



Agora Native SDK for Web Reference Manual v1.7

support@agora.io

Contents

Introduction	4
Requirements	5
<i>Compatibility.....</i>	<i>5</i>
<i>Supported Browsers</i>	<i>5</i>
<i>Operating System Requirements.....</i>	<i>5</i>
<i>Required Documents.....</i>	<i>5</i>
Capabilities and Limitations	5
Getting Started	6
Where to Get the SDK	6
About Keys	6
<i>Obtaining and Using an App ID.....</i>	<i>9</i>
<i>Obtaining and Using a Channel Key</i>	<i>9</i>
Developing and Deploying Application.....	13
Running the Sample Web Application.....	13
<i>Client</i>	<i>13</i>
<i>Server.....</i>	<i>14</i>
Encrypting Data.....	14
Installing AgoraWebAgent on Windows or Mac	15
<i>On Windows.....</i>	<i>16</i>
<i>On Mac</i>	<i>18</i>
Uninstalling AgoraWebAgent on Windows or Mac	19
<i>On Windows.....</i>	<i>19</i>
<i>On Mac</i>	<i>19</i>
Agora Native SDK for Web API Reference – JavaScript Classes.....	21
AgoraRTC API Methods	21
<i>Create Client Object (createRtcClient).....</i>	<i>21</i>
<i>Create Stream Object (createStream)</i>	<i>21</i>
Client API Methods and Callback Events.....	22
<i>Initialize Client Object (init)</i>	<i>22</i>
<i>Join AgoraRTC Channel(join)</i>	<i>23</i>
<i>Renew Channel Key (renewChannelKey).....</i>	<i>24</i>
<i>Leave AgoraRTC Channel(leave).....</i>	<i>24</i>
<i>Publish Local Stream (publish)</i>	<i>25</i>
<i>Unpublish Local Stream (unpublish)</i>	<i>25</i>
<i>Subscribe Remote Stream (subscribe).....</i>	<i>26</i>
<i>Unsubscribe Remote Stream (unsubscribe)</i>	<i>27</i>
<i>Enumerate Platform Device (getDevices)</i>	<i>27</i>
<i>Enable Built-in Encryption(setEncryptionSecret).....</i>	<i>27</i>
<i>Set Built-in Encryption Mode(setEncryptionMode).....</i>	<i>28</i>

Select Device (<i>selectdevice</i>)	28
Start Recording(<i>startRecording</i>)	29
Stop Recording (<i>stopRecording</i>).....	29
Query Recording Status(<i>queryRecordingStatus</i>)	30
Get Desktop Window List(<i>getWindows</i>)	30
Start Screen Sharing(<i>startScreenSharing</i>).....	30
Set Shared Window(<i>setScreenSharingWindow</i>).....	31
Stop Screen Sharing(<i>stopScreenSharing</i>)	31
Release Resources(<i>close</i>).....	31
Remote Stream Added Event (<i>stream-added</i>)	32
Remote Stream Subscribed Event (<i>stream-subscribed</i>)	32
Peer User Left Channel Event (<i>peer-leave</i>)	32
Remote User Muted Video Event(<i>peer-mute-video</i>).....	33
Remote User Muted Audio Event(<i>peer-mute-audio</i>).....	33
Connection Lost/Interrupted Event (<i>error</i>).....	34
Stream API Methods	34
Initialize Local Stream Object(<i>init</i>)	34
Close Video Stream (<i>close</i>).....	35
Play Video or Audio Stream(<i>play</i>).....	35
Stop Playing Video Stream(<i>stop</i>).....	35
Enable Audio Stream (<i>enableAudio</i>)	35
Disable Audio Stream (<i>disableAudio</i>)	35
Enable Video Stream(<i>enableVideo</i>)	36
Disable Video Stream (<i>disableVideo</i>)	36
Retrieve User ID (<i>getId</i>)	36
Set Video Profile (<i>setVideoProfile</i>)	36
Error and Warning Messages.....	37
Error Messages	38
Warning Messages	41

Introduction

The Agora Native SDK for Web includes the Agora JavaScript SDK and one application called AgoraWebAgent.

The **Agora Native SDK for Web** allows you 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 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 optimize Agora SDK 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, muted, and 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 to the users who have adopted the Recording Key schema.
- **Data Encryption:** Encrypt the audio and video packets. You cannot use the recording function if the data encryption function is enabled in a channel.
- **Whiteboard:** Users from different locations can draw, annotate, and share PDF documents on the whiteboard to visualize and simplify the communication.

The **Agora Native SDK for Web** provides three straightforward JavaScript classes to deliver these features, which support:

- A single *Client* object for establishing and controlling sessions.
- Multiple *Stream* objects for managing different voice and video media streams.

Top-level *AgoraRTC* object for creating the appropriate *Client* and *Stream* objects. For details on the API definition and sample code, see [Agora Native SDK for Web API Reference – JavaScript Classes](#).

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

Requirements

Compatibility

Native SDK 1.3 or later

Supported Browsers

Do not use an unsupported browser, for example, Opera and Firefox are not supported and will behave unpredictably.

Platform	Browser	Requirements
Windows	Edge	25.10586.0.0
Windows 10	Chrome	51.0.2704.103 (64 bit) <small>note</small>
Mac	Safari	9.1.1(11601.6.17)

Note:

Chrome is supported on Windows at 640x480p 30 fps resolution. Other higher resolutions will be supported in future releases.

Operating System Requirements

OS	Requirements
Windows	Windows 7 or later
Mac	OS X v10.10 or later

Required Documents

- 🔗 **This reference manual:** Mandatory
- 🔗 **Agora Recording Server User Guide:** Read it together with this reference manual to enable the recording function if necessary.
- 🔗 **Agora Whiteboard SDK Reference Manual:** Read it together with this reference manual to enable the whiteboard function if necessary.

All documents can be downloaded at agora.io/developer.

Capabilities and Limitations

- 🔗 The maximum supported video resolution : 720p 15fps and 480p 30fps.
- 🔗 A PC can only have one instance running.

Getting Started

This section explains how to deploy and use the Agora Native SDK for Web.

Where to Get the SDK

The **Agora Native SDK for Web** is available from agora.io/developer or contact sales@agora.io to get access to the latest version.

The SDK is delivered as a package of the following files:

Component	Description
./agent1	It includes the following applications: AgoraWebAgent.pkg AgoraWebAgentSetup.exe
./client	Sample web application.
./doc	Agora Web SDK Documentation: Agora Native SDK for Web Reference Manual v1_7_EN.pdf Agora Native SDK for Web Reference Manual v1_7_CHS.pdf
./lib	Required libraries
./server	Web service-side sample code

1. After the deployment, you can also download the application package from the client system prompt. For more information, see [Installing AgoraWebAgent on Windows or Mac](#).

About Keys

This section describes the concepts and use of App ID, APP Certificate, Channel Key and Recording Key.

Static Key:

Each Agora account can create multiple projects, and each project has a unique App ID. Be sure to obtain an **App ID**, required when you use APIs to access the Agora Global Network. In our network, the App ID sets you apart from others. There is no overlap, even when channels have the same name. An App ID is a static key, and if someone else illicitly obtains your static App ID, 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.

