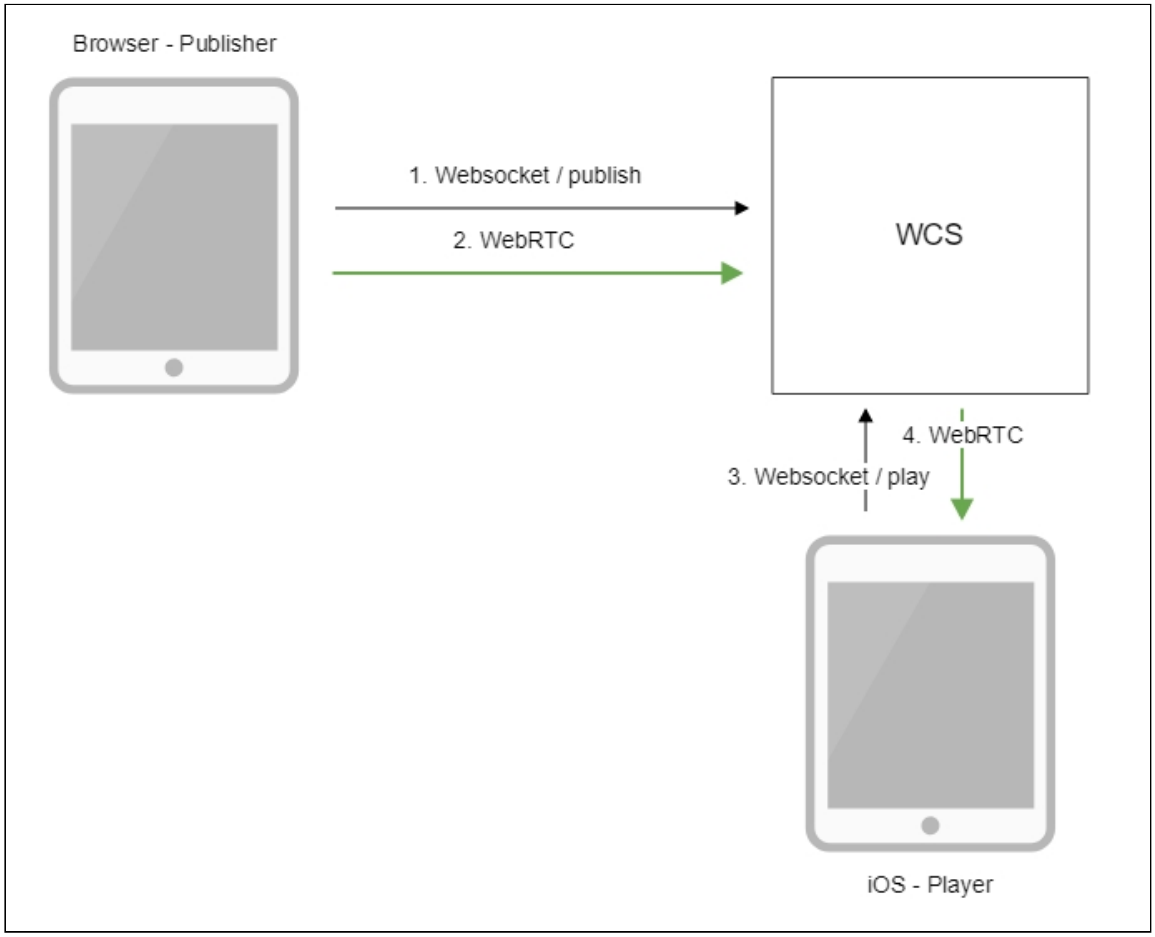
In an iOS mobile application via WebRTC

Overview

WCS provides SDK to develop client applications for the iOS platform.

