Agora Web SDK

API Reference – v1.0.0

support@agora.io

Contents

Introduction to Agora Web SDK

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 will be available for Mac in the future. Applications using the Native SDK link it directly into the application executable when they are built. These other SDKs are documented elsewhere in separate API References

The **Agora Web SDK** documented here is an additional Agora CaaS capability that 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.

Agora Web SDK is a JavaScript library that is loaded by an HTML web page, the same as for any other JavaScript library, and which then 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 conferenced together.
- Set various voice and video parameters which 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 straight-forward 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, and a top-level **AgoraRTC** object for creating the appropriate **Client** and **Stream** objects.

Agora Web SDK - JavaScript Classes

The Agora Web SDK library includes the following classes:

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

1. AgoraRTC API Methods

`createClient()`

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

Sample code:

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

`createStream(spec)`

This method creates and returns a Stream object.

Parameter Name	Type	Description
spec	Object	This object contains the following properties: streamID: represents the stream ID, normally set to uid which can be retrieved from the `client.join` callback. audio: (flag) true/false, marks whether this stream contains an audio track. video: (flag) true/false, marks whether this stream contains a video track. screen: (flag) true/false, marks whether this stream contains a screen sharing track. It should be set to false in the current version.

Sample code:

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

2. Client API Methods and Callback Events

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

Methods:

`init(key, onSuccess, onFailure)`

This method initializes the Client object.

Parameter Name	Type	Description
key	String	Key could be one of the following: - vendor key: provided by Agora during registration. - 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.
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(channel, uid, onSuccess, onFailure)`

This method lets the user join an AgoraRTC channel.

Parameter Name	Type	Description
channel	String	A string providing the unique channel name for the AgoraRTC session.
uid	String	Specifies the user ID. If not given here (set to 'undefined'), the server will generate one and return it to the onSuccess callback.
onSuccess	function	(optional) The function to be called when the method succeeds, will return the uid which represents the identity of the user.

Parameter Name	Type	Description
onFailure	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(onSuccess, onFailure)`

This method lets the user leave an AgoraRTC channel.

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:

```
client.leave(function() {
  log("client leaves channel");
  ……
}, function(err) {
  log("client leave failed ", err);
  //error handling
});
```

`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(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(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(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:

stream-published

Notify the application that the local stream has been published.

Sample code:

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

stream-added

Notify the application that the remote stream has been added.

Sample code:

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

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

`stream-removed`

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

`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-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);
  ……
})
```

3. Stream API Methods

`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
        });
```

`getId()`

This method retrieves the stream id.

`getAttributes()`

This method retrieves the stream attributes.

`hasVideo()`

This method retrieves video flag.

`hasAudio()`

This method retrieves the audio flag.

`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.

`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.

`enableAudio()`

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

`disableAudio()`

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

`setVideoProfile(profile)`

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

Parameter Name	Type	Description
profile	String	video profile, can be set to one of the following: '120p_1' : 160x120, 15fps, 80kbps '240p_1' : 320x240, 15fps, 200kbps '480p_1' : 640x480, 15fps, 500kbps '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 By default, video profile will be set to '480p_1'.

Sample code:

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

`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 resource file, by default the files are located under /assets/ folder.

Sample code:

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

`close()`

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