

Flash Video Chat

Example of a two-directional video chat in a native Flash / Flex application

This example is a two-directional video chat using a client Flash application executed as a simple swf file. The example demonstrates operation of a Flash video chat that allows two-directional video communication to the same example for [iOS](#), [Android](#) or [Web SDK](#).

The screenshot displays operation of the Flash video chat.

WCS URL



rtmfp://192.168.1.59:1935

Login

22

Leave

ESTABLISHED



222

Publish

UNPUBLISHED

11:56 chat - participants: 222

Send

Invite

http://192.168.1.59:9091/client2/examples/demo/streaming/flash_client/chat.html?roomName=53F89D

The interface contains input fields to authorize in the video chat:

- WCS server address
- user name (any unique one will do for the sake of testing)

Below the windows, there is a simple text chat to exchange messages.

The `Invite` box contains a link to send to the second participant of the chat to invite him or her.

Example files

This example is a compiled SWF file embedded to an HTML page using Flex / ActionScript3 and MXML available at:

/usr/local/FlashphonerWebCallServer/client/examples/demo/streaming/flash_client/chat.html

- chat.html - example page
- chat/bin-debug/chat.swf - application file

Working with the source code of the example

To analyze the code, let's take this version of the `chat.mxml` file with the hash `8b4baf2766e0a1b485c41a8c64da80c74070ff1`, located [here](#). The result of compiling the `chat.mxml` file is the example application `chat.swf`. The compiled swf and the source code are available for download in the corresponding build [0.5.3.1894](#).

The main example file `chat.mxml` is based on several files implementing ROOM API identical to [the room-module.js for Web SDK](#) implementation.

- Participant.as - an object describing a participant of the video chat
- RestAppCpmmunicator - an object responsible for sending sendData to the WCS server and receiving inbound messages
- Room.as - an object describing the "room" where all participants of the chat are
- RoomStatus.as - room statuses
- Session.as - an object describing connection to the server
- SessionStatus.as - session statuses

1. Accessing camera and microphone

During initialization the application gets access to the camera and the microphone

[line 65](#)

```
cam = Camera.getCamera();
localDisplay.attachCamera(cam);
mic = Microphone.getEnhancedMicrophone();
remoteDisplayHolder.addChild(remoteDisplay);
```

2. Connecting to the server

Then, we create a Session object and connect to the WCS server [line 144](#)

If connection to the server is successful, the joinRoom() method is invoked to join to the room.

```
session = new Session(url, username);
session.on(SessionStatus.FAILED, function():void{
    setStatus(sessionStatus, SessionStatus.FAILED);
    onLeft();
}).on(SessionStatus.DISCONNECTED, function():void {
    setStatus(sessionStatus, SessionStatus.DISCONNECTED);
    onLeft();
}).on(SessionStatus.ESTABLISHED, function():void {
    setStatus(sessionStatus, SessionStatus.ESTABLISHED);
    joinRoom();
});
session.connect();
```

3. Events handling

When the participant joins the room, the reactions to various events in the room are added

[line 150](#)

- **JOINED** - a new participant has joined the room
- **LEFT** - a participant has left the room
- **PUBLISHED** - a participant has published a video stream
- **FAILED** - an error occurred while communicating with the room
- **MESSAGE** - an inbound message from a participant in the room

```
session.join(this.roomName).on(RoomStatus.STATE, function(room:Room):void{
    var participants:Array = room.getParticipants();
    setInviteAddress(room);
    if (participants.length > 0) {
        var chatState:String = "participants: ";
        for (var i:Number = 0; i < participants.length; i++) {
            installParticipant(participants[i]);
            chatState += participants[i].getName();
            if (i != participants.length - 1) {
                chatState += ",";
            }
        }
        addMessage("chat", chatState);
    } else {
        addMessage("chat", " room is empty");
    }
    publishLocalMedia(room);
    onJoined(room);
}).on(RoomStatus.JOINED, function(participant:Participant):void{
```

```

        installParticipant(participant);
        addMessage(participant.getName(), "joined");
    }).on(RoomStatus.LEFT, function(participant:Participant):void{
        removeParticipant();
        addMessage(participant.getName(), "left");
    }).on(RoomStatus.PUBLISHED, function(participant:Participant):void{
        playParticipantsStream(participant);
    }).on(RoomStatus.FAILED, function(room:Room, info:Object):void{
        failedInfo.text = info.info;
        session.disconnect();
    }).on(RoomStatus.MESSAGE, function(message:Object):void{
        addMessage(message.from.getName(), message.text);
    });

```

4. Stream publishing

Publishing the video stream from the web camera to the WCS server

line 232

```

private function publishLocalMedia(room:Room):void {
    var stream:NetStream = room.publish(mic, cam);
    stream.addEventListener(NetStatusEvent.NET_STATUS,
function(event:NetStatusEvent):void{
    Logger.info("handlePublishStreamStatus: "+event.info.code);
    switch (event.info.code) {
        case "NetStream.Publish.BadName":
            setStatus(streamStatus, "FAILED");
            onMediaStopped(room);
            break;
        case "NetStream.Unpublish.Success":
            setStatus(streamStatus, "UNPUBLISHED");
            onMediaStopped(room);
            break;
        case "NetStream.Publish.Start":
            setStatus(streamStatus, "PUBLISHING");
            onMediaPublished(stream);
            break;
    }
    });
}

```

5. Stream playback

Playing the stream of another participant

line 207

```

private function playParticipantsStream(p:Participant):void
{
    var stream:NetStream = p.play();
    if (stream != null) {
        remoteDisplay.attachNetStream(stream);
    }
}

```

```
        stream.addListener(NetStatusEvent.NET_STATUS,
function(event:NetStatusEvent):void{
    Logger.info("handlePlayStreamStatus: "+event.info.code);
    switch (event.info.code) {
        case "NetStream.Video.DimensionChange":
            var res:Object =
downScaleToFitSize(remoteDisplay.videoWidth, remoteDisplay.videoHeight,
display.width, display.height);
            remoteDisplay.width = res.w;
            remoteDisplay.height = res.h;
            remoteDisplayHolder.width = res.w;
            remoteDisplayHolder.height = res.h;
            break;
        case "NetStream.Play.UnpublishNotify":
        case "NetStream.Play.Stop":
            remoteDisplay.clear();
            break;
    }
});
    }
}
```