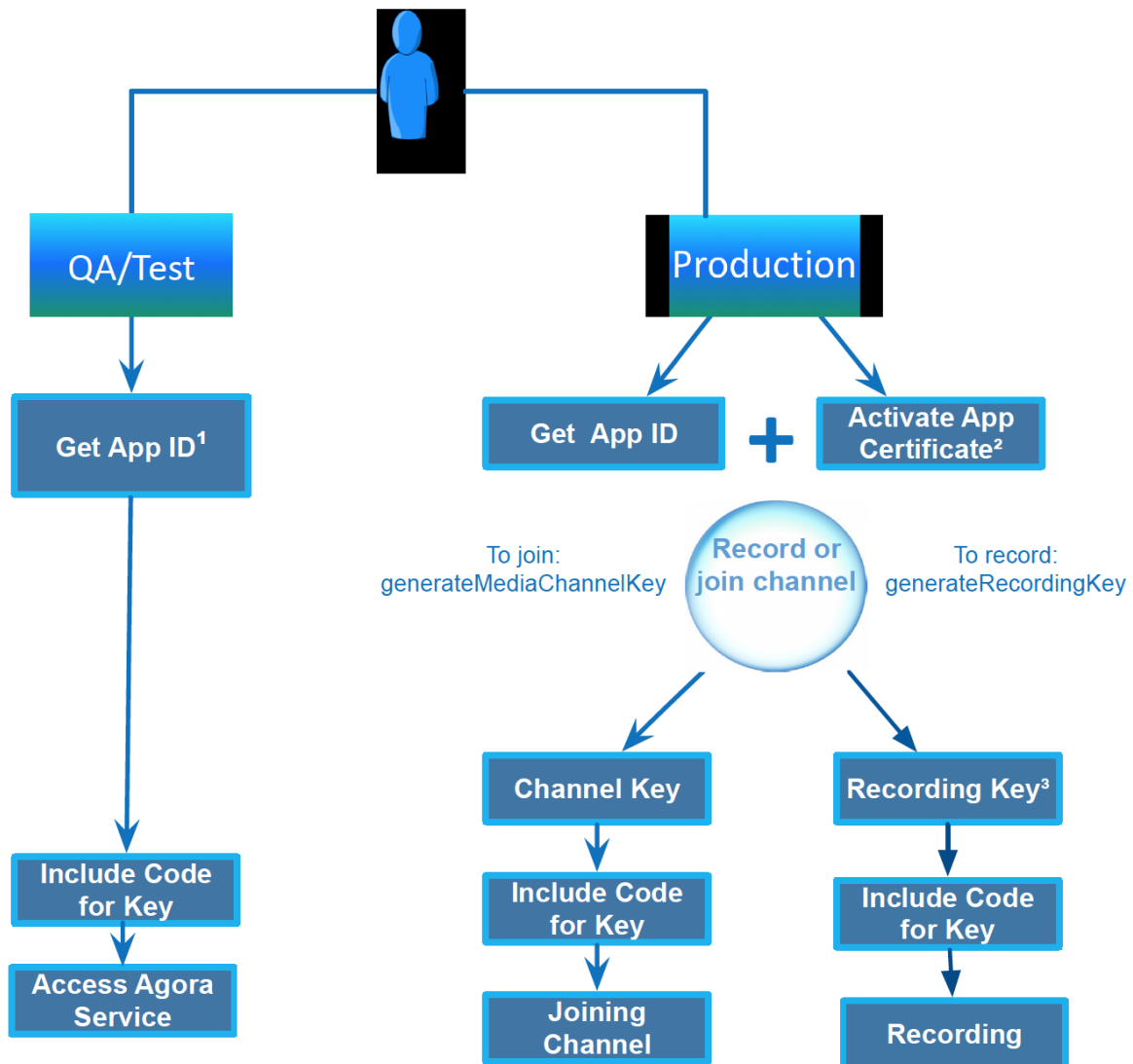
Dynamic Key:

Dynamic Key is a more secure user authentication schema for the Agora SDK. Dynamic Key is a general name compared with Static Key, and it has different and specific

names for different services. The Agora Native SDK may need the following dynamic keys according to your actual needs:

- **Channel Key:** When a user tries to enter a channel to access the Agora service, the back-end services use the App ID and App Certificate to generate a new Channel Key based on the HMAC encryption algorithm. The Channel Key is then passed to the client application. The client application calls the *join* interface function and passes the encoded Channel Key to the Agora server that authenticates users.
- **Recording Key:** When a user tries to use the recording function, the back-end services use the App ID and App Certificate to generate a new Channel Key based on the HMAC encryption algorithm. The Channel Key is then passed to the client application. The client application calls the *startRecording* interface function and passes the encoded Recording Key to the Agora server that authenticates users.

The following diagram depicts the key relations and applicable scenarios:



¹ App ID for test/lab or non-recording, low security purposes

² Cannot be used alone

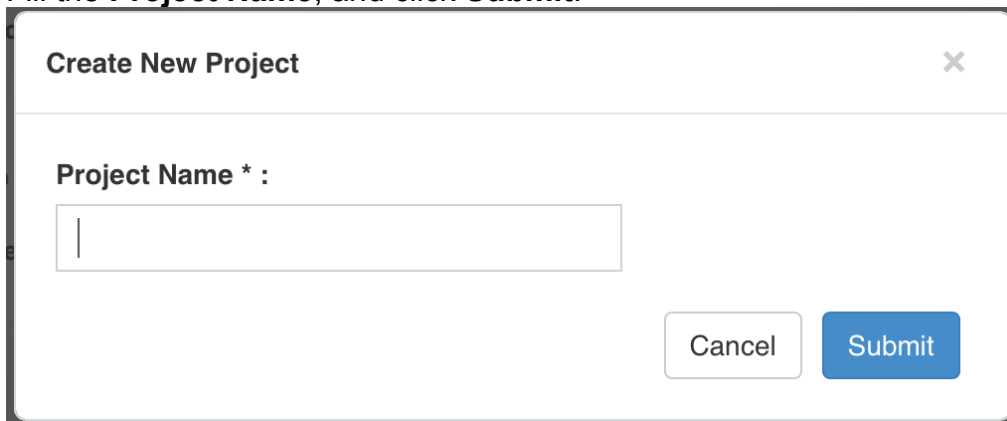
³ Required for recording

Obtaining and Using an App ID

Obtaining an App ID

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

1. Sign up for a new account at <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 image shows a 'Create New Project' dialog box. It has a title bar with the text 'Create New Project' and a close button (X). Below the title bar, there is a label 'Project Name * :' followed by a text input field. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Submit'.

4. Find your App ID under the project that you have created.

Using an App ID

Access the Agora services by using your unique App ID.

1. Enter the App ID in the starting window to enable audio or video communication in the demo.
2. Add the App ID to your code during your development.
3. Set the parameter `appld` as the App ID when calling `AgoraRTC.client.init()`.
4. Set the parameter `Key` as the App ID when calling `AgoraRTC.client.join()`.

