



# **Agora Native SDK for iOS**

## **Reference Manual v1.5**

[support@agora.io](mailto:support@agora.io)

# Content

Introduction .....	5
Agora CaaS .....	5
Agora Native SDK for iOS .....	5
Requirements.....	6
Required Development Environment .....	6
Required Libraries.....	6
Required Documents .....	6
Getting Started .....	8
Where to Get the SDK.....	8
About Keys .....	8
Obtaining and Using a Vendor Key .....	10
Obtaining and Using a Dynamic Key .....	11
Creating a New Project .....	14
Using the Demo Application .....	14
Compiling the Demo Program .....	15
Executing the Demo Program .....	16
Agora Native SDK for iOS API Reference .....	20
AgoraRtcEngineKit Interface Class.....	20
Initialize (sharedEngineWithVendorKey).....	20
Enable Video Mode (enableVideo) .....	20
Enable Audio Mode (disableVideo) .....	20
Start Video Preview (startPreview) .....	21
Stop Video Preview (stopPreview).....	21
Join Channel (joinChannelByKey) .....	21
Leave Channel (leaveChannel) .....	22
Renew Channel Dynamic Key (renewChannelDynamicKey) .....	23
Retrieve Current Call ID (getCallId).....	23
Rate the Call (rate) .....	23
Complain about Call Quality (complain) .....	24
Start Audio Call Test (startEchoTest).....	24
Stop Audio Call Test (stopEchoTest).....	24
Enable Network Test (enableNetworkTest).....	25
Disable Network Test (disableNetworkTest) .....	25
Generate Call Quality Report URL (makeQualityReportUrl).....	25
Mute Local Audio Stream (muteLocalAudioStream) .....	26
Mute All Remote Audio Streams (muteAllRemoteAudioStreams) .....	26
Mute Certain Remote Audio Stream (muteRemoteAudioStream) .....	26
Enable Speakerphone (setEnableSpeakerphone).....	27

<i>Is Speakerphone Enabled? (isSpeakerphoneEnabled)</i> .....	27
<i>Enable Audio Volume Indication (enableAudioVolumeIndication)</i> .....	27
<i>Register Packet Observer (registerAgoraPacketObserver)</i> .....	28
<i>Enable Built-in Encryption(setEncryptionSecret)</i> .....	28
<i>Set Video Profile (setVideoProfile)</i> .....	28
<i>Set Local Video View (setupLocalVideo)</i> .....	30
<i>Set Remote Video View (setupRemoteVideo)</i> .....	30
<i>Set Local Video Display Mode (setLocalRenderMode)</i> .....	31
<i>Set Remote Video Display Mode (setRemoteRenderMode)</i> .....	32
<i>Switch Between Front and Back Cameras (switchCamera)</i> .....	32
<i>Mute Local Video Stream (muteLocalVideoStream)</i> .....	32
<i>Mute All Remote Video Streams (muteAllRemoteVideoStreams)</i> .....	33
<i>Mute Certain Remote Video Stream (muteRemoteVideoStream)</i> .....	33
<i>Start Audio Recording (startAudioRecording)</i> .....	33
<i>Stop Audio Recording (stopAudioRecording)</i> .....	34
<i>Start Audio Mixing (startAudioMixing)</i> .....	34
<i>Stop Audio Mixing (stopAudioMixing)</i> .....	35
<i>Set Log File (setLogFile)</i> .....	35
<i>Set Log Filter (setLogFilter)</i> .....	35
<i>Start Server Recording Service (startRecordingService)</i> .....	36
<i>Stop Server Recording Service (stopRecordingService)</i> .....	36
<i>Refresh Server Recording Service Status (refreshRecordingServiceStatus)</i> .....	36
<i>Destroy Engine Instance (destroy)</i> .....	37
<b>Delegate Methods (AgoraRtcEngineDelegate)</b> .....	37
<i>Warning Occurred Callback (didOccurWarning)</i> .....	37
<i>Error Occurred Callback (didOccurError)</i> .....	37
<i>Join Channel Callback (didJoinChannel)</i> .....	38
<i>Rejoin Channel Callback (didRejoinChannel)</i> .....	38
<i>Audio Volume Indication Callback (reportAudioVolumeIndicationOfSpeakers)</i> .....	38
<i>First Local Video Frame Displayed Callback (firstLocalVideoFrameWithSize)</i> .....	39
<i>First Remote Video Frame Received and Decoded Callback (firstRemoteVideoDecodedOfUid)</i> .....	39
<i>First Remote Video Frame Displayed Callback (firstRemoteVideoFrameOfUid)</i> .....	40
<i>Other User Joined Callback (didJoinedOfUid)</i> .....	40
<i>Other User offline Callback (didOfflineOfUid)</i> .....	40
<i>Other User Muted Audio Callback (didAudioMuted)</i> .....	41
<i>Other User Paused/Resumed Video Callback (didVideoMuted)</i> .....	41
<i>Other User Enabled/Disabled Video Callback(didVideoEnabled)</i> .....	41
<i>Local Video Statistics Callback (localVideoStats)</i> .....	42
<i>Remote Video Statistics Callback (remoteVideoStatOfUid)</i> .....	42
<i>Camera Ready Callback (rtcEngineCameraDidReady)</i> .....	42
<i>Video Stopped Callback (rtcEngineVideoDidStop)</i> .....	42
<i>Connection Interrupted Callback (rtcEngineConnectionDidInterrupted)</i> .....	42
<i>Connection Lost Callback (rtcEngineConnectionDidLost)</i> .....	43
<i>Rtc Engine Statistics Callback (reportRtcStats)</i> .....	43
<i>Audio Quality Callback (audioQualityOfUid)</i> .....	43
<i>Network Quality Callback (networkQuality)</i> .....	44
<i>Recording Service Status Refreshed Callback (didRefreshRecordingServiceStatus)</i> .....	44
<i>Recording Started/Ended/Status Returned (didApiCallExecute)</i> .....	44
<b>Callback Methods</b> .....	46

<i>Other User Joined Callback (userJoinedBlock)</i> .....	46
<i>Other User offline Callback (userOfflineBlock)</i> .....	46
<i>Rejoin Channel Callback (rejoinChannelSuccessBlock)</i> .....	47
<i>User Left Channel Callback (leaveChannelBlock)</i> .....	47
<i>Rtc Engine Statistics Callback (rtcStatsBlock)</i> .....	47
<i>Audio Volume Indication Callback (audioVolumeIndicationBlock)</i> .....	47
<i>First Local Video Frame Displayed Callback (firstLocalVideoFrameBlock)</i> .....	48
<i>First Remote Video Frame Displayed Callback (firstRemoteVideoFrameBlock)</i> .....	48
<i>First Remote Video Frame Received and Decoded Callback (firstRemoteVideoDecodedBlock)</i> .....	48
<i>Other User Muted Audio Callback (userMuteAudioBlock)</i> .....	49
<i>Other User Paused/Resumed Video Callback (userMuteVideoBlock)</i> .....	49
<i>Local Video Statistics Callback (localVideoStatBlock)</i> .....	49
<i>Remote Video Statistics Callback (remoteVideoStatBlock)</i> .....	49
<i>Camera Ready Callback (cameraReadyBlock)</i> .....	50
<i>Audio Quality Callback (audioQualityBlock)</i> .....	50
<i>Network Quality Callback (networkQualityBlock)</i> .....	50
<i>Connection Lost Callback (connectionLostBlock)</i> .....	51
<b>Error and Warning Messages</b> .....	<b>52</b>
<b>Error Messages</b> .....	<b>52</b>
<b>Warning Messages</b> .....	<b>54</b>





# Introduction

---

## Agora CaaS






Agora Communications as a Service (CaaS) provides ensured Quality of Experience for worldwide, Internet-based voice and video communications through the Agora Global Network. The network optimizes real-time, mobile communications and solves quality of experience challenges for mobile devices in networks such as 3G/4G/Wi-Fi that perform erratically and Internet bottlenecks worldwide.

Agora CaaS includes the following SDKs and the applications using the Native SDK link it directly into the application when they are built:

-  **Agora Native SDK for iOS and Agora Native SDK for Android**  
Mobile-optimized for smartphones, allowing access to the Agora Global Network along with device-specific mobile optimizations.  
See [Getting Started](#) for details about how to use the Agora Native SDK for iOS.
-  **Agora Native SDK for Windows and Agora Native SDK for Mac**
-  **Agora Native SDK for Web**  
It provides web browsers with open access to the Agora Global Network.
-  **Agora Whiteboard SDK**  
The Agora Whiteboard SDK is an addition to the Agora CaaS capabilities. The SDK provides a simple collaboration platform on the whiteboard where users from different locations can draw, annotate, and share PDF documents to visualize and simplify the communication. The SDK provides open access to the Agora Global Network from any device that supports a standard HTML5-compliant web browser, without requiring any downloads or plugins.

## Agora Native SDK for iOS

The **Agora Native SDK for iOS** allows your Java code to perform the following operations:

-  **Session setup.** Join and leave shared Agora conferencing sessions (identified by unique channel names), where there may be many global users conferenced together or simply one other user for one-to-one communication. Your application code should create and manage unique channel names; these usually originate in user, session, and date/time information that your application is already managing.
-  **Media control.** Enable and disable voice and video (allowing muting) and set various voice and video parameters that help the Agora SDK optimize communications. Many of the SDK operations are automated and do not require developer intervention if these parameter settings are not provided.
-  **Device control.** Access the microphone or speakerphone, set the volume, select from alternative cameras, and set the video window display.
-  **Session control.** Receive events when other users join or leave a conferencing session (channel), and understanding who's speaking, who's muted, and who's viewing the session. These events allow the application to decide when to make changes to the delivery of video and audio streams, and other application-specific user and status information.
-  **Quality management.** Obtain statistics and quality indicators about network conditions, run built-in tests, submit ratings and complaints about sessions and enable different levels of error logging.

- 🔗 **Recording.** Record audio or video and audio in one or multiple channels simultaneously. This function is only applicable if you use Dynamic Key security.
- 🔗 **Data Encryption.** Encrypt the audio and video packets. You cannot use the recording function when the data encryption function is enabled in a channel.

The Agora Native SDK for iOS provides two Java abstract classes to deliver these features:

- 🔗 The **RtcEngine** class provides all the methods that can be invoked by your application.
- 🔗 The **IRtcEngineEventHandler** class enables callback event notifications to your application.

Delegate methods are replacing the use of some Block callbacks from version 1.1 of the SDK. The Block callbacks are also documented below. However, where appropriate we recommend replacing them with Delegate methods. The SDK calls the Block method if a callback is defined in both Block and Delegate.

For detailed API usage, see [Agora Native SDK for iOS API Reference](#).

The Agora SDK returns some error codes or warning codes when calling APIs or running. For details, see [Error and Warning Messages](#).

## Requirements

### Required Development Environment

Apple XCode version 6.0 or higher

iOS simulator or real devices with audio functionality

**Note:** Video is supported only on real devices.

### Required Libraries

**Agora Native SDK for iOS** requires iOS 6.0 SDK or a later version. Make sure that your main project is linked to the following libraries in the SDK:

- 🔗 AgoraRtcEngineKit.framework (the Agora Native SDK)
- 🔗 AudioToolbox.framework
- 🔗 VideoToolbox.framework
- 🔗 AVFoundation.framework
- 🔗 CoreMedia.framework
- 🔗 CoreTelephony.framework
- 🔗 CoreMotion.framework
- 🔗 SystemConfiguration.framework
- 🔗 libc++.dylib

**Note:** By default, **Agora Native SDK** uses libc++ (LLVM). Contact [support@agora.io](mailto:support@agora.io) if you prefer to use libstdc++ (GNU).




