



# **Agora Web SDK Reference Manual v1.2**

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

May 2016

## Content

<b>Introduction .....</b>	<b>4</b>
<b>Agora CaaS.....</b>	<b>4</b>
<b>Agora Web SDK .....</b>	<b>4</b>
<b>Requirements.....</b>	<b>5</b>
Compatibility.....	5
Supported Browsers.....	5
<b>Known Issues and Limitations .....</b>	<b>6</b>
<b>Getting Started .....</b>	<b>7</b>
<b>Obtaining the SDK .....</b>	<b>7</b>
<b>Obtaining a Vendor Key .....</b>	<b>7</b>
<b>Developing and Deploying Application .....</b>	<b>8</b>
<b>Running the Sample Web Application .....</b>	<b>9</b>
Client .....	9
Server.....	9
<b>Using Dynamic Keys for Increased Security.....</b>	<b>10</b>
What is Dynamic Key.....	10
How to Use Dynamic Key .....	10
Increased Security with Dynamic Keys.....	13
<b>Agora Web SDK API Reference – JavaScript Classes .....</b>	<b>15</b>
<b>AgoraRTC API Methods.....</b>	<b>15</b>
Create Client Object for Real-Time Communication (createRtcClient) .....	15
Create Client Object for Live Broadcasting (createLiveClient) .....	15
Create Client Object for Cross Platform Compatibility (createClient).....	15
Enumerate Platform Devices (getDevices).....	16
Create Stream Object (createStream).....	16
<b>Client API Methods and Callback Events.....</b>	<b>17</b>
Initialize Client Object (init).....	17
Join AgoraRTC Channel (join) .....	17
Leave AgoraRTC Channel (leave).....	18
Publish Local Stream (publish) .....	18
Unpublish Local Stream (unpublish) .....	19
Subscribe Remote Stream(subscribe) .....	19
Unsubscribe Remote Stream (unsubscribe).....	20
Local Stream Published Event (stream-published) .....	20
Remote Stream Added Event (stream-added) .....	20
Remote Stream Removed Event (stream-removed) .....	21
Remote Stream Subscribed Event (stream-subscribed).....	21
Peer User Left Channel Event (peer-leave) .....	21
Error Code .....	21
<b>Stream API Methods .....</b>	<b>23</b>
Initialize Stream Object (init) .....	23

Retrieve Stream ID (getId).....	23
Retrieve Stream Attributes.....	23
Retrieve Video Flag (hasVideo) .....	23
Retrieve Audio Flag (has Audio) .....	23
Enable Video (enableVideo).....	23
Disable Video (disableVideo) .....	24
Enable Audio(enableAudio) .....	24
Disable Audio (disableAudio) .....	24
Set Video Profile (setVideoProfile).....	24
Play Video/Audio Stream (play) .....	24
Stop Video/Audio Stream(stop) .....	25
Close Video/Audio Stream (close).....	25

## Introduction

---

### Agora CaaS

Agora Communications as a Service (CaaS) provides ensured Quality of Experience for worldwide Internet-based voice and video communications through a virtual Agora Global Network that is especially optimized for real-time and mobile-to-mobile communications. Agora CaaS solves quality of experience challenges for mobile devices, 3G/4G/Wi-Fi networks with varying performance, and global Internet bottlenecks.

Agora CaaS includes mobile-optimized Agora Native SDKs for iOS and Android smartphones that provide access to the Agora Global Network along with device-specific mobile optimizations. Agora Native SDK is also available for Windows and Mac. Applications using the Native SDK link it directly into the application when they are built.

The **Agora Web SDK** is an addition to the Agora CaaS capability. The SDK provides open access to the Agora Global Network from any device that supports a standard WebRTC-compliant web browser, without requiring any downloads or plugins. See [Getting Started](#) for details on how to deploy and use the Agora Web SDK.

### Agora Web SDK

The **Agora Web SDK** is a JavaScript library loaded by an HTML web page, as with any other JavaScript library. It also provides a set of simple high-level JavaScript APIs for establishing voice and video communications with other users across the Agora Global Network. The Agora Web SDK library uses the primitive WebRTC APIs in the browser to establish connections and control the voice and video media, while providing a simpler high-level interface for developers.

