

SFU Two Way Streaming 2.0.248 and later

- [Example source code](#)
- [Analyzing the code](#)
 - [1. Local variables](#)
 - [2. Default configuration](#)
 - [3. Object to store current publishing/playback state](#)
 - [4. Initialization](#)
 - [5. Establishing server connection](#)
 - [6. Publishing or playback start after session establishing](#)
 - [7. Streams publishing](#)
 - [7.1. Media tracks addition to WebRTC connection](#)
 - [7.2. Tracks onended event subscription](#)
 - [8. Streams playback](#)
 - [9. Publishing stopping](#)
 - [10. Publish/Play click action](#)
 - [11. Stop click actions](#)
 - [12. Websocket session disconnection actions](#)
 - [13. Helper functions](#)
 - [13.1. Start publishing or playback](#)
 - [13.2. Stop publishing or playback](#)

The example shows how to publish and play a number of streams in one WebRTC connection with simulcast. A room is considered to be a publishing unit, that is, viewers who connect to this room receive all the streams published in it.

On the screenshots below:

- [Server url](#) - Websocket URL of WCS server
- [Room name](#) - room name
- [Publisher](#) - publisher user name

SFU Two-way Streaming

Server url

wss://test1.flashphoner.com:8443

Room name

ROOM1-16cc

Publisher

Publisher1-2b12

Stop

ESTABLISHED

Publisher1-2b12 cam1 1280x720

Mute mic1



- Player - viewer nickname
- 180p send, 360p send, 720p send - quality switch buttons
- Track - video tracks switch button if there are more than one

Player

Player1-4db7
Stop

ESTABLISHED

Meeting: ROOM1-16cc

Name: Publisher1-2b12#d702


320x180

Current video track: 0

mute

180p send
360p send
720p send

Track №0: cam1



Note that audio tracks are playing in a separate `audio` tags.

Example source code

The source code consists of the following modules:

- two-way-streaming.html - HTML page
- two-way-streaming.css - HTML page styles
- two-way-streaming.js - main application logic
- config.json - client configuration file, contains streams publishing description

Analyzing the code

To analyze the example source code, take the file `two-way-streaming.js` version available [here](#)

1. Local variables

Local variables declaration to work with constants, SFU SDK, to display video and to work with client configuration

[code](#)

```
const constants = SFU.constants;
const sfu = SFU;
let mainConfig;
let localDisplay;
let remoteDisplay;
let publishState;
let playState;
const PUBLISH = "publish";
const PLAY = "play";
const STOP = "stop";
const PRELOADER_URL = "../commons/media/silence.mp3"
```

2. Default configuration

Default room configuration and stream publishing configuration to use if there is no config.jsonfile found

[code](#)

```
const defaultConfig = {
  room: {
    url: "wss://127.0.0.1:8888",
    name: "ROOM1",
    pin: "1234",
    nickName: "User1",
    failedProbesThreshold: 5,
    pingInterval: 5000
  },
  media: {
    audio: {
      tracks: [
        {
          source: "mic",
          channels: 1
        }
      ]
    },
    video: {
      tracks: Array(1).fill({
        source: "camera",
        width: 1280,
        height: 720,
        codec: "H264",
        constraints: {
          frameRate: 25
        }
      }),
      encodings: [
        {rid: "180p", active: true, maxBitrate: 200000, scaleResolutionDownBy: 4},
        {rid: "360p", active: true, maxBitrate: 500000, scaleResolutionDownBy: 2},
        {rid: "720p", active: true, maxBitrate: 900000}
      ],
      type: "cam1"
    })
  }
};
```

3. Object to store current publishing/playback state

The object should keep Websocket session data, WebRTC connection data and room data, and should form HTML tags ids to access them from code