Obtaining and Using a Channel Key

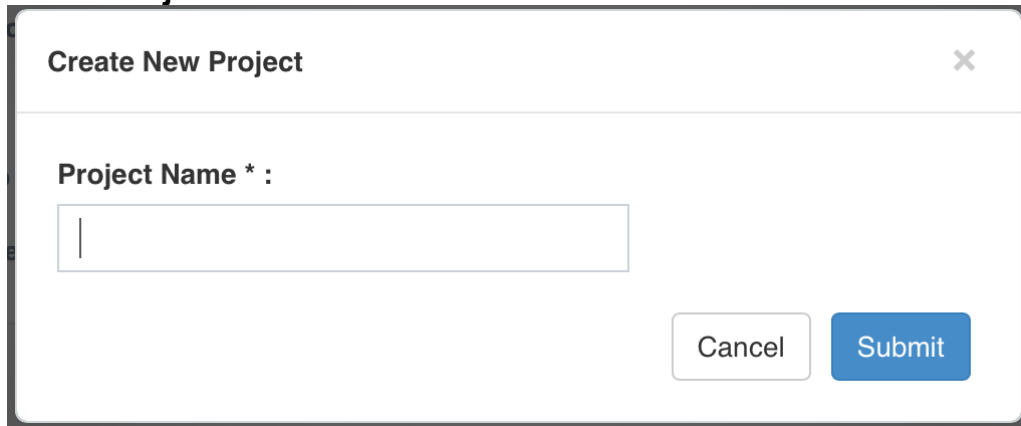
The following sections describe the use of Channel Keys:

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.

Do the following to create a project to obtain an App ID and an App Certificate at the same time:

1. Login <https://dashboard.agora.io>.
2. Click **Add New Project** on the **Projects** page of the dashboard.
3. Fill the **Project Name** and click **Submit**.



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 .
- 🔗 Keep your App Certificate on the server and never on any client machine.
- 🔗 See [Increased Security with Channel Keys](#) for the App Certificate usage.

Implementing the Channel Key Scheme

Integrate the Channel 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 directly in your application. For the sample code, refer to the following site: <https://github.com/AgoraLab/AgoraDynamicKey>

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

DynamicKey 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 to upgrade to DynamicKey4.

Using a Channel Key

Before a user joins a channel(that has started a call or received a meeting invitation), the following sequence occurs :

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 Channel Key and then passes the Channel Key back down to the client application.
The Channel Key is based on the App Certificate, App ID, Channel Name, Current Timestamp, Client User ID, Lifespan Timestamp, and so on.
3. The client application calls the AgoraRTC.client.join method, which requires the Channel Key as the first parameter.
4. The Agora server receives the Channel Key and confirms that the call comes from a legal user, and then allows it to access the Agora Global Network.
Note: With the Channel Key scheme implemented, the Channel Key replaces the original App ID when the user joins a channel.

Channel Keys expire after a certain amount of time and your application must call `renewChannelKey()` when a time-out occurs. The `onError` callback returns `ERR_CHANNEL_KEY_TIMEOUT(109)`. API Method names may vary across different platform SDKs.

Increased Security with Channel Keys

If your organization chooses Channel Keys, before a user joins a channel or starts the recording function, the client application must provide a new Channel Key to ensure that every call of the user is authorized by the signaling server. The Channel Key uses the HMAC/SHA1 signature schema to increase security for communications within your organization.

Channel Key Structure

Connect all fields in the sequence below (103 bytes in total).

Field	Type	Length	Description
Version	String	3	Channel Key version information of the algorithm
Sign	String	40	Hex code for the signature. It is a 40-byte string represented by hex code and calculated by HMAC algorithm based on inputs including the App Certificate and the following fields: <ul style="list-style-type: none">● Service Type: The visible ASCII string provided by Agora.io. See Service Type.● App ID: The 32-bit App ID string.● Timestamp: The timestamp created when the Channel Key is generated.● Random Number: A 32-bit integer in hex code; generated upon each query.● Channel: The channel name specified by the user with a maximum length of 64 bytes.● User ID: The User ID defined by the client.● Service Expiration Timestamp: The timestamp indicates that from the specific moment the user cannot use the Agora service any more.
App ID	String	32	App ID
Authorized Timestamp	Number	10	The timestamp represented by the number of seconds elapsed since 1-1-1970. The user can use the Channel Key to join a channel within 5 minutes after the Channel Key is generated. If the user does not join a channel after 5 minutes, this Channel Key is no longer valid.
Random Number	Integer	8	A 32-bit integer in hex code; generated upon each query.
Service Expiration Timestamp	Number	10	This timestamp indicates when the user cannot use the Agora service any more (for example, the user must leave an ongoing call). Set the value to 0 if there is no time limit. ¹

Table 1

1. When the value is set for Call Expiration Timestamp, it does not mean Channel Key will be expired, but only means the user will be kicked out from the channel. For example, if a conference call is around 1 hour, but if after the call is closed, someone is still in the channel, you will still be charged. By setting this timestamp, the users who have not left the channel after a call is closed will automatically be kicked out.

Service Type

Service	Value	Description
Session	ACS	The audio and video services provided by Agora.io. For example, when calling the AgoraRTC.client.join API, if a Channel Key is required, use this service type to generate the Channel Key.

Sign Encryption Algorithm: HMAC/SHA1

The Channel Key encoding uses the industry-standard HMAC/SHA1 approach for which there are libraries on most common server-side development platforms, such as Node.js, Java, Python, C++, and so on. For more information, refer to:

http://en.wikipedia.org/wiki/Hash-based_message_authentication_code

Contact www.agora.io for more information or technical support.

Developing and Deploying Application

Applications using the **Agora Native SDK for Web** are standard JavaScript applications. To deploy the application, you need to load the Agora JS library and also need access to the JS extension libraries provided with the SDK.

Load the Agora JS library AgoraRTCSdk-1.7.0.js. You can download it from

<https://rtcsdk.agora.io/AgoraRtcAgentSDK-1.7.0.js> or

<http://rtcsdk.agora.io/AgoraRtcAgentSDK-1.7.0.js>

Running the Sample Web Application

Client

The default sample code only requires your static **App ID** (which is simply entered into the sample application web page). To run the provided sample web application:

1. Ensure that you have a local web server installed, such as Apache, NginX, or Node.js.
2. Deploy the files under **/client/** to your web server.
3. Access the sample application page on your web server using one of the browsers listed in [Supported Browsers](#).

Do the following if you want to run the sample application with a **Channel Key**:

Note: For more information, see [Obtaining and Using a Channel Key](#).

1. Set up and launch the key-generation server.
2. Make an http/https request from the client to the key-generation server to fetch the generated Channel Key.
3. Call `AgoraRTC.client.join`.

Server

This code is only required when you want to experiment with using more secure **Channel Keys**. In production use, you integrate this logic into your own server-side applications and (re)code this in the programming languages you are already using for your server-based functionality.

Note: For more information, see [Using a Channel Key for Increased Security](#).

The sample code is in JavaScript and requires a standard Node.js server.

1. Install a standard Node.js server in your server or cloud infrastructure.
2. Run `npm install` under `../server/nodejs/`.
3. Fill in the values of your APP_ID (Previously called VENDOR_KEY) and APP_CERTIFICATE(Previously called SIGN_KEY) in `../server/nodejs/DemoServer.js`.
4. Launch the server with **`node DemoServer.js`**.