The Agora Web SDK allows JavaScript code to:

- Join and leave shared Agora sessions (identified by unique channel names) where there may be many global users on a conference together.
- Set various voice and video parameters that help the Agora SDK optimize communications. Most of the SDK operations are automated and do not require any developer intervention if these parameter settings are not provided.
- Manipulate voice and video media streams, controlling whether they are muted or seen and where within the page's HTML framework they will be seen or heard.
- Support multiple voice and video streams simultaneously which will occur in multi-party conference applications.

The Web SDK 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 Web SDK API Reference – JavaScript Classes](#).

## Requirements

### Compatibility

This v1.2 release of the Agora Web SDK is compatible with the use of the Agora Native SDK v1.2 and v1.3, which supports compatible voice and video capabilities. The Native SDK is used to build native applications for Android, iOS, Windows and Mac, while the Web SDK described here is used for browser-apps running within WebRTC-compatible browsers.

For how to develop both native and web browser applications, refer to:

- Agora Native SDK for iOS Reference Manual - v1.2 or later
- Agora Native SDK for Android Reference Manual - v1.2 or later
- Agora Native SDK for Windows Reference Manual - v1.2 or later

This documentation, as well as complete SDK packages for all platforms, can be downloaded at [www.agora.io/developer](http://www.agora.io/developer).

### Supported Browsers

The Agora Web SDK requires a WebRTC-compliant web browser. The following browser types and versions are tested and verified which are available across a wide range of platforms except for Apple iOS.

Recent browser releases, starting from Chrome v47 and Opera v34, have increased security by disabling the use of WebRTC from HTTP domains. It is recommended that developers follow Google's lead by moving to HTTPS domains for all WebRTC and Agora Web SDK applications.

	Chrome v42-46	Chrome v47	Firefox v42	Opera v34	QQ v9.0	360 v8.5	Baidu v7.6	Sougou v6.0
<b>HTTP</b>	Y	N	Y	N	Y	Y	Y	Y
<b>HTTPS</b>	Y	Y	Y	Y	Y	Y	Y	Y

**Note:** All the browser versions listed have been tested and verified. Earlier versions of these browsers that support WebRTC may work as well, but not verified by Agora.io.

## Known Issues and Limitations

1. The **Agora Web SDK** is based on WebRTC technology. The following browsers and platforms currently do not support WebRTC:

- **Browser:** Microsoft Internet Explorer and Apple Safari.
- **Platform:** Apple iOS.

See **Section Supported Browsers** for more details.

2. The **Agora Web SDK** supports video profiles up to 1080p resolution if the client has a true HD camera installed. However, the maximum resolution is limited by the camera device capabilities.
3. The maximum video bitrate setting is currently not supported in Mozilla Firefox.
4. The **Agora Web SDK** currently does not provide the following functionalities available in the Agora Native SDK: quality indicators, testing services, providing feedback ratings for sessions, recording and logging.

## Getting Started

---

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

### Obtaining the SDK

Contact the [sales@agora.io](mailto:sales@agora.io) to obtain the latest SDK.

Component	Description
./doc	Agora Web SDK Documentation:  Agora_Web_SDK_Release_Notes_v1_2_EN.pdf  Agora_Web_SDK_Reference_Manual_v1_2_EN.pdf
./client	sample web application
./server	Web server-side sample code and libraries for dynamic key generation.

