

Two-way Streaming

Пример стримера и плеера на одной странице

Данный пример показывает, как воспроизводить видеопоток с одновременной публикацией другого потока, используя одну web-страницу.

The screenshot displays a web application titled "Two-way Streaming" with two main sections: "Local" and "Player".

- Local Section:** Features a video player showing a 3D animated character in a forest. Below the video is a text input field containing "test" and a "Stop" button.
- Player Section:** Features a video player showing a black screen with the time "7:38". Below the video is a text input field containing "rtsp://demo.flash|", a "Stop" button, and an "Available" button.
- Local Publishing:** A section labeled "PUBLISHING" with a text area containing the JSON object `{"count": 23}` and a "Send payload as object" button.
- Player Playback:** A section labeled "PLAYING" with an empty text area.
- Connection:** A status bar at the bottom shows the connection URL `ws://localhost:8080` and a "Disconnect" button. The status "ESTABLISHED" is displayed in green text below the connection bar.

Код примера

Код данного примера находится на WCS-сервере по следующему пути:

`/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/two_way_streaming`

- `two_way_streaming.css` - файл стилей
- `two_way_streaming.html` - страница клиента
- `two_way_streaming.js` - скрипт, обеспечивающий работу примера

Тестировать данный пример можно по следующему адресу:

https://host:8888/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html

Здесь host - адрес WCS-сервера.

Работа с кодом примера

Для разбора кода возьмем версию файла `two_way_streaming.js` с хешем `ecbadc3`, которая находится [здесь](#) и доступна для скачивания в соответствующей сборке [2.0.212](#).

1. Инициализация API

`Flashphoner.init()` [code](#)

```
Flashphoner.init();
```

2. Подключение к серверу

`Flashphoner.createSession()` [code](#)

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
});
```

3. Получение от сервера события, подтверждающего успешное соединение

`ConnectionStatusEvent ESTABLISHED` [code](#)

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function(session){
    setStatus("#connectStatus", session.status());
    onConnected(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
});
```

4. Публикация видеопотока

`Session.createStream()`, `Stream.publish()` [code](#)

При создании передаются:

- `streamName` - имя видеопотока

- `localVideo` - `div`, в котором будет отображаться видео с камеры.

```
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true
  ...
}).publish();
```

5. Получение от сервера события, подтверждающего успешную публикацию потока

`StreamStatusEvent PUBLISHING` [code](#)

```
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true
  ...
}).on(STREAM_STATUS.PUBLISHING, function(stream){
  setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
  onPublishing(stream);
}).on(STREAM_STATUS.UNPUBLISHED, function(){
  ...
}).on(STREAM_STATUS.FAILED, function(){
  ...
}).publish();
```

6. Воспроизведение видеопотока

`Session.createStream()`, `Stream.play()` [code](#).

При создании передаются:

- `streamName` - имя видеопотока (в том числе, это может быть имя потока, опубликованного выше)
- `remoteVideo` - `div`, в котором будет отображаться видео.

```
session.createStream({
  name: streamName,
  display: remoteVideo
  ...
}).play();
```

7. Получение от сервера события, подтверждающего успешное воспроизведение потока

`StreamStatusEvent PLAYING` [code](#)

```
session.createStream({
  name: streamName,
```

```

    display: remoteVideo
    ...
  }).on(STREAM_STATUS.PLAYING, function(stream) {
    setStatus("#playStatus", stream.status());
    onPlaying(stream);
  }).on(STREAM_STATUS.STOPPED, function() {
    ...
  }).on(STREAM_STATUS.FAILED, function() {
    ...
  }).play();

```

8. Остановка воспроизведения видеопотока

`Stream.stop()` [code](#)

```

function onPlaying(stream) {
  $("#playBtn").text("Stop").off('click').click(function(){
    $(this).prop('disabled', true);
    stream.stop();
  }).prop('disabled', false);
  $("#playInfo").text("");
}

```

9. Получение от сервера события, подтверждающего успешную остановку воспроизведения потока

`StreamStatusEvent STOPPED` [code](#)

```

session.createStream({
  name: streamName,
  display: remoteVideo
  ...
}).on(STREAM_STATUS.PLAYING, function(stream) {
  ...
}).on(STREAM_STATUS.STOPPED, function() {
  setStatus("#playStatus", STREAM_STATUS.STOPPED);
  onStopped();
}).on(STREAM_STATUS.FAILED, function() {
  ...
}).play();

```

10. Остановка публикации видеопотока

`Stream.stop()` [code](#)

```

function onPublishing(stream) {
  $("#publishBtn").text("Stop").off('click').click(function(){
    $(this).prop('disabled', true);
    stream.stop();
  }).prop('disabled', false);
  $("#publishInfo").text("");
  ...
}

```

11. Получение от сервера события, подтверждающего успешную остановку публикации потока

`StreamStatusEvent UNPUBLISHED` [code](#)

```
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true
  ...
}).on(STREAM_STATUS.PUBLISHING, function(stream){
  ...
}).on(STREAM_STATUS.UNPUBLISHED, function(){
  setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
  onUnpublished();
}).on(STREAM_STATUS.FAILED, function(){
  ...
}).publish();
```

12. Отправка данных, привязанных к потоку

`Stream.sendData()` [code](#)

```
function onPublishing(stream) {
  ...
  $('#sendDataBtn').off('click').click(function(){
    var streamData = field('streamData');
    stream.sendData(JSON.parse(streamData));
  }).prop('disabled', false);
}
```

13. Получение данных, привязанных к потоку

`STREAM_EVENT`, `STREAM_EVENT_TYPE.DATA` [code](#)

```
session.createStream({
  name: streamName,
  display: remoteVideo
}).on(STREAM_STATUS.PENDING, function (stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
  ...
}).on(STREAM_STATUS.STOPPED, function () {
  ...
}).on(STREAM_STATUS.FAILED, function (stream) {
  ...
}).on(STREAM_EVENT, function(streamEvent) {
  switch (streamEvent.type) {
    case STREAM_EVENT_TYPE.DATA:
      addPayload(streamEvent.payload);
      break;
  }
  console.log("Received streamEvent ", streamEvent.type);
}).play();
```

