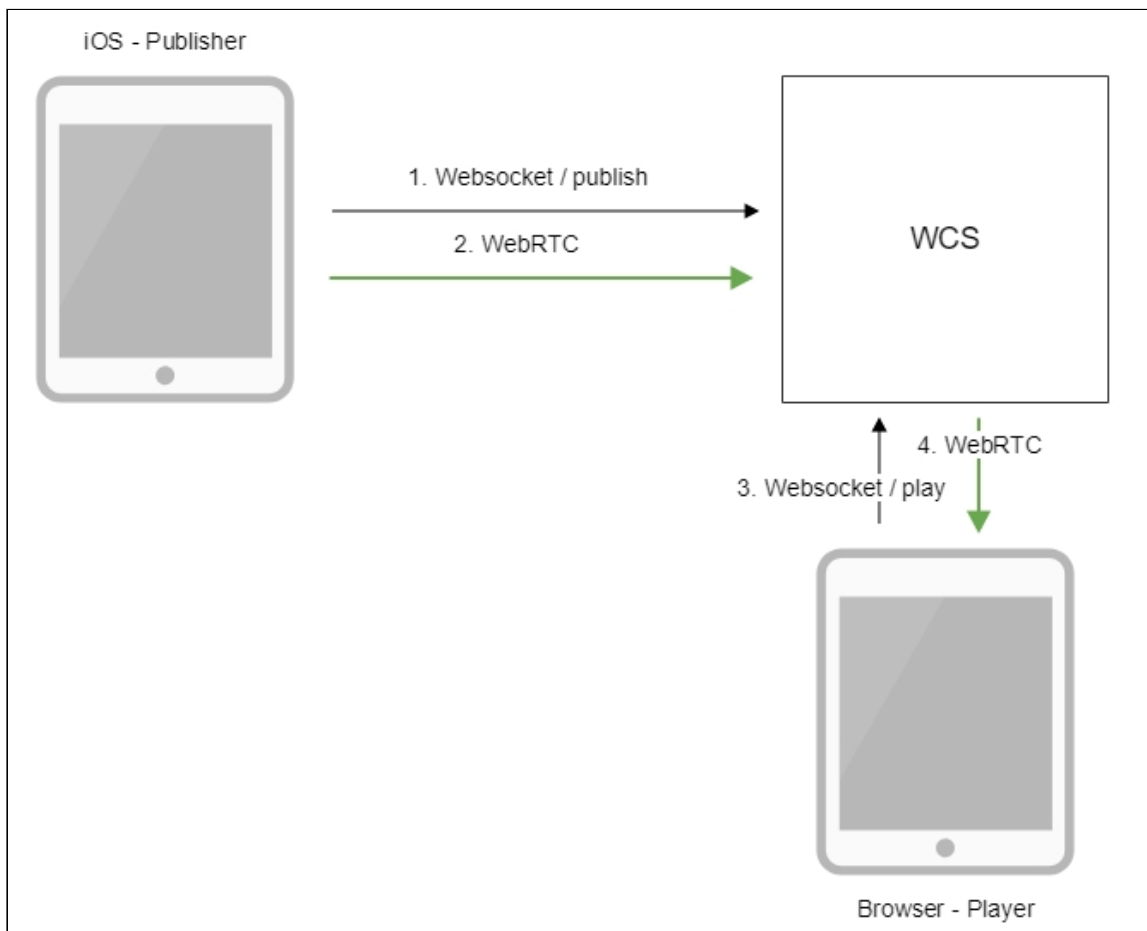


# From an iOS mobile app via WebRTC

## Overview

WCS provides SDK to develop client applications for the iOS platform

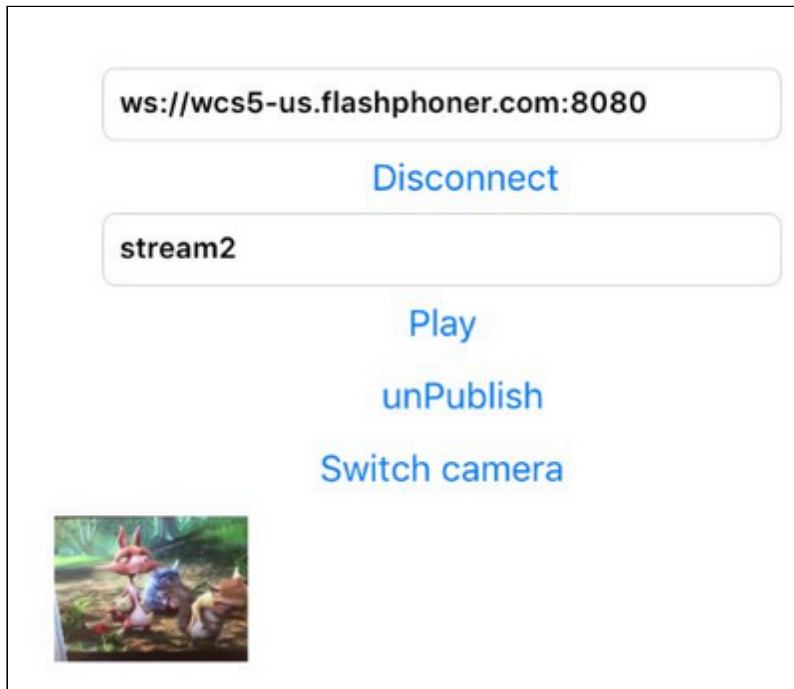
## Operation flowchart



1. The iOS device connects to the server via the Websocket protocol and sends the `publishStream` command.
2. The iOS device captures the microphone and the camera and sends the WebRTC stream to the server.
3. The browser establishes a connection via WebSocket and sends the `playStream` command.
4. The browser receives the WebRTC stream and plays that stream on the page

