

# **Tencent Effect SDK**

## **Beauty AR Web**

### **Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Beauty AR Web

- Overview

- Quick Start

- SDK Integration

  - Overview

  - Integrating Beauty AR Web

  - Web Integration

    - Built-in Camera

    - Custom Stream

    - Loading Optimization

    - Configuring Effects

    - Configuring Segmentation

    - Using Animojis and Virtual Avatars

  - Integration into Mini Program

  - Release Notes

  - API Error Codes

- Parameters and APIs

- Console Guide

  - Customizing Materials

    - 3D Effects

    - Material Customization Guide

    - Face Stickers

  - Material management

- Demos

- Practical Tutorial

  - Publishing over WebRTC

  - Publishing over WebRTC (Preinitialization Scheme)

  - Publishing Using TRTC

  - Using Beauty AR Web with Mini Program

- FAQs

# Beauty AR Web

## Overview

Last updated : 2024-05-08 16:30:10

As a key part of Tencent Cloud RT-Cube, Beauty AR Web provides an easy way to implement AR effects such as beauty filters and special effects on the web.

With the **SDK** and the **material customization and management tool**, you can easily add AR effects such as AI beautification, filters, makeup styles, stickers, animojis, and virtual avatars to your mobile or desktop webpages.

## Commercial editions

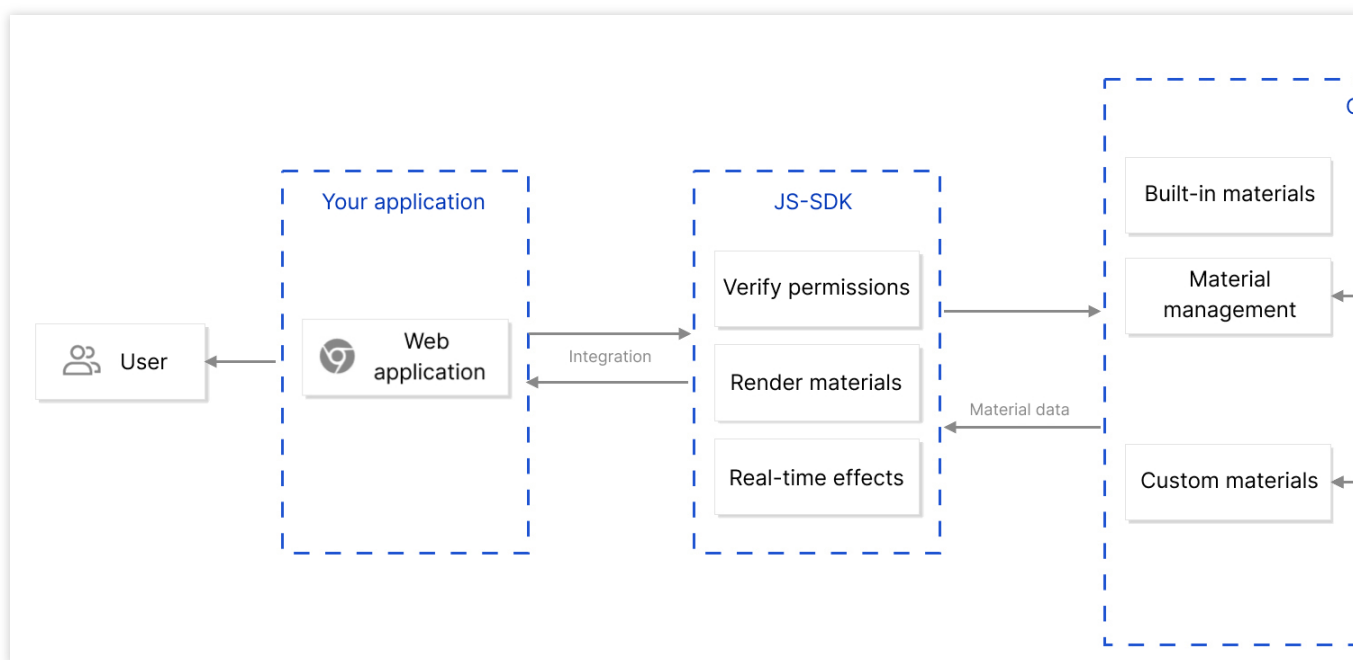
Beauty AR Web has been commercially available since February 2023 and comes in Standard and Advanced editions. You can purchase a license for either edition based on your business needs.