In the source file, use the following command to import AgoraRtcEngineKit:

```
#import <AgoraRtcEngineKit/AgoraRtcEngineKit.h>
```

The SDK provides FAT Image libraries with multi-architecture support for both 32/64-bit audio simulators and 32/64-bit audio/video real devices.

### Required Documents

Read the following documents included in your download SDK (see [Where to Get the SDK](#)):

-  This reference manual
-  **Agora Recording Server User Guide:** Read if you plan to record.
-  **Data Encryption User Guide:** Read if you plan to enable data encryption.

# Getting Started

---

## Where to Get the SDK

The Agora Native SDK for iOS available from [agora.io/developer](https://agora.io/developer), or contact [sales@agora.io](mailto:sales@agora.io).

## About Keys

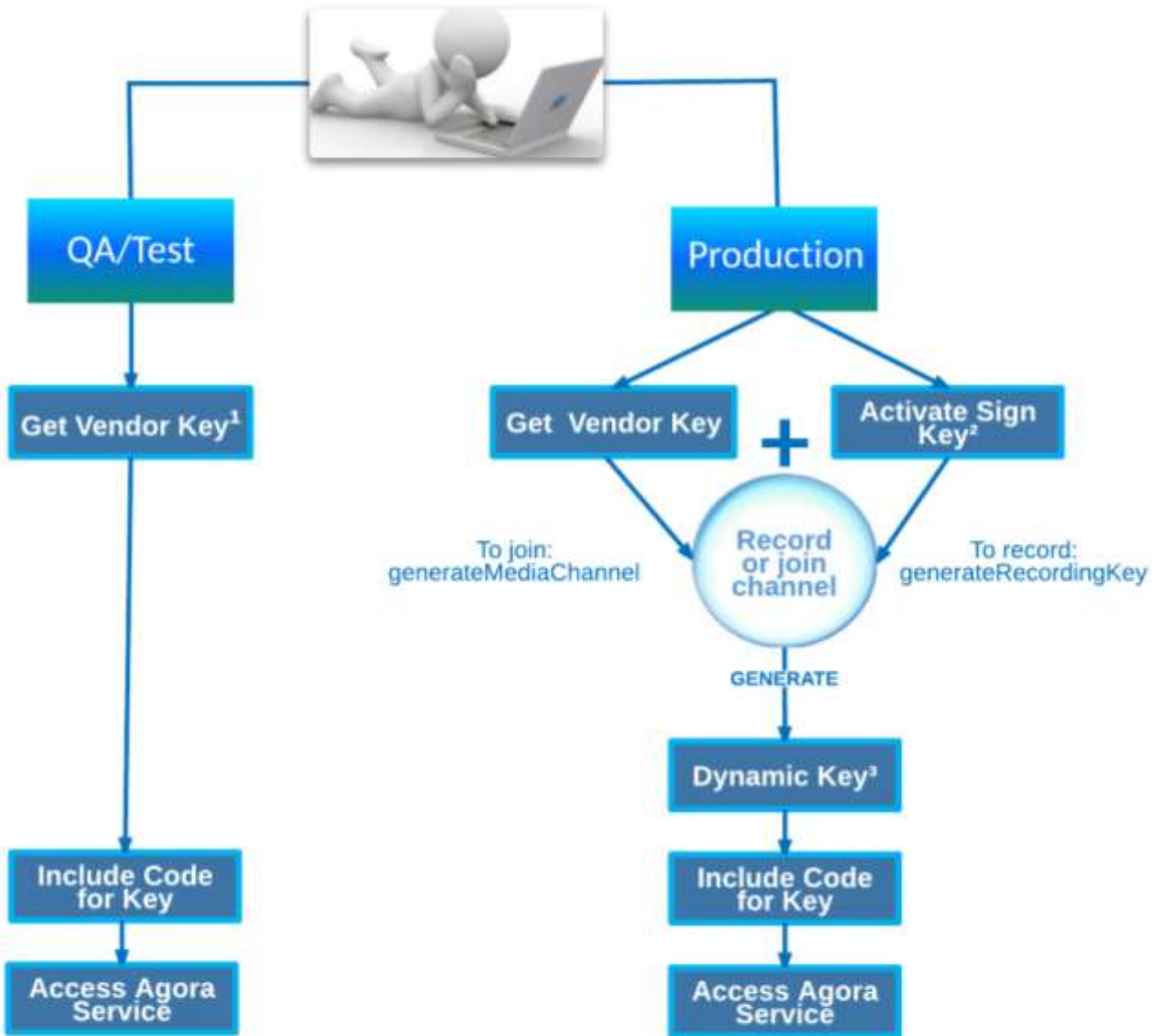
This section describes the concepts and use of vendor, sign, and dynamic keys.

Each Agora account can create multiple projects, and each project has a unique vendor key. Be sure to obtain a **Vendor Key**, required when you use APIs to access the Agora Global Network. In our network, the vendor key sets you apart from others. There is no overlap, even when channels have the same name.

But a vendor key is a static key, and if someone else illicitly obtains your static vendor key, then they can use it for their own Agora SDK client applications. If they find out the channel names of your organization, they can even interfere with your communication sessions.

A dynamic key is a more secure user authentication schema for the Agora SDK. Whenever a user tries to enter a channel to access the Agora service, the back-end services use the vendor key and a sign to generate a new dynamic key based on the HMAC encryption algorithm. The dynamic key is then passed to the client application. The client application calls the `joinChannel` or `startRecordingService` interface function and passes the encoded dynamic key to the Agora server that authenticates users.





<sup>1</sup> Vendor key for test/lab or non-recording, low security purposes

<sup>2</sup> Cannot be used alone

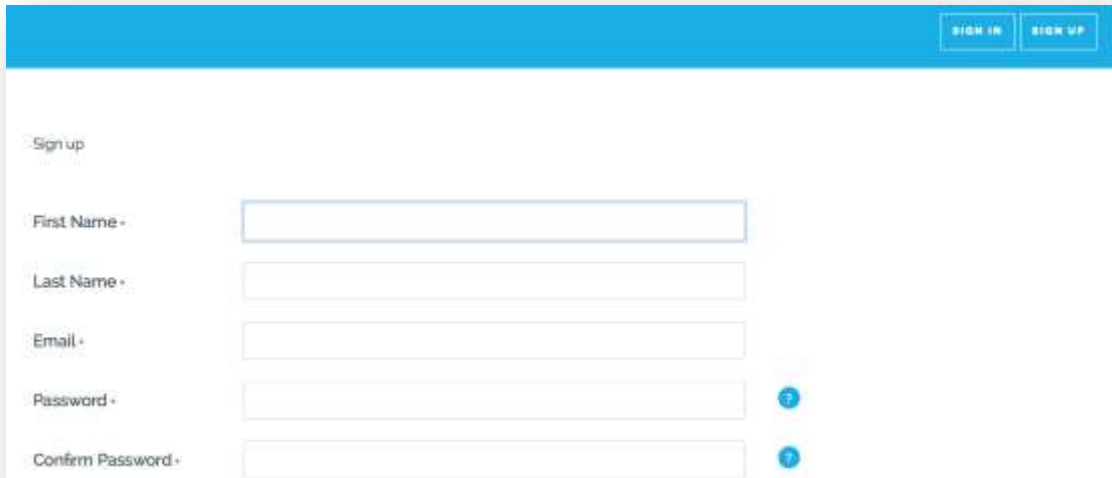
<sup>3</sup> Required for recording

## Obtaining and Using a Vendor Key

### Obtaining a Vendor Key

Each Agora account can create multiple projects, and each project has a unique vendor key.

1. Sign up for a new account at <https://dashboard.agora.io/>.

A screenshot of the Agora dashboard's sign-up page. The page has a blue header with 'SIGN IN' and 'SIGN UP' buttons. The main content area is white and titled 'Sign up'. It contains five input fields: 'First Name', 'Last Name', 'Email', 'Password', and 'Confirm Password'. The 'Password' and 'Confirm Password' fields have a small blue icon with a question mark to their right. The 'Sign up' text is in a light gray font.

2. Click **ADD NEW** on the **PROJECTS** page of the dashboard.
3. Fill in the **Project Name**. Skip the **Enable Sign Key** unless you plan to generate a dynamic key, and click **SAVE**.

A screenshot of the 'Add New Project' dialog box. It has a white background and a gray border. The 'Project Name' label is above a text input field containing 'TEST1'. Below the input field is a checkbox labeled 'Enable Sign Key'. At the bottom right are two buttons: 'CLOSE' and 'SAVE'. The 'Project Name' label and the 'TEST1' text are highlighted with a red rectangular box.

4. Find your vendor key under the project that you have created.

### Using a Vendor Key

Access the Agora services by using your unique vendor key:

1. Enter the vendor key in the starting window to enable audio or video communication in the demo.
2. To use the Agora SDK, add the vendor key to your code during development.
3. Set the parameter vendorKey as the Vendor Key when calling create().
4. Set the Key parameter to NULL when calling joinChannel().

## Obtaining and Using a Dynamic Key

### Obtaining a Vendor Key and a Sign Key

Each Agora account can create multiple projects, and each project has a unique vendor key and sign key. Follow the steps below to create a project to obtain a vendor key and a sign key at the same time:

1. Log in to <https://dashboard.agora.io>.

Click **ADD NEW** on the PROJECTS page of the dashboard. Fill in the Project Name, check the Enable Sign Key, and click **SAVE**.

A project is added.

2. Find the vendor and sign keys of the new project as follows:



3. Click the lock icon to show the sign key, and re-click it to hide the sign key again.
4. If you did not select **Enable Sign Key** when creating the project, click **ABOUT** to view the sign key after the project is created:





### Notes

- If you want to renew a sign key, contact [support@agora.io](mailto:support@agora.io).
- Keep your sign key on the server and never on any client machine.
- See the table in [Dynamic Key Structure](#) for sign key use.

### Implementing the Dynamic Key Scheme

Integrate the dynamic key scheme into your organization's signaling service. Agora.io provides sample server-side code covering the following languages: Java, C++, Python, node.js, and so on, which you can use to integrate this code directly into your application. When joining a channel or using the recording function, you will use the same vendor key and sign key, but the method to generate the dynamic key is different. Use

generateMediaChannelKey to join a channel, and use generateRecordingKey to use the recording function. For the sample code, refer to either of the following sites:

-  Github: <https://github.com/AgoraLab/AgoraDynamicKey>
-  Sign key Page at dashboard: <https://dashboard.agora.io/signkey>

If you want to verify the User ID (UID), check the following compatibilities:

Dynamic Key Version	User ID (UID)	SDK Version
DynamicKey4	UID of the specific user	1.3 or later
DynamicKey3	UID of the specific user	1.2.3 or later
DynamicKey	N/A	N/A

If you do not want to verify the User ID (UID), there is no compatibility issue, but we recommend that you upgrade to DynamicKey4.

