

# SFU Player

Пример демонстрирует проигрывание одного или нескольких потоков в нескольких качествах в одном WebRTC соединении. Единицей публикации считается комната, то есть зрители, подключившись к этой комнате, получают все опубликованные в ней потоки.

На скриншотах ниже:

- Server url - WebSocket URL WCS сервера
- Room name - имя комнаты
- Player - имя пользователя, который играет потоки
- 180p send, 360p send , 720p send - кнопки переключения принимаемого качества

# SFU Player

Server url

wss://test1.flashphoner.com:8443

Room name

ROOM1-16cc

Player

Player1-6930

Stop

ESTABLISHED

Meeting: ROOM1-16cc

Name: Publisher1-2b12#b9d5

320x180

Current video track: 0

mute

180p send

360p send

720p send

Track №0: cam1



Обратите внимание, что аудио потоки проигрываются в отдельных элементах.

## Исходный код примера

Исходный код разбит на следующие модули:

- player.html - HTML страница
- player.css - стили HTML страницы

- player.js - основная логика приложения
- config.json - файл конфигурации клиента, содержит описание комнаты

## Анализ исходного кода

Для работы с исходным кодом примера возьмем версию файла player.js, доступную [здесь](#)

### 1. Локальные переменные

Объявление локальных переменных для работы с константами, SFU SDK, для отображения видео и работы с конфигурацией клиента

[code](#)

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

### 2. Конфигурация по умолчанию

Объявление конфигурации комнаты по умолчанию, на случай, если нет файла конфигурации config.json

[code](#)

```
const defaultConfig = {
  room: {
    url: "ws://127.0.0.1:8080",
    name: "ROOM1",
    pin: "1234",
    nickName: "User1"
  }
};
```

### 3. Объект для хранения текущего состояния проигрывания

Хранит данные Websocket сессии, WebRTC соединения и комнаты, формирует идентификаторы элементов на странице для доступа к ним

[code](#)

```

const CurrentState = function(prefix) {
  let state = {
    prefix: prefix,
    pc: null,
    session: null,
    room: null,
    roomEnded: 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;
    }
  };
  return state;
}

```

## 4. Инициализация

`init()` [code](#)

Функция `init()` вызывается после того, как страница загрузится:

- инициализирует объекты состояния
- загружает `config.json` или конфигурацию по умолчанию
- инициализирует поля ввода

```
const init = function() {
  let configName = getUrlParam("config") || "./config.json";
  ...
  playState = CurrentState(PLAY);
  $.getJSON(configName, function(cfg){
    mainConfig = cfg;
    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(playState);
  });
  $("#url").val(setURL());
  $("#roomName").val("ROOM1-"+createUUID(4));
  $("#playName").val("Player1-"+createUUID(4));
}
```

## 5. Соединение с сервером

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

Функция `connect()` вызывается по нажатию кнопки Play:

- создает объект `PeerConnection`
- очищает отображение статуса предыдущей сессии
- настраивает конфигурацию комнаты и создает Websocket сессию
- подписывается на события Websocket сессии

```
const connect = async function(state) {
  //create peer connection
  const 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");
}
}

```

## 1. Запуск проигрывания при установке соединения

`onConnected()` [code](#)

Функция `onConnected()`:

- настраивает действия по нажатию кнопки Stop
- подписывается на события об ошибках комнаты
- вызывает функцию проигрывания

```

const onConnected = async 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 (e) {
        setStatus(state.errInfoId(), "Room "+state.room.name()+" has ended",
"red");
        state.setRoomEnded();
        onStopClick(state);
    }).on(constants.SFU_ROOM_EVENT.DROPPED, function (e) {
        setStatus(state.errInfoId(), "Dropped from the room
"+state.room.name()+" due to network issues", "red");
        state.setRoomEnded();
        onStopClick(state);
    });
    await playStreams(state);
    // Enable button after starting playback #WCS-3635
    $("#" + state.buttonId()).prop('disabled', false);
}

```

## 7. Проигрывание потоков

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

Функция `playStreams()`:

- инициализирует базовый элемент для отображения входящих медиа потоков
- настраивает WebRTC соединение в комнате

```

const playStreams = async function(state) {
    //create remote display item to show remote streams
    try {
        remoteDisplay = initDefaultRemoteDisplay(state.room,
document.getElementById("remoteVideo"), {quality: true});
        // 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);
        }
    }
}

```

## 8. Остановка проигрывания

`stopStreams()`, `remoteDisplay.stop()` [code](#)

```

const stopStreams = function(state) {
    if (remoteDisplay) {
        remoteDisplay.stop();
    }
}

```

```
}  
}
```

## 1. Действия по нажатию кнопки Play

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

Функция `onStartClick()`:

- проверяет правильность заполнения полей ввода
- перед стартом воспроизведения, в браузере Safari вызывает функцию `playFirstSound()` для автоматического проигрывания аудио
- вызывает функцию `connect()`

```
const onStartClick = function(state) {  
  if (validateForm("connectionForm") && validateForm(state.formId())) {  
    $("#" + state.buttonId()).prop('disabled', true);  
    if (state.is(PLAY) && Browser().isSafariWebRTC()) {  
      playFirstSound(document.getElementById("main"),  
PRELOADER_URL).then(function () {  
        connect(state);  
      });  
    } else {  
      connect(state);  
    }  
  }  
}
```

## 10. Действия по нажатию кнопки Stop

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

Функция `onStopClick()`:

- останавливает воспроизведение
- разрывает Websocket сессию

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

## 11. Действия при разрыве Websocket сессии

`onDisconnected()` [code](#)



Функция `onDisconnected()`:

- настраивает действия по нажатию кнопки Play
- открывает доступ к полям ввода Server url и Room name, если нет параллельной сессии

```
const onDisconnected = function(state) {
  state.clear();
  $("# +
state.buttonId()).text(state.buttonText()).off('click').click(function () {
  onStartClick(state);
}).prop('disabled', false);
$('#url').prop('disabled', false);
$("#roomName").prop('disabled', false);
$("# + state.inputId()).prop('disabled', false);
}
```