Agora Signaling SDK for Windows Reference Manual v1.0

support@agora.io

Contents

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 Windows to access the Agora Signaling System.

Agora Signaling SDK for Windows

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:

Platform	System Requirement	Network Requirement
Windows	XP SP3 or later	Port: HTTP 80， TCP 8181

Required SDK

The Agora Signaling SDK is available from agora.io/developer, or contact 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

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.

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 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.

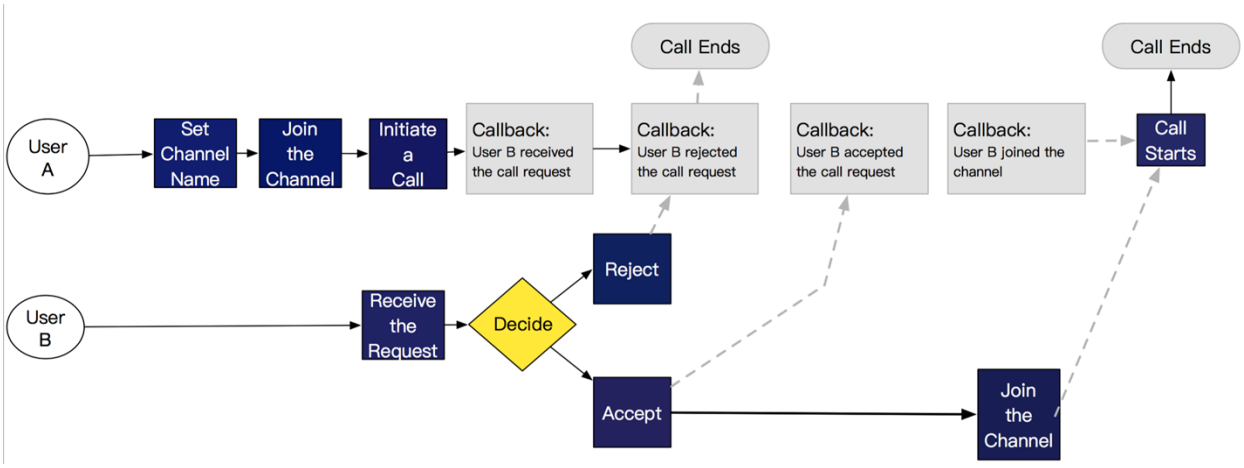
Call Flow

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.

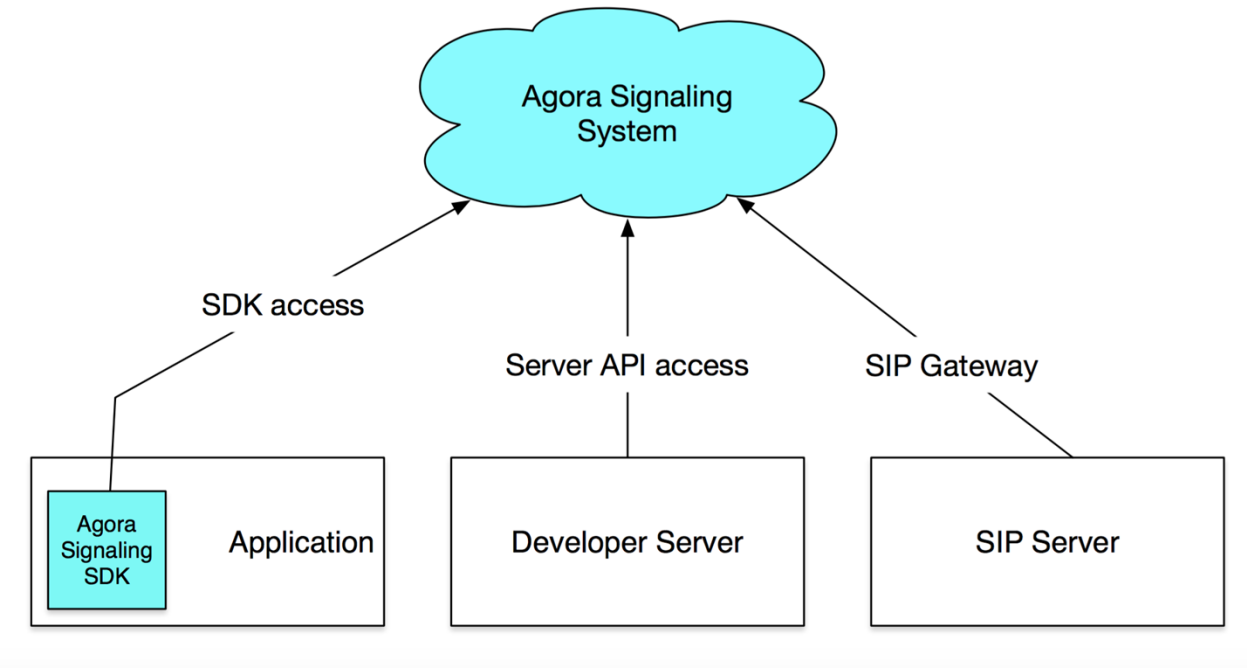
The following is the call flow of joining a channel first before initialing a call:



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 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.

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

Notes

- If you want to renew the App Certificate for some reason, contact 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.ioC5D15F8FD394285DA5227B533302
A518fe1a0437bf217bdd34cd65053fb0fe1d2592000)"
= "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 https://dashboard.agora.io.
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: • account: The User ID defined by the client. • appId: 32-character App ID string. • appCertificate: 32-character App Certificate string. • expiredTime: The UTC timestamp indicates that from the specific moment the user cannot access the Agora Signaling System any more

Signaling SDK API Reference

Methods

Login(login)

public virtual void login (char const * appId, size_t appId_size,char const * account, size_t account_size, char const * token, size_t token_size,uint32_t uid,char const * deviceID, size_t deviceID_size)=0;

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 https://dashboard.agora.io.
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 *account* parameter mentioned across this document follow the same rule.
token	Signaling Key generated by App ID and App Certificate. See Obtaining and Using a Signaling Key for details.
uid	Expired, set it as 0.
deviceID	Set it as NULL.

Log out(logout)

public virtual void logout () = 0;

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)

public virtual void channelJoin (char const * channelID, size_t channelID_size) = 0;

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

Leave Channel(channelLeave)

public virtual void channelLeave (char const * channelID, size_t channelID_size) = 0;

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

Query Channel User Number(channelQueryUserNum)

public virtual void channelQueryUserNum (char const * channelID, size_t channelID_size) = 0;

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)

public virtual void channelSetAttr (char const * channelID, size_t channelID_size,char const * name, size_t name_size,char const * value, size_t value_size) = 0;

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)

public virtual void channelSetAttr (char const * channelID, size_t channelID_size,char const * name, size_t name_size,char const * value, size_t value_size) = 0;

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)

public virtual void channelClearAttr (char const * channelID, size_t channelID_size) = 0;

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(channelInviteUser)

public virtual void channelInviteUser (char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid=0) = 0;

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 (channelInviteUser2)

public virtual void channelInviteUser2 (char const * channelID, size_t channelID_size,char const * account, size_t account_size,char const * extra, size_t extra_size) = 0;

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)

public virtual void channelInviteDTMF (char const * channelID, size_t channelID_size,char const * phoneNum, size_t phoneNum_size,char const * dtmf, size_t dtmf_size) = 0;

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)

public virtual void channelInviteAccept (char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid) = 0;

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)

public virtual void channelInviteRefuse (char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid) = 0;

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)

public virtual void channelInviteEnd (char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid) = 0;

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)

public virtual void channelInviteEnd (char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid) = 0;

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

Send Channel Message(messageChannelSend)

public virtual void messageChannelSend (char const * channelID, size_t channelID_size,char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;

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)

public virtual void set_attr (char const * name, size_t name_size,char const * value, size_t value_size) = 0;

This method sets the user attributes.

Parameter	Description
name	Attribute name. It can be up to 256 visible characters
value	Attribute value. It can be up to 128 visible characters

Get Your Own User Attributes(get_attr)

public virtual void get_attr (char const * name, size_t name_size) = 0;

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)

public virtual void get_attr_all () = 0;

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

Get Attributes of Specific User(get_user_attr)

public virtual void get_user_attr (char const * account, size_t account_size,char const * name, size_t name_size) = 0;

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)

public virtual void get_user_attr_all (char const * account, size_t account_size) = 0;

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)

public virtual bool isOnline () = 0;

