



# **Agora Signaling SDK for Mac Reference Manual v1.0**

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

## Contents

<b>Introduction</b>	<b>4</b>
Agora CaaS	4
Agora Signaling SDK for Mac	4
Requirements	5
<i>Compatibility</i>	5
<i>Required Development Environments</i>	5
<i>Required SDK</i>	5
<i>Required Documents</i>	5
<b>Function</b>	<b>5</b>
Account Information System	6
<i>Account</i>	6
<i>Message Queue</i>	6
<i>Login</i>	6
<i>Attributes</i>	7
<i>Event and Query</i>	7
<i>Plain-text Messages</i>	7
Channel System	7
<i>Channel</i>	7
<i>Joining a Channel</i>	7
<i>Channel Events</i>	8
<i>Channel Messages</i>	8
<i>Channel Attributes</i>	8
Call System	8
<i>Call System</i>	8
<i>Call Timed-out</i>	9
<i>Call Flow</i>	9
<b>Access</b>	<b>10</b>
SDK Access	10
Server API Access	11
SIP Server Access	11
<b>Obtaining and Using a Signaling Key</b>	<b>11</b>
Obtaining an App ID and an App Certificate	12
Integrating the Signaling Key Schema	13
Using a Signaling Key	13
Increased Security with Signaling Key	14
<b>Integrating and Initializing SDK</b>	<b>14</b>
<b>Signaling SDK API Reference</b>	<b>15</b>
Methods	15
<i>Login(login)</i>	15
<i>Log out(logout)</i>	16
<i>Join Channel(channelJoin)</i>	16
<i>Leave Channel(channelLeave)</i>	16
<i>Query Channel User Number(channelQueryUserNum)</i>	16
<i>Set Channel Attributes(channelSetAttr)</i>	17
<i>Delete Channel Attributes(channelDelAttr)</i>	17

Delete All Channel Attributes(channelClearAttr) .....	17
Initiate a Call(channelInviteUser).....	18
Initiate a Call (channelInviteUser2).....	18
Send DTMF Message to Other User(channelInviteDTMF) .....	19
Accept Call(channelInviteAccept).....	19
Reject Call(channelInviteRefuse) .....	19
End Call(channelInviteEnd) .....	20
Send Point-to-Point Message(messageInstantSend).....	20
Send Channel Message(messageChannelSend) .....	21
Set User Attributes(set_attr) .....	21
Get Your Own User Attributes(get_attr) .....	22
Get All of Your Own Attributes(get_attr_all) .....	22
Get Attributes of Specific User(get_user_attr).....	22
Get All attributes of Specific User(get_user_attr_all).....	23
Check Login Status(isOnline) .....	23
Internal-test or Obsolete Interface .....	23
Events .....	24
Connection Lost Event(onReconnecting) .....	24
Connection Recovered Event(onReconnected) .....	24
User Online Event(onLoginSuccess).....	25
User Offline Event(onLogout) .....	25
User Login Failed Event(onLoginFailed) .....	25
User Joined Channel Event(onChannelJoined) .....	25
User Failed to Join Channel Event(onChannelJoinFailed) .....	26
User Left Channel Event(onChannelLeaved).....	26
Other User Joined Channel Event(onChannelUserJoined).....	26
Other User Left Channel Event(onChannelUserLeaved) .....	27
User List Received Event(onChannelUserList) .....	27
User Number Received Event(onChannelQueryUserNumResult) .....	27
Channel Attribute Changed Event(onChannelAttrUpdated) .....	28
Call Invitation Received Event(onInviteReceived).....	28
Other User Received Call Event(onInviteReceivedByPeer) .....	28
Other User Accepted Call Event(onInviteAcceptedByPeer) .....	29
Other User Rejected Call Event(onInviteRefusedByPeer) .....	29
Call Failed Event (onInviteFailed).....	29
Other User Ended Call Event(onInviteEndByPeer) .....	30
Call Ended Event(onInviteEndByMyself).....	30
Message Failed Event(onMessageSendError).....	31
Message Sent Event(onMessageSendSuccess) .....	31
Other User Received Message Event(onMessageInstantReceive) .....	31
Channel Message Received Event(onMessageChannelReceive) .....	31
Log Printed Event(onLog).....	32
Attribute Received Event(onUserAttrResult) .....	32
All Attribute Received Event(onUserAttrAllResult) .....	32
Internal-test or Obsolete Interface .....	33
<b>Error Code .....</b>	<b>33</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:

SDK	Android	iOS	Windows	Mac	Web
Agora Native SDK	√	√	√	√	√
Agora Signaling SDK	√	√	√	√	

Agora Native SDK provides the audio and video communication functions, and the applications using the Native SDK link it directly into the application when they are built.

Agora Signaling SDK provides reliable audio and video control services.

This document introduces how to use the Agora Signaling SDK for Mac to access the Agora Signaling System.

## Agora Signaling SDK for Mac

The Agora Signaling SDK allows you to realize the following functions (See [Function](#) for details):

- Account Information System: the account information system is the basis of channel system and call system. The users can reliably send and receive messages, and carry out other business after login.
- Channel System: support a large number of users simultaneously online, and provide reliable channel events, properties and messages. Typical scenarios include audio and video conferencing messages, microphone control, video broadcast channel news etc.
- Call System: provide audio and video calls and external SIP calls, including hybrid call session with VOIP users and PSTN phone users.