**Note:**

A free trial license used during a trial period cannot be renewed upon expiration. If you still need to use the product after a trial license expires, you need to purchase an official license.

Package Edition	Feature	Supported Domains	Billing Type
Standard	Beautification, makeup, filters, 2D stickers, and virtual backgrounds	Single domains	Annual subscription
Advanced	Beautification, makeup, filters, 2D stickers, and virtual backgrounds 3D stickers, animojis, and virtual avatars	Single domains and wildcard domains	Annual subscription

## Architecture



## Demos

Platform	Address	Runtime Environment
Web	<a href="#">Click here to try</a>	We recommend that you use desktop Chrome 90 or later in a PC environment. For mobile, we recommend that you use the latest version of Safari or Chrome

## Quick Connection

You can run a demo within just a few minutes to try out the capabilities of Beauty AR Web. For more information, see [Quick Start](#).

## Technical Support

If you have any questions, feel free to [contact us](#).

# Quick Start

Last updated : 2024-07-11 17:02:47

**Beauty AR Web** allows you to implement AR beautification, filters, makeup, stickers, and other features on websites. This document describes how to quickly run a web application that supports real-time beautification locally. Based on this application, you can implement other features as instructed in relevant documents.

## Note:

This project is a test project intended for local testing only. To officially launch your project with Beauty AR Web capabilities, you need to purchase an official license and configure the website domain in the product console. For more information, see [Upgrade the trial license](#).

## Step 1. Create a license

### 1. Applying for a trial license

make sure you have created a web license as instructed in [Applying for a trial license](#).

### 2. Getting the license key and token

After the creation, you can see the information of the created test project in the project list and get the **License Token** for Beauty AR Web and the **License key** for the test project. you also can get the **App ID** .

## Note:

The license token is used to calculate the authentication signature, so make sure to keep it secure and confidential. Here, the token is used to calculate the signature on the frontend just to help you run the demo locally. In a real production environment, you need to implement the signature algorithm on the server.

## License Management

Web Licenses

Mobile Licenses

Create Official License

Create Trial License

Search by name

test-chunduan

Trial license

License Information

Domain

1234.com

Creation Time

May 10, 2024 16:41:23 (UTC+08:00) Asia/Shanghai

Basic information

App ID

15...

License Key

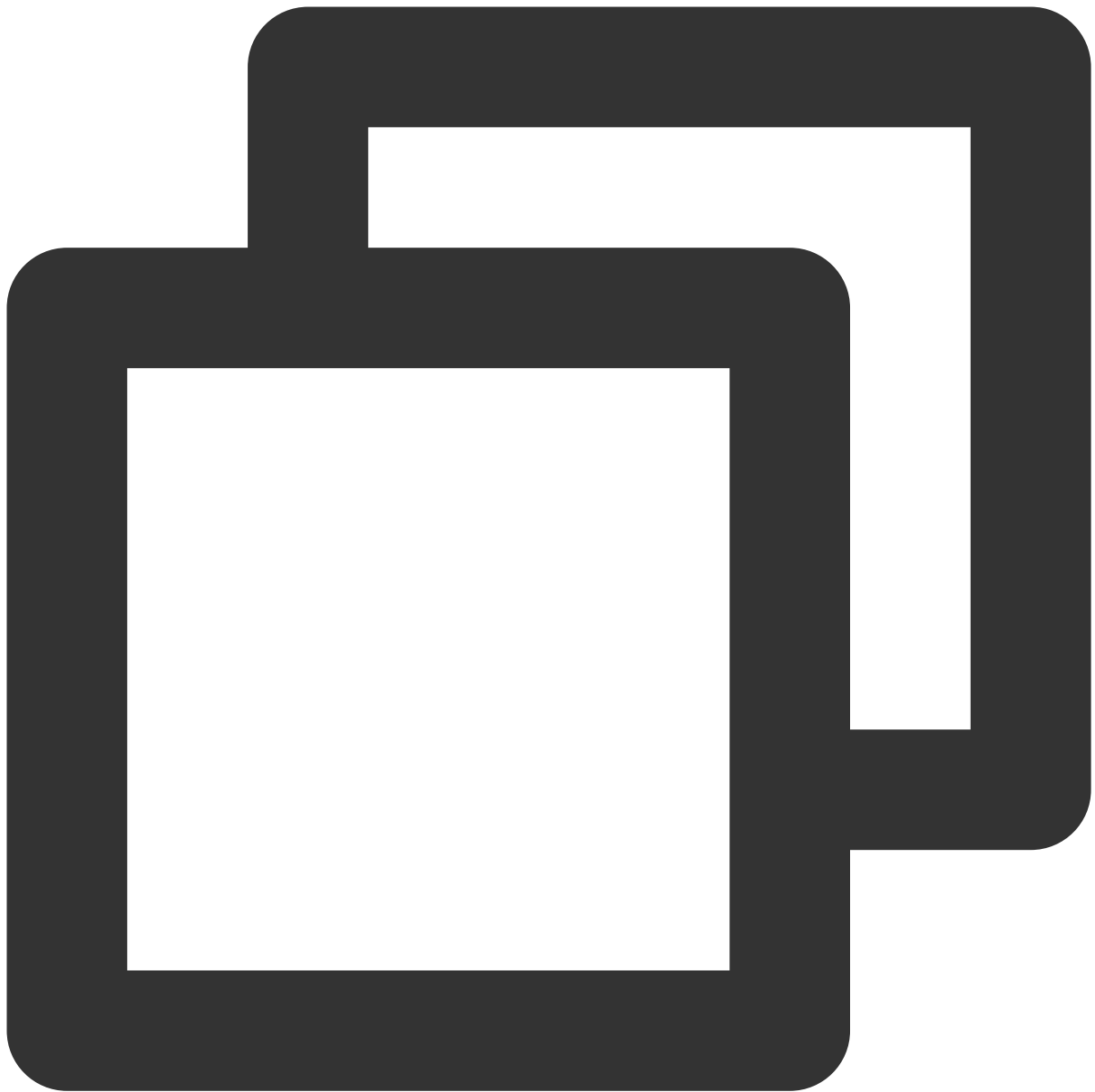
8621e2b1e539f196...