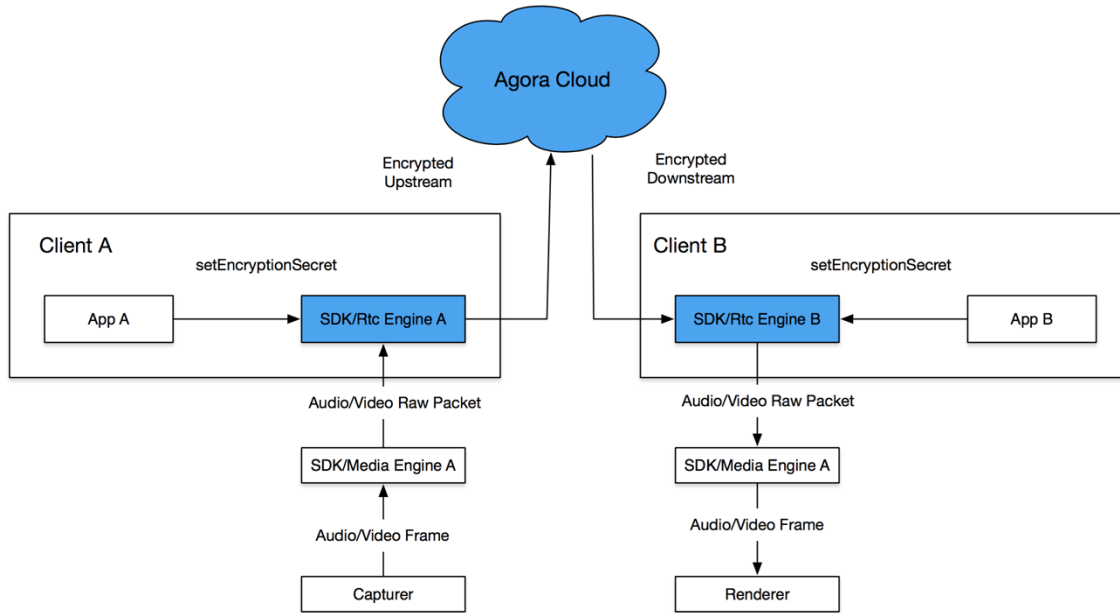
Encrypting Data

Agora SDKs allow your application to encrypt audio and video packets using the built-in Encryption of Agora SDK. If your application has integrated the built-in encryption of Agora SDK, then users are able to record an encrypted channel.

Agora SDKs support built-in encryption using the AES-128 or AES-256 encryption algorithm.

You can call `setEncryptionSecret` to enable the encryption function and call `setEncryptionMode` to set the mode to be used. For details, see [Agora Native SDK for Web API Reference – JavaScript Classes](#).

The following depicts the encryption process:



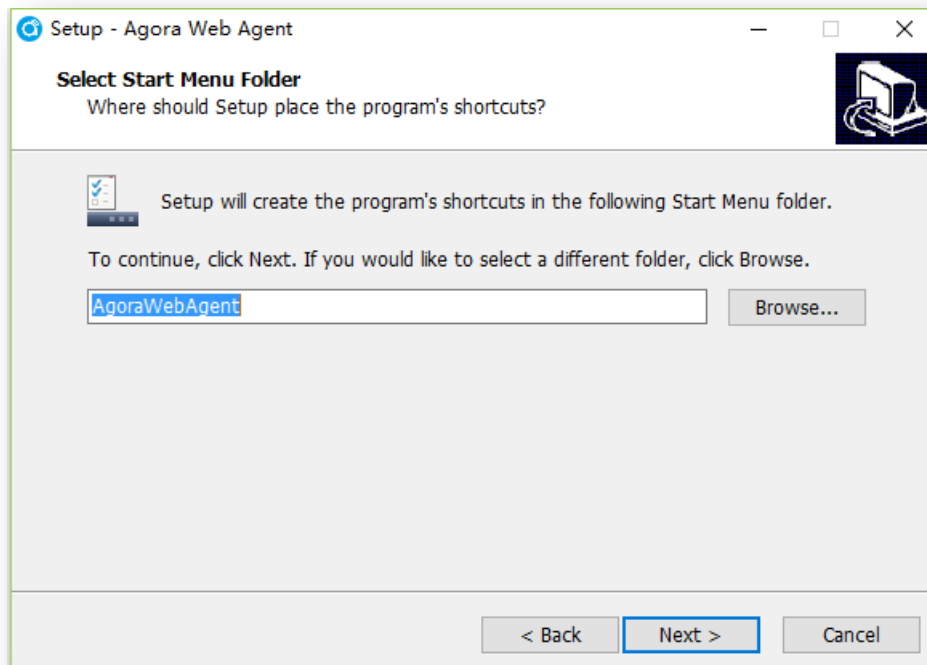
Installing AgoraWebAgent on Windows or Mac

After your deployment, the client will ask your users to install the AgoraWebAgent application on their Windows or Mac according to the system prompt. The system prompt includes a download link and an installation guide provided by Agora, and your users can directly use them or you can customize them to redirect the users to your website.

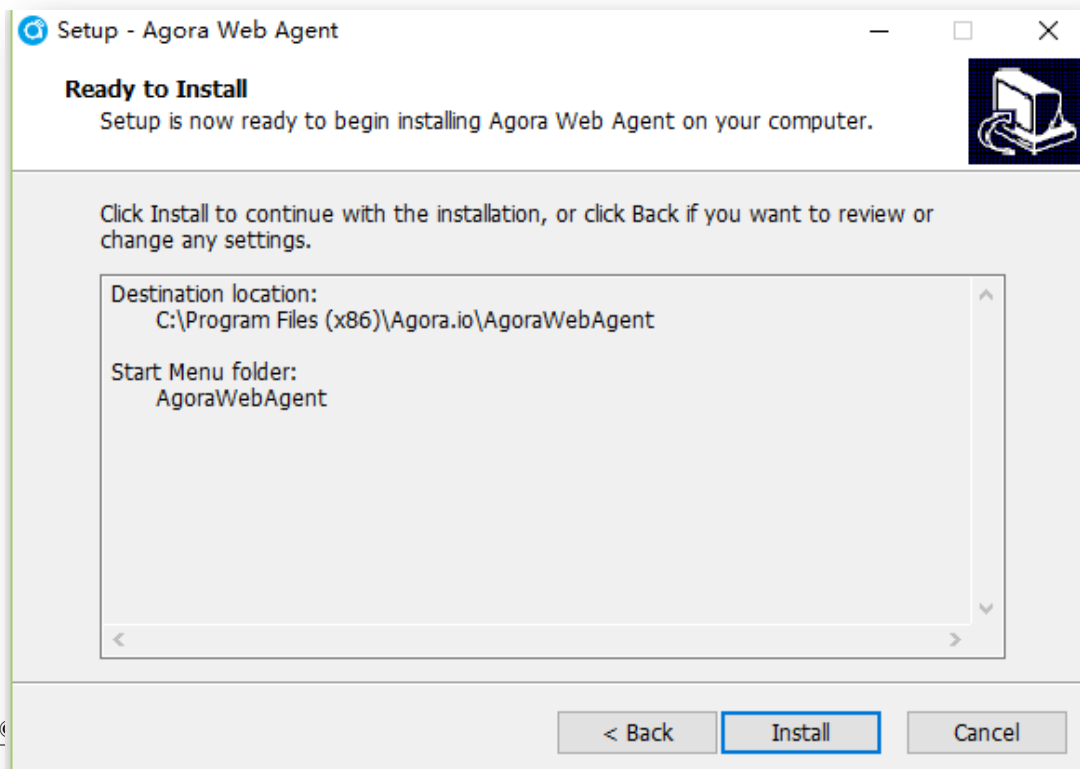
The following shows the installation guide again for your customization convenience:

On Windows

1. Double-click the AgoraWebAgent.exe downloaded from the system prompt.



2. Select the language for installation: **English**.
3. Click **Browse...** if you want to select a different folder. Otherwise, click **Next>**.
4. Click **Install** and wait until the progress bar is completed.



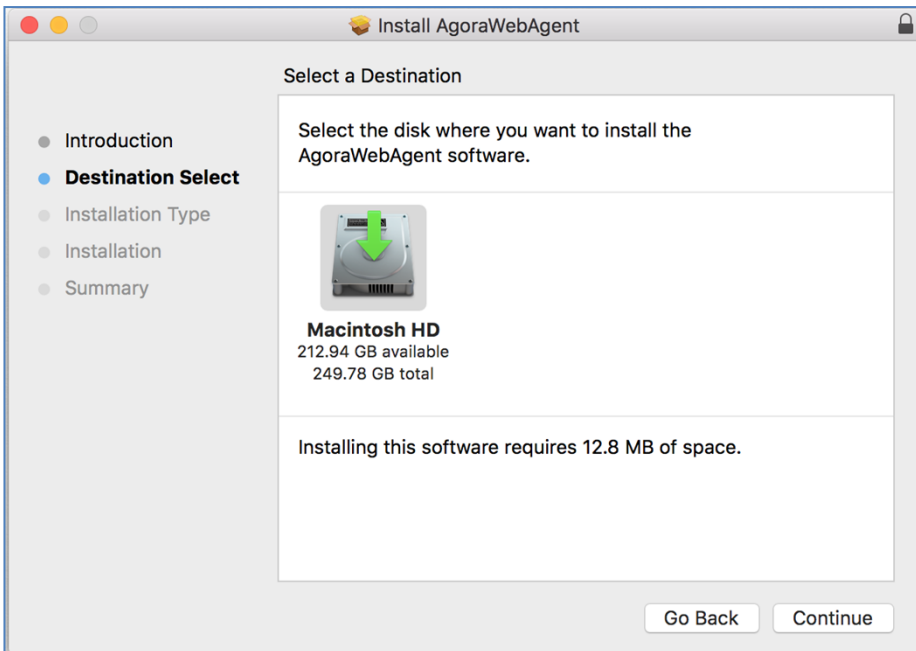
5. Select **Run AgoraWebAgent.exe** and then click **Finish**.

Note: Once the program is installed, a shortcut is displayed on the Desktop and AgoraWebAgent starts up automatically whenever starting the computer. An icon will display at the lower right part of the toolbar if AgoraWebAgent is running. Users can disable/enable the automatic start-up in the task manager. If the plugin program is closed while using the Agora Video Call on the browser, restart it by clicking the shortcut on desktop. Once it is restarted, it starts up automatically again whenever starting the computer.



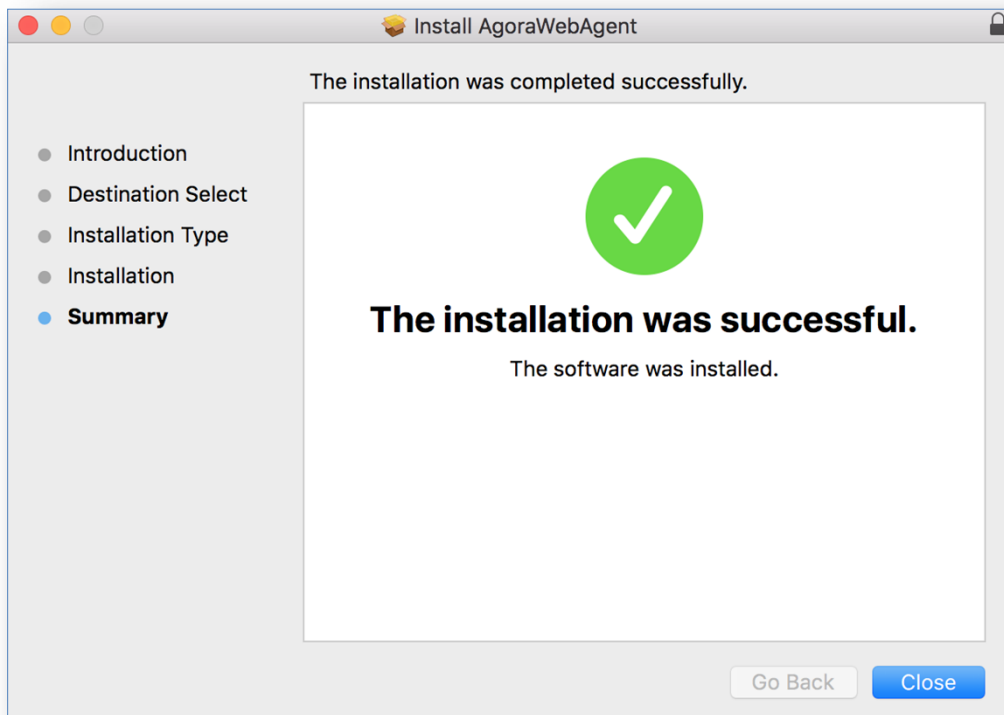
On Mac

1. Double-click the AgoraWebAgent.pkg downloaded from the system prompt.
2. Click **Continue** on the installer.



3. Click **Continue**.
4. Click **Install** to install the application on the default location.
5. Enter your computer username and password. Proceed with **Install Software**.

6. When the following message displays, the installation is complete. Click **Close**.



Note: Once AgoraWebAgent is installed, it is in the Applications directory by default, and it starts up automatically whenever starting the computer. An icon will display at the upper right part of the toolbar if AgoraWebAgent is running. You can disable/enable the automatic start-up by right-clicking the icon. If the program is closed while using the Agora Video Call on browser, restart it by clicking the icon in '/Applications' . Once it is restarted, it starts up automatically again whenever starting the computer.

Uninstalling AgoraWebAgent on Windows or Mac

On Windows

The uninstallation procedure is the same as uninstalling any other application on Windows XP, Windows 7, Windows 8, and so on.

Take Windows 8 for example:

1. Go to **Desktop> Control Panel**.
2. Click Uninstalling the Applications under Applications.
3. Find AgoraWebAgent.exe and right click it to select uninstall.
4. Wait until the progress is finished.