### Using a Dynamic Key

Before a user joins a channel or start the recording function (that is, has started a call, received a meeting invitation or wants to use the recording function), the following sequence occurs:

1. The client application requests authentication from your organization's signaling server.
2. The server, upon receiving the request, uses the algorithm provided by Agora.io to generate a dynamic key and then passes the dynamic key back down to the client application.
3. The dynamic key is based on the sign key, vendor key, Channel Name, Current Timestamp, Client User ID, Lifespan Timestamp, and so on.
4. The client application calls the joinChannel or startRecordingService method, which requires the dynamic key as the first parameter.
5. The Agora server receives the dynamic key and confirms that the call comes from a legal user, and then allows it to access the Agora Global Network.

**Note:** When you deploy the dynamic key, it replaces the original vendor key as someone joins a channel. Dynamic keys expire after a certain amount of time. Your application must call renewChannelDynamicKey() when a time-out occurs. The didOccurError callback returns ERR\_DYNAMIC\_KEY\_TIMEOUT (109).

### Increased Security with Dynamic Keys

If your organization chooses to use dynamic keys, before a person joins a channel or when you start the recording function, the client application must provide a new dynamic key. The signaling server verifies each user identity. The dynamic key uses the HMAC/SHA1 signature schema to increase security for communications within your organization.

Field	Type	Length	Description
Version	String	3	Dynamic key version information

<b>Sign</b>	String	40	<p>Hex code for the signature. A 40-byte string calculated by the HMAC algorithm based on inputs including the sign key and the following fields:</p> <ul style="list-style-type: none"> <li>Service Type: Visible ASCII string provided by Agora.io. See <a href="#">Service Type</a>.</li> <li>Vendor Key: 32-character vendor key string.</li> <li>Timestamp: The timestamp created when the dynamic key is generated.</li> <li>Random Number: A 32-bit integer in hex code; generated upon each query.</li> <li>Channel: The channel name specified by the user, maximum length: 64-bytes.</li> <li>User ID: The User ID defined by the client.</li> <li>Call Expiration Timestamp: The timestamp indicates that from the specific moment the user cannot communicate in the channel any more.</li> </ul>
<b>Vendor Key</b>	String	32	Vendor Key
<b>Authorized Timestamp</b>	Number	10	The timestamp, represented by the number of seconds elapsed since 1-1-1970. It allows access to the Agora service for 5 minutes.
<b>Random Number</b>	Integer	8	A 32-bit integer in hex code; generated upon each query.
<b>Call Expiration Timestamp</b>	Number	10	<p>Set the value to 0 for no limitation to the time of termination.</p> <p>Indicates the exact time when a party can no longer use the Agora service (for example, when someone is forced to leave an ongoing call).</p>

The table below outlines the structure of the dynamic key. Connect all fields in the sequence shown.

#### Service Type

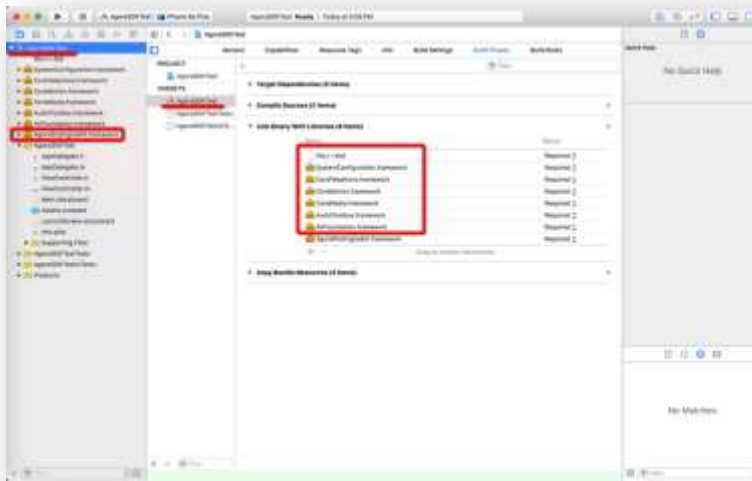
Service	Value	Description
Session	ACS	Audio and video services provided by Agora.io. For example, when you call the joinChannel API, and a dynamic key is required, use this service type to generate the dynamic key.
Recording	ARS	The audio and video recording services provided by Agora.io. For example, when you call the startRecordingService API, use this service type to generate the dynamic key.

The dynamic key encoding uses the industry-standard HMAC/SHA1 approach; libraries are available on most common server-side development platforms, such as node.js, PHP, Python, Ruby and others. For more information, refer to:

[http://en.wikipedia.org/wiki/Hash-based\\_message\\_authentication\\_code](http://en.wikipedia.org/wiki/Hash-based_message_authentication_code)

## Creating a New Project

1. Create a new project in XCode.
2. After you create a project, the development environment interface is displayed.
3. Navigate to `libs\AgoraRtcEngineKit.framework` in **Agora Native SDK for iOS**.



4. Right click the file **AgoraRtcEngineKit.framework** and select **Add Files to Project** from the pop-up menu to add AgoraRtcEngineKit.framework to the project.
5. AgoraRtcEngineKit.framework requires some system framework support as listed above under **Required Libraries**. Add the system framework with the **Link Binary with Libraries** in the Build Phase.
6. Include the header file into your source code with `#import <AgoraRtcEngineKit/AgoraRtcEngineKit.h>` to enable the use of **Agora Native SDK for iOS**.

Refer to the [API Reference](#) section below for a complete reference of the SDK methods.

## Using the Demo Application

**Agora** provides the following demo application in the **Agora Native SDK for iOS** zip file (describing here the FULL package with both video and voice):

AgoraDemo includes basic methods for entering and leaving a call, demonstrates the core operations of Agora Native SDK for iOS, and enables audio and video calls with simple method calls.

AgoraDemo/AgoraDemo: The source file folder of the demo program

AgoraDemo/AgoraDemo.xcodeproj: The project file of the demo program

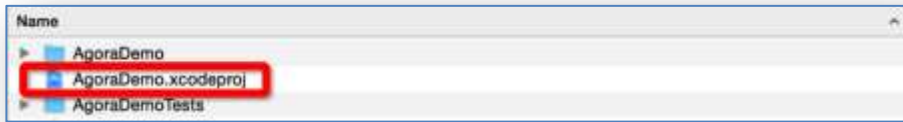
AgoraDemo/AgoraDemoTests: The test file folder of the demo program

libs/AgoraRtcEngineKit.framework: The SDK framework

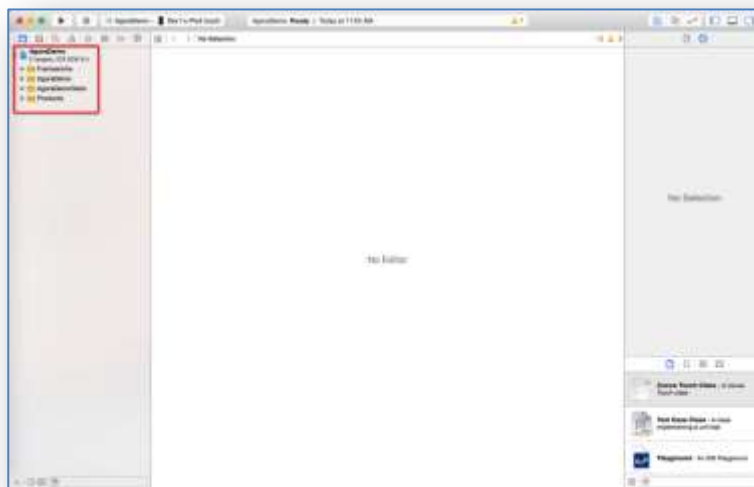
**Note:** Visit <https://github.com/AgoraLab/> for more open source demo applications.

## Compiling the Demo Program

1. Run the XCode.
2. Open the AgoraDemo.xcodeproj file.



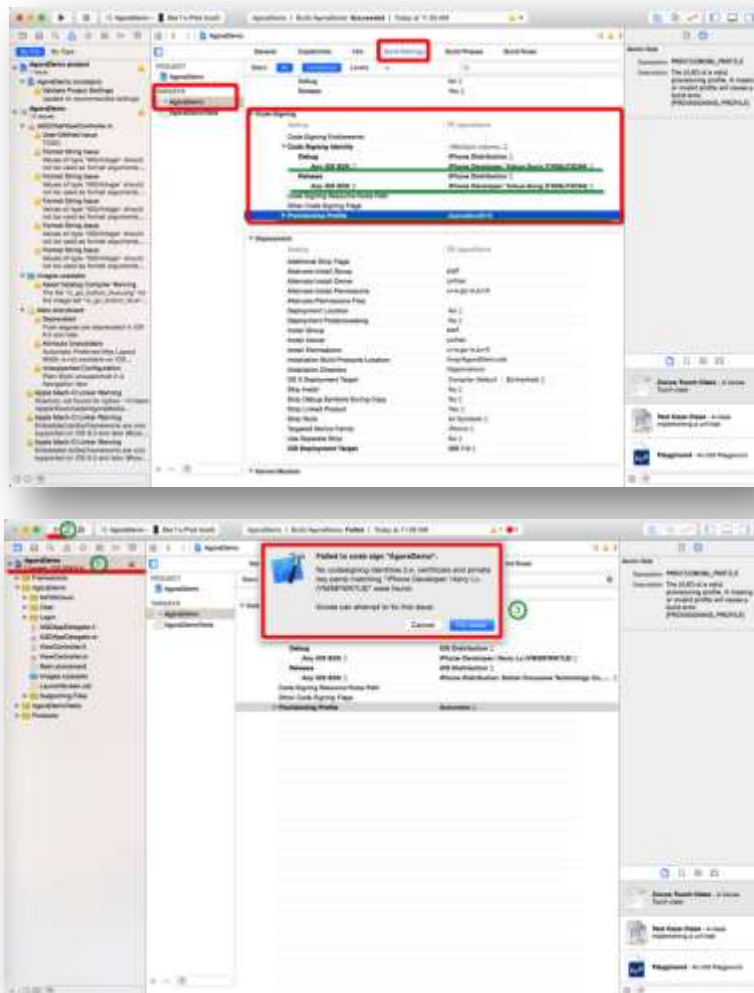
3. Open the project file.  
The Navigation panel on the left displays the file structure of AgoraDemo highlighted as follows:
4. Select the project as shown in the first screenshot below.



5. Click on **Build and Run** button as shown in the same screenshot to start compiling.

### Notes

- AgoraDemo uses both video and voice and only supports real devices. It does not support simulators.
- If a “Failed to code sign” error occurs as shown in the second screenshot, change the code signing to your activated device, or use your Apple developer account



## Executing the Demo Program

Run the AgoraDemo demo program after the deployment to your iOS device is complete.

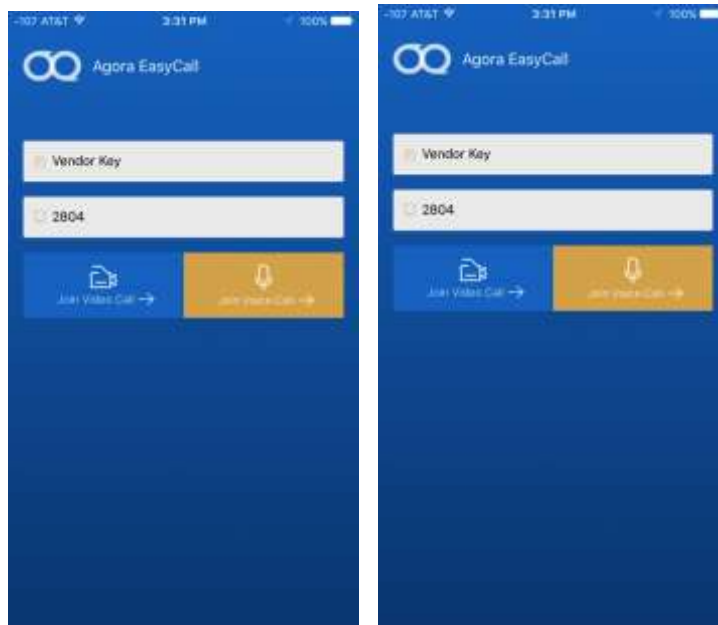
1. The following shows the first page of the demo application. You need two devices to run the demo for both voice and video calls.