Agora Signaling SDK provides APIs to realize all the mentioned function in this document.

For details, see [Signaling SDK API Reference](#).

## Requirements

### Compatibility

Agora Signaling SDK is compatible with all platform SDKs (v1.2 or later).

## Required Development Environments

According to [Access](#), the users can access the Agora signaling system in three ways, in which the system and the network requirements using the Agora Signaling SDK are listed as follows:

System Requirement		
Platform	Network Requirement	
OS X	v10.6 "Snow Leopard" or later	Port: HTTP 80 , TCP 8181

## Required SDK

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

## Required Documents

- This reference manual
- If you want to use Agora SDK to realize audio and video communication function on your application, read this document together with the specific platform SDK reference manual.

All platform SDK and related reference manuals can be downloaded at:  
[agora.io/developer](https://agora.io/developer)

# Function

By accessing the Agora Signaling System, you will be able to use the following functions:

- Account Information System
- Channel System
- Call System

## Account Information System

This section introduces the account information system in details and things that the users must pay attention to.

## Account

The account of the Agora signaling system is the unique identity for each user, which does not require users to register:

- **SDK access:** the users can use the generated Signaling Key to log in. For details, see [Obtaining and Using a Signaling Key](#).
- **Server API access:** login is not required.
- **SIP Gateway access:** contact [support@agora.io](mailto:support@agora.io).

## Message Queue

Each user has his/her own message queue:

- Message will not be lost regardless of whether the user is online or not.
- Message may be timed out, and the time limits for different messages are not the same.

## Login

Users can send or receive messages or perform other actions only after logging into the Agora signaling system.

After login, a network outage might occur temporarily. During this period, users can neither receive any message nor perform any action. When the network recovers, the user can then receive messages. The messages will not be lost if it is not timed-out.

The SDK triggers the *onLogout* event if the network has been interrupted for too long, and the server also marks the user as “Logout”. If so, the users must call *login* again before they can continue the operations.

One account can only log onto one device instead of multiple devices simultaneously. Whenever you log in, you will be kicked out from the last login device (SDK triggers the *onLogout* event) automatically.

## Attributes

Each user has his/her own attribute table:

Users can set some attributes, which they or other users can read. The attributes usually store contents like the address of the user profile picture, nickname and etc.

## Event and Query

- The users can query the status and attributes of other users.
- The users can subscribe the status and attribute changes of other users(only applicable to Server API). For details, see **Server API Reference Manual**.

## Plain-text Messages

The users can send plain text messages to each other, for example, text, web site, image address, json and etc.

## Channel System

### Channel

The signaling channel, based on a reliable account information system, ensures zero package loss, on which the users can build the service system in a reliable way.

When the network is temporarily interrupted, the users are unable to receive any message. Once the network recovers, the users receive all the messages during the interruption. If the network has been blocked, the SDK triggers *onLogout* event and the server background sets the user as “Logout”, and automatically kicks the users out of the channel.

### Joining a Channel

- The users must log into the Agora signaling system before joining any channel.
- The users can only join one channel, and whenever the users join a channel, they will be automatically kicked out from the channel they have joined previously.
- The users leave a channel automatically upon logout.
- The users get a complete user list in the channel automatically once they have joined in the channel.

### Note

The users will not receive the user list automatically in a large channel (for example, with over 200 people).

### Channel Events

The users can receive all the channel events (other user joined, left, and etc) once joining a channel. All channel events are reliable without package loss.

### Channel Messages

The channel messages are broadcasting messages, which means all users in the same channel will receive them.

### Channel Attributes

Each channel has an attribute table, and all users can get this table once joining a channel. Each user can modify the attributes and all users in the same channel will receive notification whenever a change is made on the attributes.

## Call System

### Call System

The users can use video or audio communication without the call system if all sides join the same channel.

Call system is used to initiate a call, invite other users to join a channel in one-on-one video or audio communication or conference call:

- The users must stay as “Login” to receive a call;
- The users can call Agora internal users. Internal user refers to the users registered to the Agora signaling system.
- The users can call the external users via SIP gateway. They can use SIP gateway to call external users at PSTN, call centers, PBX, IM and etc.
- Each call only reaches one person, and a group communication can initiate multiple calls. Each call is independent, and the users can call Agora internal users, or call external users via SIP gateway.
- The users can initiate a call using Server API, and it is a more reliable way to initiate a call using Server API for a multi-party communication by assigning the call status to other users through channel attributes and channel messages.

### Call Timed-out

Once you have initiated a call, and if you have not received the confirmation from the other user within 30 seconds, the call fails as timed-out.

Upon receipt of the confirmation, you, as the caller, decides by yourself when to end the call if the other user does not accept or reject the call within a certain period.

