing after finishing a conference. To do this, a first participant should set room record option while creating a room:

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

In this case, all the room streams will be recorded to one file. When the room is finalized, the recording will also be stopped, and the script set in the following parameter will be automatically launched

on_multiple_record_hook_script=on_multiple_record_hook.sh

The script will mix multiple streams recorded acoording to mixer settings defined in /usr/local/FlashphonerWebCallServer/conf/offline_mixer.json file, by default



Room recording testing

- 1. For test we use:
- 2. WCS server, for example test1.flashphoner.com
- 3. Conference web application example
- 4. Open Conference example in browser, enter participant name Alice and set Record checkbox



5. Click Join. Stream publishing will start



6. Open Invite link in another browser window

Total Tooline of			
			Seno

7. Enter participant name Bob and click Join

	Co	onference
WCS URL	wss://test1.flashphoner.com:8443	
Login	Bob	Join
L	NONE	NONE

8. Bob joined to the room

	Confe	erence	
WCSURL	wss://test1.flashphoner.com:8443		
Login	Alice	Leave Record ESTABLISHED	
	Bob	NONE	
	PUBL Mute A	ISHING Mute V Stop	

		Confe	rence		
WCS URL	wss://test1.flashph	oner.com:8443			
Login	Alice		Join	Record ESTABLISHED	
	NONE			NONE	
		UNPUBL Mute A	ISHED Mute V		
		mate A	mato v		

9. Click Leave in Alice participant window

		Confe	rence		
WCS URL	wss://test1.flashpho	pner.com:8443			
Login	Bob		Join	ESTABLISHED	
	NONE		<u></u>	NONE	
		UNPUBL	ISHED		
		Mute A	Mute V		
		Pub	lish		

10. Recording file mixing may take a long time depending on recording length, CPU and dick I/O performance. When mixing is done, download the file from



Room finalizing

Room exists until at least one participant is connected. When the lastparticipant calls Room.leave() function, the room is finalized.

If the last participant refreshes web page or loses server connection without calling Room.leave(), the room will be active during the time interval set by the following parameter in milliseconds

room_idle_timeout=60000

By default, the interval is 60 seconds. When time is expired, the room is finalized.

Known issues

1. Non-latin characters should be encoded while messaging

🗿 Symptoms

When message sent contains non-latin characters, they are replaced with questionmarks on receiving end

✓ Solution
Use JavaScript functions encodeURIComponent() while sending a message
<pre>var participants = room.getParticipants(); for (var i = 0; i < participants.length; i++) { participants[i].sendMessage(encodeURIComponent(message)); }</pre>
and decodeURIComponent() while receiving message
<pre>connection.join({name: getRoomName(), record: isRecord()}).on(ROOM_EVENT.STATE, function(room) { }).on(ROOM_EVENT.MESSAGE, function(message){ addMessage(message.from.name(), decodeURIComponent(message.text)); });</pre>

2. A race condition may occur when Room.leave() and then Session.join() are subsequently called too fast

When <u>Session.join()</u> and then <u>Room.leave()</u> are called too fast, it is possible to send join command to the server while it still handles previous leave command for this user

6 Symptoms
When Session.join() is called right after Room.leave(), client may receive a message
Room already has user with such login

Solution

Use at least 1 second interval between Room.leave() and Session.join() subsequent calls