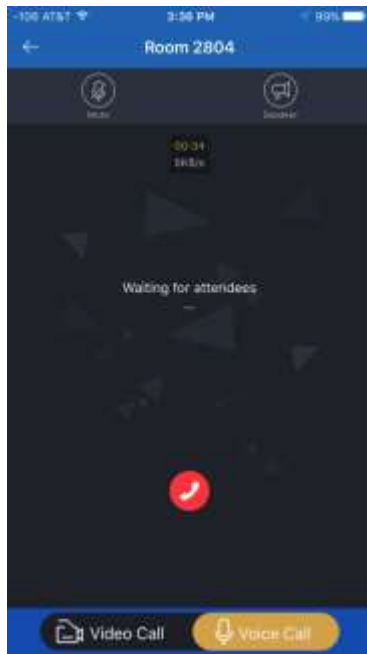
2. Make a voice call:

- a. On each device, enter your vendor key and meeting room name (channel name), for example,



2804. The entries should match, as shown above.

- b. Click or tap the **Join Voice Call** button to enter the audio call page.



- c. The call page features the following functions: mute/unmute, enable/disable speaker, leave the room.

### 3. Make a video call:

- a. Click or tap on the **Video Call** button to enter the main call page.
- b. On each device, enter your vendor key and meeting room name (channel name). The entries



should match.

- c. Click or tap **Join Video Call** to enter the video call page.

The call page features the following functions: mute/unmute, enable/disable speaker, open/close camera, switch cameras, and leave the room.

4. Switch between Video and Voice calls if necessary. Use the **Video Call** and **Voice Call** buttons at the bottom of the screen as shown in the above figure.

#### Note

The SDK and the AgoraDemo application support up to 5-way group video sessions, as shown in the following figure. Multiple users join using the same vendor key and room number (channel name in API).



# Agora Native SDK for iOS API Reference

## AgoraRtcEngineKit Interface Class

AgoraRtcEngineKit is the basic interface class of Agora Native SDK. Creating an AgoraRtcEngineKit instance and then calling the methods of this instance enables the use of Agora Native SDK's communication functionality. In previous versions this class was named differently, and it is now renamed to AgoraRtcEngineKit from version 1.0 AgoraAudioKit.

### Initialize (sharedEngineWithVendorKey)

```
(id) (instancetype)sharedEngineWithVendorKey:(NSString*)vendorKey
delegate:( id<AgoraRtcEngineDelegate>) delegate;
```

This method initializes the AgoraRtcEngineKit class as a singleton instance.

Call this method to initialize service before using AgoraRtcEngineKit.

The SDK uses Delegate to inform the application on the engine runtime events. All methods defined in Delegate are optional implementation methods.

Name	Description
vendorKey	The vendor key issued to the application developers by Agora. Apply for a new one from Agora if the key is missing from your kit.
delegate	

### Enable Video Mode (enableVideo)

```
(int)enableVideo
```

This method enables video mode.

The application can call this method either before entering a channel or during a call. If it is called before entering the channel, the service starts in video mode; if it is called during a call, it switches from audio to video mode. To disable video mode, call the disableVideo method.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Enable Audio Mode (disableVideo)

```
int disableVideo()
```

This method disables video and enables audio mode.

The application can call this method either before entering a channel or during a call. If it is called before entering the channel, the service starts in audio mode; if it is called during a call, it switches from video to audio mode. To enable video mode, call the enableVideo method.

Name	Description
Return Value	<b>0:</b> Method call succeeded.

Name	Description
	<b>&lt;0:</b> Method call failed.

## Start Video Preview (startPreview)

### (int)startPreview

This method starts the local video preview. Before starting the preview, always call `setupLocalVideo` to setup the preview window and configure its attributes and also call the `enableVideo` method to enable video. If before calling `joinChannelByKey` to join the channel, you have called `startPreview` to start the local video preview, then the local preview is still in the started state after you call `leaveChannel` to exit the channel. You can call `stopPreview` to close the local preview.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Stop Video Preview (stopPreview)

### (int)stopPreview

This method stops the local video preview and closes the video.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Join Channel (joinChannelByKey)

```
(int)joinChannelByKey:(NSString *)key channelName:(NSString *)channelName
info:(NSString *)info uid:(NSUInteger)uid joinChannelSuccess:(void (^)(NSString*
channel, NSUInteger uid, NSInteger elapsed))joinChannelSuccessBlock;
```

This method lets a user join a channel.

Users in the same channel can talk to each other, and multiple users in the same channel can start a group chat. Users using different vendor keys cannot call each other. Once in a call, the user must call the `leaveChannel` method to exit the current call before entering another channel. This method is called asynchronously; therefore, it can be called in the main user interface thread.

**Note:** The SDK uses iOS `AVAudioSession` shared object for audio recording and playing, and so operating on this object may affect the SDK's audio functions. When joining a channel, the SDK calls `setCategory AVAudioSessionCategoryPlayAndRecord` to set `AVAudioSession` to `PlayAndRecord` mode. The application should not set it to any other mode. When setting this mode, the sound being played (for example, a ring tone), if any, is interrupted.

Name	Description
key	The token is a dynamic key generated by the application. This parameter is optional if the user uses a static vendor key. In this case, pass <b>NULL</b> as the parameter value.

Name	Description
	<p>If the user uses a dynamic key, <b>Agora.io</b> issues an additional Sign Key to the application developers. The developers can then generate a user key using algorithm and Sign Key provided by <b>Agora.io</b> for user authentication on the server.</p> <p>See the discussion above in the <b>Getting Started</b> section on dynamic keys. In most developer circumstances, the static vendor key should suffice. For users who have a high security requirement, use the Dynamic Key. For users who want to use the recording function, Dynamic Key is mandatory.</p>
channelName	<p>A string providing the unique channel name for the AgoraRTC session. The length must be within 64 bytes.</p> <p>The following is the supported scope:</p> <p>a-z A-Z 0-9 space !#\$%&amp; ( )+, - ;,&lt;=. &gt;? @[] ^ _` { } ~</p>
info	(Optional) Any additional information the developer wants to add. It can be set as NULL Sting or channel related information. Other users in the channel won't receive this information.
uid	Optional) User ID: a 32-bit unsigned integer ranges from 1 to (2 <sup>32</sup> -1). It must be unique. If not assigned (or set to 0), the SDK allocates one and returns it in joinSuccessBlock callback. The app must record and maintain the returned value, as the SDK does not maintain it.

## Leave Channel (leaveChannel)

**(int)leaveChannel:(void(^)(AgoraRtcStats\* stat))leaveChannelBlock;**

This method lets the user leave a channel by hanging up or exiting a call.

After joining a channel, the user must call the leaveChannel method to end the call before joining another one. Calling the leaveChannel method even when not in a call is harmless. The leaveChannel method releases all resources related to the call. The leaveChannel method is called synchronously, and the user actually does not exit the channel when the call returns. Once the user exited the channel, the SDK triggers didLeaveChannelWithstats callback.

Name	Description
leaveChannelBlock	The callback on user successfully left the channel.
Return Value	0: Method call succeeded.

Name	Description
	<0: Method call failed.

## Renew Channel Dynamic Key (renewChannelDynamicKey)

**(int)renewChannelDynamicKey(NSString\*) key;**

This method updates dynamic key. The key expires after a certain period of time once the Dynamic Key scheme is enabled.

When the didOccurError callback reports the error ERR\_DYNAMIC\_KEY\_TIMEOUT(109), the application should retrieve a new token and then call this method to renew it. Failure to do so will result in SDK disconnecting with the server.

Name	Description
key	The Dynamic Key to be renewed.
Return Value	0: Method call succeeded. <0: Method call failed.

## Retrieve Current Call ID (getCallId)

**(NSString\*) getCallId;**

Every time a user joins a channel on a client machine by calling joinChannelByKey, a CallId is generated to identify the call from this client. Some methods such as rate and complain submit feedback to the SDK and are invoked after the call ends. These methods require assigned values of the CallId parameters. To use these feedback methods, call the getCallId method to retrieve the CallId during the call, and then pass the value as an argument in the feedback methods after the call ends.

Name	Description
Return Value	The current call ID.

## Rate the Call (rate)

**(int) rate:(NSString\*)callId rating:(NSInteger) rating description(NSString\*) description;**

This method lets the user rate the call. It is usually called after the call ends.

Name	Description
callId	Call ID retrieved from the getCallId method.
rating	Rating for the call between 1 (lowest score) to 10 (highest score).
description	A given description for the call with a length less than 800 bytes. This parameter is optional.
Return Value	0: Method call succeeded. <0: Method call failed. <b>ERR_INVALID_ARGUMENT (-2):</b> The passed argument is invalid, for example, callId invalid.

Name	Description
	<b>ERR_NOT_READY (-3):</b> The SDK status is incorrect, for example, initialization failed.

### Complain about Call Quality (complain)

```
(int) complain:(NSString*) callId description:(NSString*) description;
```

This method allows the user to complain about the call quality. It is usually called after the call ends.

Name	Description
callId	Call ID retrieved from the getCallId method.
description	A given description for the call with a length less than 800 bytes. This parameter is optional.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed. <b>ERR_INVALID_ARGUMENT (-2):</b> The passed argument is invalid, for example, callId invalid. <b>ERR_NOT_READY (-3):</b> The SDK status is incorrect, for example, initialization failed.

### Start Audio Call Test (startEchoTest)

```
(int)startEchoTest :(void(^)(NSString* channel, NSUInteger uid, NSInteger elapsed))successBlock;
```

This method launches an audio call test to check whether the audio devices (for example, headset and speaker) and the network connection work properly. In the test, the user speaks first, and the recording will be played back in 10 seconds. If the user can hear what he said in 10 seconds, it indicates that the audio devices and network connection work properly.

Note: After calling the startEchoTest method, always call stopEchoTest to end the test, otherwise the application will not be able to run the next echo test, nor can it call the joinChannel method to start a new call.

Name	Description
successBlock	Callback on successfully starting the echo test. See joinSuccessBlock in joinChannelByKey for a description of the callback parameters.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed. <b>ERR_REFUSED (-5):</b> Failed to launch the echo test, for example, initialization failed.

### Stop Audio Call Test (stopEchoTest)

```
(int)stopEchoTest;
```

This method stops an audio call test.



Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed. <b>ERR_REFUSED(-5):</b> Failed to stop the echo test. It could be that the echo test is not running.

## Enable Network Test (enableNetworkTest)

### (int)enableNetworkTest

This method enables the network quality test to test the user's network connection quality. When enabled, the SDK uses the networkQualityBlock method to notify the application about the network connection quality.