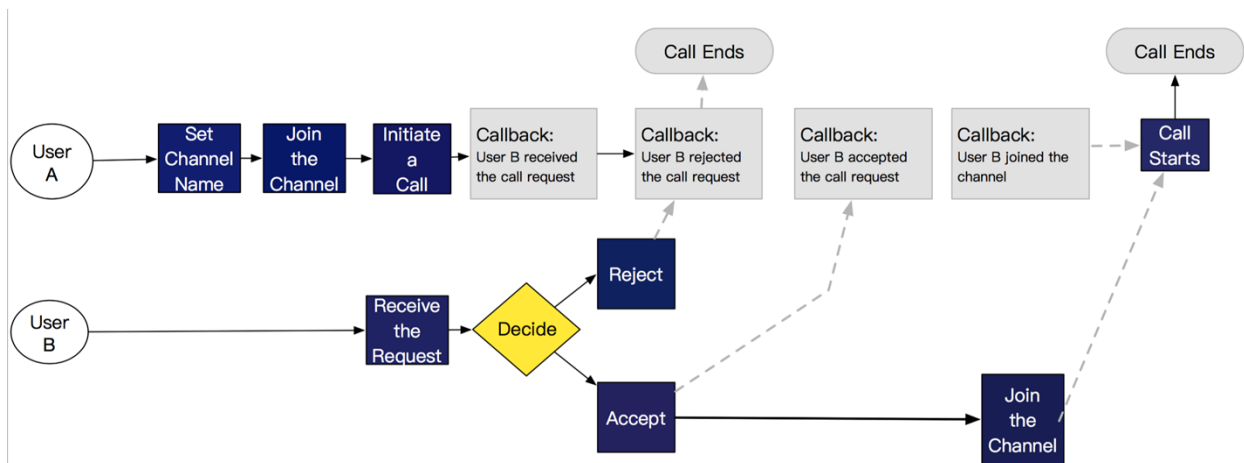
### Call Flow

Call and Channel are independent. The users can select either of the following way to call:

- Call first, then join the same channel after both sides are connected;
- Join a channel first, then initiate a call so that the other user can receive an invitation/request. It makes call faster and the other user can watch the fuzzy video of the caller before he/she decides to accept or reject the call. But if you have joined a channel, while the other user does not accept the call invitation, you will still be charged as a single channel.

The following is the call flow of joining a channel first before initialing a call:



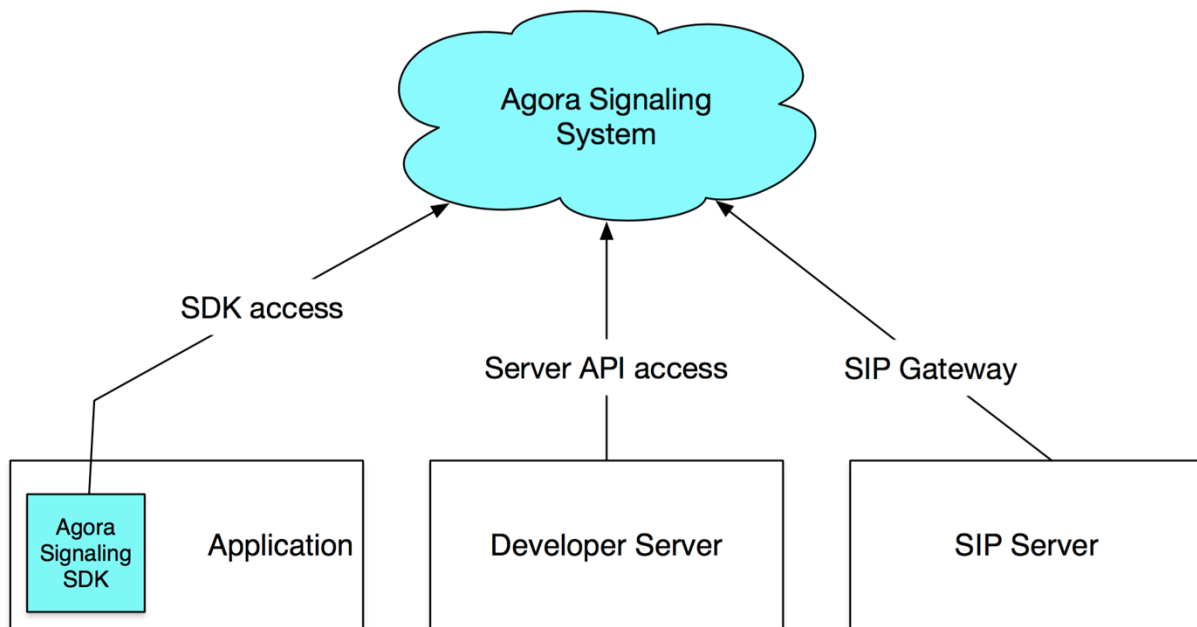


## Access

---

You can access the Agora signaling system in one of the following ways:

- SDK access
- Server API access
- SIP Gateway access



### SDK Access

The following functions are supported:

- Account Information System:
  - Login
  - Point-to-point messaging
- Call System
- Channel System

For more information, see [Signaling SDK API Reference](#).

## Server API Access

The following functions are supported:

- Account Information System:
  - Server API: no logins required
  - Query the user status
  - Subscribe the changes of user status and attributes
  - Send messages to users
- Channel System:
  - Query the channel users
- Call System:
  - Subscribe the calling events
  - Initiate a call

For more information, see ***Server API Reference Manual***.

