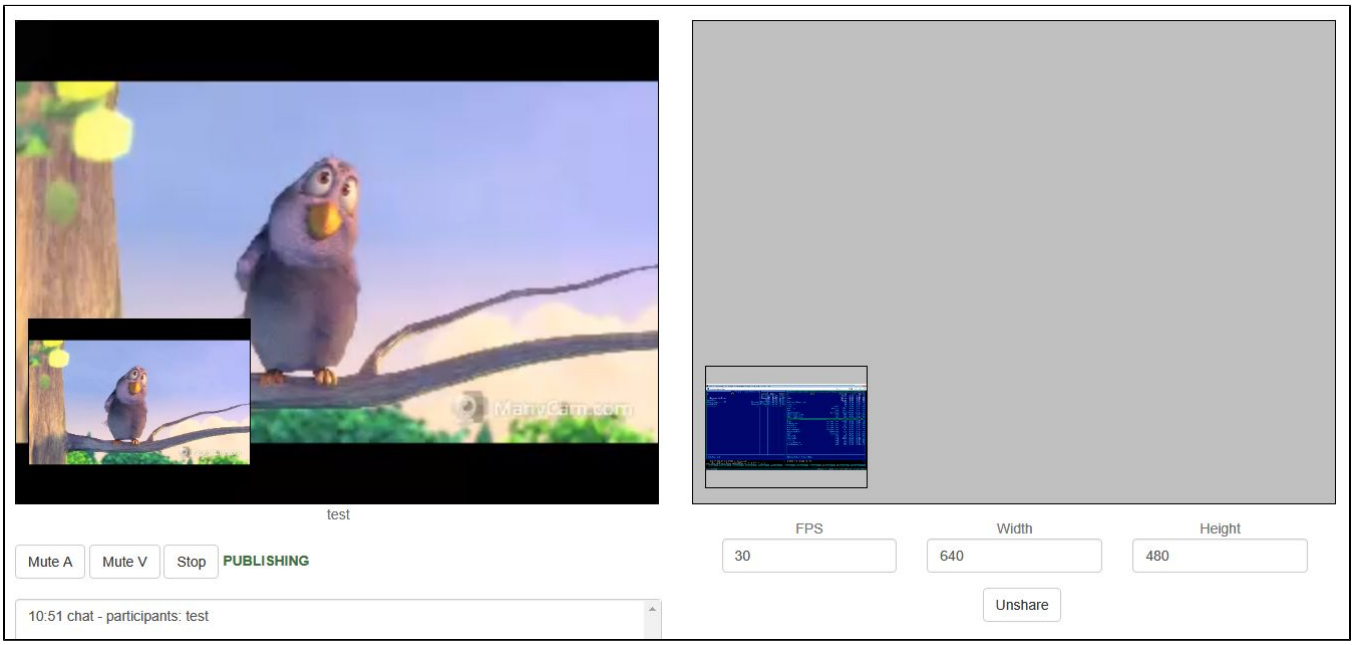
Video Chat and Screen Sharing

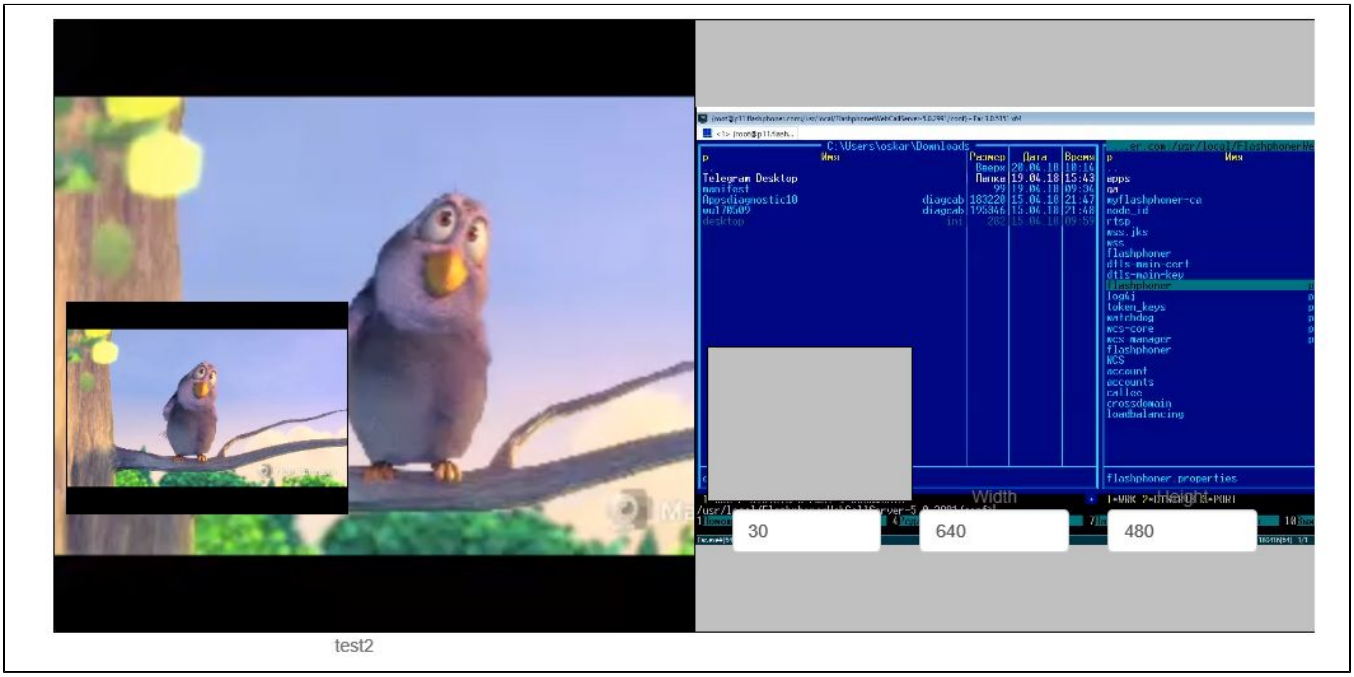
The example of video chat with screen sharing