### Obtaining a Vendor Key

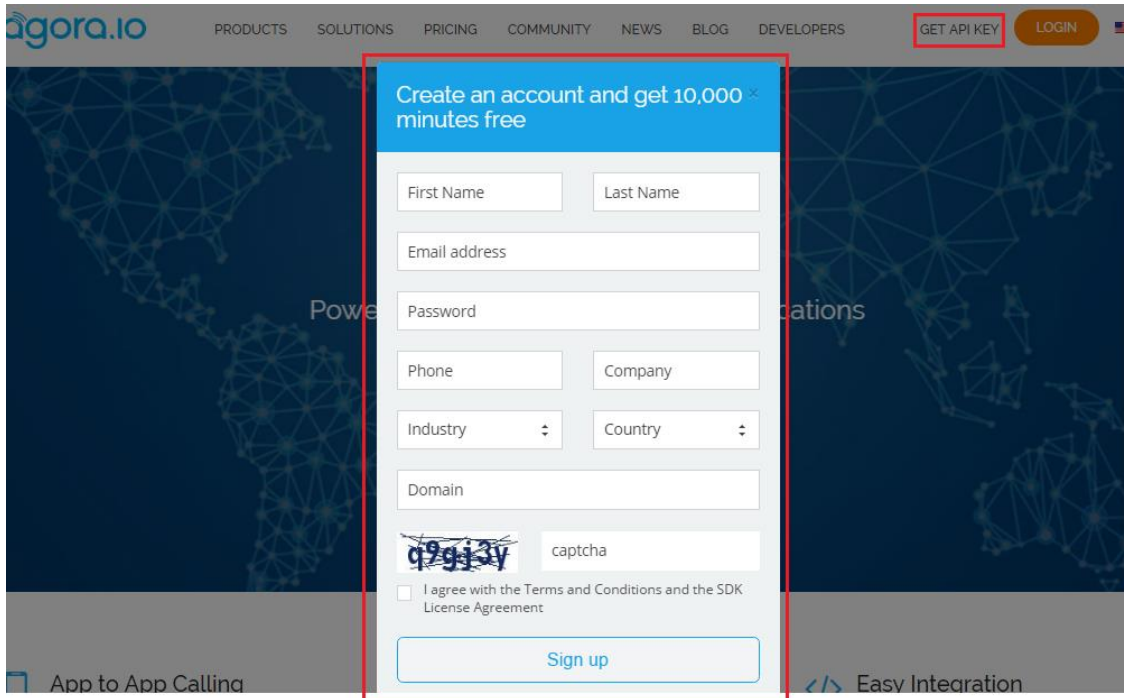
Developers must obtain a **vendor key**. The key is required when you use APIs to access the Agora Global Network.

This section only describes how to use the Vendor Key directly, but you can also combine it with a sign key within your web server code to generate a more secure dynamic key:

- For why and how to use dynamic keys (**recommended**), see [Using Dynamic Keys for Increased Security](#).
- For a sample of how to run the sample applicable for both Vendor Key and Dynamic Key, see [Running the Sample Web Application](#).

#### To obtain your vendor key:

1. Sign up for a new account at [agora.io](https://agora.io).



2. Complete the sign-up process.

Once the new account is created, you will find your API vendor key on the completion page at <http://dashboard.agora.io/>. You will also receive an email with your API vendor key.



#### API CREDENTIALS

To use Agora API you will need your API Keys

Vendor Key: 70ca35de60c44645bbae84215061b33b

3. Once the Vendor Key is assigned, you can start using Agora.io services with this key.

## Developing and Deploying Application

Applications using the **Agora Web SDK** 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.

1. Load the Agora JS library AgoraRTCSDK-1.2.5.js, you can download it from

<https://rtc.sdk.agora.io/AgoraRTCSDK-1.2.5.js> or  
<http://rtc.sdk.agora.io/AgoraRTCSDK-1.2.5.js12>.



2. Required JS extension libraries (can be found under the **/client/js/** folder in the release package):

jquery.js – version  $\geq$  v2.0.0

socket.io.js – version  $\geq$  1.3.6

adapter.js – version  $\geq$  0.2.5

## Running the Sample Web Application

### Client

The default sample code only requires your static **vendor key** (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, and then launch your http/https service.
3. Access the sample application page on your web server using one of the browsers listed in [Supported Browsers](#). Google Chrome is recommended.

Do the following if you want to run the sample application with a **dynamic key**:

**Note:** For more information, see [Running the Sample Web Application](#).

1. Uncomment the code marked by 'for dynamic key' in index.html
2. Replace the 'ip:port' with your dynamic key-generation server URL (below)
3. Set up and launch the key-generation server as described below

### Server

This code is only needed if you want to experiment with using more secure **dynamic keys**. In production use, you (developers) 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 Dynamic Keys 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 `VENDOR_KEY` and `SIGN_KEY` in `../server/nodejs/DemoServer.js`.
4. Launch the server with `'node DemoServer.js'`.

## Using Dynamic Keys for Increased Security

Each “vendor” organization using the Agora SDK has a unique **Vendor Key** that identifies that organization. Communications in the Agora Global Network are separated by Vendor Keys. Users with different Vendor Keys are isolated and won’t interfere with each other, even when the names of the channels they join are the same.