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	public virtual void messageAppSend (char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;	Agora internal test only
messageChannelSendFast	public virtual void messageChannelSendFast (char const * channelID, size_t channelID_size,char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;	Agora internal test only
messagePushSend	public virtual void messagePushSend (char const * account, size_t account_size,uint32_t uid,char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;	Agora internal test only
messageChatSend	public virtual void messageChatSend (char const * account, size_t account_size,uint32_t uid,char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;	Agora internal test only

messageDTMFSend	public virtual void messageDTMFSend (uint32_t uid,char const * msg, size_t msg_size,char const * msgID, size_t msgID_size) = 0;	Obsolete
setBackground	public virtual void setBackground (uint32_t bOut) = 0;	Agora internal test only
setNetworkStatus	public virtual void setNetworkStatus (uint32_t bOut) = 0;	Agora internal test only
Ping	virtual void ping () = 0;	Agora internal test only
channelInvitePhone	public virtual void channelInvitePhone (char const * channelID, size_t channelID_size,char const * phoneNum, size_t phoneNum_size,uint32_t uid=0) = 0;	Obsolete
channelInvitePhone2	public virtual void channelInvitePhone2 (char const * channelID, size_t channelID_size,char const * phoneNum, size_t phoneNum_size,char const * sourcesNum, size_t sourcesNum_size) = 0;	Obsolete
channelInvitePhone3	public virtual void channelInvitePhone3 (char const * channelID, size_t channelID_size,char const * phoneNum, size_t phoneNum_size,char const * sourcesNum, size_t sourcesNum_size,char const * extra, size_t extra_size) = 0;	Obsolete

Events

Connection Lost Event(onReconnecting)

public virtual 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)

public virtual 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)

public virtual 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)

public virtual void onLogout(int ecode) {}

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

Parameter	Description
ecode	error code

User Login Failed Event(onLoginFailed)

public virtual void onLoginFailed(int ecode) {}

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)

public virtual void onChannelJoined(char const * channelID, size_t channelID_size) {}

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

Parameter	Description
channelID	Channel name

User Failed to Join Channel Event(onChannelJoinFailed)

public virtual void onChannelJoinFailed(char const * channelID, size_t channelID_size,int ecode) {}

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)

public virtual void onChannelLeaved(char const * channelID, size_t channelID_size,int 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)

public virtual void onChannelUserJoined(char const * account, size_t account_size,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)

public virtual void onChannelUserLeaved(char const * account, size_t account_size,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)

public virtual void onChannelUserList(int n,char** accounts,uint32_t* 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)

public virtual void onChannelQueryUserNumResult(char const * channelID, size_t channelID_size,int 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)

public virtual void onChannelAttrUpdated(char const * channelID, size_t channelID_size,char const * name, size_t name_size,char const * value, size_t value_size,char const * type, size_t type_size) {}

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)

public virtual void onInviteReceived(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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)

public virtual void onInviteReceivedByPeer(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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)

public virtual void onInviteAcceptedByPeer(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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)

public virtual void onInviteRefusedByPeer(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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

Parameter	Description
uid	obsolete

Call Failed Event (onInviteFailed)

public virtual void onInviteFailed(char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid,int 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)

public virtual void onInviteEndByPeer(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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)

public virtual void onInviteEndByMyself(char const * channelID, size_t channelID_size,char const * account, size_t account_size,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)

public virtual void onMessageSendError(char const * messageID, size_t messageID_size,int ecode) {}

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

Parameter	Description
messageID	Message ID
ecode	Error code

Message Sent Event(onMessageSendSuccess)

public virtual void onMessageSendSuccess(char const * messageID, size_t messageID_size) {}

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

Parameter	Description
messageID	Message ID

Other User Received Message Event(onMessageInstantReceive)

public virtual void onMessageInstantReceive(char const * account, size_t account_size,uint32_t uid,char const * msg, size_t msg_size) {}

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)

public virtual void onMessageChannelReceive(char const * channelID, size_t channelID_size,char const * account, size_t account_size,uint32_t uid,char const * msg, size_t msg_size) {}

This event is trigged upon receiving the channel message.

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

Log Printed Event(onLog)

public virtual void onLog(char const * txt, size_t txt_size) {}
This event is trigged upon a line of log is printed out.

Parameter	Descripton
txt	Content of a log line

Attribute Received Event(onUserAttrResult)

public virtual void onUserAttrResult(char const * account, size_t account_size,char const * name, size_t name_size,char const * value, size_t value_size) {}

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)

public virtual void onUserAttrResult(char const * account, size_t account_size,char const * name, size_t name_size,char const * value, size_t value_size) {}

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
value	The json value of all attributes

Internal-test or Obsolete Interface

Interface	Prototype	Description
onMessageAppReceived	public virtual void onMessageAppReceived(char const * msg, size_t msg_size) {}	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