## SIP Server Access

SIP Gateway is used to connect the external systems, including:

- Signaling: turn the private signaling to SIP protocol
- Media: turn the private media to RTP protocol

The following functions are supported currently:

- Calling function
- Call parameters customization

### Note

Contact [support@agora.io](mailto:support@agora.io) to enable the SIP gateway access to the Agora Signaling System if required.

## Obtaining and Using a Signaling Key

---

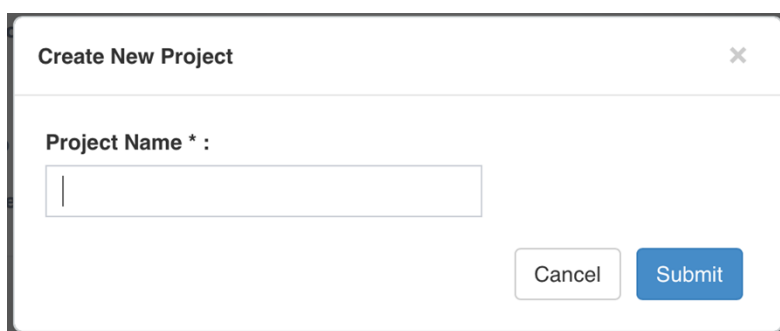
The users must provide a Signaling Key when using Agora Signaling SDK to access the Agora signaling system to ensure that each call service is verified by the Agora signaling server.

## Obtaining an App ID and an App Certificate

Each Agora account can create multiple projects, and each project has a unique App ID and App Certificate.

Follow the steps below to create a project to obtain an App ID and an App Certificate at the same time.

1. Log into <https://dashboard.agora.io>.
2. Click **Add New Project** on the **Projects** page of the dashboard.
3. Fill the **Project Name** and click **Submit**. The App ID is available now.



4. Enable the App Certificate for the project.
  - a. Click **Edit** on the top right of the project.
  - b. Click **Enable** to the right of App Certificate. Read the description **About App Certificate** carefully before you confirm the operation.

**App Certificate:** App Certificate Not Enabled **Enable**

- c. Click the “eye” icon to view the App Certificate. You can re-click this icon to hide the App Certificate.

**App Certificate:**



### Notes

- 🔗 If you want to renew the App Certificate for some reason, contact [support@agora.io](mailto:support@agora.io).
- 🔗 Keep your App Certificate on the server and never on any client machine.

- See [Signaling Key Structure](#) for the App Certificate usage.

## Integrating the Signaling Key Schema

Use the following algorithm to generate a token(Signaling Key) required for logging onto the Agora Signaling System:

### Input:

```
appId      = "C5D15F8FD394285DA5227B533302A518" //App ID
appCertificate = "fe1a0437bf217bdd34cd65053fb0fe1d" //App Certificate
expiredTime  = "2592000" // expiredTime
account      = "test@agora.io" //User ID defined by the client
```

### Output:

token

```
= "1:appId:expiredTime:md5(account + appId + appCertificate + expiredTime)"
="1:C5D15F8FD394285DA5227B533302A518:2592000:md5(test@agora.ioC5D15F8FD394285DA5227
B533302A518fe1a0437bf217bdd34cd65053fb0fe1d2592000)"
= "1:C5D15F8FD394285DA5227B533302A518:2592000:5c0ee12fdf2020d0d0fdad04d6395473"
```

### Note

For the detailed explanation of the above fields, see [Signaling Key Structure](#).

## Using a Signaling Key

Before a user request to log into the Agora signaling system:

1. The client application requests authentication from the signaling server of your organization.
2. The server, upon receiving the request, uses the algorithm provided by Agora.io to generate a Signaling Key and then passes the Signaling Key back down to the client application.

The Signaling Key is based on the App Certificate, App ID, User ID defined by the client and Lifespan Timestamp.

3. The client application calls *login*, and it is required to set the parameter *token* as the Signaling Key generated above.
4. The Agora server receives the Signaling Key and confirms that the call comes from a legal user, and then allows it to access the Agora Signaling System.

## Increased Security with Signaling Key

The table below outlines the structure of the Signaling Key. Connect all fields in the sequence shown.

Field	Type	Length	Description
Version	String		Signaling Key version number, fixed as 1.
App ID	String	32	App ID provided by Agora, which you can obtain at <a href="https://dashboard.agora.io">https://dashboard.agora.io</a> .
Authorized Timestamp	Number	10	The UTC timestamp represented by the number of seconds elapsed since 1-1-1970. 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).
Sign	String	32	Hex code for the signature. A string calculated by the MD5 algorithm based on inputs including the App Certificate and the following fields: <ul style="list-style-type: none"><li>• account: The User ID defined by the client.</li><li>• appId: 32-character App ID string.</li><li>• appCertificate: 32-character App Certificate string.</li><li>• expiredTime: The UTC timestamp indicates that from the specific moment the user cannot access the Agora Signaling System any more</li></ul>

## Integrating and Initializing SDK

1 Add the libraries under **libs/nomedia/** to the client.

2 Initialize the code:

```
inst = [AgoraAPI getInstanceWithoutMedia:appId];  
instMedia = [AgoraRtcEngineKit sharedEngineWithappId:appId delegate:nil];
```

3 Save the uid upon a successful login.

```
inst.onLoginSuccess = ^(uint32_t uid, int fd){  
    my_uid = uid;  
};
```

4 The media and signaling channels must join or leave the channel simultaneously.

```
[inst channelJoin:self.txtChannel.text];  
[instMedia joinChannelByKey:appId channelName:self.txtChannel.text info:nil]
```

```
uid:my_uid joinSuccess:nil];
```

### Note

Refer to the demo included in the Agora Signaling SDK package for detailed codes.