**Note:** Once the network test is enabled, it uses the network bandwidth even when the application is not in a call. We recommend the following practices:

When the application is in the front end, call the enableNetworkTest method to enable the network connection test; and when the application is switched to the back end, call the disableNetworkTest method to disable the network test in order to reduce network traffic. By default, the network test is disabled.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Disable Network Test (disableNetworkTest)

### (int)disableNetworkTest;

This method disables the network connection quality test.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Generate Call Quality Report URL (makeQualityReportUrl)

```
(NSString*) makeQualityReportUrl:(NSString*) channel
listenerUid:(NSUInteger) listenerUid
speakerUid:(NSUInteger) speakerUid
reportFormat:(AgoraRtcQualityReportFormat) reportFormat;
```

This method generates a URL pointing to the call quality reports. The application then uses the returned URL to retrieve detailed call quality data. The application must know the channel name and both of the callers' IDs to use this method. For instance, if A and B are in the same channel, the application retrieves a report on A speaking and B listening, and also a report on B speaking and A listening.

Name	Description
channel	Channel name specified in the joinChannel method.

Name	Description
listenerUid	User ID of the listener.
speakerUid	User ID of the speaker.
reportFormat	Format of the report. <b>AgoraRtc_QualityReportFormat_Json (0): JSON.:</b> Returns the quality report data. The application displays this information to users upon request. <b>AgoraRtc_QualityReportFormat_Html (1): HTML.:</b> Returns a report in HTML format, displayed on a web browser or WebVIEW components.
Return Value	Generated URL.

### Mute Local Audio Stream (muteLocalAudioStream)

**(int)muteLocalAudioStream:(BOOL)muted;**

This method mutes/unmutes local audio.

Name	Description
mute	Yes: Mutes local audio. No: Unmutes local audio.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Mute All Remote Audio Streams (muteAllRemoteAudioStreams)

**(int)muteAllRemoteAudioStreams:(BOOL)muted;**

This method mutes/unmutes all remote callers' audio streams.

Name	Description
muted	Yes: Stops playing all the received audio streams. No: Resumes playing all the received audio streams.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Mute Certain Remote Audio Stream (muteRemoteAudioStream)

**(int)muteRemoteAudioStream :(NSUInteger)uid  
muted:(BOOL)muted;**

Mute/unmute a specified remote user's audio stream.

**Note:** When set to Yes, this method stops playing audio streams without affecting the audio stream receiving process.

Name	Description
uid	User ID whose audio streams the user intends to mute.
mute	Yes: Stops playing a specified user's audio streams. No: Allows playing a specified user's audio streams.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Enable Speakerphone (setEnabledSpeakerphone)

**(int)setEnableSpeakerphone:(BOOL)enableSpeaker;**

This method enables audio output to the speaker.

**Note:** The SDK calls setCategory AVAudioSessionCategoryPlayAndRecord withOptions to configure the headset/speaker, and so any sound is interrupted when calling this method.

Name	Description
enableSpeaker	Yes: Switches to speaker. No: Switches to headset.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Is Speakerphone Enabled? (isSpeakerphoneEnabled)

**(BOOL)isSpeakerphoneEnabled;**

This method tests whether the speakerphone is enabled or not.

Name	Description
Return Value	Yes: Yes, enabled. No: No, not enabled.

### Enable Audio Volume Indication (enableAudioVolumeIndication)

**(int)enableAudioVolumeIndication:(NSInteger)interval  
smooth:(NSInteger)smooth;**

This method enables the SDK to regularly report to the application on who is speaking and the volume of the speaker.

Name	Description
interval	Specifies the time interval between two consecutive volume indications. <b>&lt;=0</b> : Disables volume indication. <b>&gt;0</b> : The volume indication interval in milliseconds. We recommend setting it to a minimum of 200ms.

Name	Description
smooth	The smoothing factor. Recommended default value is 3.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Register Packet Observer (registerAgoraPacketObserver)

**int registerAgoraPacketObserver(void\* nativeHandle, agora::rtc::IPacketObserver\* observer)**

This method registers a packet observer. When sending/receiving (Audio or Video) network packets, the Agora SDK triggers a callback on the interface that IPacketObserver defines. The application can use this interface to process data, for example, data encryption and decryption. The API can be used directly in Objective-C with an .mm file.

Notes

- Ensure that AgoraRtcEngineKit has been initialized before calling this API.
- The maximum size of the processed network packet is 1200 bytes. A packet larger than the maximum size may not be sent successfully.

Name	Description
nativeHandle	Retrieve by calling the API: (void*)getNativeHandle;
observer	Callback interface of the send/receive packet.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Enable Built-in Encryption(setEncryptionSecret)

**(int)setEncryptionSecret:(NSString\*)secret;**

Ensure that your application calls setEncryptionSecret to specify a secret for AES-128 encryption before joining a channel; otherwise the communication is unencrypted. All users in a channel must set the same secret. The secret is automatically cleared when a user leaves the channel. If the secret is not specified or set to empty, the encryption function is disabled.

Name	Description
secret	Parameter to enable built-in encryption.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

### Set Video Profile (setVideoProfile)

**(int)setVideoProfile:(AgoraRtcVideoProfile)profile;**

This method sets the video encoding profile. Each profile includes a set of parameters such as resolution, frame rate, bitrate, and son.

**Note:** Always set the video profile before calling the joinChannel method. When the device camera does not

support the specified resolution, the SDK automatically chooses a suitable camera resolution, but the encoder resolution still uses the one specified by `setVideoProfile`.

Name	Description
profile	Video profile. See the table below for more details.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

#### Video Profile Definition

Video Profile	Enum value	Resolution (width * height)	Frame Rate (fps)	Bitrate (kbps)
120P	0	160x120	15	80
120P_2	1	120x160	15	80
120P_3	2	120x120	15	60
180P	10	320x180	15	160
180P_2	11	180x320	15	160
180P_3	12	180x180	15	120
240P	20	320x240	15	200
240P_2	21	240x320	15	200
240P_3	22	240x240	15	160
360P	30	640x360	15	400
360P_2	31	360x640	15	400
360P_3	32	360x360	15	300
360P_4	33	640x360	30	800
360P_5	34	360x640	30	800
360P_6	35	360x360	30	600
480P	40	640x480	15	500
480P_2	41	480x640	15	500
480P_3	42	480x480	15	400
480P_4	43	640x480	30	1000
480P_5	44	480x640	30	1000
480P_6	45	480x480	30	800
480P_7	46	640x480	15	1000
720P	50	1280x720 *	15	1000
720P_2	51	720x1080 *	15	1000
720P_3	52	1280x720 *	30	2000
720P_4	53	720x1280 *	30	2000

\* Achieving 720P resolution depends on the performance and capabilities of the device and may not currently be possible on lower performance devices. If a device has inadequate performance, low frame

rates may result when using 720P. Agora.io is continuing to optimize video for lower-end devices in future releases – contact [support@agora.io](mailto:support@agora.io) to discuss particular device needs.

## Set Local Video View (setupLocalVideo)

**(int)setupLocalVideo:(AgoraRtcVideoCanvas\*)local;**

This method configures the video display settings on local machine. The application calls this method to bind with the video window (view) of local video streams and configures the video display settings. In the application development, call this method after initialization to configure local video display settings before entering a channel. The bind is still valid after exiting the channel. To unbind the view, set the view value to NULL when calling setupLocalVideo.

Name	Description
local	<p>Configures the video settings. AgoraRtcVideoCanvas types:</p> <p>view: The video display window (view). The SDK does not maintain the lifecycle of the view, and it is the responsibility of the application to ensure that the view is valid during the call, otherwise it may cause the SDK to crash. The view can be safely released after calling leaveChannel with a returned value.</p> <p>renderMode: The video display mode.</p> <ul style="list-style-type: none"><li>• AgoraRtc_Render_Hidden(1): If the video size is different than that of the display window, crops the borders of the video (if the video is bigger) or stretch the video (if the video is smaller) to fit it in the window.</li><li>• AgoraRtc_Render_Fit(2): If the video size is different than that of the display window, resizes the video proportionally to fit the window.</li><li>• AgoraRtc_Render_Adaptive(3): Adjusts based on the orientations. If both callers use the same screen orientation, that is, both use vertical screens or both use horizontal screens, the AgoraRtc_Render_Hidden mode applies; if they use different screen orientations, that is, one vertical and one horizontal, the AgoraRtc_Render_Fit mode applies.</li></ul> <p>uid: The local user ID as set in the joinchannel method.</p>
Return Value	<p><b>0</b>: Method call succeeded.</p> <p><b>&lt;0</b>: Method call failed.</p>

## Set Remote Video View (setupRemoteVideo)

**(int)setupRemoteVideo:(AgoraRtcVideoCanvas\*)remote;**

This method binds the remote user to the video display window, that is, sets the view for the user of the specified uid. Usually the application should specify the uid of the remote video in the method call before the user enters a channel. If the remote uid is unknown to the application, set it later when the application receives the onUserJoined event. If the Video Recording function is enabled, the Video Recording Service joins the channel as a dumb client, which means others clients will also receive its didJoinedOfUid event. Your application should not bind it with the view, because it does not send any video stream. If your application cannot recognize the dumb client, bind it with the view when the firstRemoteVideoDecodedOfUid. To unbind the user from the view, set the view to null. After the user left the channel, the SDK unbinds the remote user.

Name	Description
remote	<p>Configures the video settings. AgoraRTCVideoCanvas types:</p> <p>view: The video display window (view). The SDK does not maintain the lifecycle of the view, and it is the responsibility of the application to ensure that the view is valid during the call, otherwise it may cause the SDK to crash. The view can be safely released after calling leaveChannel with a returned value.</p> <ul style="list-style-type: none"> <li>AgoraRtc_Render_Hidden(1): If the video size is different than that of the display window, crops the borders of the video (if the video is bigger) or stretch the video (if the video is smaller) to fit it in the window.</li> <li>AgoraRtc_Render_Fit(2): If the video size is different than that of the display window, resizes the video proportionally to fit the window.</li> <li>AgoraRtc_Render_Adaptive(3): Adjusts based on the orientations. If both callers use the same screen orientation, that is, both use vertical screens or both use horizontal screens, the AgoraRtc_Render_Hidden mode applies; if they use different screen orientations, that is, one vertical and one horizontal, the AgoraRtc_Render_Fit mode applies.</li> </ul>
Return Value	<p><b>0</b>: Method call succeeded.</p> <p><b>&lt;0</b>: Method call failed.</p>

## Set Local Video Display Mode (setLocalRenderMode)

**(int)setLocalRenderMode:(AgoraRtcRenderMode) mode;**

This method configures the local video display mode.

The application can call this method multiple times to change the display mode.

Name	Description
mode	<p>Configures the video mode. AgoraRtcRenderMode types:</p> <p>renderMode: The video display mode.</p> <p>AgoraRtc_Render_Hidden(1): If the video size is different than that of the display window, crops the borders of the video (if the video is bigger) or stretch the video (if the video is smaller) to fit it in the window.</p> <p>AgoraRtc_Render_Fit(2): If the video size is different than that of the display window, resizes the video proportionally to fit the window.</p> <p>AgoraRtc_Render_Adaptive(3): Adjusts based on the orientations. If both callers use the same screen orientation, that is, both use vertical screens or both use horizontal screens, the AgoraRtc_Render_Hidden mode applies; if they use different screen orientations, that is, one vertical and one horizontal, the AgoraRtc_Render_Fit mode applies.</p>

Name	Description
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Set Remote Video Display Mode (setRemoteRenderMode)

**(int)setRemoteRenderMode: (NSUInteger)uid mode:(AgoraRtcRenderMode) mode;**

This method configures the remote video display mode.

The application can call this method multiple times to change the display mode.