However, if someone else illicitly obtained your static **Vendor Key** and then they can use it on their own Agora SDK client applications. If they find out the channel names of your organization, they can even interfere with your communication sessions. To increase the security of applications, it is recommended that you consider using **dynamic keys** in your large-scale production applications.

### What is Dynamic Key

Dynamic Key is a more secure user authentication schema for the Agora SDK. Whenever a user tries to enter a channel to access the Agora service, the back-end services use the Vendor Key and a Sign Key to generate a new dynamic key based on the HMAC encryption algorithm. The dynamic key is then passed to the client application. The client application calls the `AgoraRtc.client.join` API interface function and passes the encoded dynamic key to the Agora server for the user authentication.

#### Note: (Terminology)

- **Vendor Key:** The “vendor” ID used to identify your organization. If the Dynamic Key scheme is not implemented, the Vendor Key is then used to access the Agora service.
- **Sign Key:** The “sign” key used to generate your dynamic key (see [Table 1](#) for its usage). Sign Key should always be stored on the server and invisible to all clients.

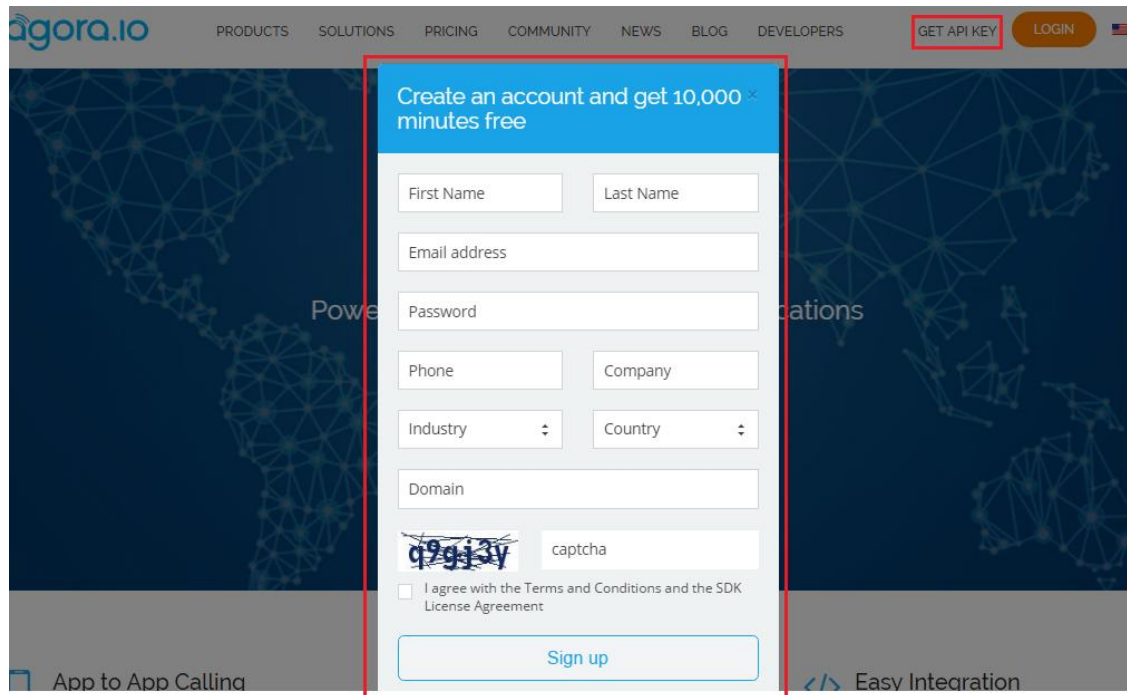
### How to Use Dynamic Key

The following sections describe the use of the dynamic keys:

- [Obtaining a Vendor Key](#)
- [Obtaining a Sign Key](#)
- [Implementing the Dynamic Key Scheme](#)
- [Using the Dynamic Key](#)