## Signaling SDK API Reference

---

### Methods

#### Login(login)

```
(void) login:(NSString*)appId account:(NSString*)account token:(NSString*)token  
uid:(uint32_t)uid deviceID:(NSString*)deviceID ;
```

This method allows the users to log into the Agora signaling system. The users must log in before they can perform any action.

It triggers the *onLoginSuccess* callback upon a successful execution, otherwise it *returns onLoginFailed*. If the application lost the connection with the server, it returns the *onLogout* callback.

Parameter	Description
appId	App ID provided by Agora at <a href="https://dashboard.agora.io">https://dashboard.agora.io</a> .
account	User ID defined by the client. It can be up to 128 visible characters (space is not permitted currently). It can be the uid, nickname, guid and other content as long as it is unique. The <i>account</i> parameter mentioned across this document follow the same rule.
token	Signaling Key generated by App ID and App Certificate. See <a href="#">Obtaining and Using a Signaling Key</a> for details.
uid	Expired, set it as 0.
deviceID	Set it as NULL.

## Log out(`logout`)

(void) `logout`;

This method allows the users to log out of the Agora signaling system. The user leaves the joined channel automatically upon logout.

## Join Channel(`channelJoin`)

(void) `logout`; (void) `channelJoin:(NSString*)channelID`;

This method allows the users to join a specified channel. Whenever he/she joins a specified channel, he/she will be disconnected automatically from the previously joined channel, if any.

Once the user joins a channel, he/she receives the `onChannelJoined` callback, and other users in the same channel receive the `onChannelUserJoined` callback. If the user fails to join a specified channel, he/she receives the `onChannelJoinFailed` callback.

Parameter	Description
<code>channelID</code>	Channel name. It can be up to 128 visible characters.

## Leave Channel(`channelLeave`)

(void) `channelLeave:(NSString*)channelID`;

This method allows the users to leave the current channel. Once the user left the channel, all rest users in the channel receive the `onChannelUserLeaved` callback.

Parameter	Description
<code>channelID</code>	Channel name. It can be up to 128 visible characters .

## Query Channel User Number(`channelQueryUserNum`)

(void) `channelQueryUserNum:(NSString*)channelID`;

This method allows the users to query the number of users in a specified channel. Upon a successful execution, the users will receive the `onChannelQueryUserNumResult` callback.



Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .

## Set Channel Attributes(channelSetAttr)

(void) channelSetAttr:(NSString\*)channelID name:(NSString\*)name value:(NSString\*)value;

This method sets the channel attributes. Whenever a change is made on the attributes, all users in the channel receive the onChannelAttrUpdated event.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
name	Attribute name. It can be up to 128 visible characters .
value	Attribute value. It can be up to 1024 visible characters .

## Delete Channel Attributes(channelDelAttr)

(void) channelDelAttr:(NSString\*)channelID name:(NSString\*)name;

This method deletes the specific attribute of a specified channel.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
name	Attribute name. It can be up to 128 visible characters .

## Delete All Channel Attributes(channelClearAttr)

(void) channelClearAttr:(NSString\*)channelID;

This method deletes all the attributes of a specified channel.




Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .

## Initiate a Call(channellInviteUser)

```
(void) channellInviteUser:(NSString*)channelID account:(NSString*)account uid:(uint32_t)uid ;
```

This method initiates a call, that is, inviting other user to join a specific channel.

If the call fails, the user will receive onInviteFailed event. Possible reasons are listed as follows:

-  The other user is offline;
-  Network issue in the local;
-  Server error;

If the user(calling) gets confirmation from the other user(called), the user will receive onInviteReceivedByPeer event, and the other will receive onInviteReceived.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
account	User ID defined by the client.
uid	Obsolete. Set it as 0.

## Initiate a Call (channellInviteUser2)

```
(void) channellInviteUser2:(NSString*)channelID account:(NSString*)account  
extra:(NSString*)extra;
```

Same as above, but parameter change.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
account	User ID defined by the client.
extra	Set it as NULL

## Send DTMF Message to Other User(channelInviteDTMF)

```
(void) channelInviteDTMF:(NSString*)channelID phoneNum:(NSString*)phoneNum  
dtmf:(NSString*)dtmf;
```

This method sends DTMF message to the other user for calling via SIP gateway.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
phoneNum	The phone number of the other user.
dtmf	The subsequent analog phone number(DTMF) to be entered after the call gets through.

## Accept Call(channelInviteAccept)

```
(void) channelInviteAccept:(NSString*)channelID account:(NSString*)account uid:(uint32_t)uid ;
```

This method accepts the received call request/invitation.  
Once this API is called, the other user will receive onInviteAcceptedByPeer event.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
account	User ID defined by the client.
uid	Obsolete. Set it as 0.

## Reject Call(channelInviteRefuse)

```
(void) channelInviteRefuse:(NSString*)channelID account:(NSString*)account uid:(uint32_t)uid ;
```

This method rejects the received call invitation/request.  
Once this API is called, the other user will receive the onInviteRefusedByPeer event.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
account	User ID defined by the client.
uid	Obsolete. Set it as 0.