[code](#)

```

const CurrentState = function (prefix) {
  let state = {
    prefix: prefix,
    pc: null,
    session: null,
    room: null,
    display: null,
    roomEnded: false,
    starting: false,
    set: function (pc, session, room) {
      state.pc = pc;
      state.session = session;
      state.room = room;
      state.roomEnded = false;
    },
    clear: function () {
      state.room = null;
      state.session = null;
      state.pc = null;
      state.roomEnded = false;
    },
    setRoomEnded: function () {
      state.roomEnded = true;
    },
    buttonId: function () {
      return state.prefix + "Btn";
    },
    buttonText: function () {
      return (state.prefix.charAt(0).toUpperCase() + state.prefix.slice(1));
    },
    inputId: function () {
      return state.prefix + "Name";
    },
    statusId: function () {
      return state.prefix + "Status";
    },
    formId: function () {
      return state.prefix + "Form";
    },
    errInfoId: function () {
      return state.prefix + "ErrorInfo";
    },
    is: function (value) {
      return (prefix === value);
    },
    isActive: function () {
      return (state.room && !state.roomEnded && state.pc);
    },
    isConnected: function () {
      return (state.session && state.session.state() === constants.SFU_STATE.CONNECTED);
    },
    isRoomEnded: function () {
      return state.roomEnded;
    },
    setStarting: function (value) {
      state.starting = value;
    },
    isStarting: function () {
      return state.starting;
    },
    setDisplay: function (display) {
      state.display = display;
    },
    disposeDisplay: function () {
      if (state.display) {
        state.display.stop();
        state.display = null;
      }
    }
  }
};

```

```
    return state;
  }
}
```

4. Initialization

`init()`[code](#)

The `init()` function is called on page load and:

- initializes state objects
- reads `config.json` file or default configuration
- initializes input fields

```
const init = function () {
  let configName = getUrlParam("config") || "./config.json";
  $("#publishBtn").prop('disabled', true);
  $("#playBtn").prop('disabled', true);
  $("#url").prop('disabled', true);
  $("#roomName").prop('disabled', true);
  $("#publishName").prop('disabled', true);
  $("#playName").prop('disabled', true);
  publishState = CurrentState(PUBLISH);
  playState = CurrentState(PLAY);
  $.getJSON(configName, function (cfg) {
    mainConfig = cfg;
    onDisconnected(publishState);
    onDisconnected(playState);
  }).fail(function (e) {
    //use default config
    console.error("Error reading configuration file " + configName + ": " + e.status + " " + e.statusText);
    console.log("Default config will be used");
    mainConfig = defaultConfig;
    onDisconnected(publishState);
    onDisconnected(playState);
  });
  $("#url").val(setURL());
  $("#roomName").val("ROOM1-" + createUUID(4));
  $("#publishName").val("Publisher1-" + createUUID(4));
  $("#playName").val("Player1-" + createUUID(4));
}
```

5. Establishing server connection

`connect()`, `SFU.createRoom()` [code](#)

The `connect()` function is called by Publish or Play click:

- creates `PeerConnection` object
- cleans previous session state displayed
- sets up room configuration and creates Websocket session
- subscribes to Websocket session events

```

const connect = async function (state) {
  //create peer connection
  let pc = new RTCPeerConnection();
  //get config object for room creation
  const roomConfig = getRoomConfig(mainConfig);
  roomConfig.url = $("#url").val();
  roomConfig.roomName = $("#roomName").val();
  roomConfig.nickname = $("#" + state.inputId()).val();
  // clean state display items
  setStatus(state.statusId(), "");
  setStatus(state.errInfoId(), "");
  // connect to server and create a room if not
  try {
    const session = await sfu.createRoom(roomConfig);
    // Set up session ending events
    session.on(constants.SFU_EVENT.DISCONNECTED, function () {
      onStopClick(state);
      onDisconnected(state);
      setStatus(state.statusId(), "DISCONNECTED", "green");
    }).on(constants.SFU_EVENT.FAILED, function (e) {
      onStopClick(state);
      onDisconnected(state);
      setStatus(state.statusId(), "FAILED", "red");
      if (e.status && e.statusText) {
        setStatus(state.errInfoId(), e.status + " " + e.statusText, "red");
      } else if (e.type && e.info) {
        setStatus(state.errInfoId(), e.type + ": " + e.info, "red");
      }
    });
    // Connected successfully
    onConnected(state, pc, session);
    setStatus(state.statusId(), "ESTABLISHED", "green");
  } catch (e) {
    onDisconnected(state);
    setStatus(state.statusId(), "FAILED", "red");
    setStatus(state.errInfoId(), e, "red");
  }
}