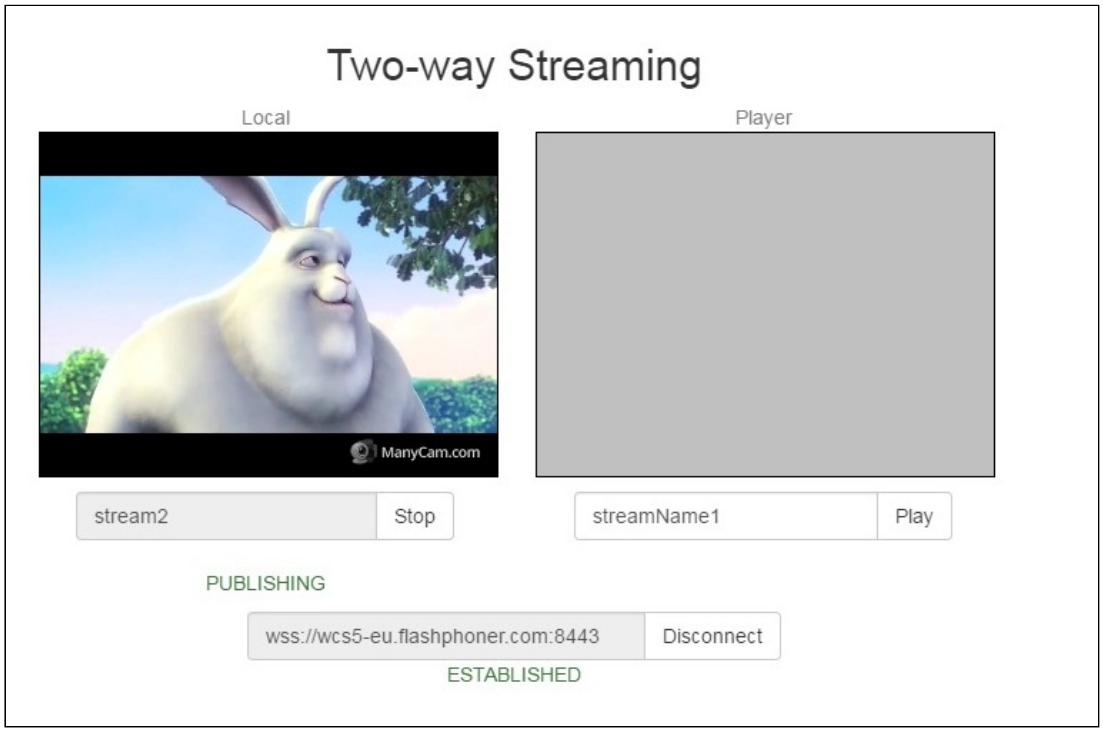
Operation flowchart



1. The browser connects to the server via the Websocket protocol and sends the `publishStream` command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The iOS device connects to the server via the Websocket protocol and sends the `playStream` command.
4. The iOS device receives the WebRTC stream from the server and plays it in the application.

