iOS Media Devices

Example of iOS application managing media devices

This example allows to publish WebRTC stream on Web Call Server and demonstrates selection of source camera and specification of the following parameters for published and played video

- resolution (width, height)
- bitrate
- FPS (Frames Per Second) - for published video
- quality - for played video

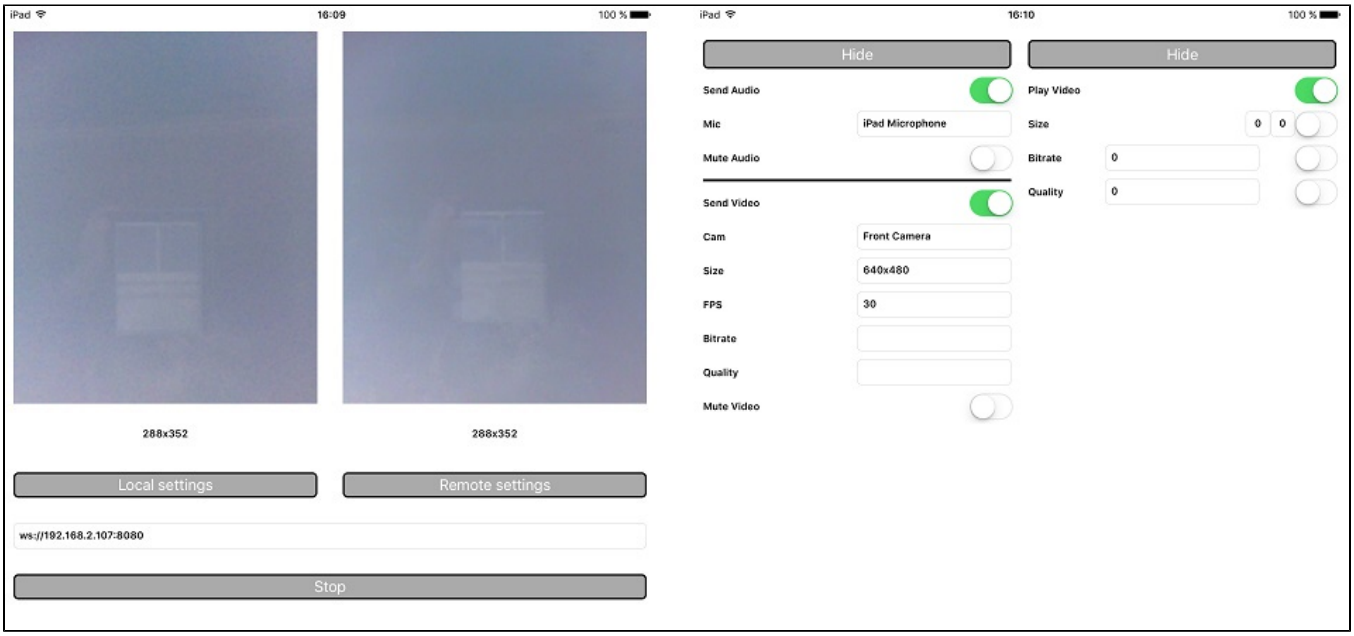
As well as publishing streams with audio and video, it allows to publish audio-only and video-only streams.
Audio and video can be muted when publishing is started (if corresponding controls has been set to ON before streaming was started), or while stream is being published.
Video streams can be played with or without video.

On the screenshot below the example is displayed when a stream is being published and played.
In the URL specified in the input field, 192.168.2.107 is the address of the WCS server.

Two videos are displayed

- left - video from the camera
- right - the published video stream is played from the server

View with controls for publishing settings is displayed when 'Local settings' button is tapped, and view with controls for playback settings - when 'Remote settings' button is tapped.



Work with code of the example

To analyze the code, let's take MediaDevices example, which can be downloaded with corresponding build 2.5.2.

View classes

- class for the main view of the application: ViewController (header file ViewController.h; implementation file ViewController.m)
- class for view with publishing settings: WCSLocalVideoControlView (header file WCSLocalVideoControl.h; implementation file WCSLocalVideoControl.m)
- class for view with playback settings: WCSRemoteVideoControlView (header file WCSRemoteVideoControl.h; implementation file WCSRemoteVideoControl.m)

1. Import of API.code

```
#import <FPWCSApi2/FPWCSApi2.h>
```

2. List available media devices.

FPWCSApi2 getMediaDevicescode

```
localDevices = [FPWCSApi2 getMediaDevices];
```