On Mac

1. Go to **Finder** on the Docker.
2. Click and enter the folder **Applications**.

3. Find the installed AgoraWebAgent and right click it to select **Move to Trash**.

Agora Native SDK for Web API Reference – JavaScript Classes

The Agora Web SDK library includes the following classes:

AgoraRTC	Use the AgoraRTC object to create Client(s) and Stream objects.
Client	Represents the web client object that provides access to core AgoraRTC functionality.
Stream	Represents the local/remote media streams in a session.

AgoraRTC API Methods

Create Client Object (createRtcClient)

createRtcClient()

This method creates and returns a Client object. It should be called only once per session.

Sample code:

```
var client = AgoraRTC.createClient();
```

Create Stream Object (createStream)

createStream(spec)

This method creates and returns a Stream object.

Parameter	Type	Description
spec	Object	<p>This object contains the following properties:</p> <ul style="list-style-type: none">● <u>streamID</u>: represents the stream ID, normally set to uid which can be retrieved from the client.join callback.● <u>local</u>: represents whether the stream is local or remote, if local, set it as true. The remote stream is created inside of the SDK, there is no need to set it as false.● <u>video</u>: represents whether the user opens the camera or not. If true, then video function is enabled, which turns on the camera. If false, the video function is disabled, which does not turn on the camera.

Sample code:

```
var stream = AgoraRTC.createStream({streamID: uid, local:true, video:true});
```

Client API Methods and Callback Events

Represents the web client object that provides access to core AgoraRTC functionality.

Methods

Initialize Client Object (init)

init(appld, onSuccess, onFailure)

This method initializes the Client object.

Parameter	Type	Description
appld	String	The App ID, a static key, is provided by Agora during registration. For users who do not have a high security requirement in most developer circumstances, the App ID should suffice.
onSuccess	function	(optional) the function to be called when the method succeeds.
onFailure	function	<p>(optional) the function to be called when an error occurs in the lifetime of a client object.</p> <p>When the onFailure callback reports CLOSE_BEFORE_OPEN, it means the application cannot connect to AgoraWebAgent. The following lists the possible reasons:</p> <ol style="list-style-type: none">1. No available network. The application requires DNS service to connect to AgoraWebAgent due to the adoption of a secure web socket local connection. The application cannot connect to AgoraWebAgent when the DNS service is unavailable.2. AgoraWebAgent is not launched.3. AgoraWebAgent is not installed.4. The application cannot connect to AgoraWebAgent because proxy is used. <p>When the onFailure callback reports INCOMPATIBLE_WEBAGENT, it may be caused by the v1.6 JS is incompatible with the previous AgoraWebAgent, then you need to upgrade the AgoraWebAgent according to the url returned in this callback.</p>

Sample code:

```

client.init(applId, function() {
  log("client initialized");
  //join channel
  .....
}, function(err) {
  log("client init failed ", err);
  //error handling
});

```

Join AgoraRTC Channel(join)

join(channelKey,channelName, uid, onSuccess, onFailure)

This method allows the user to join an AgoraRTC channel.

Parameter	Type	Description
channelKey	string	For users who have a high security requirement, use the Channel Key, otherwise use an App ID. You can switch the use of App ID and Channel Key on Dashboard . The Channel Key is generated according to security algorithms provided by Agora.io with an App ID and a App Certificate fetched from Dashboard .
channelName	string	A string providing the unique channel name for the AgoraRTC session. The length must be within 64 bytes. The following is the supported scope: a-z A-Z 0-9 space !#\$%& ()+, - :;<=. >? @[] ^ _` { } ~
uid	integer	The user ID: a 32-bit unsigned integer ranges from 1 to (2 ³² -1). It must be unique. If set to 'undefined (the value is set to 0)', the server assigns one and returns it in onSuccess callback.
onSuccess	function	(optional) The function to be called when the method succeeds, returns the uid, which represents the identity of the user.
onFailure	function	(optional) the function to be called when the method fails.

Sample code:

```
client.join(key,'1024', 0, function(uid) {
    log("client " + uid + " joined channel");
    //create local stream
    .....
}, function(err) {
    log("client join failed ", err);
    //error handling
});
```

Renew Channel Key (renewChannelKey)

renewChannelKey(channelKey,onSuccess,onFailure)

This method allows the users to renew the Channel Key. The descriptions of the parameters are the same as above.

When the onFailure callback after calling client.init returns

ERR_CHANNEL_KEY_TIMEOUT = 109 or

ERR_INVALID_CHANNEL_KEY = 110, your application must call this method to renew the Channel Key.

Parameter	Type	Description
channelkey	string	Channel Key. See Obtaining and Using a Channel Key for details.
onSuccess	function	(optional) The function to be called when the method succeeds returns the uid which represents the identity of the user.
onFailure	function	(optional) the function to be called when the method fails.

Leave AgoraRTC Channel(leave)

leave(onSuccess, onFailure)

This method allows the user to leave an AgoraRTC channel.

Parameter	Type	Description
onSuccess	function	(optional) The function to be called when the method succeeds.
onFailure	function	(optional) The function to be called when the method fails.

Sample code:

```
client.leave(function() {
    log("client leaves channel");
    .....
});
```



```

    }, function(err) {
      log("client leave failed ", err);
      //error handling
    });

```

Publish Local Stream (publish)

publish(stream, onSuccess, onFailure)

This method publishes a local stream to the server.

Parameter	Type	Description
stream	object	Stream object, which represents the local stream.
onSuccess	function	The function to be called when publish local stream successfully
onFailure	function	(optional) The function to be called when the method fails.

Sample code:

```

client.publish(stream, function(err) {
  console.log ("stream published");
  .....
}, function (err) {
  console.log ("failed to publish stream");
});

```

Unpublish Local Stream (unpublish)

unpublish(stream, onSuccess, onFailure)

This method unpublishes the local stream.

Parameter	Type	Description
stream	object	Stream object representing local stream.
onSuccess	function	(optional) the function to be called when the method succeeds.
onFailure	function	(optional) the function to be called when the method fails.

Sample code:

```

client.unpublish(stream, function(err) {
  console.log("stream unpublished5.79");
}, function (err) {
  console.log("failed to unpublish stream");
});

```

Subscribe Remote Stream (subscribe)

subscribe(stream, onFailure)

This method subscribes remote stream from the server.

Parameter	Type	Description
stream	object	Stream object representing the remote stream.
onFailure	function	(optional) Function to be called when the method fails.

Sample code:

```
client.subscribe(stream, function(err) {  
  log("stream unpublished");  
  .....  
})
```

Unsubscribe Remote Stream (unsubscribe)

unsubscribe(stream, onFailure)

This method unsubscribes the remote stream.

Parameter	Type	Description
stream	object	Stream object representing remote stream.
onFailure	function	(optional) The function to be called when the method fails.

Sample code:

```
client.unsubscribe(stream, function(err) {  
  log("stream unpublished");  
  .....  
})
```

Enumerate Platform Device (getDevices)

getDevices(callback)

This method enumerates platform camera/microphone devices. It is not applicable to Mac yet.