Name	Description
uid	ID of the user whose video streams are received.
mode	Configures video mode. AgoraRtcRenderMode types: <ul style="list-style-type: none"> <li>AgoraRtc_Render_Hidden(1): If the video size is different than that of the display window, crops the borders of the video (if the video is bigger) or stretch the video (if the video is smaller) to fit it in the window.</li> <li>AgoraRtc_Render_Fit(2): If the video size is different than that of the display window, resizes the video proportionally to fit the window.</li> <li>AgoraRtc_Render_Adaptive(3): Adjusts based on the orientations. If both callers use the same screen orientation, that is, both use vertical screens or both use horizontal screens, the AgoraRtc_Render_Hidden mode applies; if they use different screen orientations, that is, one vertical and one horizontal, the AgoraRtc_Render_Fit mode applies.</li> </ul>
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Switch Between Front and Back Cameras (switchCamera)

**(int)switchCamera;**

This method switches between front and back cameras.

Name	Description
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Mute Local Video Stream (muteLocalVideoStream)

**(int)muteLocalVideoStream: (BOOL)muted;**

This method enables/disables sending the local video stream to the network.

**Note:** When set to Yes, this method does not disable the camera and thus does not affect the retrieval of local



video stream.

Name	Description
mute	Yes: Stops sending local video stream to the network. No: Allows sending local video stream to the network.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Mute All Remote Video Streams (muteAllRemoteVideoStreams)

**(int)muteAllRemoteVideoStreams:(BOOL)muted;**

This method enables/disables playing all remote callers' video streams.

**Note:** When set to Yes, this method stops playing video streams without affecting the video stream receiving process.

Name	Description
mute	Yes: Stops playing all received video streams. No: Allows playing all received video streams.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Mute Certain Remote Video Stream (muteRemoteVideoStream)

**(int)muteRemoteVideoStream :(NSUInteger)uid  
muted:(BOOL)muted;**

This method pauses/resumes playing a specified user's video, that is, enables/disables playing a specified user's video stream.

**Note:** When set to Yes, this method stops playing video streams without affecting the video stream receiving process.

Name	Description
uid	User ID of the specified user.
mute	Yes: Stops playing a specified user's video stream. No: Allows playing a specified user's video stream.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

### Start Audio Recording (startAudioRecording)

**(int)startAudioRecording:(NSString\*)filePath;**

This method starts audio recording.

The SDK allows recording during a conversation. The recording file format is .wav. Ensure that the saving directory in the application exists and is writable. This method is usually called after the joinChannel method.

**Note:** The recording automatically stops when the leaveChannel method is called.

Name	Description
filePath	Full file path of the recording file.
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Stop Audio Recording (stopAudioRecording)

**(int)stopAudioRecording;**

This method stops recording on the client machine.

**Note:** Call this method before calling leaveChannel; otherwise the recording automatically stops when the leaveChannel method is called.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Start Audio Mixing (startAudioMixing)

**(int) startAudioMixing: (NSString\*) filePath**  
**loopback: (BOOL) loopback**  
**replace: (BOOL) replace**  
**cycle: (NSInteger) cycle**

This method mixes the specified local audio file with the audio stream collected by the microphone, or replaces the audio stream collected by the microphone with the specified local audio file. You can decide whether the other user can hear the local audio playback and specify the number of loop playbacks.

Name	Description
filePath	Specify the name and path of the local audio file to be mixed. The following lists the supported audio formats: <ul style="list-style-type: none"> <li>• mp3</li> <li>• aac</li> <li>• m4a</li> <li>• 3gp</li> <li>• wav</li> </ul>
loopback	True: Only the local user can hear the remix or the replaced audio stream False: Both users can hear the remix or the replaced audio stream
replace	True: the content of the local audio file replaces the audio stream collected by the microphone False: Mix the content of the local audio file with the audio stream collected by the microphone
cycle	Specify the number of loop playbacks: <ul style="list-style-type: none"> <li>• 0: No loop</li> </ul>

	<ul style="list-style-type: none"> <li>• Positive integer: the number of loop playbacks</li> <li>• -1: infinite loop</li> </ul>
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

## Stop Audio Mixing (stopAudioMixing)

**(int)stopAudioMixing**

This method stops the mixing the specified local audio with the microphone input.

Name	Description
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

## Set Log File (setLogFile)

**(int)setLogFile:(NSString\*)filePath;**

This method specifies the SDK output log file.

The log file records all the log data of the SDK operation. Ensure that the saving directory in the application exists and that your app can write to it.

Name	Description
filePath	Full file path of the log file.
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

## Set Log Filter (setLogFilter)

**(int)setLogFilter:(NSUInteger)filter;**

This method sets the SDK output log filter. You can use one filter or a combination of the filters.

Name	Description
filter	Filters 1: INFO 2: WARNING 4: ERROR 8: FATAL 0x800: DEBUG
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

## Start Server Recording Service (startRecordingService)

**(int)startRecordingService: (NSString\*)key**

This method starts the server recording function. It requires a correct Dynamic key to start the service. This method is called asynchronously, and the execution result (successful or failed) returns in the didApiCallExecute callback.

Name	Description
key	Dynamic Key
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Stop Server Recording Service (stopRecordingService)

**(int)startRecordingService: (NSString\*)key**

This method stops the server recording function. It requires a correct Dynamic Key to start the service. This method is called asynchronously, and the execution result (successful or failed) returns in the didApiCallExecute callback.

Name	Description
key	Dynamic Key
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Refresh Server Recording Service Status (refreshRecordingServiceStatus)

**(int)refreshRecordingServiceStatus**

This method refreshes the server recording service status. This method is called asynchronously. The successful execution result returns in the didRefreshRecordingServiceStatus callback and unsuccessful execution result returns in the didApiCallExecute callback.

Name	Description
Return Value	<b>0:</b> Method call succeeded. <b>&lt;0:</b> Method call failed.

## Destroy Engine Instance (destroy)

**public void destroy()**

This method releases all the resources being used by the Agora SDK, and may be helpful for applications that perform voice or video communications from time to time according to user needs, but at other times want to maintain resources for other application functions. Once the application has called `destroy()` to destroy the created `RtcEngine` instance, no other methods in the SDK are available, and no callbacks occur. To start communications again, re-initialize (`sharedEngineWithVendorKey`) to establish a new `AgoraRtcEngineKit` instance.

Name	Description
Return Value	<b>0</b> : Method call succeeded. <b>&lt;0</b> : Method call failed.

## Delegate Methods (AgoraRtcEngineDelegate)

The SDK uses Delegate methods to report runtime events to the application. From version 1.1 on, some Block callbacks in the SDK are replaced with Delegate. The old Block callbacks are therefore being depreciated, but can still be used in current versions. However, we recommend replacing them with Delegate methods. The SDK calls the Block method if a callback is defined in both Block and Delegate.

### Warning Occurred Callback (didOccurWarning)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine  
didOccurWarning:(AgoraRtcErrorCode)warningCode**

This callback method indicates that some warning occurred during the runtime of the SDK. In most cases the application can ignore the warnings reported by the SDK because the SDK usually can fix the issue and resume running. For instance, the SDK may report an `AgoraRtc_Error_OpenChannelTimeout(106)` warning upon disconnection with the server, and in the meantime the SDK will attempt to reconnect.

Name	Description
warningCode	Warning code.

### Error Occurred Callback (didOccurError)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine  
didOccurError:(AgoraRtcErrorCode)errorCode**

This callback indicates that a network or media error occurred during the runtime of the SDK. In most cases reporting an error means that the SDK cannot fix the issue and resume running, and therefore requires actions from the application or simply informs you about the issue. For instance, the SDK reports an `AgoraRtc_Error_StartCall(1002)` error when it fails to initialize a call. In this case, the application informs the user that the call initialization failed, and at the same time it also calls the `leaveChannel` method to exit the channel.

Name	Description
Engine	The <code>AgoraRtcEngineKit</code> Object.
errorCode	Error code.

### Join Channel Callback (didJoinChannel)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine didJoinChannel:(NSString*)channel  
withUid:(NSUInteger)uid elapsed:(NSInteger) elapsed
```

Same as joinSuccessBlock in the joinChannelByKey API. This callback indicates that the user has successfully joined the specified channel.

Name	Description
channel	Channel name.
uid	User ID. If the uid is specified in the joinChannel method, returns the specified ID; if not, returns an ID that is automatically allocated by the Agora server.
elapsed	Time elapsed in milliseconds from calling joinChannel until this event occurs.

### Rejoin Channel Callback (didRejoinChannel)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine didRejoinChannel:(NSString*)channel  
withUid:(NSUInteger)uid elapsed:(NSInteger) elapsed
```

In times when the client machine loses connection with the server because of network problems, the SDK automatically attempts to reconnect, and then triggers this callback method upon reconnection, indicating that the user has rejoined the channel with assigned channel ID and user ID.

Name	Description
channel	Channel name.
uid	User ID.
elapsed	Time elapsed in milliseconds from calling joinChannel until this event occurs.

### Audio Volume Indication Callback (reportAudioVolumeIndicationOfSpeakers)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine  
reportAudioVolumeIndicationOfSpeakers:(NSArray*)speakers  
totalVolume:(NSInteger)totalVolume
```

Same as audioVolumeIndicationBlock.

By default the indication is disabled. If needed, use the enableAudioVolumeIndication method to configure it.

Name	Description
speakers	Speakers (array). Each speaker (): uid: User ID of the speaker (that is, the user who is speaking). volume: The volume of the speaker between 0 (lowest volume) to 255 (highest volume).

Name	Description
totalVolume	Total volume after audio mixing between 0 (lowest volume) to 255 (highest volume).

### First Local Video Frame Displayed Callback (firstLocalVideoFrameWithSize)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine
firstLocalVideoFrameWithSize:(CGSize)size elapsed:(NSInteger)elapsed
```

Same as firstLocalVideoFrameBlock. Indicates that the first local video frame is displayed on the video window.

Name	Description
size	Size of the video (width and height).
elapsed	Time elapsed in milliseconds from calling joinChannel until this callback is triggered.

### First Remote Video Frame Received and Decoded Callback (firstRemoteVideoDecodedOfUid)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine
firstRemoteVideoDecodedOfUid:(NSUInteger)uid size:(CGSize)size
elapsed:(NSInteger)elapsed
```

Same as firstRemoteVideoDecodedBlock. Indicates that the first remote video frame is received and decoded.

Name	Description
uid	User ID of the user whose video stream is received.
size	Size of the video (width and height).
elapsed	Time elapsed in milliseconds from calling joinChannel until this callback is triggered.

## First Remote Video Frame Displayed Callback (firstRemoteVideoFrameOfUid)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine  
firstRemoteVideoFrameOfUid:(NSUInteger)uid size:(CGSize)size  
elapsed:(NSInteger)elapsed
```

Same as firstRemoteVideoFrameBlock. Indicates that the first remote video frame is displayed on the screen.

Name	Description
uid	ID of the user whose video stream is received.
size	Size of the video (width and height).
elapsed	Time elapsed in milliseconds from calling joinChannel until this callback is triggered.

