# Seajei Developer Guide iOS

Version 3.1


**Introduction** **1**

**SeajeiDemo XCode sample programs** **2**

**Getting started with the Seajei solution** **3**

Import the necessary frameworks 3

**Initializing frameworks** **4**

**Establish network connection to Smart Home Device** **5**

**Display incoming video stream** **5**

**Play incoming audio stream** **5**

**Capture audio and stream to Smart Home Device** **5**

**Push Notifications** **6**

Enable push notifications to work in your app 6

Create app and set Bundle Identifier 6

Create new identifier in developer account 6

Create app in iTunes Connect 6

Create Certificate and set up AWS SNS 7

Receive and deal with push notifications 7


# Introduction

The Seajei SDK enables split-second connection and highly reliable video/audio streaming with ultra-low lag between phone apps and embedded systems such as Smart Home video cameras.

It can establish direct connections over the same Wifi, or connections across the internet using Wifi or cellular data (it will use peer-to-peer when possible, or server relay otherwise). The only requirement is that both devices are connected to the network.

The SDK has been designed to be easy to integrate and has a simple API that only requires a few lines of code and then it just works!

The SDK contains frameworks for iOS, and libraries for Linux / embedded systems (see SeajeiDeveloperGuideLinux).

The SDK currently contains 5 frameworks for iOS:

- CjConnectivity
- CjVideoPlayer
- CjVideoCapture
- CjAudioPlayer
- CjAudioCapture

The communication framework is called CjConnectivity. It is responsible for establishing the connection and streaming video, audio and any other type of data. It will resend lost packets, and indicate to the host device whether to lower or increase bitrate depending on network conditions.

The CjVideoPlayer framework displays H.264 video frames in a UIView.

The CjVideoCapture framework enables the iOS device to capture H.264 video frames at 30 frames per second. Resolution currently can be set between 240p and 1080p, and can be updated on the fly.

The CjAudioPlayer framework can play audio frames in PCM or Opus formats.

And finally, the CjAudioCapture framework can capture audio frames from the iOS device's microphone, in PCM or Opus formats.

# SeajeiDemo XCode sample programs

Compile and run the included SeajeiDoorbellSimulator (*SamplePrograms/iOS/SeajeiDoorbellSimulator*) and SeajeiDemoApp (*SamplePrograms/iOS/SeajeiDemoApp*) XCode projects for a working sample on how to connect, stream and display video, and stream and play audio both ways, as well as send push notifications and act upon receiving them.

Run the samples on 2 separate phones, so you can press the doorbell simulator button, open the app via notification, and see video and 2 way-audio.

Run the Raspberry Pi doorbell sample (*SamplePrograms/RaspberryPi/doorbell.c*) on your Raspberry Pi and connect to it from the SeajeiDemoApp.

*Note:*

- *if running the demo app on the iOS Simulator, the video may not play smoothly. This appears to be an issue with the iOS Simulator. Run the demo app on a phone for proper results.*
- *if running the doorbell simulator on the iOS simulator, because live video capture won't work, it will stream a video from a file instead. Run the doorbell simulator on a phone for live video to work.*

# Getting started with the Seajei solution

This tutorial shows how to establish a network connection from an iOS app to an embedded system such as a Smart Home doorbell (running the Linux version of the SDK).

Next it shows how to display the incoming video stream in real-time, and how to play the incoming audio stream.

Then it shows how to capture audio on the iOS device and stream it to the embedded system for a 2-way audio experience.

Lastly it shows how to receive and act upon a push notification received from the embedded system after its button was pressed.