The example may be used for video chat between two participants with sharing screen of one of them. The chat participant can publish WebRTC stream from web camera and, simultaneously, WebRTC stream from screen or application window.

Example of client streaming its' screen in Chrome browser window:



Example of client playing screen stream in Chrome browser window:



The code of the example

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

/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/video-chat-and-screen-sharing/

video-chat.css - CSS style file
video-chat-and-screen-sharing.html - chat participant page
video-chat-and-screen-sharing.js - script for chat to work

The example can be tested at this URL:

https://host:8888/client2/examples/demo/streaming/video-chat-and-screen-sharing/video-chat-and-screen-sharing.html

where host is your WCS server address.

Analyzing the code

To analyze the code get video-chat-and-screen-sharing.js file version with hash 90771d4 that can be found here and is available to download in build 2.0.218.

1. API initializing.

Flashphoner.init() code

```
    try {
        Flashphoner.init();
    } catch(e) {
        $("#notifyFlash").text("Your browser doesn't support WebRTC technology needed for this example");
        return;
    }
```

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 the 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 to the room

connection.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()}).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"
- otherwise, the user starts publishing video stream

```
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);
         return false;
     }
     room_ = room;
     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. Screen streaming

room.publish() code

These parameters are passed to room.publish() method:

- video constraints: width, height, FPS, source (screen)
- page element to display preview

```
var constraints = {
    video: {
        width: parseInt($('#width').val()),
        height: parseInt($('#height').val()),
        frameRate: parseInt($('#fps').val()),
        withoutExtension: true
    },
    audio: $("#useMic").prop('checked')
};
constraints.video.type = "screen";
if (Browser.isFirefox()){
    constraints.video.mediaSource = "screen";
}
var options = {
    name: "screenShare",
    display: document.getElementById("preview"),
    constraints: constraints,
    cacheLocalResources: false
}
if (isSafariMacOS()) {
    options.disableConstraintsNormalization = true;
}
room.publish(options).on(STREAM_STATUS.FAILED, function (stream) {
    ...
});
```

8. Receiving the event notifying that other participant joined to the room

RoomStatusEvent 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){
    ...
});
```

9. Receiving the event notifying that other participant published video stream

RoomStatusEvent PUBLISHED code

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

10. Playback of video stream.

participant.play() code

 element, in which the video will be displayed depending on source (web camera or screen), is passed to the participant.play() method.

```
function playParticipantsStream(participant) {
    if (participant.getStreams().length > 0) {
        for (var i=0; i<participant.getStreams().length; i++) {
            $("[id$=Name]").each(function (index, value) {
                if ($(value).text() == participant.name()) {
                    var p = value.id.replace('Name', '');
                    var pDisplay = p + 'Display';
                    // check if we already play this stream
                    if (document.getElementById(participant.getStreams()[i].id()) == null) {
                        // setup 1st stream to main div
                        if (participant.getStreams()[i].streamName().indexOf("screenShare") == -1) {
                            participant.getStreams()[i].play(document.getElementById(pDisplay)).on
(STREAM_STATUS.PLAYING, function (playingStream) {
                                document.getElementById(playingStream.id()).addEventListener('resize', function
(event) {
                                    resizeVideo(event.target);
                                });
                            });
                        } else {
                            participant.getStreams()[i].play(document.getElementById("sharedDisplay")).on
(STREAM_STATUS.PLAYING, function (playingStream) {
                                document.getElementById(playingStream.id()).addEventListener('resize', function
(event) {
                                    resizeVideo(event.target);
                                });
                            });
                        }
                    }
                }
            });
        }
    }
}
```

11. Stop of screen sharing.

stream.stop() code

```
    room.publish(options).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        /*
         * User can stop sharing screen capture using Chrome "stop" button.
         * Catch onended video track event and stop publishing.
         */
        document.getElementById(stream.id()).srcObject.getVideoTracks()[0].onended = function (e) {
            stream.stop();
        };
        ...
    }).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
        ...
    });
```

12. Receiving the event confirming successful screen sharing stop.

StreamStatusEvent UNPUBLISHED code

```
    room.publish(options).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        ...
    }).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
        onStopSharing();
    });
```

13. Leaving chat 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);
    ...
}
```