Parameter	Type	Description
callback	Function	The callback function to retrieve the devices information. For ex. callback(devices): devices[0].deviceId: device ID devices[0].label: device name in string devices[0].kind: 'audioinput', 'videoinput'

Sample code:

```
client.getDevices (function(devices) {  
  var dev_count = devices.length;  
  var id = devices[0].deviceId;  
});
```

Enable Built-in Encryption(setEncryptionSecret)

function setEncryptionSecret(secret)

Ensure that your application calls setEncryptionSecret to specify a secret and enable the 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 once the user has left the channel. If the secret is not specified or set to empty, the encryption function is disabled.

Name	Description
secret	Encryption password
Return Value	0 : Method call succeeded. <0 : Method call failed.

Sample Code:

```
client.setEncryptionSecret("1234");
```

Set Built-in Encryption Mode(setEncryptionMode)




function setEncryptionMode (encryptionMode)

Agora Native SDK supports built-in encryption, and it is in AES-128-XTS mode by default.

If you want to use another mode, call this API to set the encryption mode.

All users in the same channel must use the same encryption mode and secret. You can refer to AES encryption algorithm related information on the differences of the several encryption modes.

Note: Call setEncryptionSecret to enable the built-in encryption function before calling this API.

Name	Description
encryptionMode	Encryption mode. The following modes are supported currently: <ul style="list-style-type: none">  "aes-128-xts": 128-bit AES encryption, XTS mode  "aes-256-xts": 256-bit AES encryption, XTS mode  "": When it is set to NUL string, the encryption is in "aes-128-xts" by default.
Return Value	0 : Method call succeeded. <0 : Method call failed.

Select Device (selectdevice)

selectDevice(device)

Select audio/video device to use in current session.

Parameter	Type	Description
device	Object	Device selected from device list returned by getDevices. Agora uses the selected audio/video device in the current session.

Sample code:

```
client.selectDevice(device);
```

Start Recording(startRecording)

startRecording(key, onSuccess, onFailure)

This method requests the Agora Recording Server (ARS) to record all the audios and videos.

Parameter	Type	Description
key	String	Recording Key for Recording. The app backend must be integrated with the Channel Key for recording algorithms to authorize user access to ARS.
onSuccess	Function	Callback when starting successfully.
onFailure	Function	Callback when failed to start.

Sample code:

```
client.startRecording(key, function(data) {  
    console.log("start recording successfully.");  
}, function(err) {  
    console.log("failed to start recording" + err);  
});
```

Stop Recording (stopRecording)

stopRecording(key, onSuccess, onFailure)

This method requests the ARS to stop recording all the audios and videos. stopRecording and startRecording use the same recording key, and the SDK will store the key.

Parameter	Type	Description
key	String	Recording Key used for Recording. App backend must integrated Channel Key for Recording algorithms to authorize user the access to ARS.
onSuccess	Function	Callback when stopping successfully.
onFailure	function	Callback when failed to stop.

Sample code:

```
client.stopRecording(key, function(data) {  
    console.log("stop recording successfully.");  
}, function(err) {  
    console.log("failed to stop recording" + err);  
});
```

Query Recording Status(queryRecordingStatus)

queryRecordingStatus (onStatus)

This method sends a request to ARS to query the recording status.

Parameter	Type	Description
onStatus	Function	Callback with recording status.

Sample code:

```
client.queryRecordingStatus(function(result) {  
  console.log(result);  
  switch(result.status) {  
    case 0:  
      // TODO:...  
      break;  
    default:  
      break;  
  }  
  
});
```

Get Desktop Window List(getWindows)

This method allows the users to obtain the desktop window of the operating system. The application can obtain the window list and specify the window to be shared using this method.

Parameter	Type	Description
callback	function	Use callback function to obtain the window list, for example, callback(windows): windows[0].windowId: window id in string windows[0].title: window title in string

Start Screen Sharing(startScreenSharing)

startScreenSharing (window, onSuccess, onFailure)

This method allows the users to share the screens. The shared window is specified by the Window Handle **window**. Once the screen sharing function is enabled, the screen of the other user will display your shared desktop or window, instead of the camera video.

Parameter	Type	Description
window	string	Window ID. When window="", the entire screen is shared.
onSuccess	function	The function to be called when the method succeeds.

onFailure	function	The function to be called when the method fails.
-----------	----------	--

Set Shared Window(setScreenSharingWindow)

setScreenSharingWindow (window, onSuccess, onFailure)

This method allows users to switch the shared screen or specified window after enabling the screen sharing function. The shared window is specified by the Window Handle window.

Parameter	Type	Description
window	string	Window ID. When window="", the entire screen is shared.
onSuccess	function	The function to be called when the method succeeds.
onFailure	function	The function to be called when the method fails.

Stop Screen Sharing(stopScreenSharing)

stopScreenSharing (onSuccess, onFailure)

This method stops the screen sharing.

Parameter	Type	Description
onSuccess	function	The function to be called when the method succeeds.
onFailure	function	The function to be called when the method fails.

Release Resources(close)

close();

This method releases all the resources in the client. Once called, the users cannot call any other method including init. If the users need to use the Client object again, they must create the object and initialize it again.

Sample code:

```
var client = AgoraRTC.createRtcClient();
client.init(appID, onInitClientSuccess, onInitClientFailure);
client.close();
client = undefined;

// create a new client
client = AgoraRTC.createRtcClient();
client.init(appID, onInitClientSuccess, onInitClientFailure);
```

Remote Stream Added Event (stream-added)

Notifies the application that the remote stream has been added.

Sample code:

```
client.on('stream-added', function(evt) {  
  var stream = evt.stream;  
  log("new stream added ", stream.getId());  
  //subscribe the stream  
  .....  
})
```

Remote Stream Subscribed Event (stream-subscribed)

Notifies the application that the remote stream has been subscribed, which means you can play the remote videos now.

Sample code:

```
client.on('stream-subscribed', function(evt) {  
  var stream = evt.stream;  
  log("new stream subscribed ", stream.getId());  
  //play the stream  
  .....  
})
```

Peer User Left Channel Event (peer-leave)

Notifies the application that the other user has left the room, (called client.leave()), which means that you can stop playing the remote video now.

Sample code:

```
client.on('peer-leave', function(evt) {  
  var uid = evt.uid;  
  log("remote user left ", uid);  
  .....  
})
```

Remote User Muted Video Event(peer-mute-video)

Notifies the application that the remote user has chosen not to send or to resend the local video stream.

Parameter	Description
muted	When muted is true , indicating the remote user has chosen not to send the local video stream, which means the other users cannot see his/her image.
	When muted is false , indicating the remote user has chosen

	to resend the local video stream, which means the other users cannot see his/her image.
--	---

Sample code:

```
client.on("peer-mute-video", function (event) {
  var message = event.msg;
  if (message.muted) {
    console.log("remote user " + message.uid + " muted video");
  } else {
    console.log("remote user " + message.uid + " unmuted video");
  }
}
```

Remote User Muted Audio Event(peer-mute-audio)

Notifies the application that the remote user has chosen not to send or to resend the local audio stream.

