

iOS Media Devices Swift

- [Пример iOS-приложения для управления медиа-устройствами](#)
- [Работа с кодом примера](#)

Пример iOS-приложения для управления медиа-устройствами

Данный пример может использоваться как стример для публикации WebRTC-видеопотока с Web Call Server и позволяет выбрать медиа-устройства и следующие параметры для публикуемого и проигрываемого видео

- разрешение (ширина, высота)
- скорость передачи (bitrate)
- FPS (Frames Per Second) - для публикуемого видео
- quality - для проигрываемого видео

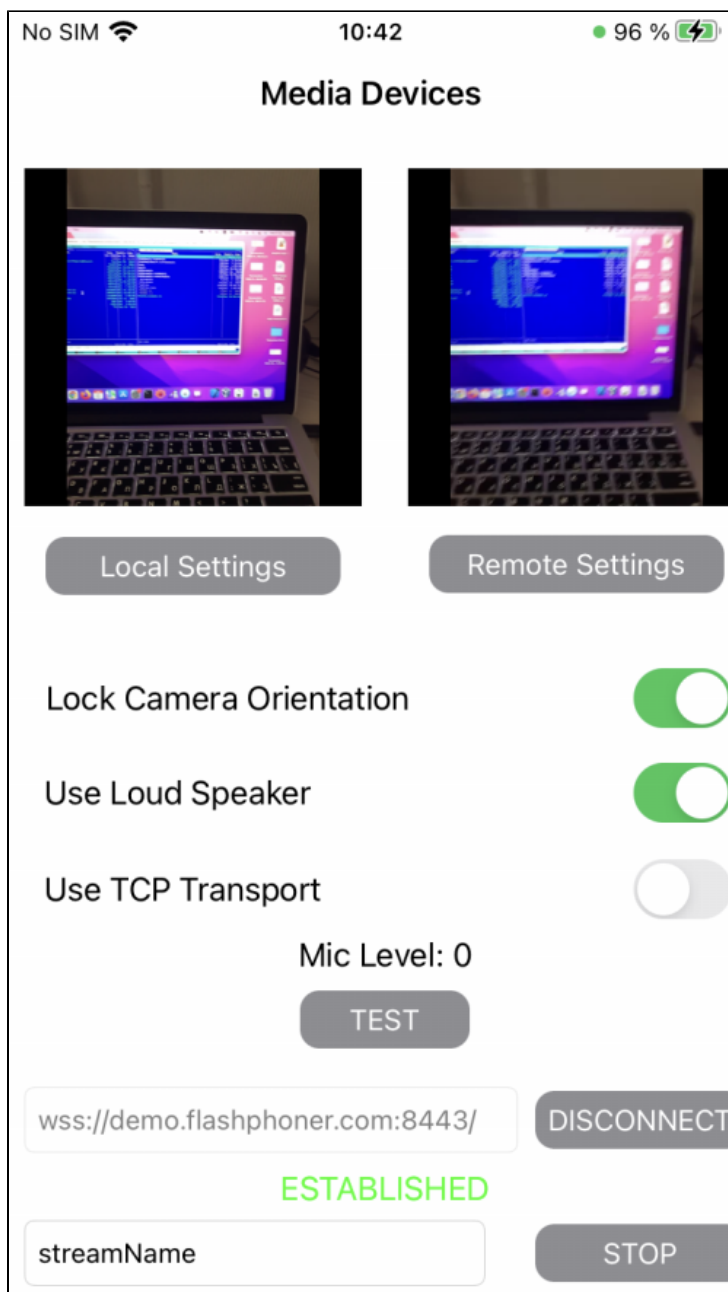
Поток может быть опубликован как с аудио и видео, так и без аудио или видео (переключатели 'Send Audio' и 'Send Video').

Аудио и видео в публикуемом потоке могут быть выключены/включены соответствующими переключателями 'Mute Audio' и 'Mute Video' во время или до начала публикации.

Видео потоки могут воспроизводиться с видео или без видео (переключатель 'Play Video').

При необходимости, определенные кодеки можно исключить из публикации или воспроизведения (поле 'Strip codecs')

На скриншоте ниже представлен пример во время публикации потока. Слева отображается видео с камеры, справа воспроизводится опубликованный поток.



Вид с настройками публикации показывается при нажатии на кнопку 'Local settings'

