

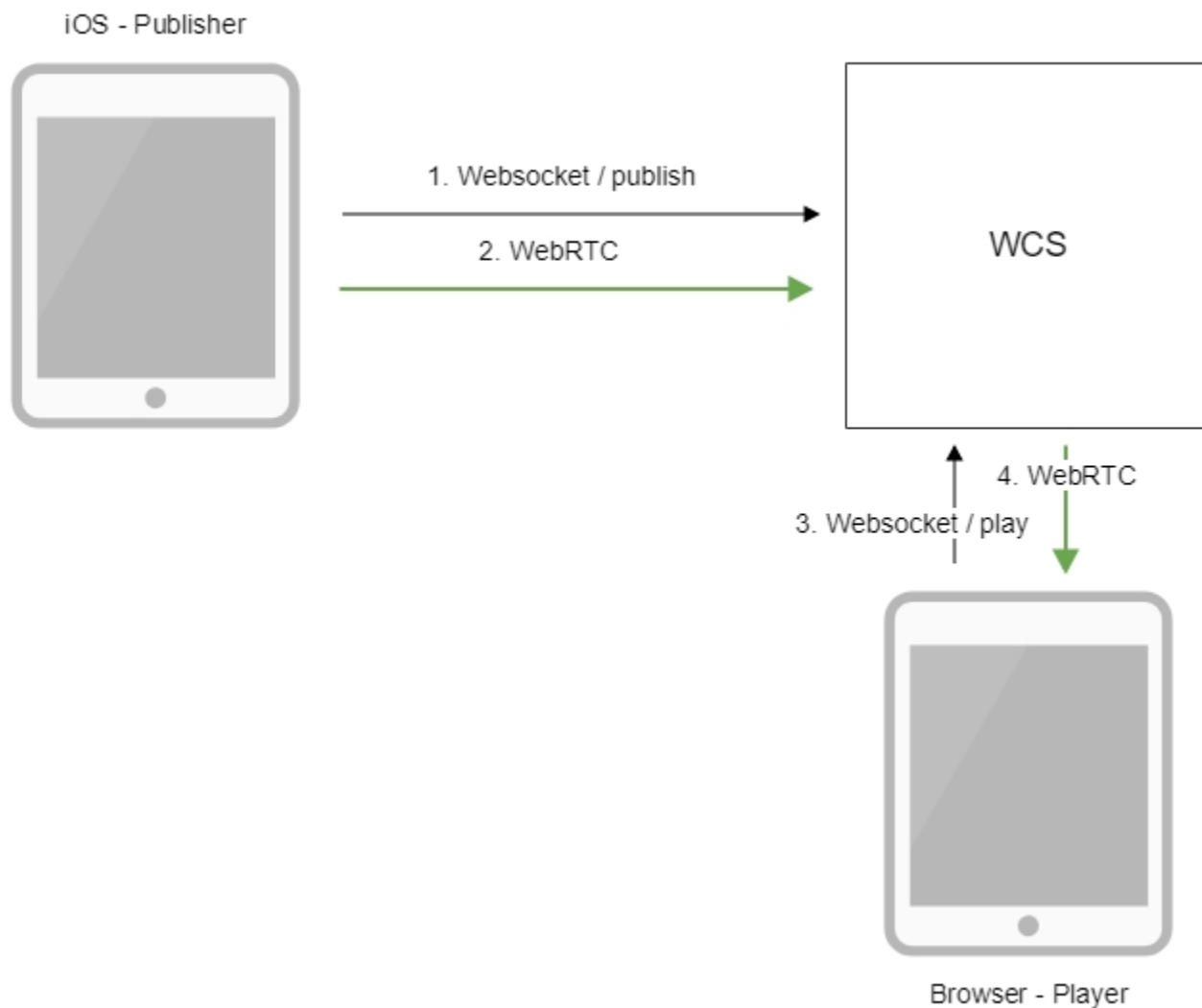
С мобильного приложения iOS по WebRTC

- [Описание](#)
 - [Схема работы](#)
- [Краткое руководство по тестированию](#)
- [Последовательность выполнения операций \(Call flow\)](#)

Описание

WCS предоставляет SDK для разработки клиентских приложений на платформе iOS

Схема работы



1. iOS-устройство соединяется с сервером по протоколу Websocket и отправляет команду publish.
2. iOS-устройство захватывает микрофон и камеру и отправляет WebRTC поток на сервер.
3. Браузер устанавливает соединение по Websocket и отправляет команду play.
4. Браузер получает WebRTC поток и воспроизводит этот поток на странице.

Краткое руководство по тестированию

1. Для теста используем:

- демо-сервер wcs5-us.flashphoner.com;

- мобильное приложение iOS;
- веб-приложение [Two Way Streaming](#) для отображения захваченного потока

2. Запустите приложение на iPhone. Введите URL WCS-сервера и имя потока. Опубликуйте поток, нажав "Publish":

