



**AlapKorea**

# **Web SDK API Reference**

**v1.0.0**

[support@alapkorea.com](mailto:support@alapkorea.com)

# Contents

---

<b>Contents</b>	<b>1</b>
<b>Introduction to AlapKorea Web SDK</b>	<b>2</b>
<b>AlapKorea Web SDK - JavaScript Classes</b>	<b>4</b>
1. AlapKoreaRTC API Methods	4
createClient()	4
createStream(spec)	4
2. Client API Methods and Callback Events	5
init(key, onSuccess, onFailure)	5
join(channel, uid, onSuccess, onFailure)	6
leave(onSuccess, onFailure)	6
publish(stream, onFailure)	7
unpublish(stream, onFailure)	7
subscribe(stream, onFailure)	8
unsubscribe(stream, onFailure)	8
Events	9
stream-published	9
stream-added	9
stream-removed	9
stream-subscribed	10
peer-leave	10
3. Stream API Methods	11
init(onSuccess, onFailure)	11
getId()	11
getAttributes()	11
hasVideo()	11
hasAudio()	11
enableVideo()	11
disableVideo()	12
enableAudio()	12
disableAudio()	12
setVideoProfile(profile)	12
play(elementID, assetsURL)	12
close()	13
4. User Authentication & Authorization	13
5. Session Management	13
6. Error Handling & Logging	14
7. Scaling & Load Management	14
8. Advanced Media Controls	14
9. Cross-Browser & Mobile Support	15
10. API Rate Limits & Security	15
11. Screen Sharing	15

## Introduction to AlapKorea Web SDK

---

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

AlapKorea CaaS includes mobile-optimized AlapKorea Native SDKs for iOS and Android smartphones that provide access to the AlapKorea Global Network along with device-specific mobile optimizations. AlapKorea 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 AlapKorea Web SDK documented here is an additional AlapKorea CaaS capability that provides open access to the AlapKorea Global Network from any device that supports a standard WebRTC-compliant web browser, without requiring any downloads or plugins.

AlapKorea 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 AlapKorea Global Network. The AlapKorea 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 AlapKorea Web SDK allows JavaScript code to:

- Join and leave shared AlapKorea sessions (identified by unique channel names) where there may be many global users conferenced together.
- Set various voice and video parameters which help the AlapKorea 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 [AlapKoreaRTC](#) object for creating the appropriate [Client](#) and [Stream](#) objects.

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

© 2025 AlapKorea.com. All rights reserved

## AlapKorea Web SDK - JavaScript Classes

---

The AlapKorea Web SDK library includes the following classes:

<a href="#">AlapKoreaRTC</a>	Use the AlapKoreaRTC object to create Client and Stream objects.
Client	Represents the web client object which provides access to core AlapKoreaRTC functionality.
Stream	Represents the local/remote media streams in a session.

### 1. AlapKoreaRTC API Methods

#### **createClient()**

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

Sample code:

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

#### **createStream(spec)**

This method creates and returns a Stream object.

Parameter Name	Type	Description
spec	Object	<p>This object contains the following properties:</p> <p><u>streamID</u>: represents the stream ID, normally set to uid which can be retrieved from the client.join callback.</p> <p><u>audio</u>: (flag) true/false, marks whether this stream contains an audio track.</p> <p><u>video</u>: (flag) true/false, marks whether this stream contains a video track.</p> <p><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.</p>

Sample code:

```
var stream = AlapKoreaRTC.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 AlapKoreaRTC functionality.

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

This method initializes the Client object.

Parameter	Type	Description
key	String	Key could be one of the following: <ul style="list-style-type: none"> <li>- vendor key: provided by AlapKorea during registration.</li> <li>- dynamic key: the token generated with vendor key and sign key. A NodeJS implementation of the token-gen algorithm is provided. This is the safest and recommended way to access the AlapKorea 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(channel, uid, onSuccess, onFailure)**

This method lets the user join an AlapKoreaRTC channel.

Parameter	Type	Description
channel	String	A string providing the unique channel name for the AlapKoreaRTC 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.
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 AlapKoreaRTC 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(stream, onFailure)**

This method publishes a local stream to the server.

Parameter	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	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 to a remote stream from the server.

Parameter	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	Type	Description
stream	object	Stream object, which represents a 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-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).

Sample code:

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

**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.

Sample code:

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:

```
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 the 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 a 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 a video pause.

