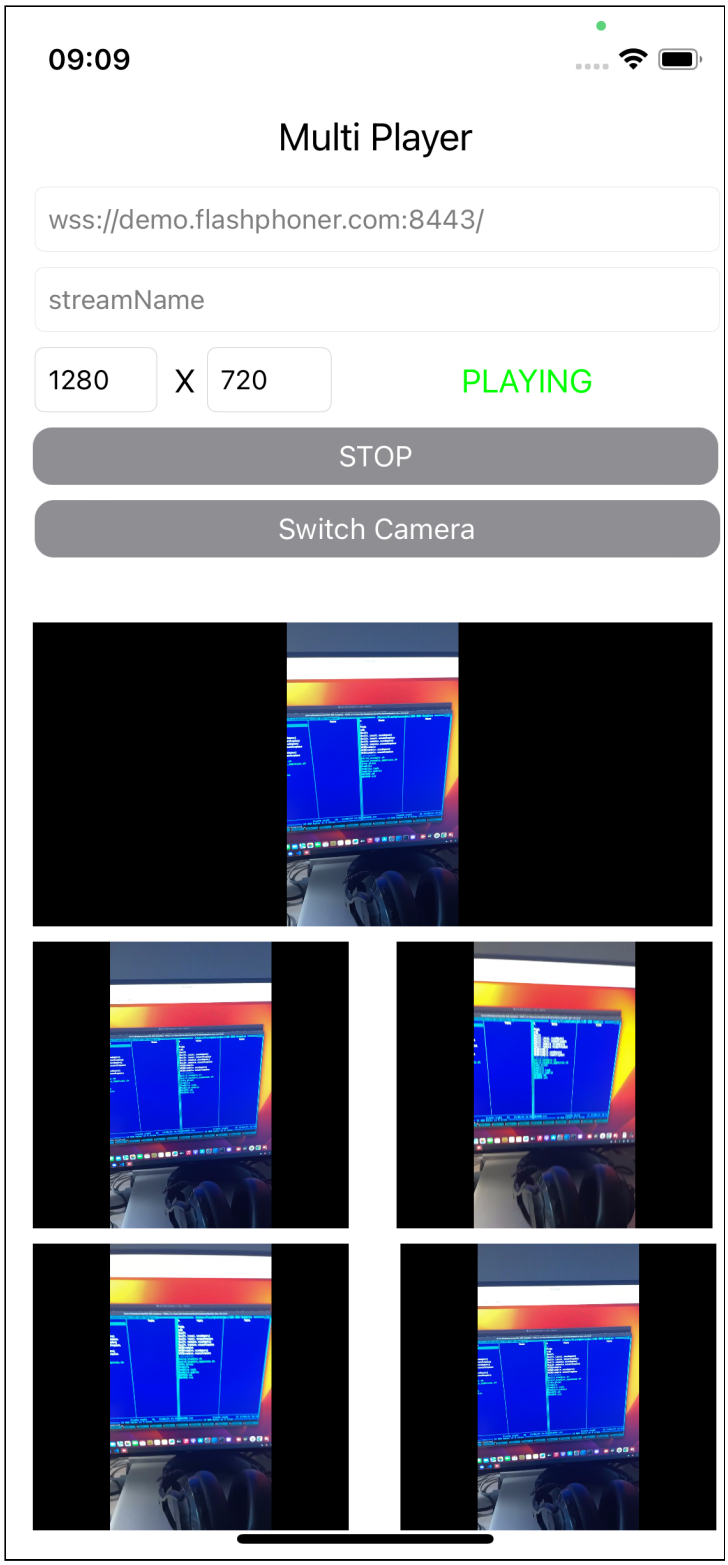
iOS MultiPlayer Swift

Overview

The example shows how to publish a WebRTC stream to a server and play multiple streams from the server (one-to-mant videochat). It may be also used as memory consumption test while decoding multiple high resolution streams.