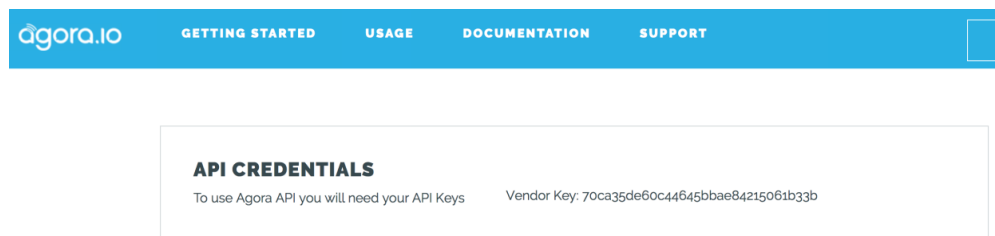
## Obtaining a Vendor Key

1. Sign up for a new account at [agora.io](http://agora.io).



2. Complete the sign-up process.

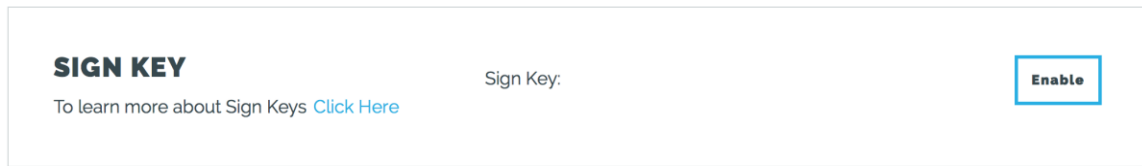
Once the new account is created, find your API vendor key on the completion page at <http://dashboard.agora.io/>. You also receive an email with your API vendor key.



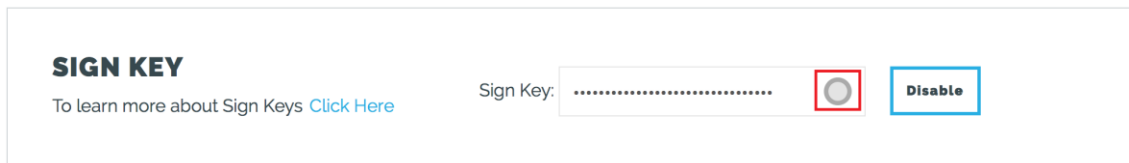
3. Once a **Vendor Key** is assigned, you can start using Agora.io services with this key.

## Obtaining a Sign Key

1. Generate the Sign Key by clicking **Enable** at [dashboard.agora.io](http://dashboard.agora.io).



2. Click the button highlighted in the following figure to display the Sign Key.  
Click the button again to hide the Sign Key.



**Caution:** Keep your Sign Key on the server and never on any client machine.

3. If you want to renew your Sign Key for some reason, contact [support@agora.io](mailto:support@agora.io).

## Implementing the Dynamic Key Scheme

Integrate the dynamic key scheme into your organization's signaling service. Agora.io provides sample server-side code covering the following languages: Java, C++, Python, node.js, and so on, which you can use this code directly in your application. For the sample code, refer to any of the following sites:

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

## Using the Dynamic Key

Every time before a user joins a channel (i.e., has started a call or received a meeting invitation) :

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 Dynamic Key **based on the following information** and then passes the dynamic key back down to the client application:

The dynamic key is based on the Sign Key, Vendor Key, 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 dynamic key as the first parameter.
4. The Agora server receives the dynamic key and confirms that the client application comes from your organization, and then allows access to Agora Global Network.

**Note:** With the Dynamic Key scheme implemented, Dynamic Key will replace the original Vendor Key when the user joins a channel.

## Increased Security with Dynamic Keys

If your organization chooses to use dynamic keys, every time before a user joins a channel, the client application must provide a new Dynamic Key. The signaling server verifies each user identity. The dynamic key uses the HMAC/SHA1 signature schema to increase security for communications within your organization.

### Dynamic Key Structure

Table 1 shows the structure of the dynamic key. Connect all fields in the sequence below (103 bytes in total):

Field	Type	Length	Description
Version	String	3	Dynamic 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 Sign Key and the following fields: <ul style="list-style-type: none"><li>✓ <b>Service Type:</b> The visible ASCII string provided by Agora.io. See <a href="#">Service Type</a>.</li><li>✓ <b>Vendor Key:</b> The 32-bit vendor key string.</li><li>✓ <b>Timestamp:</b> The timestamp created when the Dynamic Key is generated.</li><li>✓ <b>Random Number:</b> A 32-bit integer in hex code; generated upon each query.</li><li>✓ <b>Channel:</b> The channel name specified by the user with a maximum length of 64-byte.</li><li>✓ <b>User ID:</b> The User ID defined by the client.</li><li>✓ <b>Service Expiration Timestamp:</b> The timestamp indicates that from the specific</li></ul>

			moment the user cannot communicate in the channel any more.
Vendor Key	String	32	Vendor Key
Authorized Timestamp	Number	10	The timestamp represented by the number of seconds elapsed since 1-1-1970. It is authorized that the user can access the Agora service for 5 minutes.
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 no there is no time limit.