## End Call(channelInviteEnd)

```
(void) channelInviteEnd:(NSString*)channelID account:(NSString*)account uid:(uint32_t)uid;
```

This method closed the call after the call has been connected.  
The user receives the onInviteEndByMyself event, and the other user receives onInviteEndByPeer.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
account	User ID defined by the client.
uid	Obsolete. Set it as 0.

## Send Point-to-Point Message(messageInstantSend)

```
(void) messageInstantSend:(NSString*)account uid:(uint32_t)uid msg:(NSString*)msg  
msgID:(NSString*)msgID;
```

This method send point-to-point message to a specific account.

Upon a successful execution, the user receives onMessageSendSuccess event, and the other user receives onMessageInstantReceive.

If the execution fails, the user receives onMessageSendError event.

Parameter	Description
account	User ID defined by the client.

Parameter	Description
uid	Obsolete. Set it as 0.
msg	Message body. Each message must be no greater than 8K visible characters.

### Send Channel Message(messageChannelSend)

```
(void) messageChannelSend:(NSString*)channelID msg:(NSString*)msg
msgID:(NSString*)msgID;
```

This method sends channel message, and all users in the same receive onMessageChannelReceive event.

The user who sent the message receives onMessageSendSuccess event upon a successful execution, otherwise, receives onMessageSendError.

Parameter	Description
channelID	Channel name. It can be up to 128 visible characters .
msg	Message body. In a large group call, each message must be no greater than 1K visible characters. Each user cannot send more than 1 message per second, the entire channel can not send more than 100 messages per second. In a one-onone call, each message must be no greater than 8K visible characters.
msgID	Visible characters, can be set as NULL. It is used to mark the callback message.

### Set User Attributes(set\_attr)

```
(void) set_attr:(NSString*)name value:(NSString*)value;
```

This method sets the user attributes.

Parameter	Description
name	Attribute name. It can be up to 256 visible characters

Parameter	Description
value	Attribute value. It can be up to 128 visible characters

## Get Your Own User Attributes(**get\_attr**)

(void) get\_attr:(NSString\*)name;

This method gets your own user attributes. You will receive the onUserAttrResult event.

Parameter	Description
name	Attribute name. It can be up to 256 visible characters

## Get All of Your Own Attributes(**get\_attr\_all**)

(void) get\_attr\_all;

This method gets all of your attributes.  
You will receive the onUserAttrResult upon a successful execution.

## Get Attributes of Specific User(**get\_user\_attr**)

(void) get\_user\_attr:(NSString\*)account name:(NSString\*)name;

This method allows the user to get the attributes of a specific user. The user will receive *onUserAttrResult* event.

Parameter	Description
account	Used ID defined by the client.
name	Attribute name. It can be up to 256 visible characters

## Get All attributes of Specific User(**get\_user\_attr\_all**)

(void) get\_user\_attr\_all:(NSString\*)account;

This method allows the user to get all the attributes of a specific user. The user will receive the onUserAttrAllResult callback upon a successful execution.

Parameter	Description
account	User ID defined by the client.

## Check Login Status(isOnline)

(bool) isOnline;

This method checks whether the user is logged in the Agora signaling system or not, online or offline.

Parameter	Description
Return Value	1: Online 0: Offline

## Internal-test or Obsolete Interface

Interface	Prototype	Description
messageAppSend	(void) messageAppSend:(NSString*)msg msgID:(NSString*)msgID ;	Agora internal test only
messageChannelSendFast	(void) messageChannelSendFast:(NSString*)channelID msg:(NSString*)msg msgID:(NSString*)msgID ;	Agora internal test only
messagePushSend	(void) messagePushSend:(NSString*)account uid:(uint32_t)uid msg:(NSString*)msg msgID:(NSString*)msgID ;	Agora internal test only
messageChatSend	(void) messageChatSend:(NSString*)account uid:(uint32_t)uid msg:(NSString*)msg msgID:(NSString*)msgID ;	Agora internal test only
messageDTMFSend	(void) messageDTMFSend:(uint32_t)uid msg:(NSString*)msg msgID:(NSString*)msgID ;	Obsolete
setBackground	(void) setBackground:(uint32_t)bOut ;	Agora internal test only
setNetworkStatus	(void) setNetworkStatus:(uint32_t)bOut ;	Agora internal test only
Ping	(void) ping;	Agora internal test

		only
channelInvitePhone	(void) channelInvitePhone:(NSString*)channelID phoneNum:(NSString*)phoneNum uid:(uint32_t)uid ;	Obsolete
channelInvitePhone2	(void) channelInvitePhone2:(NSString*)channelID phoneNum:(NSString*)phoneNum sourcesNum:(NSString*)sourcesNum ;	Obsolete
channelInvitePhone3	(void) channelInvitePhone3:(NSString*)channelID phoneNum:(NSString*)phoneNum sourcesNum:(NSString*)sourcesNum extra:(NSString*)extra ;	Obsolete

## Events

### Connection Lost Event(onReconnecting)

@property (copy) void(^onReconnecting)(uint32\_t nretry) ;

This event is triggered when the connection with the Agora signaling system is lost after login.

