

# 1) Clone Project & add Agora RTC SDK via CocoaPods

**git clone <https://github.com/sidsharma27/OpenLive-iOS> -b StartProject**

---

**Podfile should already be in your starter project, open the Podfile via Terminal**

`open Podfile`

**Double check the dependency is included in the Podfile**

```
target 'OpenLive' do
  use_frameworks!
  pod 'AgoraRtcEngine_iOS'
end
```

**Install the pod & open the newly created .xcworkspace file**

```
pod install
```

**Add Privacy Settings – Info.plist**

---



**2) Create & Initialize  
'AgoraRtcEngineKit' object**

---

## Create the 'AgoraRtcEngineKit' object and initialize it class as a singleton instance

---

```
import AgoraRtcEngineKit

var rtcEngine : AgoraRtcEngineKit!
func loadAgoraKit() {
    rtcEngine = AgoraRtcEngineKit.sharedEngine(withAppId: KeyCenter.AppID, delegate: s
}
```

## 3) Set Channel Profile

---

### Set the Channel Profile to Live Broadcast

---

```
rtcEngine.setChannelProfile(.channelProfile_LiveBroadcasting)
```

## 4) Enable Video Mode

---

Enables video data to be sent to stream, without this  
-> Audio Only

---

```
rtcEngine.enableVideo()
```

## 5) Set Channel Profile

---

Tells AgoraRtcEngine what type of channel it is in

order to optimize the particular call

---

```
engine.setChannelProfile(.channelProfile_LiveBroadcasting)
```

## 6) Set video resolution

---

**Sets the video encoding profile (FPS / Resolution)**

---

```
engine.setVideoProfile(videoProfile, swapWidthAndHeight:true)
```

## 7) Set user role

---

**Tells RtcEngine which role the member is: Audience or Broadcaster**

---

```
rtcEngine.setClientRole(clientRole, withKey: nil)
```

## 8) Enable dual stream mode

---

**Allows the ability to have two different types of stream modes (high and low)**

---

```
rtcEngine.enableDualStreamMode(true)
```

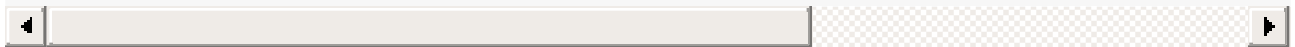
## 9) Set Remote Video Stream

---

**Adjust stream type based on size of UI windows to save bandwidth & calculation resources (in `setStreamType()`)**

---

```
rtcEngine.setRemoteVideoStream(UInt(session.uid), type: (session == fullScreenSession ?  
rtcEngine.setRemoteVideoStream(UInt(session.uid), type: .videoStream_High)
```



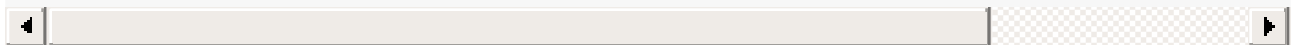
## 10) Join Channel

---

**Join the channel with App ID as Key (unsecure alternative instead of Dynamic Key - demo purposes only)**

---

```
rtcEngine.joinChannel(byKey: KeyCenter.AppID, channelName: roomName, info: nil, uid: 0
```



## 11) Leave Channel

---

**Stop the local preview, unbind the view, and leave the channel ( in `leaveChannel()` )**

---

```
rtcEngine.setupLocalVideo(nil)  
rtcEngine.leaveChannel(nil)
```

## 12) Set the local/remote video view

---

**VideoSession** is an object that contains the information regarding an individual video session

---

```
var canvas: AgoraRtcVideoCanvas!  
canvas = AgoraRtcVideoCanvas()  
canvas.uid = UInt(uid)  
canvas.renderMode = .render_Hidden  
canvas.view = hostingView
```

### Set remote video view in rtcEngine callback (didJoinedOfUid)

---

```
rtcEngine.setupRemoteVideo(userSession.canvas)
```

### Set local video view in addLocalSession()

---

```
rtcEngine.setupLocalVideo(localSession.canvas)
```

## 13) Setup additional features

---

### Switch camera (toggle front/back)

---

```
@IBAction func doSwitchCameraPressed (_sender:UIButton) {  
    rtcEngine?.switchCamera()  
}
```

## Mute Local Audio Stream (don't send audio stream)

---

```
fileprivate var isMuted = false {  
    didSet{  
        rtcEngine?.muteLocalAudioStream(isMuted)  
    }  
}
```

## Join broadcast (audience member joins as Guest broadcaster)

---

```
@IBAction func doBroadcastPressed(_sender:UIButton) {  
    if isBroadcaster{  
        clientRole= .clientRole_Audience  
    } else {  
        clientRole= .clientRole_Broadcaster  
    }  
    rtcEngine.setClientRole(clientRole, withKey: nil)  
}
```

## Run the app!

---