No SIM 10:41 96 %

< Back

Local Settings

Send Audio

Microphone

iPhone Microphone

FEC

Stereo

Bitrate

Mute Audio

Send Video

Camera

Back Camera

Size

640

x

360

FPS

25

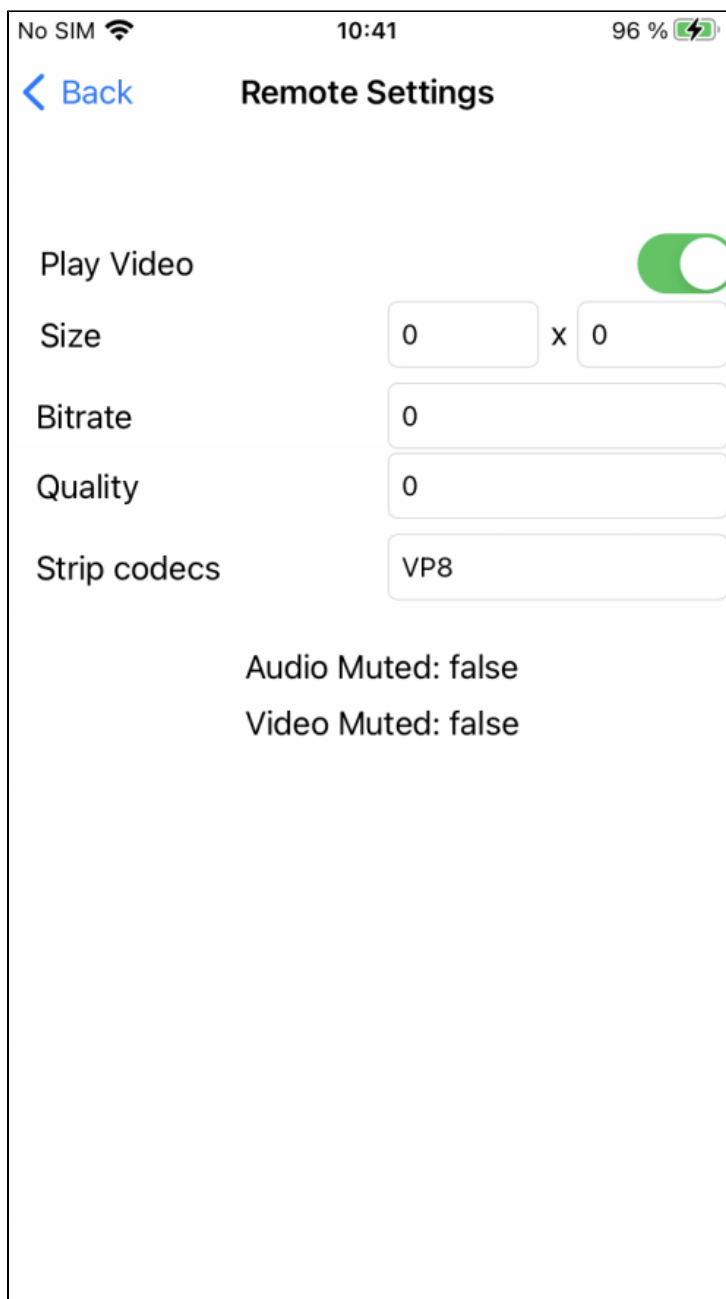
Min Bitrate (kbps)

400

Max Bitrate (kbps)

Mute Video

а вид с настройками воспроизведения - при нажатии на кнопку 'Remote settings'.



Работа с кодом примера

Для разбора кода возьмем версию примера MediaDevices Swift, которая доступна для скачивания на [GitHub](#).

Классы видов

- класс для основного вида приложения: ViewController (файл имплементации [ViewController.swift](#))
- класс для вида с настройками публикации: LocalViewController (файл имплементации [LocalViewController.swift](#))
- класс для вида с настройками воспроизведения: RemoteViewController (файл имплементации [RemoteViewController.swift](#))

1. Импорт API.

[code](#)

```
import FPWCSPi2Swift
```

2. Получение списка доступных медиа-устройств.

WCsApi2.getMediaDevices [code](#)

```
localDevices = WCsApi2.getMediaDevices()
```

3. Определение камеры и микрофона для использования по умолчанию

FPWCsApi2MediaDeviceList.audioFPWCsApi2MediaDeviceList.video[code](#)

```
if (localDevices?.audio?.count ?? 0 > 0) {  
    microphone.text = (localDevices?.audio[0] as AnyObject).label  
}  
if (localDevices?.video?.count ?? 0 > 0) {  
    camera.text = (localDevices?.video[0] as AnyObject).label  
}
```

4. Определение параметров аудио и видео для публикуемого потока.

FPWCsApi2VideoConstraints, FPWCsApi2AudioConstraints [code](#)

```
func toMediaConstraints() -> FPWCsApi2MediaConstraints {  
    let ret = FPWCsApi2MediaConstraints()  
    if (self.audioSend.isOn) {  
        let audio = FPWCsApi2AudioConstraints()  
        audio.useFEC = audioFEC.isOn  
        audio.useStereo = audioStereo.isOn  
        audio.bitrate = Int(audioBitrate.text ?? "0") ?? 0  
        ret.audio = audio  
    }  
    if (self.videoSend.isOn) {  
        let video = FPWCsApi2VideoConstraints()  
        for device in localDevices!.video {  
            if ((device as AnyObject).label == camera.text) {  
                video.deviceID = (device as AnyObject).deviceID;  
            }  
        }  
        video.minWidth = Int(videoWidth.text ?? "0") ?? 0  
        video.maxWidth = video.minWidth  
        video.minHeight = Int(videoHeight.text ?? "0") ?? 0  
        video.maxHeight = video.minHeight  
        video.minFrameRate = Int(videoFPS.text ?? "0") ?? 0  
        video.maxFrameRate = video.minFrameRate  
        video.minBitrate = Int(videoMinBitrate.text ?? "0") ?? 0  
        video.maxBitrate = Int(videoMaxBitrate.text ?? "0") ?? 0  
        ret.video = video;  
    }  
    return ret;  
}
```

5. Определение параметров для проигрываемого потока.