## Other User Joined Callback (didJoinedOfUid)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine didJoinedOfUid:(NSUInteger)uid  
elapsed:(NSInteger)elapsed
```

Same as userJoinedBlock. This callback indicates that a user has successfully joined the channel. If there are other users in the channel when this user joins, the SDK also reports to the application on the existing users who are already in the channel.

Name	Description
uid	User ID. If the uid is specified in the joinChannel method, returns the specified ID; if not, returns an ID that is automatically allocated by the Agora server.
elapsed	Time elapsed in milliseconds from calling joinChannel until this callback is triggered.

## Other User offline Callback (didOfflineOfUid)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine didOfflineOfUid:( NSUInteger )uid  
reason:(AgoraRtcUserOfflineReason)reason
```

Same as userOfflineBlock. This callback indicates that a user has left the call or gone offline.

**Note:** The SDK reads the timeout data to determine if a user has left the channel (or has gone offline) or not. If no data package is received from the user in 15 seconds, the SDK takes it as the user being offline. Sometimes a weak network connection may lead to false detection, therefore we recommend using signaling for reliable offline detection.

Name	Description
uid	User ID.
reason	Reasons for user going offline: USER_OFFLINE_QUIT: User has quit the call. USER_OFFLINE_DROPPED: The SDK is timeout and dropped offline because it hasn't received data package for too long. <b>Note:</b>



Name	Description
	Sometimes when the other user quits the call but the message is not passed to the SDK due to unreliable channel, the SDK may mistake it as the other user is timeout and dropped offline.

### Other User Muted Audio Callback (didAudioMuted)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine didAudioMuted:(BOOL)muted byUid:(NSUInteger)uid**

Same as userMuteAudioBlock. This callback indicates that some other user has muted/unmuted his/her audio streams.

**Note:** This callback method is valid only when the channel profile is set to Free-talk mode.

Name	Description
uid	User ID.
muted	Yes: User has muted his/her audio. No: User has unmuted his/her audio.

### Other User Paused/Resumed Video Callback (didVideoMuted)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine didVideoMuted:(BOOL)muted byUid:(NSUInteger)uid**

Same as userMuteVideoBlock. This callback indicates that some other user has paused/resumed his/her video streams.

Name	Description
uid	User ID.
muted	Yes: User has paused his/her video. No: User has resumed his/her video.

### Other User Enabled/Disabled Video Callback(didVideoEnabled)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine didVideoEnabled:(BOOL)enabled byUid:(NSUInteger)uid**

This callback indicates that other users enabled/disabled the video function. Once the video is disabled, users can only perform audio calls. They cannot see any video from either themselves or others.

Name	Description
uid	User ID
enabled	Yes: User has enabled the video function. No: User has disabled the video function.

### Local Video Statistics Callback (localVideoStats)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine  
localVideoStats:(AgoraRtcLocalVideoStats*)stats
```

Same as localVideoStatBlock. The SDK updates the application on local video streams uploading statistics once every 2 seconds.

Name	Description
sentBytes	Total bytes sent since last count.
sentFrames	Total frames sent since last count.

### Remote Video Statistics Callback (remoteVideoStatOfUid)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine  
remoteVideoStats:(AgoraRtcRemoteVideoStats*)stats
```

Same as remoteVideoStatBlock. The SDK updates the application on the statistics about receiving remote video streams once every 2 seconds.

Name	Description
uid	User ID of the user whose video streams are received.
delay	Time delay (in milliseconds).
width	Width (in pixel) of the video stream.
height	Height (in pixel) of the video stream.
receivedBitrate	The data receive bitrate (in kbps) since last count.
receivedFrameRate	The data receive frame rate (in fps) since last count.

### Camera Ready Callback (rtcEngineCameraDidReady)

```
(void)rtcEngineCameraDidReady:(AgoraRtcEngineKit *)engine
```

Same as cameraReadyBlock. This callback indicates that the camera is opened and ready to capture video.

### Video Stopped Callback (rtcEngineVideoDidStop)

```
(void)rtcEngineVideoDidStop:(AgoraRtcEngineKit *)engine
```

This callback indicates that the video is stopped. The application can use this callback to change the configuration of the view (for example, displaying other pictures on the view) after the video stops.

### Connection Interrupted Callback (rtcEngineConnectionDidInterrupted)

```
(void)rtcEngineConnectionDidInterrupted:(AgoraRtcEngineKit *)engine
```

This callback indicates that connection is lost between the SDK and server. Once the connection is lost, the SDK tries to reconnect it repeatedly until the application calls `leaveChannel`.

## Connection Lost Callback (rtcEngineConnectionDidLost)

**(void)rtcEngineConnectionDidLost:(AgoraRtcEngineKit \*)engine**

When the connection between the SDK and server is lost, the rtcEngineConnectionDidInterrupted callback is triggered and the SDK reconnects automatically. If the reconnection fails within a certain period (10s by default), the callback rtcEngineConnectionDidLost is triggered. The SDK continues to try reconnecting until the application calls leaveChannel.

## Rtc Engine Statistics Callback (reportRtcStats)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine reportRtcStats:(AgoraRtcStats\*)stats**

Same as rtcStatsBlock. The SDK updates the application on the statistics of the Rtc Engine runtime status once every 2 seconds.

Name	Description
stat	See table below.

AgoraRtcStats	Description
duration	Call duration in seconds, represented by an aggregate value.
txBytes	Total number of bytes transmitted, represented by an aggregate value.
rxBytes	Total number of bytes received, represented by an aggregate value.

## Audio Quality Callback (audioQualityOfUid)

**(void)rtcEngine:(AgoraRtcEngineKit \*)engine audioQualityOfUid:(NSUInteger)uid quality:(AgoraRtcQuality)quality delay:(NSUInteger)delay lost:(NSUInteger)lost**

Same as audioQualityBlock. During a call, this callback is triggered once every 2 seconds to report on the audio quality of the current call.

Name	Description
audioQualityBlock	
uid	User ID. Each callback reports on the audio quality of one user only (excluding the user himself). Every different user's statistics is reported in a separate callback.
quality	Rating of the audio quality: AgoraRtc_Quality_Excellent(1) AgoraRtc_Quality_Good(2) AgoraRtc_Quality_Poor(3) AgoraRtc_Quality_Bad(4) AgoraRtc_Quality_VBad(5) AgoraRtc_Quality_Down(6)
delay	Time delay in milliseconds.

Name	Description
lost	Packet loss rate (in percentage).

### Network Quality Callback (networkQuality)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine
networkQuality:(AgoraRtcQuality)quality
```

Same as networkQualityBlock. This callback is triggered once every 2 seconds during a call to report on the network quality.

Name	Description
quality	Rating of the network quality: AgoraRtc_Quality_Unknown = 0 AgoraRtc_Quality_Excellent = 1 AgoraRtc_Quality_Good = 2 AgoraRtc_Quality_Poor = 3 AgoraRtc_Quality_Bad = 4 AgoraRtc_Quality_VBad = 5 AgoraRtc_Quality_Down = 6

### Recording Service Status Refreshed Callback (didRefreshRecordingServiceStatus)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine
didRefreshRecordingServiceStatus:(NSInteger)status;
```

This callback returns the status code after executing the refreshRecordingServiceStatus method successfully.

Name	Description
Status Value	0: Recording is stopped. 1: Recording is ongoing.

### Recording Started/Ended/Status Returned (didApiCallExecute)

```
(void)rtcEngine:(AgoraRtcEngineKit *)engine didApiCallExecute:(NSString*)api
error:(NSInteger)error;
```

This callback returns the status code after a successful or unsuccessful execution of the startRecordingService or stopRecordingService method. It also returns the status code after an unsuccessful execution of the refreshRecordingServiceStatus method.

Method	API String Parameter	Status Code
StartRecordingService	rtc.api.start_recording_service	0: AgoraRtc_Error_NoError 1: AgoraRtc_Error_Failed 2: AgoraRtc_Error_InvalidArgument 3: ERR_NOT_READY

Method	API String Parameter	Status Code
stopRecordingService	rtc.api.stop_recording_service	7: AgoraRtc_Error_NotInitialized 10: AgoraRtc_Error_Timedout
refreshRecordingServiceStatus	rtc.api.query_recording_service_status	The code can be one of the above 1,2,3 ,7 or 10. The description is the same as above.

## Callback Methods

The SDK supports Block to report runtime events to the application. However, see the previous section on delegate methods. From version 1.1 on, some Block callbacks in the SDK are replaced with Delegate. The old Block callbacks are therefore being depreciated, but can still be used in current versions. However, we recommend replacing them with Delegate methods. The SDK calls the Block method if a callback is defined in both Block and Delegate.

The following section describes each of the callback methods in the SDK.

### Other User Joined Callback (userJoinedBlock)

**(void)userJoinedBlock:(void(^)(NSUInteger uid, NSInteger elapsed))userJoinedBlock;**

This callback indicates that a user has successfully joined the channel. If there are other users in the channel when this user joins, the SDK also reports to the application on the existing users who are already in the channel.

Name	Description
uid	User ID. If the uid is specified in the joinChannel method, returns the specified ID; if not, returns an ID that is automatically allocated by the Agora server.
elapsed	Time elapsed in milliseconds from calling joinChannel until this callback is triggered.

### Other User offline Callback (userOfflineBlock)

**(void)userOfflineBlock:(void(^)(NSUInteger uid))userOfflineBlock;**

This callback indicates that a user has left the call or gone offline.

**Note:** The SDK reads the timeout data to determine if a user has left the channel (or went offline) or not. If no data package is received from the user in 15 seconds, the SDK takes it as the user being offline. Sometimes a weak network connection may lead to false detection, therefore we recommend using signaling for reliable offline detection.

Name	Description
uid	User ID.

### Rejoin Channel Callback (rejoinChannelSuccessBlock)

```
(void)rejoinChannelSuccessBlock:(void(^)(NSString* channel, NSUInteger uid,
NSUInteger elapsed))rejoinChannelSuccessBlock;
```

In times when the local machine loses connection with the server because of network problems, the SDK automatically attempts to reconnect, and then triggers this callback method upon reconnection. The callback indicates that the user has rejoined the channel with an allocated channel ID and a user ID.

Name	Description
channel	Channel name.
uid	User ID.
elapsed	Time delay (in milliseconds) in rejoining the channel.

### User Left Channel Callback (leaveChannelBlock)

```
(void)leaveChannelBlock:(void(^)(AgoraRtcStats* stat))leaveChannelBlock;
```

This callback indicates that a user has left the channel, and it also provides call session statistics including the duration of the call and tx/rx bytes.

Name	Description
stat	See table below.

AgoraRtcStats	Description
duration	Call duration in seconds, represented by an aggregate value.
txBytes	Total number of bytes transmitted, represented by an aggregate value.
rxBytes	Total number of bytes received, represented by an aggregate value.

### Rtc Engine Statistics Callback (rtcStatsBlock)

```
(void)rtcStatsBlock:(void(^)(AgoraRtcStats* stat))rtcStatsBlock;
```

This callback is triggered once every 2 seconds to update the application on Rtc Engine runtime statistics.

Name	Description
stat	See table above.

### Audio Volume Indication Callback (audioVolumeIndicationBlock)

```
(void)audioVolumeIndicationBlock:(void(^)(NSArray *speakers, NSInteger
totalVolume))audioVolumeIndicationBlock;
```

This callback indicates who is speaking and the speaker's volume. By default it is disabled. If needed, use the enableAudioVolumeIndication method to configure it.

Name	Description
speakers	Speaker (array). Each speaker (): uid: User ID of the speaker (that is, the user who is speaking). volume: The volume of the speaker between 0 (lowest volume) to 255 (highest volume).
totalVolume	Total volume after audio mixing between 0 (lowest volume) to 255 (highest volume).