**enableAudio()**

This method enables audio track in the stream and acts like an 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	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	Type	Description
elementID	String	Represents the html element id.
assetsURL	String	(optional) the URL of the 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.

## 4. User Authentication & Authorization

AlapKorea Web SDK requires applications to authenticate users securely before joining sessions.

- Token-based Authentication (JWT/OAuth2):

Each client must obtain a signed token from your backend using your AlapKorea Vendor Key.

```
client.init(token, () => {
  console.log("Client authenticated and initialized");
},
(err) => {
  console.error("Authentication failed:", err);
});
```

- Role-based Access: Tokens can embed claims (e.g., role: "moderator" or "attendee").
- Integration with OAuth providers (Google, Facebook, Azure AD) is supported by issuing JWTs from your backend

## 5. Session Management

Beyond join and leave, advanced session controls:

- **Session Timeout:** Sessions expire after inactivity (default: 30 min, configurable).
- **Room Locking:**

```
client.lockRoom(channelId, true); // Prevents new users from joining
```

- **Automatic Reconnect:** Built-in retry logic handles network drops

## 6. Error Handling & Logging

Developers can subscribe to global error events:



```
client.on("error", (err) => {
  console.error("Global SDK Error:", err.code, err.message);
});
```

All SDK methods return detailed error codes.

- Developers are encouraged to enable **debug logs**:



```
AlapKoreaRTC.setLogLevel("debug");
```

## 7. Scaling & Load Management

For large conferences:

- **Media Server Clusters:** AlapKorea Global Network automatically routes streams via regional SFUs (Selective Forwarding Units).
- **Load Balancing:** Deploy multiple signaling nodes behind a load balancer.
- **Bandwidth Optimization:** Adaptive bitrate (ABR) and simulcast are enabled by default.

## 8. Advanced Media Controls

Modern features supported by WebRTC extensions:

- **Virtual Backgrounds:**

```
stream.enableVirtualBackground({ type: "blur", strength: "medium" });
```

### Noise Suppression & Echo Cancellation:

Enabled by default, configurable via `stream.setAudioProfile("music")`.

- **Video Effects:** Custom filters can be applied with Canvas/WebGL integration.


## 9. Cross-Browser & Mobile Support

- Supported browsers: Chrome  $\geq 80$ , Firefox  $\geq 78$ , Safari  $\geq 13$ , Edge  $\geq 88$ .
- **Mobile SDKs:** Native iOS/Android SDKs provide hardware-level optimizations.
- Performance benchmarks (720p@30fps with 10 participants) available in [Performance Guide].

## 10. API Rate Limits & Security

- **Encryption:** All streams are encrypted with DTLS-SRTP at the transport layer.
- **End-to-End Encryption (E2EE):** Optional via insertable streams API.
- **Rate Limits:**
  - Join requests: 100 requests/sec per app key.
  - Publish streams: 20 streams per channel.
  - REST API: 500 calls/min per project.


## 11. Screen Sharing



```
const screenStream = AlapKoreaRTC.createStream({
  video: true, screen: true
});
screenStream.init(() => client.publish(screenStream));
```

## 12. Live Streaming


Push channel to RTMP endpoint with:



```
client.startLiveStreaming({ url: "rtmp://youtube.com/live/key" });
```

## 13. Roles/Permissions:

Moderator can mute/remove attendee



```
client.muteUser(uid, "audio");
client.removeUser(uid);
```