## Import the necessary frameworks

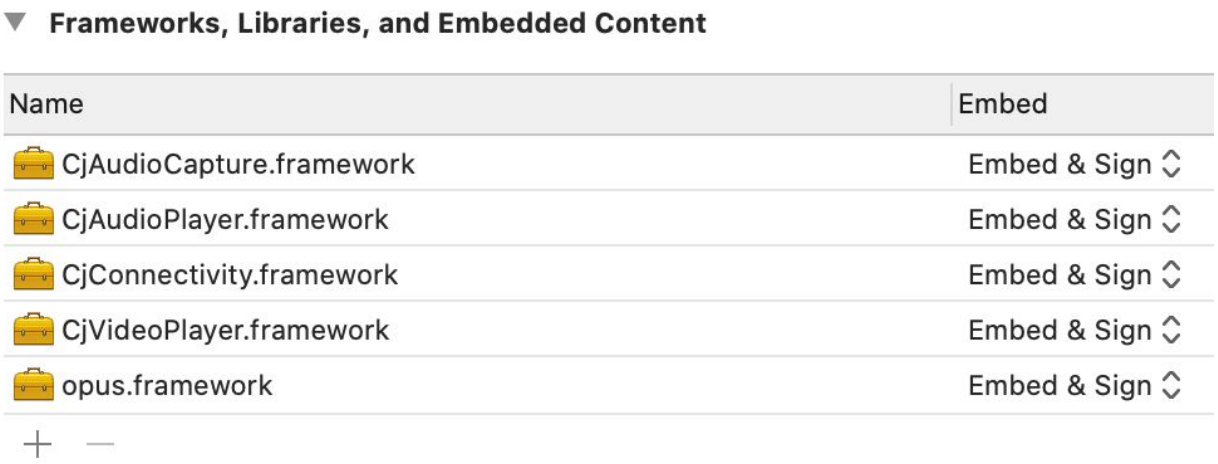
Grab the following frameworks from the SDK in the *Libraries/iOS* folder, and copy them to the folder or subfolder where your XCode project is:

- CjConnectivity.framework
- CjVideoPlayer.framework
- CjAudioPlayer.framework
- CjAudioCapture.framework

*Note: the frameworks need to be copied to your project's folder.*

If you are using one of the audio frameworks, you will also need to copy *opus.framework* found in *Libraries/iOS/OpusAudioCodec*. For development, use *opus.framework* in subfolder *iphoneos_and_iphonesimulator*. For submitting to the App Store, use *opus.framework* in subfolder *iphoneos*.

Then in your app's general settings, drag the frameworks to "Frameworks, Libraries, and Embedded Content":




Make sure that under Embed, "Embed & Sign" is selected.

If you included the CjAudioCapture or CjAudioPlayer framework, you will need to disable bitcode from your project (this is because of the prebuilt Opus binary they use). To do so, go to your Build Settings, and search for "bitcode", and set "Enable Bitcode" to NO.

# Initializing frameworks

As soon as the app has started, create the CjConnectivity object and call its `initWithToken:myDeviceId:` method.

It is important to call this method as soon as possible, because it will make the library connect to its signaling service, so that when a connection attempt is made it will happen as quickly as possible.

You can use the free trial token (*free-trial-64-4234-89c8-e1732f71059e*). It offers full features, with these exceptions:

- network bandwidth is limited to 3 Mbps when relayed via server
- session length is limited to 3 minutes
- no push notifications

If you want to remove those limitations by getting your own token, contact us at support@seajei.com.

`myDeviceId` could for example be an email or other unique ID on the phone, and a MAC address on the embedded system.

Set the delegate for your CjConnectivity object, and look for events in the delegate's `connectionEventReceived:otherDeviceId:connectionType:` method.

As soon as the signaling service has been established, the `CoConnectionEventMessagingReady` event will be received, which indicates that the library is ready to send and receive messages. Hint: do not wait for this event (or any event) to start a data connection.

Call the init functions for the other frameworks and set their delegates.

# Establish network connection to Smart Home Device

Initiate the establishment of a data connection that will enable to stream video and audio 2-way. The assumption here is that the destination unique ID is known. In a real life scenario the unique destination ID of the Smart Home camera would have been obtained during provisioning/setup. The function to call is `connectToOtherDeviceId:`.

At this point, the `connectionEventReceived:otherDeviceId:connectionType:` delegate will tell you whether the connection was successful, or what the error was in case it failed.

# Display incoming video stream

Assuming the connection was successful, the iOS device is now ready to receive any type of data (such as audio/video) via its `dataReceived:dataType:otherDeviceId:connectionType:` delegate.

When receiving video data, check if the dataType is `CoDataTypeVideo`. If that's the case, forward the data directly to the video player by calling its `displayH264Nalus:` function.

# Play incoming audio stream

This is similar to receiving and playing video, except we check if the dataType is `CoDataTypeAudio`.

We then play the audio by forwarding the data directly to the audio player by calling its `queueAudioFrames:` function.

# Capture audio and stream to Smart Home Device

After the CjAudioCapture framework has been initialized, call its `start` function so it starts capturing audio frames. Call the `pause` and `resume` functions as necessary.

The `CjAudioCaptureDelegate` will get called when audio frames have been captured.