FPWCsApi2VideoConstraints, FPWCsApi2AudioConstraints [code](#)

```

func toMediaConstraints() -> FPWCSEApi2MediaConstraints {
    let ret = FPWCSEApi2MediaConstraints();
    ret.audio = FPWCSEApi2AudioConstraints();
    if (playVideo.isOn) {
        let video = FPWCSEApi2VideoConstraints();
        video.minWidth = Int(videoWidth.text ?? "0") ?? 0
        video.maxWidth = video.minWidth
        video.minHeight = Int(videoHeight.text ?? "0") ?? 0
        video.maxHeight = video.minWidth
        video.bitrate = Int(videoBitrate.text ?? "0") ?? 0
        video.quality = Int(videoQuality.text ?? "0") ?? 0
        ret.video = video;
    }
    return ret;
}

```

6. Локальное тестирование микрофона и камеры

WCSEApi2.getMediaAccesscode

```

@IBAction func testPressed(_ sender: Any) {
    if (testButton.title(for: .normal) == "TEST") {
        let constraints = FPWCSEApi2MediaConstraints(audio: true, video: true)!
        do {
            try WCSEApi2.getMediaAccess(constraints, localDisplay.videoView)
        } catch {
            print(error)
        }
        testButton.setTitle("RELEASE", for: .normal)
    } else {
        WCSEApi2.releaseLocalMedia(display: localDisplay.videoView);
        testButton.setTitle("TEST", for: .normal)
    }
}

```

7. Создание сессии и подключение к серверу.

WCSSession,WCSSession.connectcode

В параметрах сессии указываются:

- URL WCS-сервера
- имя серверного приложения defaultApp

```

@IBAction func connectPressed(_ sender: Any) {
    changeViewState(connectButton, false)
    if (connectButton.title(for: .normal) == "CONNECT") {
        if (session == nil) {
            let options = FPWCSEApi2SessionOptions()
            options.urlServer = urlField.text
            options.appKey = "defaultApp"
            do {
                session = try WCSSession(options)
            } catch {
                print(error);
            }
        }
        ...
        changeViewState(urlField, false)
        session?.connect()
    }
}

```

8. Публикация потока.

WCSSession.createStream, WCSSStream.publishcode

Методу createStream передаются параметры:

- options.name - имя публикуемого потока
- options.display - вид для локального отображения
- options.constraints - параметры аудио и видео
- options.stripCodecs - массив кодеков, которые должны быть исключены из SDP при публикации
- options.transport - используемый WebRTC транспорт

```
@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton, false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        let options = FPWCSTransportOptions()
        options.name = publishName.text
        options.display = localDisplay.videoView
        options.constraints = localMediaConstraints;
        options.stripCodecs = localStripCodecs?.split(separator: ",")
        options.transport = tcpTransport.isOn ? kFPWCSTransport.fpwcsTransportTCP : kFPWCSTransport.fpwcsTransportUDP;
        do {
            try publishStream = session!.createStream(options)
        } catch {
            print(error);
        }
        ...
        do {
            try publishStream?.publish()
        } catch {
            print(error);
        }
    }
}
```

9. Воспроизведение видеопотока после публикации

WCSSession.createStream, WCSSStream.playcode

Методы createStream передаются параметры:

- options.name - имя публикуемого потока
- options.display - вид для локального отображения
- options.constraints - параметры аудио и видео
- options.stripCodecs - массив кодеков, которые должны быть исключены из SDP при проигрывании
- options.transport - используемый WebRTC транспорт

```
@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton, false)
    if (playButton.title(for: .normal) == "PLAY") {
        let options = FPWCSTransportOptions()
        options.name = playName.text;
        options.display = remoteDisplay.videoView;
        options.constraints = remoteMediaConstraints;
        options.stripCodecs = remoteStripCodecs?.split(separator: ",")
        options.transport = tcpTransport.isOn ? kFPWCSTransport.fpwcsTransportTCP : kFPWCSTransport.fpwcsTransportUDP;
        do {
            playStream = try session!.createStream(options)
        } catch {
            print(error);
        }
        ...
        do {
            try playStream?.play()
        } catch {
            print(error);
        }
    }
}
```

10. Остановка воспроизведения потока.

WCSSStream.stopcode

```

@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton, false)
    if (playButton.title(for: .normal) == "PLAY") {
        ...
    } else {
        do {
            try playStream?.stop();
        } catch {
            print(error);
        }
    }
}

```

11. Остановка публикации потока.

WCStream.stop[code](#)

```

@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton, false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        ...
    } else {
        do {
            try publishStream?.stop();
        } catch {
            print(error);
        }
    }
}

```