Table 1

#### 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 Dynamic Key is required, use this service type to generate the Dynamic Key.

#### Sign Encryption Algorithm: HMAC/SHA1

The dynamic 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, PHP, Python, Ruby and others. For more information, refer to:

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

## Agora Web SDK API Reference – JavaScript Classes

---

The Agora Web SDK library includes the following classes:

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

### AgoraRTC API Methods

#### Create Client Object for Real-Time Communication (createRtcClient)

##### **createRtcClient()**

This method creates and returns a Client object for real-time communication user scenarios that include video/voice chat and conferencing. It should be called only once per session.

##### **Note:**

It is recommended to use this method for small group chat and conferencing including one-to-one scenario for which this method is optimized.

Sample code:

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

#### Create Client Object for Live Broadcasting (createLiveClient)

##### **createLiveClient()**

This method creates and returns a Client object for live broadcasting (one-to-many) user scenario. It should be called only once per session.

Sample code:

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

#### Create Client Object for Cross Platform Compatibility (createClient)

##### **createClient()**

This method creates and returns a Client object for cross platform compatibility. For example, it realizes the interoperability between mobile and web users. This method is deprecated and will be removed in the later release. It should be called only once per session.

Sample code:

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

## Enumerate Platform Devices (getDevices)

### getDevices(callback)

This method enumerates platform camera/microphone devices.

Parameter Name	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:

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

## Create Stream Object (createStream)

### createStream()

This method creates and returns a Stream object.

Parameter Name	Type	Description
spec	Object	This object contains the following properties: <ul style="list-style-type: none"><li>● <u>streamID</u>: represents the stream ID, normally set to uid which can be retrieved from the <code>client.join</code> callback.</li><li>● <u>audio</u>: (flag) true/false, marks whether this stream contains an audio track.</li><li>● <u>video</u>: (flag) true/false, marks whether this stream contains a video track.</li><li>● <u>screen</u>: (flag) true/false, marks whether this stream contains a screen sharing track. It should be set to false in the current version.</li><li>● (optional) <u>cameraId</u>: the camera device ID retrieved from the <code>getDevices</code> method.</li><li>● (optional) <u>microphoneId</u>: the microphone device ID retrieved from the <code>getDevices</code> method.</li></ul>

Sample code:

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



## Client API Methods and Callback Events

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

### Methods

#### Initialize Client Object (init)

##### **init(key, onSuccess, onFailure)**

This method initializes the Client object.

Parameter Name	Type	Description
key	String	Key can be one of the following: <ul style="list-style-type: none"><li>- vendor key: provided by Agora during registration.</li><li>- dynamic key: the token generated with vendor key and sign key. A NodeJS implementation of token-gen algorithm is provided. This is the safest and recommended way to access the Agora Global Network.</li></ul>
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.init(vendorKey, function() {  
    log("client initialized");  
    //join channel  
    .....  
}, function(err) {  
    log("client init failed ", err);  
    //error handling  
});
```

#### Join AgoraRTC Channel (join)

##### **join(channel, uid, onSuccess, onFailure)**

This method allows the user to join an AgoraRTC channel.

Parameter Name	Type	Description
channel	String	A string providing the unique channel name for the AgoraRTC session.
uid	String	The user ID: a 32-bit unsigned integer ranges from 1 to (2^32-1). It must be unique. If set to 'undefined', the server will allocate one and

Parameter Name	Type	Description
		returns it in <code>onSuccess</code> callback.
<code>onSuccess</code>	function	(optional) The function to be called when the method succeeds, will return the uid which represents the identity of the user.
<code>onFailure</code>	function	(optional) the function to be called when the method fails.

Sample code:

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

## Leave AgoraRTC Channel (leave)

### `leave(onSuccess, onFailure)`

This method allows the user to leave an AgoraRTC channel.

Parameter Name	Type	Description
<code>onSuccess</code>	function	(optional) The function to be called when the method succeeds.
<code>onFailure</code>	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, onFailure)`

This method publishes a local stream to the server.

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

Sample code:

```
client.publish(stream, function(err) {
  log("stream published");
  .....
})
```

## Unpublish Local Stream (unpublish)

### **unpublish(stream, onFailure)**

This method unpublishes the local stream.

Parameter Name	Type	Description
stream	object	Stream object which represents local stream.
onFailure	function	(optional) the function to be called when the method fails.

Sample code:

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

## Subscribe Remote Stream(subscribe)

### **subscribe(stream, onFailure)**

This method subscribes remote stream from the server.

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

Sample code:

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

## Events

### Local Stream Published Event (stream-published)

Notify the application that the local stream has been published.

Sample code:

```
client.on('stream-published', function(evt) {  
  log("local stream published");  
  .....  
})
```

### Remote Stream Added Event (stream-added)

Notify 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 Removed Event (stream-removed)

Notifies the application that the remote stream has been removed, (i.e., a peer user called unpublish stream).

Sample code:

```
client.on('stream-removed', function(evt) {  
  var stream = evt.stream;  
  log("remote stream was removed", stream.getId());  
  .....  
})
```

## Remote Stream Subscribed Event (stream-subscribed)

Notifies the application that the remote stream has been subscribed.

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 peer user has left the room, (i.e., peer user called client.leave())

Sample code:

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

## Error Code

The onFailure callback function returns the following error codes:

Value	Description
10	Invalid user key.
11	Invalid user operation.
12	Invalid local stream.
13	Invalid remote stream.

Value	Description
100	Socket connection error.
101	Connect to web service failed.
102	Peer connection lost.
1000	Service not available.
1001	Join channel failed.
1002	Publish stream failed.
1003	Unpublish stream failed.
1004	Subscribe stream failed.
1005	Unsubscribe stream failed.

## Stream API Methods

### Initialize Stream Object (init)

#### **init(onSuccess, onFailure)**

This method initializes the Stream object.

Parameter Name	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:

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

### Retrieve Stream ID (getId)

#### **getId()**

Retrieves the stream id.

### Retrieve Stream Attributes

#### **getAttributes()**

This method retrieves the stream attributes.

### Retrieve Video Flag (hasVideo)

#### **hasVideo()**

This method retrieves video flag.

### Retrieve Audio Flag (has Audio)

#### **hasAudio()**

This method retrieves the audio flag.

### Enable Video (enableVideo)

#### **enableVideo()**

This method enables video track in the stream; it only works when video flag was set to true in stream.create and acts like video resume.

## Disable Video (disableVideo)

### disableVideo()

This method disables video track in the stream, only works when video flag was set to true in stream.create and acts like video pause.

## Enable Audio(enableAudio)

### enableAudio()

This method enables audio track in the stream and acts like audio resume.

## Disable Audio (disableAudio)

### disableAudio()

This method disables audio track in the stream and acts like audio pause.

## Set Video Profile (setVideoProfile)

### setVideoProfile(profile)

This method sets the video profile. It is optional and only works before calling stream.init().

Parameter Name	Type	Description
profile	String	The video profile. It can be set to one of the following: '120p_1': 160x120, 15fps, 80kbps '240p_1': 320x240, 15fps, 200kbps '480p_1': 640x480, 15fps, 500kbps (default) '480p_2': 640x480, 30fps, 1Mbps '720p_1': 1280x720, 15fps, 1Mbps '720p_2': 1280x720, 30fps, 2Mbps '1080p_1': 1920x1080, 15fps, 1.5Mbps '1080p_2': 1920x1080, 30fps, 3Mbps

Sample code:

```
stream.setVideoProfile('480p_1');
```

## Play Video/Audio Stream (play)

### play(elementID, assetsURL)

This method plays the video/audio stream.

Parameter Name	Type	Description
elementID	String	Represents the html element id.
assetsURL	String	(optional) The URL of the resource file. By default the files are



Parameter Name	Type	Description
		located under /assets/ folder.

Sample code:

```
stream.play('div_id', 'res'); // stream will be played in div_id element, resource files
under /res/assets/
```

## Stop Video/Audio Stream(stop)

### stop()

This method stops the video/audio stream and clears the display DOM tree under elementID (specified in the play method).

## Close Video/Audio Stream (close)

### close()

This method closes the video/audio stream. The camera and microphone authorization will be cancelled after calling this method.

---

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.