

iOS Streamer

Example of streamer for iOS

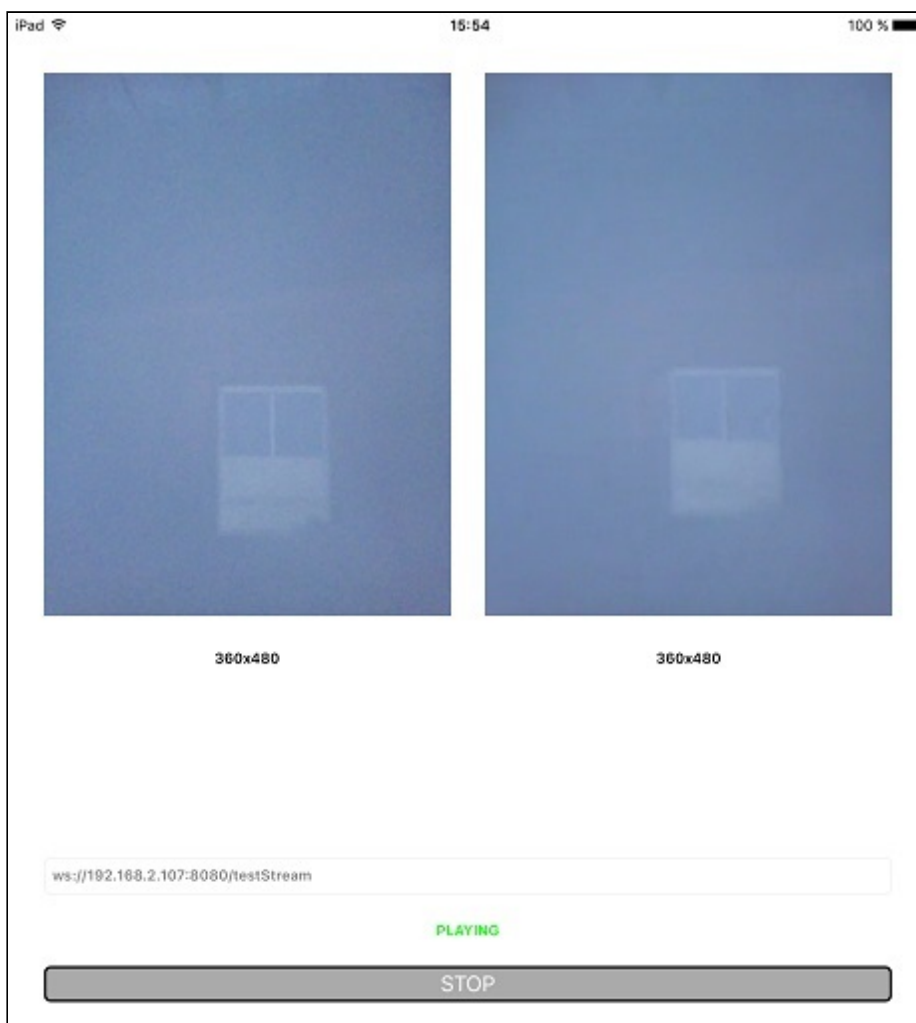
This streamer can be used to publish WebRTC video stream on Web Call Server.

On the screenshot below the example is displayed when a stream is being published. In the URL specified in the input field

- `192.168.2.107` is the address of the WCS server
- `testStream` is the stream name

Two videos are played

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



Analyzing the example code

To analyze the code, let's take Streamer example version, which is available [here](#).

View class for the main view of the application: `ViewController` (header file `ViewController.h`; implementation file `ViewController.m`).

1. Import of API

code

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

2. Session creation

`FPWCSApi2.createSession` [code](#)

The options include:

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

```
FPWCSApi2SessionOptions *options = [[FPWCSApi2SessionOptions 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 = [FPWCSApi2 createSession:options error:&error];
```

3. Connection to the server

`FPWCSApi2Session.connect` [code](#)

```
[session connect];
```

4. Receiving the event confirming successful connection

`FPWCSApi2Session.onConnected` [code](#)

On this event, `ViewController.publishStream` method is called to publish the stream

```
- (void)onConnected:(FPWCSApi2Session *)session {
    [_connectButton setTitle:@"STOP" forState:UIControlStateNormal];
}
```

```

    // [self changeViewState:_connectButton enabled:YES];
    [self publishStream];
}

```

5. Stream publishing

`FPWCSEApi2Session.createStream`, `FPWCSEApi2Stream.publish` [code](#)

Object with the following stream options is passed to `createStream` method:

- stream name
- view to display video
- video constraints (picture resolution for iPad for example)

```

- (FPWCSEApi2Stream *)publishStream {
    FPWCSEApi2Session *session = [FPWCSEApi2 getSessions][0];
    FPWCSEApi2StreamOptions *options = [[FPWCSEApi2StreamOptions alloc] init];
    options.name = streamName;
    options.display = _videoView.local;
    if ( UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad ) {
        options.constraints = [[FPWCSEApi2MediaConstraints alloc]
initWithAudio:YES videoWidth:640 videoHeight:480 videoFps:15];
    }
    NSError *error;
    publishStream = [session createStream:options error:&error];
    ...
    if(![publishStream publish:&error]) {
        UIAlertController * alert = [UIAlertController
            alertControllerWithTitle:@"Failed to
publish"
            message:error.localizedDescription
            preferredStyle:UIAlertControllerStyleAlert];

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

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

```

6. Receiving the event confirming successful stream publishing

`FPWCSEApi2Stream.onPublishing` [code](#)

On this event, `ViewController.playStream` method is called to play published stream preview

```
- (void)onPublishing:(FPWCSApi2Stream *)stream {
    [self playStream];
}
```

7. Preview stream playback

`FPWCSApi2Session.createStream`, `FPWCSApi2Stream.play` code

Object with the following stream options is passed to `createStream` method:

- stream name
- view to display video

```
- (FPWCSApi2Stream *)playStream {
    FPWCSApi2Session *session = [FPWCSApi2 getSessions][0];
    FPWCSApi2StreamOptions *options = [[FPWCSApi2StreamOptions alloc] init];
    options.name = streamName;
    options.display = _videoView.remote;
    NSError *error;
    playStream = [session createStream:options error:nil];
    ...
    if(![playStream play:&error]) {
        UIAlertController * alert = [UIAlertController
            alertControllerWithTitle:@"Failed to
play"
            message:error.localizedDescription
            preferredStyle:UIAlertControllerStyleAlert];

        UIAlertAction* okButton = [UIAlertAction
            initWithTitle:@"Ok"
            style:UIAlertActionStyleDefault
            handler:^(UIAlertAction * action) {

            }];

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

8. Disconnection

`FPWCSApi2Session.disconnect` code

```
- (void)connectButton:(UIButton *)button {
    [self changeViewState:button enabled:NO];
}
```

```

    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];
    }
}

```

9. Receiving the event confirming successful disconnection

`ViewController.onDisconnected` code

```

- (void)onDisconnected {
    [_connectButton setTitle:@"START" forState:UIControlStateNormal];
    [self changeViewState:_connectButton enabled:YES];
    [self changeViewState:_connectUrl enabled:YES];
    [self onUnpublished];
    [self onStopped];
}

```