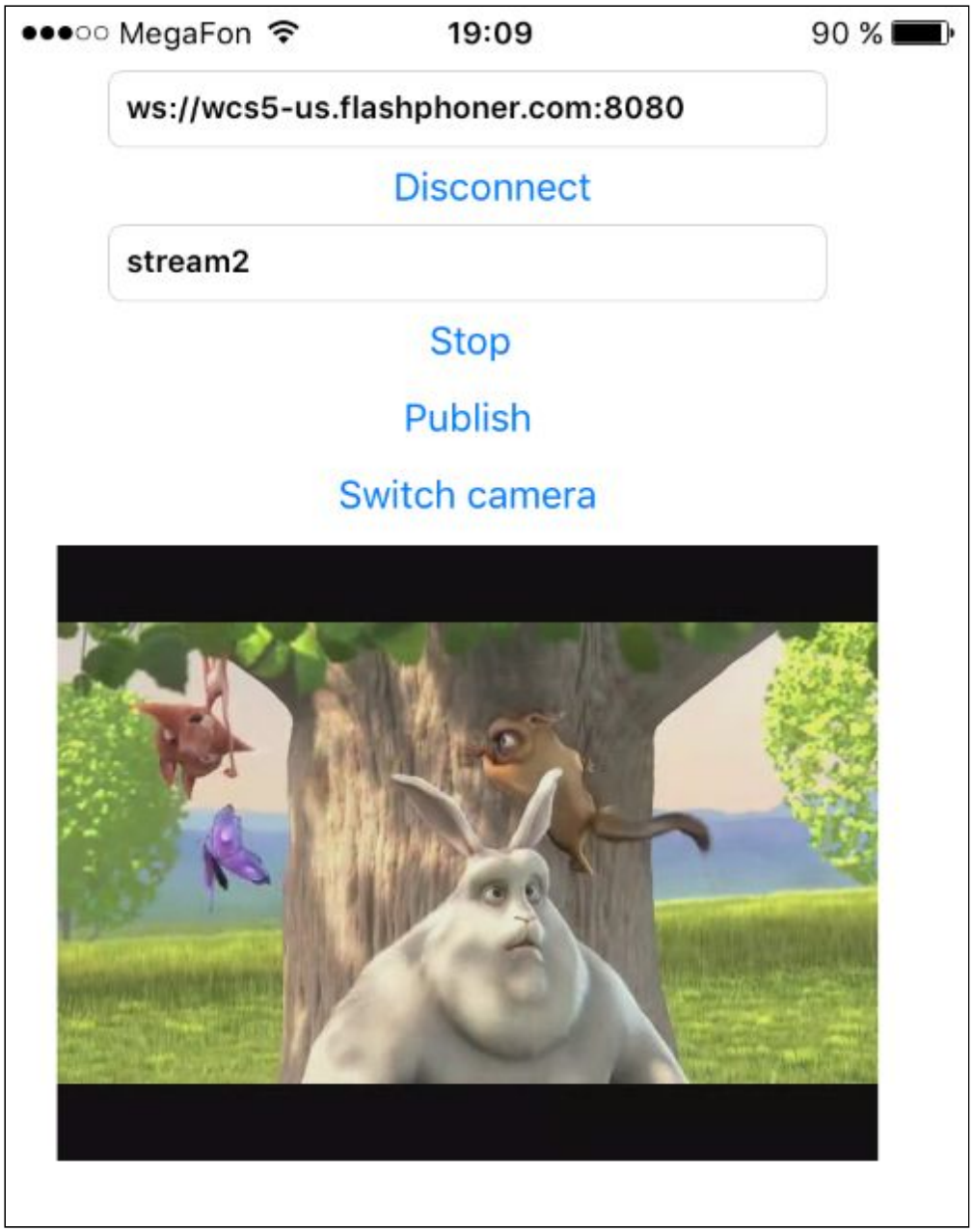
Quick manual on testing

1. For the test we use:
2. the demo server at `demo.flashphoner.com`;
3. the Two Way Streaming web application to publish the stream;
4. the iOS mobile application from AppStore to play the stream.
5. Open the Two Way Streaming web application. Click `Connect`, then `Publish`. Copy the identifier of the stream:



6. Install the iOS mobile application from AppStore. Run the application. Enter the URL of the WCS server and the name of the published stream, then click `Play`. The stream