Parameter	Description
nretry	The current reconnection number.

### Connection Recovered Event(onReconnected)

@property (copy) void(^onReconnected)(int fd) ;

This event is triggered when the application is reconnected to the Agora signaling system.

Parameter	Description
fd	For Agora internal use.

### User Online Event(onLoginSuccess)

@property (copy) void(^onLoginSuccess)(uint32\_t uid,int fd) ;

This event is triggered when the user is logged into the Agora signaling system.



Parameter	Description
uid	obsolete
fd	For Agora internal use.

## User Offline Event(onLogout)

@property (copy) void(^onLogout)(AgoraEcode ecodes) ;

This event is triggered when the user logged out the Agora signaling system.

Parameter	Description
ecode	error code

## User Login Failed Event(onLoginFailed)

@property (copy) void(^onLoginFailed)(AgoraEcode ecodes) ;

This callback is triggered when the user fails to log into the Agora signaling system.

Parameter	Description
ecode	error code

## User Joined Channel Event(onChannelJoined)

@property (copy) void(^onChannelJoined)(NSString\* channelId) ;

This event is triggered when the user has joined a channel.

Parameter	Description
channelID	Channel name

## User Failed to Join Channel Event(onChannelJoinFailed)

@property (copy) void(^onChannelJoinFailed)(NSString\* channelId,AgoraEcode ecodes) ;

This event is triggered when the user fails to join a channel.

Parameter	Description
channelID	Channel name
ecode	error code

### User Left Channel Event(`onChannelLeaved`)

```
@property (copy) void(^onChannelLeaved)(NSString* channelID,AgoraEcode ecode) ;
```

This event is triggered when the user left a channel.

Parameter	Description
channelID	Channel Name
ecode	Error code

### Other User Joined Channel Event(`onChannelUserJoined`)

```
@property (copy) void(^onChannelUserJoined)(NSString* account,uint32_t uid) ;
```

This event is triggered when other user joined the channel.

Parameter	Description
account	Used ID defined by the client.
uid	obsolete

### Other User Left Channel Event(`onChannelUserLeaved`)

```
@property (copy) void(^onChannelUserLeaved)(NSString* account,uint32_t uid) ;
```

This event is triggered when other user left the channel.

Parameter	Description
account	Used ID defined by the client.
uid	obsolete

### User List Received Event(`onChannelUserList`)

@property (copy) void(^onChannelUserList)(NSMutableArray\* accounts, NSMutableArray\* uids);

This event is triggered when the user has joined a non-large channel(less than 200 people).

Parameter	Description
accounts	List of User ID defined by the client
uids	obsolete

### User Number Received Event(`onChannelQueryUserNumResult`)

@property (copy) void(^onChannelQueryUserNumResult)(NSString\* channelId,AgoraEcode ecode,int num) ;




This event is triggered when the user has queried the user number of the channel.

Parameter	Description
channelID	Channel name
ecode	Error code
num	The query result

### Channel Attribute Changed Event(`onChannelAttrUpdated`)

@property (copy) void(^onChannelAttrUpdated)(NSString\* channelId,NSString\* name,NSString\* value,NSString\* type) ;

This event is triggered when the channel attribute is changed.

Parameter	Description
channelID	Channel name
name	Attribute name
value	Attribute value
type	Change types:  "update": Update  "del": Delete  "clear": Delete all

### Call Invitation Received Event(`onInviteReceived`)

```
@property (copy) void(^onInviteReceived)(NSString* channelID,NSString* account,uint32_t uid)
;
```

This event is triggered when the user received a call invitation/request.

Parameter	Description
channelID	Channel name
account	Used ID defined by the client.
uid	obsolete

### Other User Received Call Event(`onInviteReceivedByPeer`)

```
@property (copy) void(^onInviteReceivedByPeer)(NSString* channelID,NSString*
account,uint32_t uid) ;
```

This event is received the caller when the other user has received the call invitation/request.

Parameter	Description
channelID	Channel name
account	User ID defined by the client
uid	obsolete

## Other User Accepted Call Event(`onInviteAcceptedByPeer`)

```
@property (copy) void(^onInviteAcceptedByPeer)(NSString* channelId,NSString* account,uint32_t uid) ;
```

This event is received by the caller when the other user has accepted the call invitation/request.

Parameter	Description
channelID	Channel name
account	User ID defined by the client
uid	obsolete

## Other User Rejected Call Event(`onInviteRefusedByPeer`)

```
@property (copy) void(^onInviteRefusedByPeer)(NSString* channelId,NSString* account,uint32_t uid) ;
```

This event is received by the caller when the other user rejected the call invitation/request.

Parameter	Description
channelID	Channel name
account	Used ID defined by the client
uid	obsolete

## Call Failed Event (`onInviteFailed`)

```
@property (copy) void(^onInviteFailed)(NSString* channelId,NSString* account,uint32_t uid,AgoraEcode ecode) ;
```

This event is triggered when a call fails.

Parameter	Description
channelID	Channel name
account	Used ID defined by the client
uid	obsolete

## Other User Ended Call Event(**onInviteEndByPeer**)

@property (copy) void(^onInviteEndByPeer)(NSString\* channelId,NSString\* account,uint32\_t uid) ;

This event is triggered when the other user ended the call.