```

6. Publishing or playback start after session establishing

onConnected() [code](#)

The onConnected() function:

- sets up Stop button click actions
- subscribes to room error events
- calls publishing or playback function

```

const onConnected = function (state, pc, session) {
  state.set(pc, session, session.room());
  $("# + state.buttonId()).text("Stop").off('click').click(function () {
    onStopClick(state);
  });
  $('#url').prop('disabled', true);
  $("#roomName").prop('disabled', true);
  $("# + state.inputId()).prop('disabled', true);
  // Add errors displaying
  state.room.on(constants.SFU_ROOM_EVENT.FAILED, function (e) {
    setStatus(state.errInfoId(), e, "red");
    state.setRoomEnded();
    onStopClick(state);
  }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
    onOperationFailed(state, e);
  }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
    setStatus(state.errInfoId(), "Room " + state.room.name() + " has ended", "red");
    state.setRoomEnded();
    onStopClick(state);
  }).on(constants.SFU_ROOM_EVENT.DROPPED, function () {
    setStatus(state.errInfoId(), "Dropped from the room " + state.room.name() + " due to network issues",
"red");
    state.setRoomEnded();
    onStopClick(state);
  });
  startStreaming(state);
}

```

7. Streams publishing

`publishStreams()`, `SFURoom.join()` [code](#)

The `publishStreams()` function:

- initializes a basic HTML container tag to display local video
- gets local media access according to configuration file
- adds media tracks to WebRTC connection
- joins the room on server


```

const publishStreams = async function (state) {
  if (state.isConnected()) {
    //create local display item to show local streams
    const display = initLocalDisplay(document.getElementById("localVideo"));
    state.setDisplay(display);
    try {
      //get configured local video streams
      let streams = await getVideoStreams(mainConfig);
      let audioStreams = await getAudioStreams(mainConfig);
      if (state.isConnected() && state.isActive()) {
        //combine local video streams with audio streams
        streams.push.apply(streams, audioStreams);
        let config = {};
        //add our local streams to the room (to PeerConnection)
        streams.forEach(function (s) {
          let contentType = s.type || s.source;
          //add local stream to local display
          display.add(s.stream.id, $("#" + state.inputId()).val(), s.stream, contentType);
          //add each track to PeerConnection
          s.stream.getTracks().forEach((track) => {
            config[track.id] = contentType;
            addTrackToPeerConnection(state.pc, s.stream, track, s.encodings);
            subscribeTrackToEndedEvent(state.room, track, state.pc);
          });
        });
        //start WebRTC negotiation
        await state.room.join(state.pc, null, config);
      }
    } catch (e) {
      if (e.type === constants.SFU_ROOM_EVENT.OPERATION_FAILED) {
        onOperationFailed(state, e);
      } else {
        console.error("Failed to capture streams: " + e);
        setStatus(state.errInfoId(), e.name, "red");
        onStopClick(state);
      }
    }
  }
}

```

7.1. Media tracks addition to WebRTC connection

`addTrackToPeerConnection()`, `PeerConnection.addTransceiver()` [code](#)

```

const addTrackToPeerConnection = function(pc, stream, track, encodings) {
  pc.addTransceiver(track, {
    direction: "sendonly",
    streams: [stream],
    sendEncodings: encodings ? encodings : [] //passing encoding types for video simulcast tracks
  });
}

```

7.2. Tracks onended event subscription

`subscribeTrackToEndedEvent()`, `MediaTrack.addEventListener()`, `SFURoom.updateState()` [code](#)

```
const subscribeTrackToEndedEvent = function (room, track, pc) {
  track.addEventListener("ended", async function () {
    //track ended, see if we need to cleanup
    let negotiate = false;
    for (const sender of pc.getSenders()) {
      if (sender.track === track) {
        pc.removeTrack(sender);
        //track found, set renegotiation flag
        negotiate = true;
        break;
      }
    }
    if (negotiate) {
      //kickoff renegotiation
      await room.updateState();
    }
  });
};
```