Parameter	Description
muted	<p>When muted is true, indicating the remote user has chosen not to send the local audio stream, which means the other users cannot hear his/her voice.</p> <p>When muted is false, indicating the remote user has chosen to resend the local audio stream, which means the other users cannot hear his/her voice.</p>

Sample code:

```
client.on("peer-mute-audio", function (event) {
  var message = event.msg;
  if (message.muted) {
    console.log("remote user " + message.uid + " muted audio");
  } else {
    console.log("remote user " + message.uid + " unmuted audio");
  }
}
```

Connection Lost/Interrupted Event (error)

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

For more description on error codes, see [Error and Warning Messages](#).

Sample code:

```
client.on("error", function (event) {
  var message = event.msg;
  if (message.reason === "CONNECTION_INTERRUPTED") {
    console.log("connection interrupted");
  } else if (message.reason === "CONNECTION_LOST") {
    console.log("connection lost");
  }
}
```

Stream API Methods

Initialize Local Stream Object(init)

init(onSuccess, onFailure)

This method initializes the local Stream object. No need to initialize the remote stream, because the SDK handles it internally.

Parameter	Type	Description
onSuccess	function	(optional) The function to be called when the method succeeds.
onFailure	function	(optional) The function to be called when an error occurs in the lifetime of client

Sample code:

```
stream.init(function() {
  log("local stream initialized");
  // publish the stream
  .....
}, function(err) {
  log("local stream init failed ", err);
  //error handling
});
```

Close Video Stream (close)

close()

This method closes the video stream, and users won't see this video.

Play Video or Audio Stream(play)

play(elementID, onFailure)

This method plays the video or audio stream.

Parameter	Type	Description
elementID	String	html element ID。
onFailure	Function	(optional)

Sample Code:

```
stream.play('div_id', function (err) { // stream will be played in div_id element
  console.log('failed to play stream');
});
```

Stop Playing Video Stream(stop)

stop()

This method stops playing the video steam on the web.

Enable Audio Stream (enableAudio)

enableAudio(callback)

This method enables the audio stream, which triggers a callback after a successful execution.

Disable Audio Stream (disableAudio)

disableAudio(callback)

This method disables the audio stream, which triggers a callback after a successful execution.

Enable Video Stream(enableVideo)

enableVideo(callback)

This method enables the video stream, which triggers a callback after a successful execution. If you receive the video steam from the user continuously, the applicable will receive the stream-added event, and the application can re-play the video.

Disable Video Stream (disableVideo)

disableVideo(callback)

This method disables the video stream, which triggers a callback after a successful execution.

Retrieve User ID (getId)

getId()

This method retrieves the stream id (the corresponding user ID).

Set Video Profile (setVideoProfile)

setVideoProfile(profile)

This method sets the video profile. Each profile corresponds to a set of video parameters, for example, resolution, frame rate, bitrate etc. When the camera of the device does not support the specified resolution, SDK selects one suitable camera resolution automatically, but the encoder resolution still uses the one specified by setVideoProfile.

The method only sets the attributes of the streams encoded by the encoder which may be inconsistent with the final display. For example, when the resolution of the encoded stream is 640x480, and the rotation attribute of the stream is 90 degrees, then the resolution of the final display is in portrait mode.

The parameter profile is a json object, which includes the following attributes:

Name Enumeration Value	
profile	The video attribute. See the following Video Profile Definition for details.
swapWidthAndHeight	Whether to swap the width and height of the stream: true: swap false: not swap(default)

Note: Set the video profile before calling AgoraRTC.client.join to join the channel.

Video Profile Definition

The following table lists the definitions of the profile.

Video Profile (String)	Enumeration Value	Resolution (width x 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	720x1280	15	1000
"720P_3"	52	1280x720	30	2000
"720P_4"	53	720x1280	30	2000

Sample code:

```
stream.setVideoProfile('480p');
```

Error and Warning Messages

Agora SDK will return some error codes or warning codes when calling APIs or running:

- **Error Messages** means that the SDK encountered an error that can be not recovered automatically if without the application intervention. For example, the SDK returns an error if it fails to open a camera, then the application needs to remind of the user not to use the camera.
- **Warning Messages** means that the SDK encountered an error that might be recovered automatically. A Warning Code is just for notification, which can usually be ignored.

Error Messages

Error Code	Value	Description
ERR_OK	0	No error.
ERR_FAILED	1	General error (the reason is not classified specifically).
ERR_INVALID_ARGUMENT	2	Invalid parameter called. For example, the specific channel name includes illegal characters.
ERR_NOT_READY	3	SDK module is not ready. For example, if some API relies on a specific module, but the module is not ready to provide service yet.

ERR_NOT_SUPPORTED	4	SDK does not support this function.
ERR_REFUSED	5	Request is rejected. It is only for SDK internal use, which means it won't 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 is not initialized before calling this API.
ERR_INVALID_VIEW	8	The specified view is invalid. It is required to specify a view when using the video call function, otherwise, it returns this error.
ERR_NO_PERMISSION	9	No permission. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_TIMEDOUT	10	API call timed-out. Some API requires 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. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_TOO_OFTEN	12	Call frequency is too high. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_BIND_SOCKET	13	SDK failed to bind to the network socket. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_NET_DOWN	14	Network is unavailable. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_NET_NOBUFS	15	No network buffers available. It is only for SDK internal use, which means it won't 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	The request to join the channel is rejected. This error usually occurs when the user is already in the channel, and still calls the API to join the channel, for

		example, joinChannel.
ERR_LEAVE_CHANNEL_REJECTED	18	The request to leave the channel is rejected. This error usually occurs when the user has already left the channel, and still 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 the request probably due to too many Request. It is only for SDK internal use, which means it won't return to the application through any API or callback event.
ERR_INVALID_APP_ID	101	The specified App ID is invalid.
ERR_INVALID_CHANNEL_NAME	102	The specified Channel Name is invalid.
ERR_CHANNEL_KEY_TIMEOUT	109	The current Channel Key is expired, no longer valid.
ERR_INVALID_CHANNEL_KEY	110	The specified Channel Key is invalid. Usually the Channel Key was generated incorrectly.
ERR_CONNECTION_INTERRUPTED	111	CONNECTION_INTERRUPTED callback. It is only applicable to Agora Native SDK for Web.
ERR_CONNECTION_LOST = 112	112	CONNECTION_LOST callback. It is only applicable to Agora Native SDK for Web.
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	General error on Audio Device Module (the reason is not classified specifically).
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: The camera is not authorized

Warning Messages

Warning Code	Value	Description
WARN_PENDING	20	The request is pending. Usually it is due to some module is not ready, and the SDK postponed processing the request.
WARN_NO_AVAILABLE_CHANNEL	103	No channel resources are available. Maybe because the server cannot allocate channel resources.
WARN_LOOKUP_CHANNEL_TIMEOUT	104	Timed-out when looking up the channel. When joining a channel, the SDK looks up the specified channel. The warning usually occurs when the network condition is too bad to connect to the server.
WARN_LOOKUP_CHANNEL_REJECTED	105	The server rejected the request to look up the channel. The server cannot process this request or 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 the network condition is too bad 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 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 is detected

Agora CaaS, Agora Global Network, Agora Native SDK and Agora Web SDK are trademarks of Agora.io. Agora Lab does business as Agora.io. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

© 2016 Agora.io. All rights reserved.