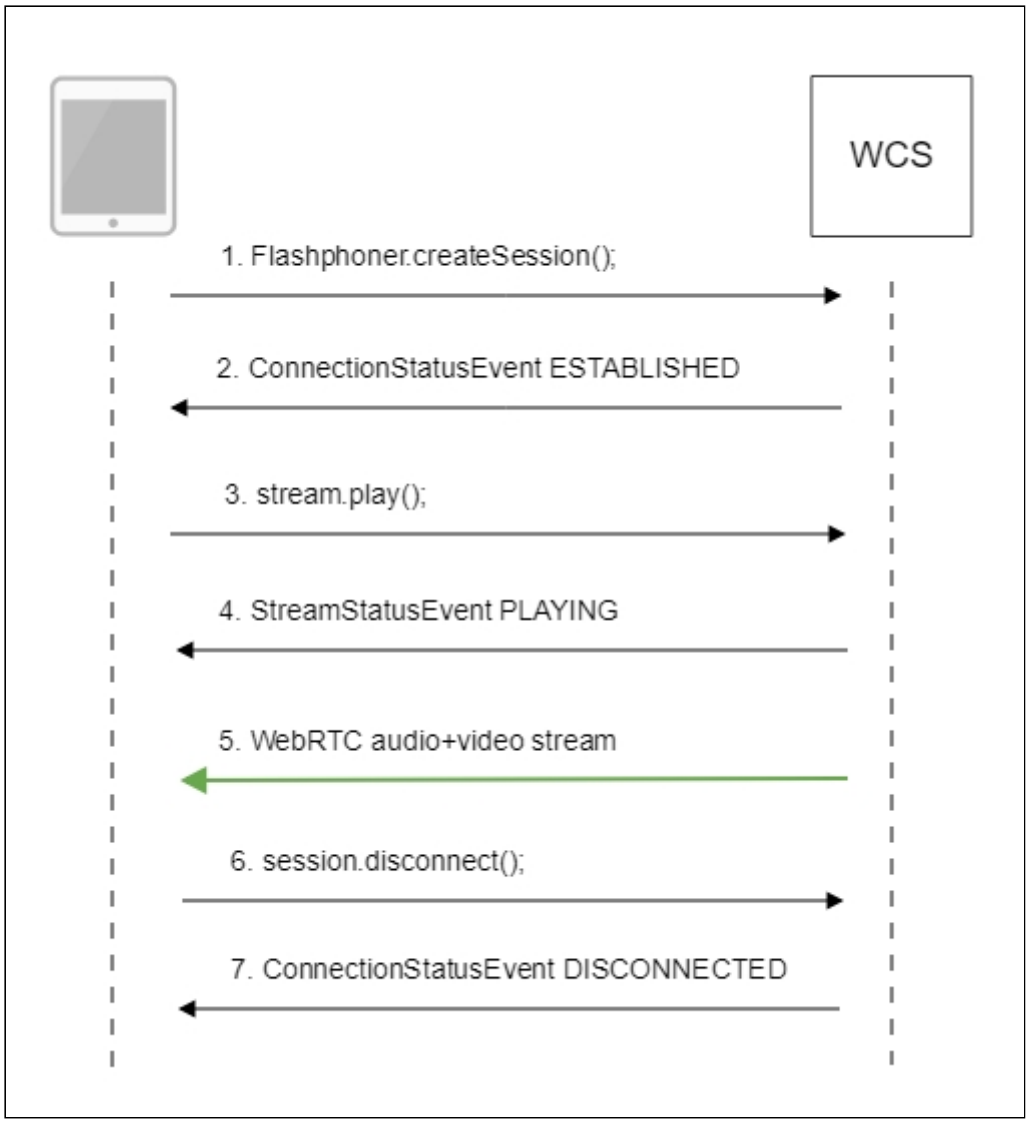
starts playing from the server:



Call flow

Below is the call flow when using the Player example.

ViewController.m



1. Establishing a connection to the server
   `FPWCSApi2.createSession()` code

```
FPWCSApi2SessionOptions *options = [[FPWCSApi2SessionOptions alloc] init];
options.urlServer = _connectUrl.text;
options.appKey = @"defaultApp";
NSError *error;
FPWCSApi2Session *session = [FPWCSApi2 createSession:options
error:&error];
```

2. Receiving from the server an event confirming successful connection
   `FPWCSApi2Session.on:kFPWCSSessionStatusEstablished callback` code

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

3. Playing the stream
`FPWCSApi2Session.createStream()` code

```
- (FPWCSApi2Stream *)playStream {
    FPWCSApi2Session *session = [FPWCSApi2 getSessions][0];
    FPWCSApi2StreamOptions *options = [[FPWCSApi2StreamOptions alloc]
init];
    options.name = _remoteStreamName.text;
    options.display = _remoteDisplay;
    NSError *error;
    FPWCSApi2Stream *stream = [session createStream:options error:nil];
    if (!stream) {
        ...
        return nil;
    }
}
```

4. Receiving from the server an event confirming successful playing of the stream
`FPWCSApi2Stream.on:kFPWCSStreamStatusPlaying callback` code

```
[stream on:kFPWCSStreamStatusPlaying callback:^(FPWCSApi2Stream *rStream){
    [self changeStreamStatus:rStream];
    [self onPlaying:rStream];
}];
```

5. Receiving the audio and video stream via WebRTC
6. Stopping the playback 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];
    }
    ...
}
```

7. Receiving from the server an event confirming the playback of the stream is stopped
`FPWCSApi2Session.on:kFPWCSSessionStatusDisconnected callback` code

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