License Token

\*\*\*\*\*

## Step 2. Run locally

1. Pull [sample project](#) code.



```
git clone https://github.com/tencentcloud-webar/web-demo-en.git  
cd quick-start
```

2. replace the specified configuration items in `index.js` with the license key, token, and `APPID` obtained in [step 1](#) as shown below:



```
/** ----- Authentication configuration ----- */

/**
 * Tencent Cloud APPID
 *
 * View the APPID at [Tencent Cloud Account Center](https://console.tencentcloud.com/developer)
 */
const APPID = '' // 'your APPID';

/**
 * Web LicenseKey
 *
 * Obtain a LicenseKey by creating a project at [Web License Management](https://console.tencentcloud.com/xmagic/web)
 */
const LICENSE_KEY = '' // 'your LicenseKey';

/**
 * This Token is the secret key used for calculating the signature.
 * Obtain a Token by creating a project at [Web License Management](https://console.tencentcloud.com/xmagic/web)
 */
const token = '' // 'your token';

/** ----- */

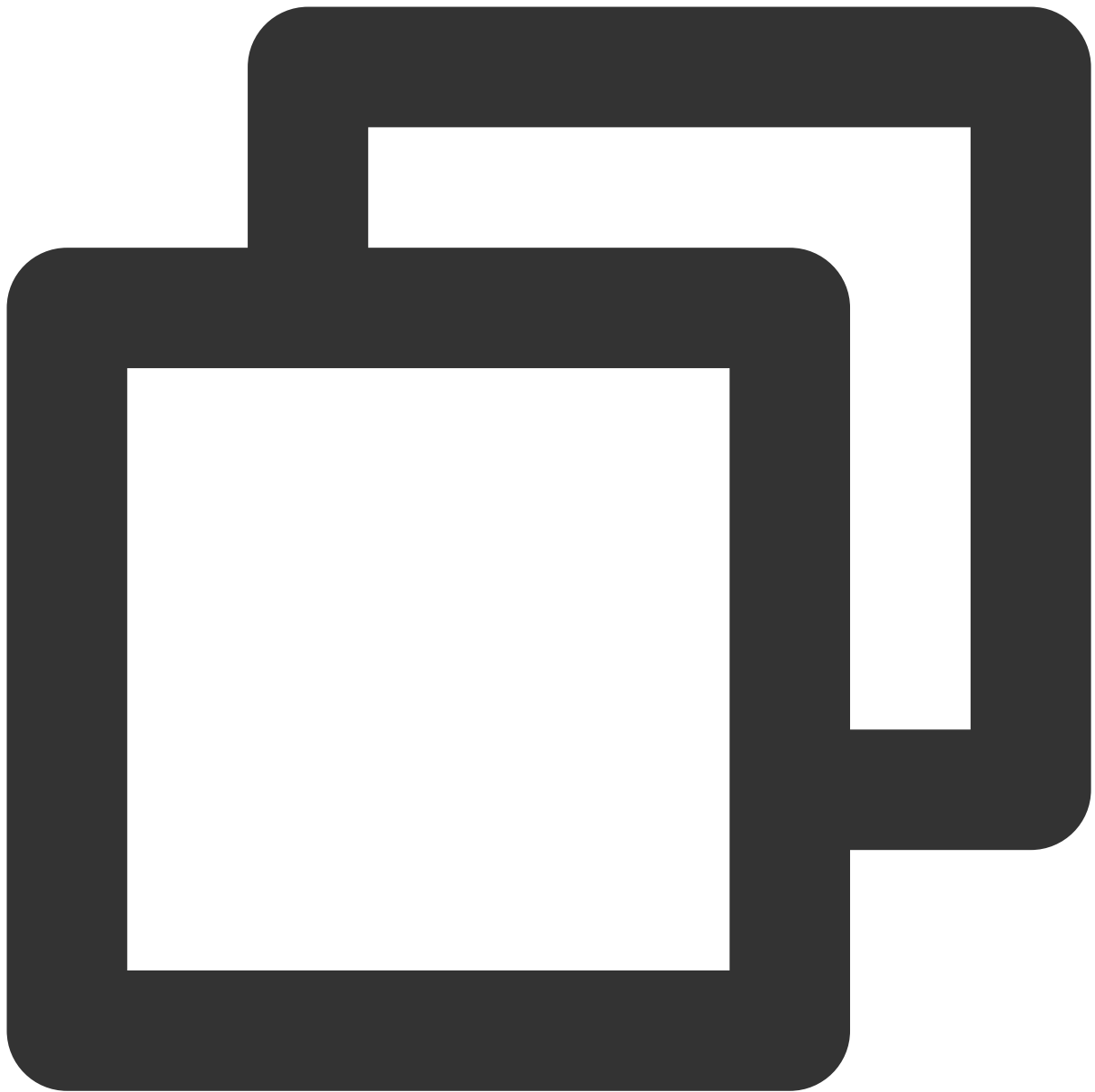
/**
 * Obtain Signature Function
 *
 * WARNING: This is only for demo and debugging purposes.
 * In the production environment, please keep the Token on the server side and migrate the signature calculation method to the server.
 * The signature can be obtained by calling the Interface from the front end.
 *
 * eg:
 * async function () {
 *   return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
  return { signature, timestamp };
};
```

3. Run the project in the local development environment.

#### Note:

Before running your project locally, make sure the `nodejs` environment is already installed on the device.

Run the following commands successively in the project directory and access `localhost:8090` in the browser to try out the capabilities of Beauty AR Web.



```
# Install dependencies
npm i

# Compile and run the code
npm run dev
```

After following the steps above, you can try out the filters and effects of the SDK for desktop browser. You can use the built-in materials to try out various makeup filters and effects as instructed in [Overview](#), or use more capabilities of Beauty AR Web such as custom stickers, makeup, and filters. For detailed directions on how to customize effect materials, please [contact us](#).

# SDK Integration Overview

Last updated : 2024-06-12 17:37:26

This document describes how to quickly and securely connect to **Beauty AR Web** and use its features. If you have any questions, please [contact us](#)

## Preparations

Before connecting to the SDK, make sure you have purchased a web license and created a project as instructed in [Activate the Service](#).

