

WebRTC ABR Player 2.0.248 and later

- [Example source code](#)
- [Analyzing the code](#)
 - [1. Local variables](#)
 - [2. Object to store current publishing/playback state](#)
 - [3. Initialization](#)
 - [4. Establishing server connection](#)
 - [5. Playback start after session establishing](#)
 - [6. Streams playback](#)
 - [7. Playback stopping](#)
 - [8. Play click action](#)
 - [9. Stop click actions](#)
 - [10. Websocket session disconnection actions](#)

The example shows how a stream published to WCS server may be played in a number of video qualities via WebRTC.

On the screenshot below:

- Server url - WCS server websocket URL
- Stream name - stream name to play
- Auto, 240p send, 480p send, 720p send - quality switching buttons named by quality profiles from `/usr/local/FlashphonerWebCallServer/conf/wcs_sfu_bridge_profiles.yml` file

WebRTC ABR Player

Meeting: test

Name: test#test

426x240

Current video track: 0



Server url

wss://test1.flashphoner.com:8443

Stream name

test

ESTABLISHED

Note that audio track is playing separately

Example source code

The source code consists of the following modules:

- player.html - HTML page
- player.css - HTML page styles
- player.js - main application logic

Analyzing the code

To analyze the example source code, take the file player.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;
const PRELOADER_URL = "../commons/media/silence.mp3";
const playStatus = "playStatus";
const playErrorInfo = "playErrorInfo";
```

2. Object to store current publishing/playback state

The object should keep Websocket session data, WebRTC connection data, room data and object to display tracks data

[code](#)

```
const CurrentState = function() {
  let state = {
    pc: null,
    session: null,
    room: null,
    display: 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;
    },
    isRoomEnded: function() {
      return state.roomEnded;
    },
    isConnected: function() {
      return (state.session && state.session.state() === constants.SFU_STATE.CONNECTED);
    },
    isActive: function() {
      return (state.room && !state.roomEnded && state.pc);
    },
    setDisplay: function (display) {
      state.display = display;
    },
    disposeDisplay: function () {
      if (state.display) {
        state.display.stop();
        state.display = null;
      }
    }
  };
  return state;
}
```

3. Initialization

`init()`[code](#)

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

- initializes state objects
- initializes input fields

```
const init = function() {
  $("#playBtn").prop('disabled', true);
  $("#url").prop('disabled', true);
  $("#streamName").prop('disabled', true);
  onDisconnected(CurrentState());
  $("#url").val(setURL());
}
```

4. Establishing server connection

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

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

- creates `PeerConnection` object
- cleans the 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();
  // Create a config to connect to SFU room
  const roomConfig = {
    // Server websocket URL
    url: $("#url").val(),
    // Use stream name as room name to play ABR
    roomName: $("#streamName").val(),
    // Make a random participant name from stream name
    nickname: "Player-" + $("#streamName").val() + "-" + createUUID(4),
    // Set room pin
    pin: 123456
  }
  // Clean state display items
  setStatus(playStatus, "");
  setStatus(playErrorInfo, "");
  try {
    // Connect to the server (room should already exist)
    const session = await sfu.createRoom(roomConfig);
    // Set up session ending events
    session.on(constants.SFU_EVENT.DISCONNECTED, function() {
      onStopClick(state);
      onDisconnected(state);
      setStatus(playStatus, "DISCONNECTED", "green");
    }).on(constants.SFU_EVENT.FAILED, function(e) {
      onStopClick(state);
      onDisconnected(state);
      setStatus(playStatus, "FAILED", "red");
      if (e.status && e.statusText) {
        setStatus(playErrorInfo, e.status + " " + e.statusText, "red");
      } else if (e.type && e.info) {
        setStatus(playErrorInfo, e.type + ": " + e.info, "red");
      }
    });
    // Connected successfully
    onConnected(state, pc, session);
    setStatus(playStatus, "CONNECTING...", "black");
  } catch(e) {
    onDisconnected(state);
    setStatus(playStatus, "FAILED", "red");
    setStatus(playErrorInfo, e, "red");
  }
}
```