### First Local Video Frame Displayed Callback (firstLocalVideoFrameBlock)

**(void)firstLocalVideoFrameBlock:(void(^)(NSInteger width, NSInteger height, NSInteger elapsed))firstLocalVideoFrameBlock;**

This callback indicates that the first local video frame is displayed on the screen.

Name	Description
width	Width (in pixels) of the video stream.
height	Height (in pixels) of the video stream.
elapsed	Time elapsed in milliseconds from the user joining the channel until this callback is triggered.

### First Remote Video Frame Displayed Callback (firstRemoteVideoFrameBlock)

**(void)firstRemoteVideoFrameBlock:(void(^)(NSUInteger uid, NSInteger width, NSInteger height, NSInteger elapsed))firstRemoteVideoFrameBlock;**

This callback indicates that the first remote video frame is displayed on the screen.

Name	Description
uid	ID of the user whose video streams are received.
width	Width (in pixels) of the video stream.
height	Height (in pixels) of the video stream.
elapsed	Time elapsed in milliseconds from the user joining the channel until this callback is triggered.

### First Remote Video Frame Received and Decoded Callback (firstRemoteVideoDecodedBlock)

**(void)firstRemoteVideoDecodedBlock:(void(^)(NSUInteger uid, NSInteger width, NSInteger height, NSInteger elapsed))firstRemoteVideoDecodedBlock;**

This callback indicates that the first remote video frame is received and decoded.

Name	Description
uid	User ID of the user whose video streams are received.



Name	Description
width	Width (in pixels) of the video stream.
height	Height (in pixels) of the video stream.
elapsed	Time elapsed in milliseconds from the user joining the channel until this callback is triggered.

### Other User Muted Audio Callback (userMuteAudioBlock)

```
(void)userMuteAudioBlock:(void(^)(NSUInteger uid, bool muted))userMuteAudioBlock;
```

This callback indicates that a user has muted/unmuted his/her audio stream.

Name	Description
uid	User ID.
muted	Yes: Muted audio. No: Unmuted audio.

### Other User Paused/Resumed Video Callback (userMuteVideoBlock)

```
(void)userMuteVideoBlock:(void(^)(NSUInteger uid, bool muted))userMuteVideoBlock;
```

This callback indicates that a user has paused/resumed his/her video stream.

Name	Description
uid	User ID
muted	Yes: User has paused his/her video. No: User has resumed his/her video.

### Local Video Statistics Callback (localVideoStatBlock)

```
(void)localVideoStatBlock:(void(^)(NSInteger sentBytes, NSInteger sentFrames))localVideoStatBlock;
```

The SDK updates the application about the statistics on uploading local video streams once every 2 seconds.

Name	Description
sentBytes	Total bytes sent since last count.
sentFrames	Total frames sent since last count.

### Remote Video Statistics Callback (remoteVideoStatBlock)

```
(void)remoteVideoStatBlock:(void(^)(NSUInteger uid, NSInteger frameCount, NSInteger delay, NSInteger receivedBytes))remoteVideoStatBlock;
```

The SDK updates the application about the statistics on receiving remote video streams once every 2 seconds.

Name	Description
uid	User ID that specifies which user's video streams are received.
frameCount	Total frames received since last count.
delay	Time delay in milliseconds.
receivedBytes	Total bytes received since last count.

### Camera Ready Callback (cameraReadyBlock)

```
(void)cameraReadyBlock:(void(^))cameraReadyBlock;
```

This callback indicates that the camera is opened and ready to capture video.

### Audio Quality Callback (audioQualityBlock)

```
(void)audioQualityBlock:(void(^)(NSUInteger uid, AgoraRtcQuality quality, NSUInteger delay, NSUInteger lost))audioQualityBlock;
```

During a conversation, this callback is triggered once every 2 seconds to report on the audio quality of the current call.

Name	Description
audioQualityBlock	
uid	User ID. Each callback reports on the audio quality of one user only (excluding the user himself). Every different user's statistics is reported in a separate callback.
quality	Rating of the audio quality: AgoraRtc_Quality_Excellent(1) AgoraRtc_Quality_Good(2) AgoraRtc_Quality_Poor(3) AgoraRtc_Quality_Bad(4) AgoraRtc_Quality_VBad(5) AgoraRtc_Quality_Down(6)
delay	Time delay in milliseconds.
lost	Packet loss rate (in percentage).

### Network Quality Callback (networkQualityBlock)

```
(void)networkQualityBlock:(void(^)(AgoraRtcQuality quality))networkQualityBlock;
```

This callback is triggered once every 2 seconds during a call to report on the network quality.

Name	Description
quality	Rating of network quality:

Name	Description
	AgoraRtc_Quality_Unknown = 0 AgoraRtc_Quality_Excellent = 1 AgoraRtc_Quality_Good = 2 AgoraRtc_Quality_Poor = 3 AgoraRtc_Quality_Bad = 4 AgoraRtc_Quality_VBad = 5 AgoraRtc_Quality_Down = 6

### Connection Lost Callback (connectionLostBlock)

**(void)connectionLostBlock:(void(^)(void))connectionLostBlock;**

This callback indicates that the SDK has lost network connection with the server.

# Error and Warning Messages

Agora SDKs return error or warning messages:

- An **error message** means that the SDK encountered an error that requires application intervention. For example, when the SDK returns an error upon failing to open a camera, the application must remind people not to use the camera.
- A **warning message** means that the SDK encountered an error that may be recovered automatically. Such messages are just for notification and can usually be ignored.

## Error Messages

Error Code	Value	Description
ERR_OK	0	No error
ERR_FAILED	1	General error
ERR_INVALID_ARGUMENT	2	Invalid parameter called. For example, the specified channel name includes illegal characters.
ERR_NOT_READY	3	SDK module is not ready. For example, an API might require a specific module, but the module is not yet ready to provide service.
ERR_NOT_SUPPORTED	4	SDK does not support this function.
ERR_REFUSED	5	Request is rejected. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_BUFFER_TOO_SMALL	6	The buffer size is not big enough to store the returned data.
ERR_NOT_INITIALIZED	7	The SDK must initialize before your app calls this API.
ERR_INVALID_VIEW	8	Specified view is invalid. You must specify a view (for example, RENDER_MODE_HIDDEN) when using the video call function; otherwise, it returns this error.
ERR_NO_PERMISSION	9	No permission. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_TIMEDOUT	10	API call timed-out. Some APIs require the SDK to return the execution result, and this error occurs if the request takes too long for the SDK to process.
ERR_CANCELED	11	Request is cancelled. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_TOO_OFTEN	12	Call frequency is too high. Only for SDK internal use, which means there is no return to the

Error Code	Value	Description
		application through any API or callback event.
ERR_BIND_SOCKET	13	SDK failed to bind to the network socket. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_NET_DOWN	14	Network is unavailable. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_NET_NOBUFS	15	No network buffers available. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_INIT_VIDEO	16	Failed to initialize the video function.
ERR_JOIN_CHANNEL_REJECTED	17	Request to join the channel is rejected. This error usually occurs when the user is already in the channel but calls the API to join the channel, for example, joinChannel.
ERR_LEAVE_CHANNEL_REJECTED	18	Request to leave the channel is rejected. This error usually occurs when the user has already left the channel but calls the API to leave the channel, for example, leaveChannel.
ERR_ALREADY_IN_USE	19	Resources have been occupied, and cannot be reused.
ERR_ALREADY_IN_USE	20	SDK gave up responding to a request, probably due to too many requests. Only for SDK internal use, which means there is no return to the application through any API or callback event.
ERR_INVALID_VENDOR_KEY	101	Specified Vendor Key is invalid.
ERR_INVALID_CHANNEL_NAME	102	Specified Channel Name is invalid.
ERR_DYNAMIC_KEY_TIMEOUT	109	Current Dynamic Key has expired and is no longer valid.
ERR_INVALID_DYNAMIC_KEY	110	The specified Dynamic Key is invalid. Usually, this means that the dynamic key was generated incorrectly.
ERR_LOAD_MEDIA_ENGINE	1001	Failed to load media engine.
ERR_START_CALL	1002	Failed to start the call after enabling the media engine.
ERR_START_CAMERA	1003	Failed to start the camera.
ERR_START_VIDEO_RENDER	1004	Failed to start the video rendering module.
ERR_ADM_GENERAL_ERROR	1005	Audio Device Module: General error

Error Code	Value	Description
ERR_ADM_JAVA_RESOURCE	1006	Audio Device Module: Error in using Java resources
ERR_ADM_SAMPLE_RATE	1007	Audio Device Module: Error in setting the sampling frequency
ERR_ADM_INIT_PLAYOUT	1008	Audio Device Module: Error in initializing the playback device
ERR_ADM_START_PLAYOUT	1009	Audio Device Module: Error in starting the playback device
ERR_ADM_STOP_PLAYOUT	1010	Audio Device Module: Error in stopping the playback device
ERR_ADM_INIT_RECORDING	1011	Audio Device Module: Error in initializing the recording device
ERR_ADM_START_RECORDING	1012	Audio Device Module: Error in starting the recording device
ERR_ADM_STOP_RECORDING	1013	Audio Device Module: Error in stopping the recording device
ERR_ADM_RUNTIME_PLAYOUT_ERROR	1015	Audio Device Module: Runtime playback error
ERR_ADM_RUNTIME_RECORDING_ERROR	1017	Audio Device Module: Runtime recording error
ERR_ADM_RECORD_AUDIO_FAILED	1018	Audio Device Module: Failed to record
ERR_ADM_INIT_LOOPBACK	1022	Audio Device Module: Error in initializing the loopback device
ERR_ADM_START_LOOPBACK	1023	Audio Device Module: Error in starting the loopback device
ERR_VDM_CAMERA_NOT_AUTHORIZED	1501	Video Device Module: Camera is not permitted.

## Warning Messages

Warning Code	Value	Description
WARN_PENDING	20	The request is pending, usually due to a module that is not ready. The SDK has postponed processing the request.
WARN_NO_AVAILABLE_CHANNEL	103	No channel resources are available. Possibly, the server cannot allocate channel resources.
WARN_LOOKUP_CHANNEL_TIMEOUT	104	Timed out when looking up the channel. When

Warning Code	Value	Description
		joining a channel, the SDK looks up the specified channel. The warning usually occurs when network conditions are too poor to connect to the server.
WARN_LOOKUP_CHANNEL_REJECTED	105	The server rejected the request to look up the channel. Either the server cannot process this request, or the request is illegal.
WARN_OPEN_CHANNEL_TIMEOUT	106	Timed-out when opening the channel. Once the specific channel is found, the SDK opens the channel. The warning usually occurs when network conditions are too poor to connect to the server.
WARN_OPEN_CHANNEL_REJECTED	107	The server rejected the request to open the channel. The server cannot process this request or the request is illegal.
WARN_ADM_RUNTIME_PLAYOUT_WARNING	1014	Audio Device Module: Warning in runtime playback device
WARN_ADM_RUNTIME_RECORDING_WARNING	1016	Audio Device Module: Warning in runtime recording device
WARN_ADM_RECORD_AUDIO_SILENCE	1019	Audio Device Module: No valid audio data collected
WARN_ADM_PLAYOUT_MALFUNCTION	1020	Audio Device Module: Playback device failure
WARN_ADM_RECORD_MALFUNCTION	1021	Audio Device Module: Recording device failure
WARN_ADM_HOWLING	1051	Audio Device Module: Howling (feedback) detected