If during initialization audio format was set to use Opus, simply send the captured audio data to the smart home device by calling the `sendAudioFrames:` function.

If the audio format is PCM, encode the audio frames with an audio codec and send data to the smart home device.

*Note: the sample programs use push-to-talk in order to have the audio stream only go in one direction at a time, which resolves audio echo problems. Some smaller embedded systems are not powerful enough to do echo cancellation and push-to-talk is a way around that problem.*

# Push Notifications

## Enable push notifications to work in your app

For push notifications to work in your own app, you need to do some setting up in iTunes Connect and obtain a security certificate. Follow these steps.

### Create app and set Bundle Identifier

When you create your app in XCode, set the Bundle Identifier in General settings. For this example let's suppose we specify the Bundle Identifier as "com.mydomain.doorbelldemo".

If you use the SeajeiDemoApp from the SDK, you can simply rename the existing identifier to your own domain. For example rename "com.seajei.doorbelldemo" to "com.mydomain.doorbelldemo".

### Create new identifier in developer account

Go to https://developer.apple.com/account/resources/identifiers/list to create the new identifier:

1. Click on +
2. Select App IDs and click continue
3. Select App and click continue
4. Give a description, and set bundle ID to same identifier as XCode (for example com.mydomain.doorbelldemo). Select Explicit. Then set Capabilities:
    a. Select "Push Notifications", then click continue
5. Click Register

### Create app in iTunes Connect

Next go to itunes connect (https://itunesconnect.apple.com/):

1. Click on My Apps.
2. Click on + and New App
3. Under Platforms select iOS. Give a name to the app and select the Bundle ID we just created. Set some unique ID for SKU

Now the app is ready to be uploaded. But there is no need to upload for now as the notifications will also work in Sandbox mode when running the app from XCode.

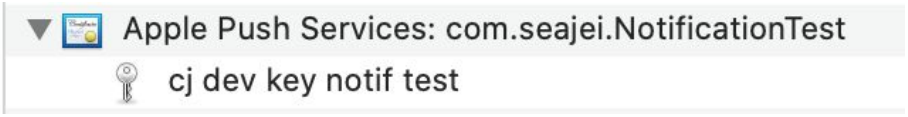
**Note**: the XCode simulator cannot receive notifications. So run the app that needs to receive notifications on an iOS device.

### Create Certificate and set up AWS SNS

Next step is to create a certificate.

Go to https://developer.apple.com/account/resources/certificates/list:

1. Click on +
2. Under Services select "Apple Push Notification service SSL (Sandbox & Production)"
3. Click Continue.
4. Pick the App ID previously created. Click Continue.
5. Click on "Learn more" link and follow instructions.
6. Click on Choose File and pick file that was just created. Click Continue.
7. Click on Download. Save .cer file somewhere. Then double click on it.
8. Re-launch Keychain Access app if it had been closed, click on "My Certificates", and look for a certificate called Apple Push Services: followed by bundle id, and expand it




9. Right click on key (in this example "cj dev key notif test"), and click on Export. A window popup will ask to create and verify a password. Leave it empty and click ok. Another window will pop up asking for your Mac account password. Put it in and click Allow. It will save a .p12 file.

Send the .p12 file to support@seajei.com. We will get back to you within 24 hours with a new token.

Replace the free trial token with your new token in your app(s) and in the raspberry pi programs, and the push notifications will now work.

## Receive and deal with push notifications

In order to receive push notifications from the smart home device, we call the `registerPushNotificationsFromDevice:myDeviceToken:useSandbox:`

function. Pass YES to useSandbox parameter when debugging, and NO when building your app for release (you can use the DEBUG flag to check):

```
#ifdef DEBUG
        [self.connectivity
registerPushNotificationsFromDevice:deviceIdToConnectTo
myDeviceToken:deviceToken useSandbox:YES];
#else
        [self.connectivity
registerPushNotificationsFromDevice:deviceIdToConnectTo
myDeviceToken:deviceToken useSandbox:NO];
#endif
```

The device token (not to be confused with the Seajei token) needs to be taken from the `application:didRegisterForRemoteNotificationsWithDeviceToken:` function in the app delegate.

After that, when a push notification is received while the app is in the foreground, the `userNotificationCenter:willPresentNotification:withCompletionHandler:` will be called in the app delegate.

If the app was in the background or not running at all when the notification was received, the `userNotificationCenter:didReceiveNotificationResponse:withCompletionHandler:` function will be called in the delegate after the user has tapped the notification.