5. Playback start after session establishing

onConnected() [code](#)

The onConnected() function:

- sets up Stop button click actions
- subscribes to SFU_ROOM_EVENT.PARTICIPANT_LIST event to check if the stream is published in the room
- subscribes to room error events
- calls playback function

```
const onConnected = async function(state, pc, session) {
  state.set(pc, session, session.room());
  $("#playBtn").text("Stop").off('click').click(function () {
    onStopClick(state);
  });
  $('#url').prop('disabled', true);
  $("#streamName").prop('disabled', true);
  // Add room event handling
  state.room.on(constants.SFU_ROOM_EVENT.PARTICIPANT_LIST, function(e) {
    // If the room is empty, the stream is not published yet
    if (!e.participants || e.participants.length === 0) {
      setStatus(playErrorInfo, "ABR stream is not published", "red");
      onStopClick(state);
    }
    else {
      setStatus(playStatus, "ESTABLISHED", "green");
      $("#placeholder").hide();
    }
  }).on(constants.SFU_ROOM_EVENT.FAILED, function(e) {
    // Display error state
    setStatus(playErrorInfo, e, "red");
  }).on(constants.SFU_ROOM_EVENT.OPERATION_FAILED, function (e) {
    onOperationFailed(state);
  }).on(constants.SFU_ROOM_EVENT.ENDED, function () {
    // Publishing is stopped, dispose playback and close connection
    setStatus(playErrorInfo, "ABR stream is stopped", "red");
    state.setRoomEnded();
    onStopClick(state);
  }).on(constants.SFU_ROOM_EVENT.DROPPED, function () {
    // Client dropped from the room, dispose playback and close connection
    setStatus(playErrorInfo, "Playback is dropped due to network issues", "red");
    state.setRoomEnded();
    onStopClick(state);
  });
  await playStreams(state);
  // Enable button after starting playback #WCS-3635
  $("#playBtn").prop('disabled', false);
}
```

6.Streams playback

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

The playStreams() function:

- initializes a base container tag to display incoming media streams
- sets up incoming streams info displaying parameters
- sets up ABR to switch automatically between available qualities when playback channel conditions are changing
- negotiates WebRTC connection

```
const playStreams = async function (state) {
  try {
    // Create remote display item to show remote streams
    const display = initRemoteDisplay(state.room, document.getElementById("remoteVideo"), {quality:true,
autoAbr: true}, {thresholds: [
      {parameter: "nackCount", maxLeap: 10},
      {parameter: "freezeCount", maxLeap: 10},
      {parameter: "packetsLost", maxLeap: 10}
    ], abrKeepOnGoodQuality: ABR_KEEP_ON_QUALITY, abrTryForUpperQuality: ABR_TRY_UPPER_QUALITY, interval:
ABR_QUALITY_CHECK_PERIOD},createDefaultMeetingController, createDefaultMeetingModel, createDefaultMeetingView,
oneToOneParticipantFactory(remoteTrackProvider(state.room)));
    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(playErrorInfo, e.name, "red");
      onStopClick(state);
    }
  }
}
}
```

7.Playback stopping

CurrentState.disposeDisplay() [code](#)

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

8.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")) {
    $("#playBtn").prop('disabled', true);
    if (Browser().isSafariWebRTC()) {
      playFirstSound(document.getElementById("main"), PRELOADER_URL).then(function () {
        connect(state);
      });
    } else {
      connect(state);
    }
  }
}
```

9.Stop click actions

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

The onStopClick() function:

- stops playback
- disconnects Websocket session

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

10.Websocket session disconnection actions

onDisconnected() [code](#)

The onDisconnected() function:

- sets up Play click actions
- enables Server url and Room name fields access

```
const onDisconnected = function(state) {
  state.clear();
  $("#placeholder").show();
  $("#playBtn").text("Play").off('click').click(function () {
    onStartClick(state);
  }).prop('disabled', false);
  $('#url').prop('disabled', false);
  $('#streamName').prop('disabled', false);
}
```