On the screenshot below the 1280x720 stream is publishing and playing. The published stream from the device camera is displayed above, and four playing streams are displayed below



Analyzing the code

To analyze the code let's take MultiPlayerSwift example, which can be downloaded from GitHub.

The class for main view of the application: MultiPlayerController (implementation file MultiPlayerController.swift).

1. Import of API

code

```
import FPWCSApi2Swift
```

2. Session creation and connection to server

`WCSSession`, `WCSSession.connect` code

The session options include:

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

```
@IBAction func startPressed(_ sender: Any) {
    ...

    if (startButton.title(for: .normal) == "PUBLISH AND PLAY") {
        if (session == nil) {
            let options = FPWCSApi2SessionOptions()
            options.urlServer = urlField.text
            options.appKey = "defaultApp"
            do {
                try session = WCSSession(options)
            } catch {
                print(error)
            }
        }
        ...
        session?.connect()
    } else {
        ...
    }

}
```

3. Stream publishing

`WCSSession.createStream`, `WCSStream.publish` code

The following stream options are passed to `createStream` method:

- stream name
- view to display video

- width and height publishing constraint

```
func publish() {
    let options = FPWCSApi2StreamOptions()
    options.name = streamName.text
    options.display = localDisplay.videoView

    let ret = FPWCSApi2MediaConstraints()

    let video = FPWCSApi2VideoConstraints()
    video.minWidth = Int(widthField.text ?? "0") ?? 0
    video.maxWidth = video.minWidth
    video.minHeight = Int(heightField.text ?? "0") ?? 0
    video.maxHeight = video.minHeight
    ret.video = video

    options.constraints = ret
    do {
        publishStream = try session!.createStream(options)
    } catch {
        print(error);
    }

    ...
    do {
        try publishStream?.publish()
    } catch {
        print(error);
    }
}
```

4. Start four stream instances playback after successful publishing

code

```
fileprivate func onPublishing(_ stream:FPWCSApi2Stream) {
    playStreamLT = self.play(display: remoteDisplayLT);
    playStreamRT = self.play(display: remoteDisplayRT);
    playStreamLB = self.play(display: remoteDisplayLB);
    playStreamRB = self.play(display: remoteDisplayRB);

    startButton.setTitle("STOP", for:.normal)
    changeViewState(startButton, true);
    changeViewState(switchCameraButton, true);
}
```

5. Stream playback

`WCSSession.createStream`, `WCSStream.play` code

The following stream options are passed to createStream method:

- stream name
- view to display video

```
func play(display: WebRTCView) -> WCSStream? {
    let options = FPWCSApi2StreamOptions()
    options.name = streamName.text;
    options.display = display.videoView;
    do {
        let playStream = try session!.createStream(options)

        ...
        try playStream.play()
        return playStream;
    } catch {
        print(error);
    }
    return nil;
}
```

6. Camera switching

`WCSStream.switchCamera` code

```
@IBAction func switchCameraPressed(_ sender: Any) {
    publishStream?.switchCamera()
}
```

7. Playback stop when publishing is stopped

`WCSStream.stop` code

```
fileprivate func onUnpublished() {
    do {
        try playStreamLT?.stop();
    } catch {
        print(error);
    }
    do {
        try playStreamRT?.stop();
    } catch {
        print(error);
    }
    do {
        try playStreamLB?.stop();
    } catch {
        print(error);
    }
    do {
        try playStreamRB?.stop();
    } catch {
        print(error);
```

```
    }
}
```

8. Closing the connection

`WCSSession.disconnect` code

```
@IBAction func startPressed(_ sender: Any) {
    changeViewState(startButton, false)
    changeViewState(urlField, false)
    changeViewState(streamName, false)

    if (startButton.title(for: .normal) == "PUBLISH AND PLAY") {
        ...
        session?.connect()
    } else {
        session?.disconnect()
    }

}
```