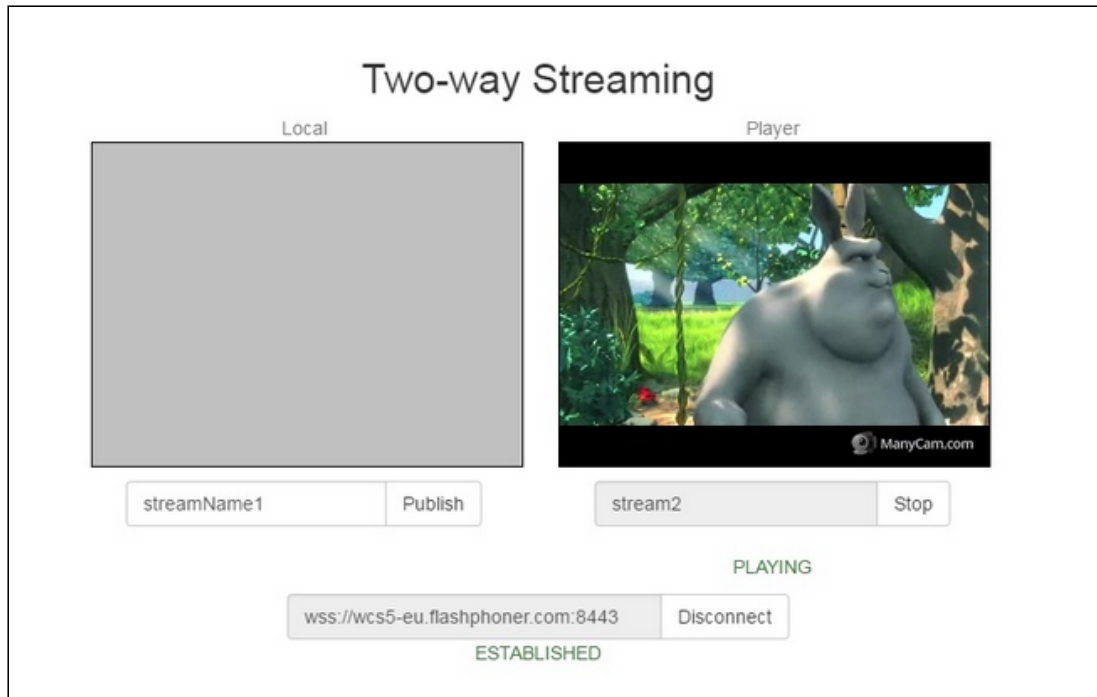
## Quick manual on testing

1. For the test we use:
2. the demo server at `wcs5-us.flashphoner.com`;
3. the mobile application for iOS;
4. the [Two Way Streaming](#) web application to display the captured stream
5. Start the application on iPhone. Enter the URL of the WCS server and the name of the stream. Publish the stream by clicking `Publish`:



6. Open the Two Way Streaming web application. In the field just below the player window enter the name of the stream broadcast from iPhone, and click the `Play` button. The

