

# iOS Player

## Example of player for iOS

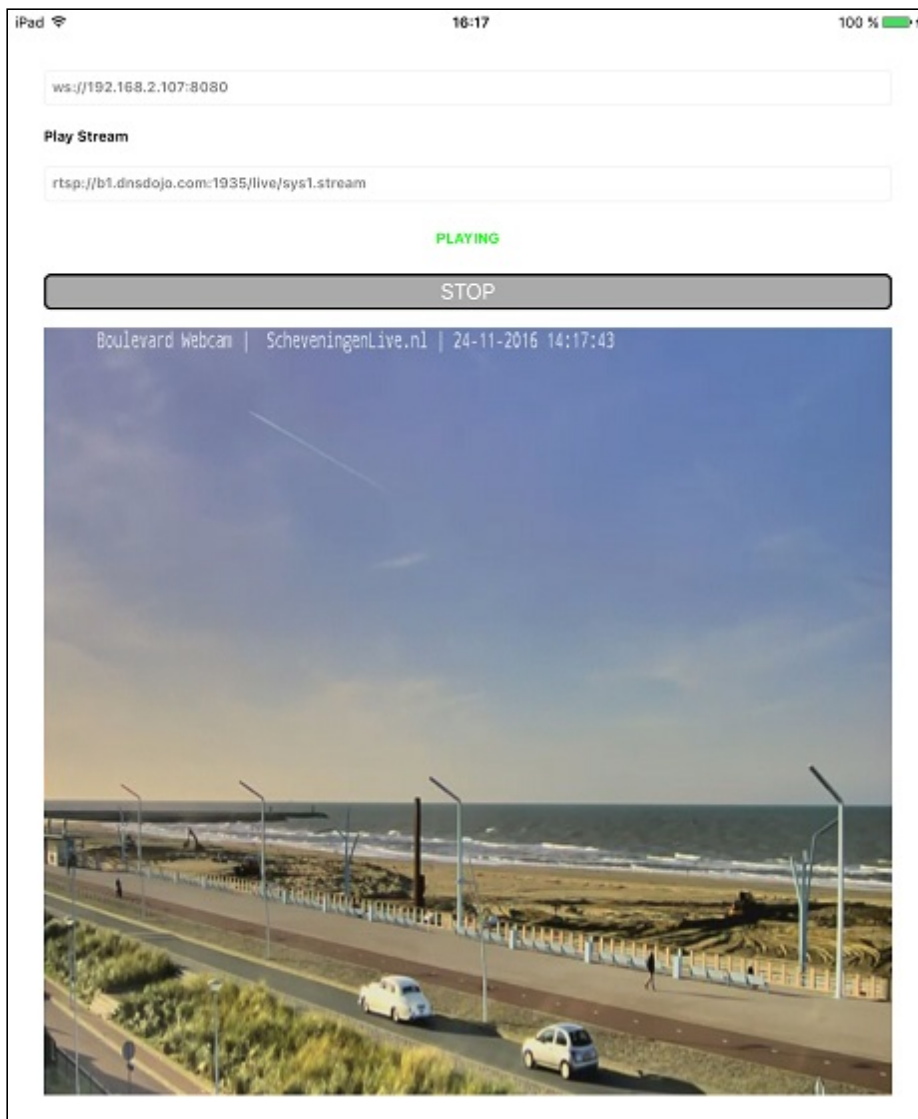
This player can be used to play any type of stream on Web Call Server

- RTSP
- WebRTC
- RTMP

On the screenshot below an RTSP stream is being playing.

In the input fields

- 192.168.2.107 in the URL is the address of the WCS server
- stream name is entered in to the 'Play Stream' field (RTSP URL in this case)



## Analyzing the example code

To analyze the code, let's take Player example, 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];
options.urlServer = _connectUrl.text;
options.appKey = @"defaultApp";
NSError *error;
FPWCSApi2Session *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, stream playback is started

```
- (void)onConnected:(FPWCSApi2Session *)session {
    [self changeViewState:_remoteStreamName enabled:NO];
    [self playStream];
}
```

### 5. 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 = _remoteStreamName.text;
    options.display = _remoteDisplay;
    NSError *error;
    FPWCSApi2Stream *stream = [session createStream:options error:nil];
    ...
    if(![stream 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 stream;
}

```

## 6. Disconnection

`FPWCSEApi2Session.disconnect` [code](#)

```

- (void)startButton:(UIButton *)button {
    [self changeViewState:button enabled:NO];
    if ([button.titleLabel.text isEqualToString:@"STOP"]) {
        if ([FPWCSEApi2 getSessions].count) {
            FPWCSEApi2Session *session = [FPWCSEApi2 getSessions][0];
            NSLog(@"Disconnect session with server %@", [session
getServerUrl]);
            [session disconnect];
        } else {
            NSLog(@"Nothing to disconnect");
            [self onDisconnected];
        }
    } else {
        [self changeViewState:_connectUrl enabled:NO];
        [self connect];
    }
}

```

## 7. Receiving the event confirming successful disconnection

`FPWCSEApi2Session.onDisconnected` [code](#)

```

- (void)onDisconnected {
    [self changeViewState:_connectUrl enabled:YES];
    [self onStopped];
}

```