3. Default microphone and camera selection

FPWCSApi2MediaDeviceList.audio[0]code

```
_micSelector = [[WCSPickerInputView alloc] initWithLabelText:@"Mic" pickerDelegate:self];
//set default mic
if (localDevices.audio.count > 0) {
    _micSelector.input.text = ((FPWCSApi2MediaDevice *)(localDevices.audio[0])).label;
}
```

FPWCSApi2MediaDeviceList.video[0]code

```
_camSelector = [[WCSPickerInputView alloc] initWithLabelText:@"Cam" pickerDelegate:self];
//set default cam
if (localDevices.video.count > 0) {
    _camSelector.input.text = ((FPWCSApi2MediaDevice *)(localDevices.video[0])).label;
}
```

4. Constraints for published stream.

FPWCSApi2MediaConstraints.audio, FPWCSApi2MediaConstraints.videocode

```
- (FPWCSApi2MediaConstraints *)toMediaConstraints {
    FPWCSApi2MediaConstraints *ret = [[FPWCSApi2MediaConstraints alloc] init];
    if ([_sendAudio.control isOn])  {
        FPWCSApi2AudioConstraints *audio = [[FPWCSApi2AudioConstraints alloc] init];
        audio.useFEC = [_useFEC.control isOn];
        audio.useStereo = [_useStereo.control isOn];
        audio.bitrate = [_audioBitrate.input.text integerValue];
        ret.audio = audio;
    }
    if ([_sendVideo.control isOn]) {
        FPWCSApi2VideoConstraints *video = [[FPWCSApi2VideoConstraints alloc] init];
        ...
        NSArray *res = [_videoResolutionSelector.input.text componentsSeparatedByString:@"x"];
        video.minWidth = video.maxWidth = [res[0] integerValue];
        video.minHeight = video.maxHeight = [res[1] integerValue];
        video.minFrameRate = video.maxFrameRate = [_fpsSelector.input.text integerValue];
        video.bitrate = [_videoBitrate.input.text integerValue];
        ret.video = video;
    }
    return ret;
}
```

5. Constraints for played stream.

FPWCSApi2MediaConstraints.audio, FPWCSApi2MediaConstraints.videocode

```
- (FPWCSApi2MediaConstraints *)toMediaConstraints {
    FPWCSApi2MediaConstraints *ret = [[FPWCSApi2MediaConstraints alloc] init];
    ret.audio = [[FPWCSApi2AudioConstraints alloc] init];
    if ([_playVideo.control isOn]) {
        FPWCSApi2VideoConstraints *video = [[FPWCSApi2VideoConstraints alloc] init];
        video.minWidth = video.maxWidth = [_videoResolution.width.text integerValue];
        video.minHeight = video.maxHeight = [_videoResolution.height.text integerValue];
        video.bitrate = [_bitrate.input.text integerValue];
        video.quality = [_quality.input.text integerValue];
        ret.video = video;
    }
    return ret;
}
```

6.Local camera and microphone testing

FPWCSApi2 getMediaAccess, AVAudioRecorder record, AVAudioRecorder stopcode

```
- (void)testButton:(UIButton *)button {
    if ([button.titleLabel.text isEqualToString:@"Test"]) {
        NSError *error;
        [FPWCSApi2 getMediaAccess:[_localControl toMediaConstraints] display:_videoView.local error:&error];
        [_testButton setTitle:@"Release" forState:UIControlStateNormal];

        [[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryRecord error:&error];

        NSURL *url = [NSURL fileURLWithPath:@"/dev/null"];

        NSDictionary *settings = [NSDictionary dictionaryWithObjectsAndKeys:
                                  [NSNumber numberWithFloat: 44100.0],                AVSampleRateKey,
                                  [NSNumber numberWithInt: kAudioFormatAppleLossless], AVFormatIDKey,
                                  [NSNumber numberWithInt: 1],                        AVNumberOfChannelsKey,
                                  [NSNumber numberWithInt: AVAudioQualityMax],        AVEncoderAudioQualityKey,
                                  nil];

        _recorder = [[AVAudioRecorder alloc] initWithURL:url settings:settings error:&error];
        [_recorder prepareToRecord];
        _recorder.meteringEnabled = YES;
        [_recorder record];
        _levelTimer = [NSTimer scheduledTimerWithTimeInterval: 0.3 target: self selector: @selector
(levelTimerCallback:) userInfo: nil repeats: YES];
    } else {
        [FPWCSApi2 releaseLocalMedia:_videoView.local];
        [_testButton setTitle:@"Test" forState:UIControlStateNormal];

        [_levelTimer invalidate];
        [_recorder stop];

    }
}
```

7. Session creation and connection to the server.

FPWCSApi2 createSession, FPWCSApi2Session connectcode

The options include:

- URL of WCS server
- appKey of internal server-side application (defaultApp)

```
- (void)start {
    if (!_session || [_session getStatus] != kFPWCSSessionStatusEstablished || ![[_session getServerUrl]
isEqualToString:_urlInput.text]) {
        ...
        FPWCSApi2SessionOptions *options = [[FPWCSApi2SessionOptions alloc] init];
        options.urlServer = _urlInput.text;
        options.appKey = @"defaultApp";
        NSError *error;
        _session = [FPWCSApi2 createSession:options error:&error];
        ...
        [_session connect];
    } else {
        [self startStreaming];
    }
}
```

8. Stream publishing.

FPWCSApi2Session createStream, FPWCSApi2Stream publishcode

Object with next stream options is passed to createStream method:

- stream name
- view to display video

- video constraints

```
- (void)startStreaming {
    FPWCSApi2StreamOptions *options = [[FPWCSApi2StreamOptions alloc] init];
    options.name = [self getStreamName];
    options.display = _videoView.local;
    options.constraints = [_localControl toMediaConstraints];
    NSError *error;
    _localStream = [_session createStream:options error:&error];
    ...
    if(![_localStream publish:&error]) {
        UIAlertController * alert = [UIAlertController
                                     alertControllerWithTitle:@"Failed to publish"
                                     message:error.localizedDescription
                                     preferredStyle:UIAlertControllerStyleAlert];

        UIAlertAction* okButton = [UIAlertAction
                                   actionWithTitle:@"Ok"
                                   style:UIAlertActionStyleDefault
                                   handler:^(UIAlertAction * action) {
                                       [self onStopped];
                                   }];

        [alert addAction:okButton];
        [self presentViewController:alert animated:YES completion:nil];
    }
}
```

9. Preview stream playback

FPWCSApi2Session createStream, FPWCSApi2Stream playcode

Object with next stream options is passed to createStream method:

- stream name
- view to display video
- video constraints

```
- (void)startPlaying {
    FPWCSApi2StreamOptions *options = [[FPWCSApi2StreamOptions alloc] init];
    options.name = [_localStream getName];
    options.display = _videoView.remote;
    options.constraints = [_remoteControl toMediaConstraints];
    NSError *error;
    _remoteStream = [_session createStream:options error:&error];
    ...
    if(![_remoteStream play:&error]) {
        UIAlertController * alert = [UIAlertController
                                     alertControllerWithTitle:@"Failed to play"
                                     message:error.localizedDescription
                                     preferredStyle:UIAlertControllerStyleAlert];

        UIAlertAction* okButton = [UIAlertAction
                                   actionWithTitle:@"Ok"
                                   style:UIAlertActionStyleDefault
                                   handler:^(UIAlertAction * action) {
                                       if (_localStream && [_localStream getStatus] ==
kFPWCSStreamStatusPublishing) {
                                           [_localStream stop:nil];
                                       }
                                   }];

        [alert addAction:okButton];
        [self presentViewController:alert animated:YES completion:nil];
    }

}
```

10. Mute/unmute audio and video.

FPWCSApi2Stream muteAudio, unmuteAudio, muteVideo, unmuteVideocode

```
- (void)controlValueChanged:(id)sender {
    if (sender == _localControl.muteAudio.control) {
        if (_localStream) {
            if (_localControl.muteAudio.control.isOn) {
                [_localStream muteAudio];
            } else {
                [_localStream unmuteAudio];
            }
        }
    } else if (sender == _localControl.muteVideo.control) {
        if (_localStream) {
            if (_localControl.muteVideo.control.isOn) {
                [_localStream muteVideo];
            } else {
                [_localStream unmuteVideo];
            }
        }
    }
}
```

11. Stream playback stop.

FPWCSApi2Stream stopcode

```
- (void)startButton:(UIButton *)button {
    button.userInteractionEnabled = NO;
    button.alpha = 0.5;
    _urlInput.userInteractionEnabled = NO;
    if ([button.titleLabel.text isEqualToString:@"Stop"]) {
        if (_remoteStream) {
            NSError *error;
            [_remoteStream stop:&error];
        } else {
            NSLog(@"No remote stream, failed to stop");
        }
    } else {
        //start
        [self start];
    }
}
```

12. Stream publishing stop.

FPWCSApi2Stream stopcode

```
[_remoteStream on:kFPWCSStreamStatusStopped callback:^(FPWCSApi2Stream *rStream){
    [self changeStreamStatus:rStream];
    [_localStream stop:nil];
    _useLoudSpeaker.control.userInteractionEnabled = NO;

}];
```