8. Streams playback

`playStreams()`, `SFURoom.join()` [code](#)

The `playStreams()` function:

- initializes a base container tag to display incoming media streams
- joins to the room on server

```
const playStreams = async function (state) {
  if (state.isConnected() && state.isActive()) {
    try {
      //create remote display item to show remote streams
      const display = initDefaultRemoteDisplay(state.room, document.getElementById("remoteVideo"), null,
null);
      state.setDisplay(display);
      //start WebRTC negotiation
      await state.room.join(state.pc, null, null, 1);
    } catch (e) {
      if (e.type === constants.SFU_ROOM_EVENT.OPERATION_FAILED) {
        onOperationFailed(state, e);
      } else {
        console.error("Failed to play streams: " + e);
        setStatus(state.errInfoId(), e.name, "red");
        onStopClick(state);
      }
    }
  }
}
```

9. Publishing stopping

`state.disposeDisplay()` [code](#)

```
const disposeStateDisplay = function (state) {
  state.disposeDisplay();
}
```

10. Publish/Play click action

`onStartClick()`, `playFirstSound()`, `connect()` [code](#)

The `onStartClick()` function:

- validates input fields

- in Safari browser, calls `playFirstSound()` before playback to automatically play incoming audio
- calls `connect()` function

```
const onStartClick = function (state) {
  if (validateForm("connectionForm", state.errInfoId())
    && validateForm(state.formId(), state.errInfoId())
    && validateName(state, state.errInfoId())) {
    state.setStarting(true);
    let otherState = getOtherState(state);
    $("##" + state.buttonId()).prop('disabled', true);
    // Disable other session button to prevent a simultaneous connections
    if (!otherState.isStarting()) {
      $("##" + otherState.buttonId()).prop('disabled', true);
    }
    if (state.is(PLAY) && Browser().isSafariWebRTC()) {
      playFirstSound(document.getElementById("main"), PRELOADER_URL).then(function () {
        connect(state);
      });
    } else {
      connect(state);
    }
  }
}
```

11. Stop click actions

`onStopClick()`, `Session.disconnect()` [code](#)

The `onStopClick()` function:

- stops playback or publishing
- disconnects Websocket session

```
const onStopClick = async function (state) {
  state.setStarting(false);
  disposeStateDisplay(state);
  if (state.isConnected()) {
    $("##" + state.buttonId()).prop('disabled', true);
    await state.session.disconnect();
    onDisconnected(state);
  }
}
```

12. Websocket session disconnection actions

`onDisconnected()` [code](#)

The `onDisconnected()` functions:

- sets up Publish/Play click actions
- enables Server url and Room name fields access, if there's no parallel session

```

const onDisconnected = function (state) {
  state.clear();
  $("#" + state.buttonId()).text(state.buttonText()).off('click').click(function () {
    onStartClick(state);
  }).prop('disabled', false);
  $("#" + state.inputId()).prop('disabled', false);
  // Enable other session buttons
  let otherState = getOtherState(state);
  if (!otherState.session) {
    $("#" + otherState.buttonId()).prop('disabled', false);
    $("#" + otherState.inputId()).prop('disabled', false);
    $('#url').prop('disabled', false);
    $("#roomName").prop('disabled', false);
  }
}

```

13. Helper functions

13.1.Start publishing or playback

startStreaming() [code](#)

```

const startStreaming = function(state) {
  if (state.is(PUBLISH)) {
    publishStreams(state);
  } else if (state.is(PLAY)) {
    playStreams(state);
  }
}

```

13.2.Stop publishing or playback

state.display.stop() [code](#)

```

const CurrentState = function (prefix) {
  let state = {
    ...
    disposeDisplay: function () {
      if (state.display) {
        state.display.stop();
        state.display = null;
      }
    }
  };
  return state;
}

```