iOS MCU Client Swift

Example of client for MCU conference participant

This example can be used to arrange an MCU video conference on Web Call Server. Each participant of such conference can publish a WebRTC stream and play a mixer stream with audio and video from the other participants and its own video (without its own audio).

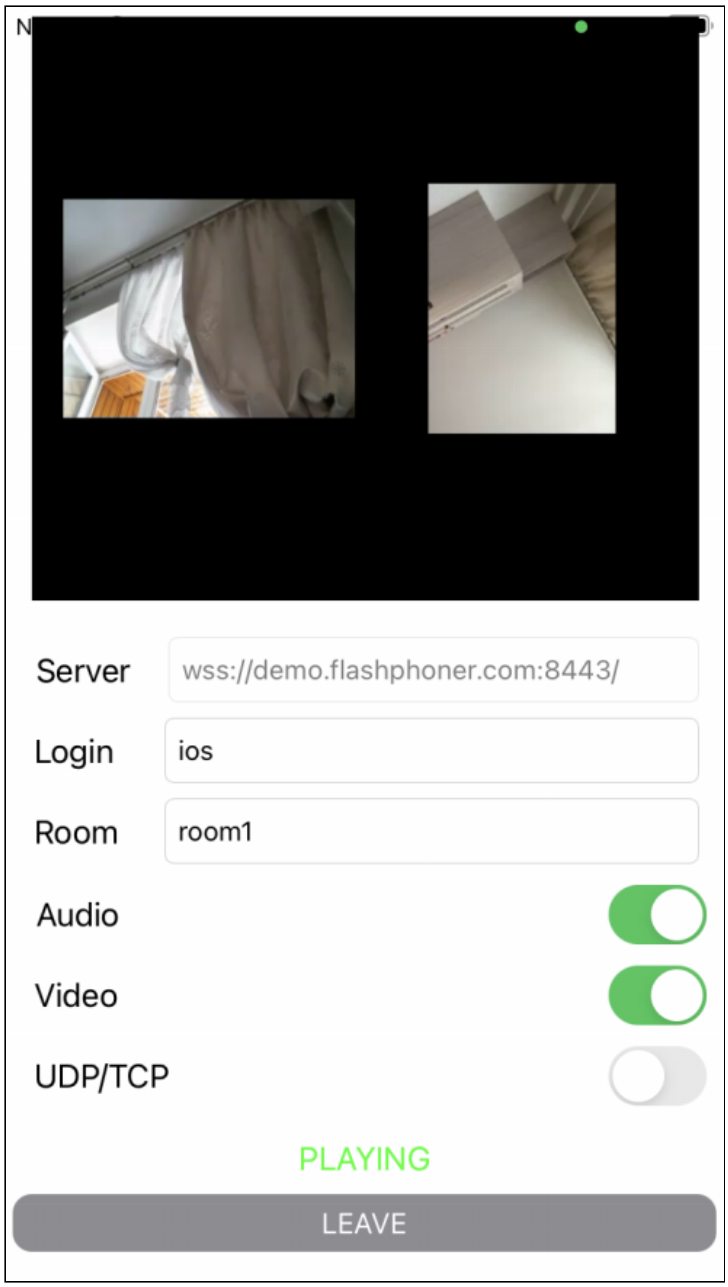
The following settings are required in WCS flashphoner.properties

```
mixer_auto_start=true
mixer_mcu_audio=true
mixer_mcu_video=true
```

When a participant joins a conference using the client

- a stream with video of the participant, named `participantName + "\#" + roomName`, is published
- the participant's stream is added to mixer named `roomName` (in case such mixer did not exist, it is auto created)
- a new mixer output stream named `roomName + "-" + participantName + roomName` and containing video from all the participants (including this one) and audio only from the other participants is created and played for the participant

On the screenshot below the participant is publishing a stream and playing his conference mixer stream:



Analyzing the example code

To analyze the code take MCUClientSwift class which is available on GitHub.

Main application view class: `MCUViewController` (implementation file MCUViewController.swift).

1. API import

code

```
import FPWCSApi2Swift
```

2. Session creation and connecting to the server

`WCSSession`, `WCSSession.connect` code

The following session parameters are set:

- WCS server URL
- server REST hook application name `defaultApp`

```
@IBAction func joinPressed(_ sender: Any) {
    self.changeViewState(joinButton, false)
    if (joinButton.title(for: .normal) == "JOIN") {
        if (session == nil) {
            let options = FPWCSApi2SessionOptions()
            options.urlServer = serverField.text
            options.appKey = "defaultApp"
            do {
                try session = WCSSession(options)
            } catch {
                print(error)
            }
        }
        ...
        self.changeViewState(serverField, false)
        session?.connect()
    } else {
        leave()
    }

}
```

3. Participant stream publishing

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

The following parameters are passed to `createStream` method:

- stream name to publish
- local view to display
- WebRTC transport type
- audio and video publishing constraints

```
func publish() {
    ...
    let constraints = FPWCSApi2MediaConstraints()
    if (audioSwitch.isOn) {
        constraints.audio = FPWCSApi2AudioConstraints()
    }
    if (videoSwitch.isOn) {
        constraints.video = FPWCSApi2VideoConstraints()
    }
```

```
    let options = FPWCSApi2StreamOptions()
    options.name = loginField.text! + "#" + roomField.text!
    options.transport = transportSwitch.isOn ?
kFPWCSTransport.fpwcsTransportTCP : kFPWCSTransport.fpwcsTransportUDP
    options.constraints = constraints
    options.display = localDisplay.videoView
    do {
        publishStream = try session!.createStream(options)
    } catch {
        print(error);
    }
    ...
    do {
        try publishStream?.publish()
    } catch {
        print(error);
    }
}
```

4. MCU mixer stream playback

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

The following parameters are passed to `createStream` method:

- stream name to play
- remote view to display
- WebRTC transport type

```
func play() {
    ...
    let options = FPWCSApi2StreamOptions()
    options.name = roomField.text! + "-" + loginField.text! + roomField.text!
    options.transport = transportSwitch.isOn ?
kFPWCSTransport.fpwcsTransportTCP : kFPWCSTransport.fpwcsTransportUDP
    options.display = remoteDisplay.videoView;
    do {
        playStream = try session!.createStream(options)
    } catch {
        print(error)
    }
    ...
    do {
        try playStream?.play()
    } catch {
        print(error);
    }
}
```

5. Stream playback stopping

`WCSStream.stop` code

```
func stopPlay() {
    do {
        try playStream?.stop();
    } catch {
        print(error);
    }
    playStream = nil;
}
```

6. Stream publishing stopping

`WCSStream.stop` code

```
func stopPublish() {
    do {
        try publishStream?.stop()
    } catch {
        print(error);
    }
    publishStream = nil;
}
```

7. Connection closing

`WCSSession.disconnect` code

```
func leave() {
    session?.disconnect()
    session = nil;
}
```