ws://wcs5-us.flashphoner.com:8080

Disconnect

stream2

Play

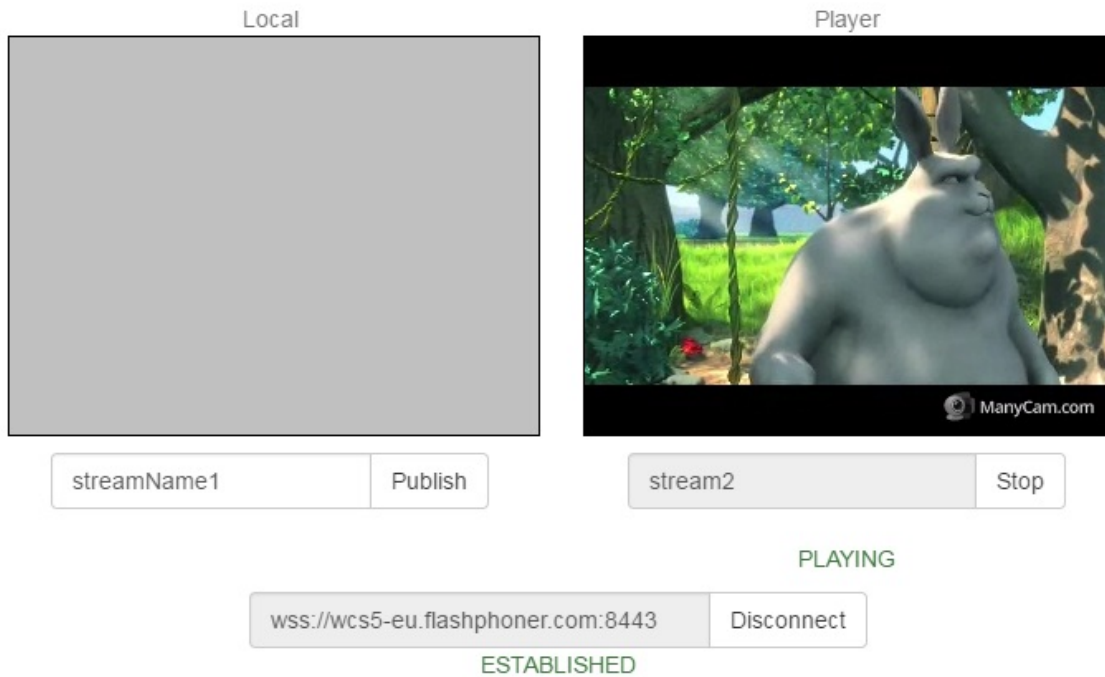
unPublish

Switch camera



3. Откройте веб-приложение Two Way Streaming. Укажите в поле под окном плеера имя потока, транслируемого с iPhone, и нажмите кнопку "Play". Браузер начнет воспроизведение потока:

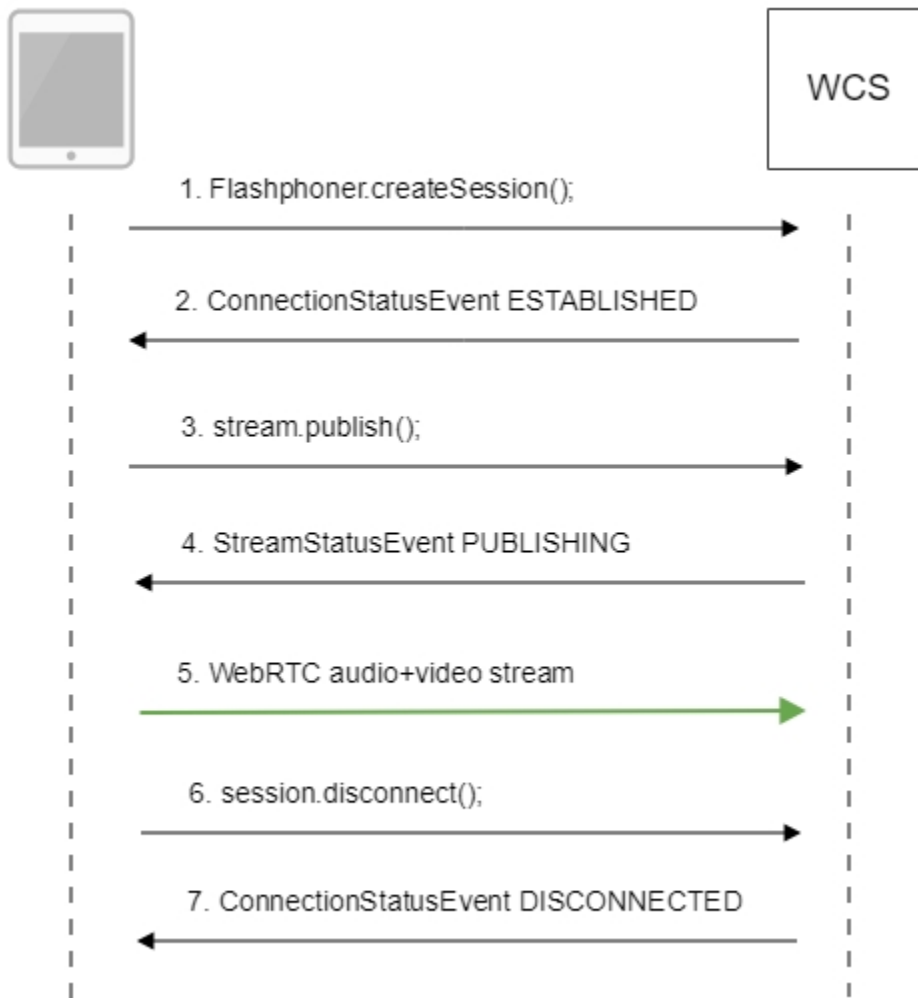
Two-way Streaming



Последовательность выполнения операций (Call flow)

Ниже описана последовательность вызовов при использовании примера Streamer

[ViewController.m](#)



1. Установка соединения с сервером.

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

```

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

2. Получение от сервера события, подтверждающего успешное соединение.

ConnectionStatusEvent ESTABLISHED[code](#)

```

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

3. Публикация потока.

session.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. Получение от сервера события, подтверждающего успешную публикацию потока.

StreamStatusEvent, статус PUBLISHING[code](#)

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

5. Отправка аудио-видео потока по WebRTC

6. Остановка публикации потока.

session.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. Получение от сервера события, подтверждающего остановку публикации потока.

ConnectionStatusEvent DISCONNECTED[code](#)

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