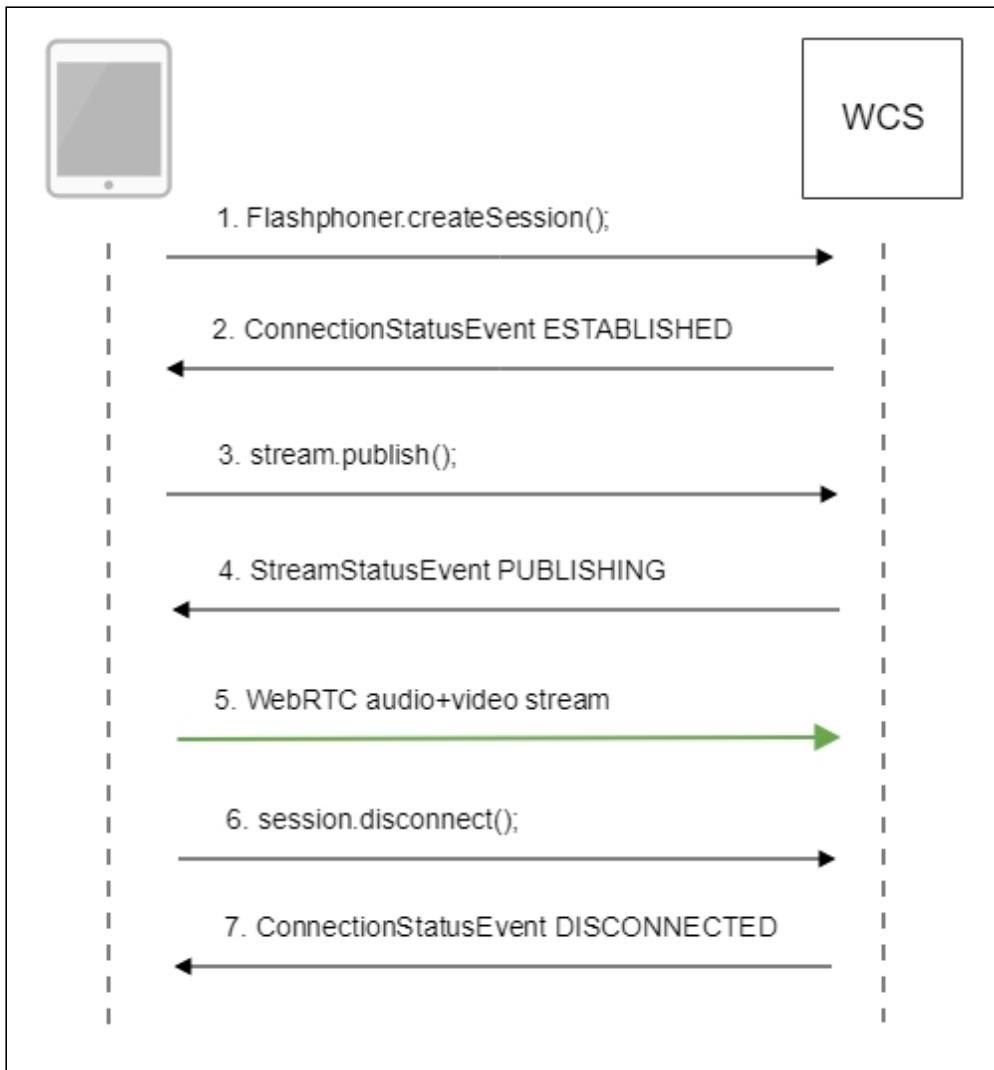
browser starts playing the stream:



## Call flow

Below is the call flow when using the Streamer example

[ViewController.m](#)



### 1. Establishing a connection to the server

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

```

FPWCSSessionOptions *options = [[FPWCSSessionOptions alloc] init];
NSURL *url = [[NSURL alloc] initWithString:_connectUrl.text];
options.urlServer = [NSString stringWithFormat:@"%@@://%@:%@", url.scheme,
url.host, url.port];
streamName = [url.path stringByDeletingPathExtension
stringByReplacingOccurrencesOfString:@"/" withString:@""];
options.appKey = @"defaultApp";
NSError *error;
session = [FPWCSSession createSession:options error:&error];
  
```

### 2. Receiving from the server an event confirming successful connection

`FPWCSSession.on:kFPWCSSessionStatusEstablished` [callback](#) [code](#)

```

[session on:kFPWCSSessionStatusEstablished callback:^(FPWCSSession
*rSession){
    [self changeConnectionStatus:[rSession getStatus]];
}];
  
```

```
[self onConnected:rSession];
}];
```

### 3. Publishing the stream

`FPWCSApi2Session.createStream()` [code](#)

```
FPWCSApi2Session *session = [FPWCSApi2 getSessions][0];
FPWCSApi2StreamOptions *options = [[FPWCSApi2StreamOptions alloc] init];
options.name = streamName;
options.display = _videoView.local;
if ( UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad ) {
    options.constraints = [[FPWCSApi2MediaConstraints alloc]
initWithAudio:YES videoWidth:640 videoHeight:480 videoFps:15];
}
NSError *error;
publishStream = [session createStream:options error:&error];
```

### 4. Receiving from the server an event confirming successful publishing of the stream

`FPWCSApi2Stream.on:kFPWCSSStreamStatusPublishing` [callback](#) [code](#)

```
[publishStream on:kFPWCSSStreamStatusPublishing callback:^(FPWCSApi2Stream
*rStream){
    [self changeStreamStatus:rStream];
    [self onPublishing:rStream];
}];
```

### 5. Sending audio and video stream via WebRTC

### 6. Stopping publishing of the stream

`FPWCSApi2Session.disconnect()` [code](#)

```
if ([button.titleLabel.text isEqualToString:@"STOP"]) {
    if ([FPWCSApi2 getSessions].count) {
        FPWCSApi2Session *session = [FPWCSApi2 getSessions][0];
        NSLog(@"Disconnect session with server %@", [session getServerUrl]);
        [session disconnect];
    } else {
        NSLog(@"Nothing to disconnect");
        [self onDisconnected];
    }
} else {
    //todo check url is not empty
    [self changeViewState:_connectUrl enabled:NO];
    [self connect];
}
```

### 7. Receiving from the server an event confirming successful unpublishing of the stream

`FPWCSApi2Session.on:kFPWCSSessionStatusDisconnected` [callback](#) [code](#)

```
[session on:kFPWCSSessionStatusDisconnected callback:^(FPWCSApi2Session
*rSession){
    [self changeConnectionStatus:[rSession getStatus]];
}
```

```
[self onDisconnected];  
}];
```