Parameter	Description
channelID	Channel name
account	Used ID defined by the client
msg	Message body

## Call Ended Event(**onInviteEndByMyself**)

@property (copy) void(^onInviteEndByMyself)(NSString\* channelId,NSString\* account,uint32\_t uid) ;

This event is received by the user when the user ended the call.

Parameter	Description
channelID	Channel name
account	User ID defined by the client
uid	obsolete
msg	Message body

## Message Failed Event(**onMessageSendError**)

@property (copy) void(^onMessageSendError)(NSString\* messageId,AgoraEcode ecode) ;

This event is triggered when the user fails to send a message.

Parameter	Description
messageID	Message ID

Parameter	Description
ecode	Error code

### Message Sent Event(**onMessageSendSuccess**)

@property (copy) void(^onMessageSendSuccess)(NSString\* messageID) ;

This event is triggered when the user sent a message successfully.

Parameter	Description
messageID	Message ID

### Other User Received Message Event(**onMessageInstantReceive**)

@property (copy) void(^onMessageInstantReceive)(NSString\* account,uint32\_t uid,NSString\* msg) ;

This event is triggered when the other user has received the message.

Parameter	Description
account	User ID defined by the client
uid	Obsolete
msg	Message body

### Channel Message Received Event(**onMessageChannelReceive**)

@property (copy) void(^onMessageChannelReceive)(NSString\* channelId,NSString\* account,uint32\_t uid,NSString\* msg) ;

This event is triggered upon receiving the channel message.

Parameter	Description
channelID	Channel name
account	User ID defined by the client

Parameter	Description
uid	obsolete
msg	Message body

## Log Printed Event(onLog)

@property (copy) void(^onLog)(NSString\* txt) ;

This event is triggered upon a line of log is printed out.

Parameter	Description
txt	Content of a log line

## Attribute Received Event(onUserAttrResult)

@property (copy) void(^onUserAttrResult)(NSString\* account,NSString\* name,NSString\* value) ;

This event is triggered when the user has queried the attribute of a specific user.

Parameter	Description
account	User ID defined by the client
name	Attribute name
value	The json value of all attributes

## All Attribute Received Event(onUserAttrAllResult)

@property (copy) void(^onUserAttrAllResult)(NSString\* account, NSString\* value) ;

This event is triggered when the user has queried all the attributes of a specific user.

Parameter	Description
account	User ID defined by the client



Parameter	Description
value	The json value of all attributes

## Internal-test or Obsolete Interface

Interface	Prototype	Description
onMessageAppReceived	@property (copy) void(^onMessageAppReceived)(NSString* msg) ;	Agora internal test only

## Error Code

Error Code	Value	Description
SUCCESS	0	No error
LOGOUT_E_OTHER	100	Unknown reason
LOGOUT_E_USER	101	User logged out
LOGOUT_E_NET	102	Network issue
LOGOUT_E_KICKED	103	The account is logged in other place
LOGOUT_E_PACKET	104	obsolete
LOGOUT_E_TOKENEXPIRED	105	Signaling Key expired
LOGOUT_E_OLDVERSION	106	obsolete
LOGOUT_E_TOKENWRONG	107	obsolete
LOGIN_E_OTHER	200	Unknown reason
LOGIN_E_NET	201	Network issue
LOGIN_E_FAILED	202	Rejected by server
LOGIN_E_CANCEL	203	User cancelled the login
LOGIN_E_TOKENEXPIRED	204	Signaling Key expired, login rejected
LOGIN_E_OLDVERSION	205	obsolete
LOGIN_E_TOKENWRONG	206	Signaling Key abnormal
LOGIN_E_TOKEN_KICKED	207	User has used a renewed Signaling Key to log in some place else.
JOINCHANNEL_E_OTHER	300	Failed to join a channel
SENDMESSAGE_E_OTHER	400	Failed to send message
SENDMESSAGE_E_TIMEOUT	401	Timed-out in sending message
QUERYUSERNUM_E_OTHER	500	Failed to query the channel user number.
QUERYUSERNUM_E_TIMEOUT	501	Timed-out in querying the channel user number
QUERYUSERNUM_E_BYUSER	501	obsolete
LEAVECHANNEL_E_OTHER	600	Left channel out of unknown reason
LEAVECHANNEL_E_KICKED	601	Kicked out by the administrator
LEAVECHANNEL_E_BYUSER	602	The user has left the channel
LEAVECHANNEL_E_LOGOUT	603	Kicked out of the channel upon logout
LEAVECHANNEL_E_DISCONN	604	Left the channel due to network outage
INVITE_E_OTHER	700	Call fails
INVITE_E_REINVITE	701	Repeated calls

INVITE_E_NET	702	Network issue
INVITE_E_PEEROFFLINE	703	The other user is offline
INVITE_E_TIMEOUT	704	Call timed-out
INVITE_E_CANTRECV	705	obsolete
GENERAL_E	1000	general error
GENERAL_E_FAILED	1001	general error - failure
GENERAL_E_UNKNOWN	1002	general error-unknown
GENERAL_E_NOT_LOGIN	1003	general error-action before login
GENERAL_E_WRONG_PARAM	1004	general error-parameter calling error