### Getting parameter information


- Getting the `App Id` , `license key` and `token` from [License Management](#)

### License Management


[Web Licenses](#) [Mobile Licenses](#)

[Create Official License](#) [Create Trial License](#)

test-chunduan Trial license

 License Information

Domain	1234.com
Creation Time	May 10, 2024 16:41:23 (UTC+08:00) Asia/Shanghai

 Basic information

App ID	15...
License Key	8621e2
License Token	*****

**Web Domain:** The domain information entered during project creation. The license can be used only under this domain.

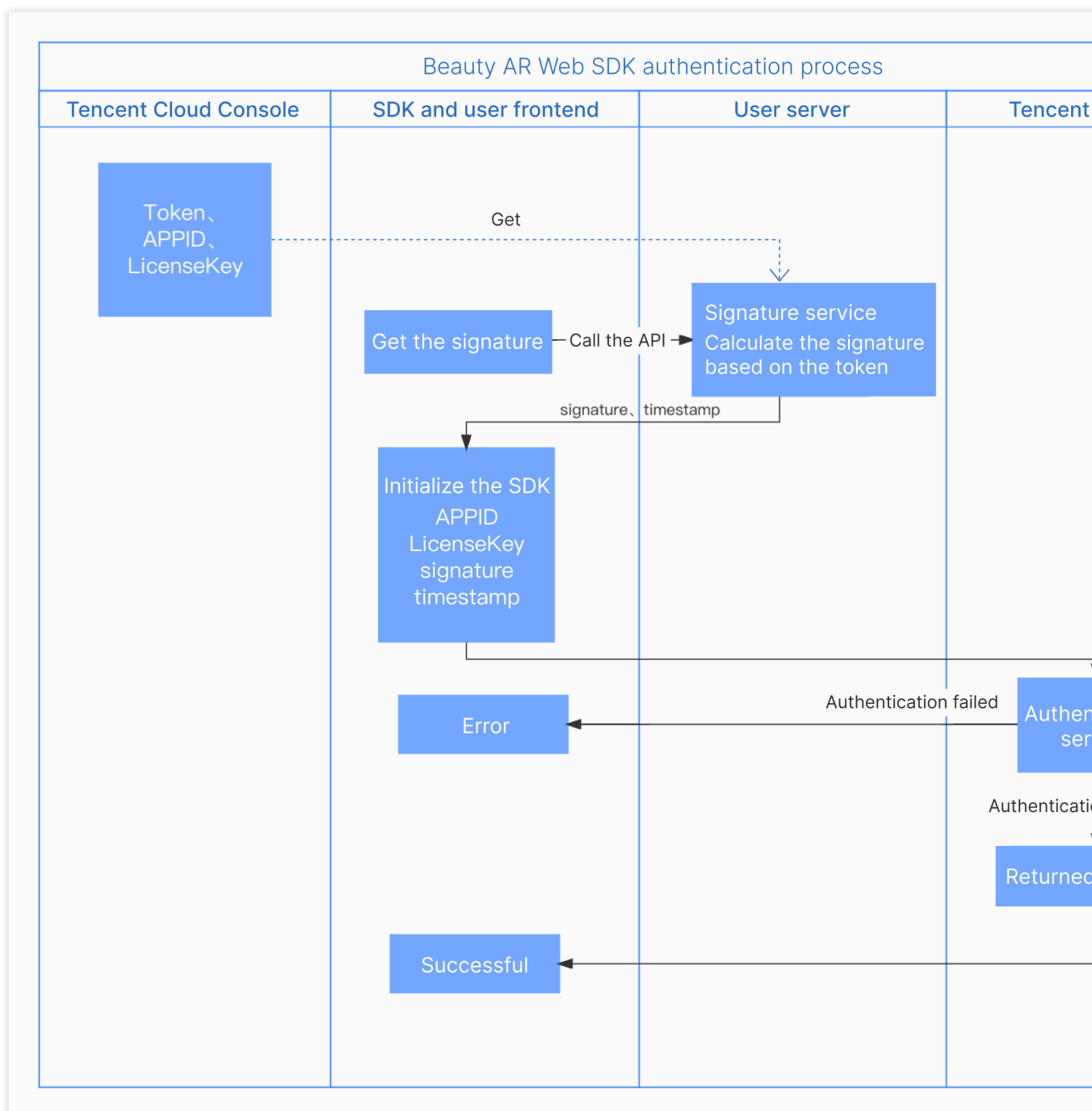
#### Note:

Be sure to use the license key and token that **match the developed domain**, otherwise authentication will fail, and the SDK cannot be properly initialized.

### Preparing signing information

In addition to the license key that is needed to authorize the SDK, you also need to use the token to sign the APIs called in the SDK.

### Signature algorithm authentication process



**Token:** Your unique ID, which is used to sign SDK APIs.

**App ID:** The `APPID` displayed in the Beauty AR console.

**Timestamp:** The current time accurate to the second (10 digits).

**Signature:** The signature used for signing, which expires after five minutes.

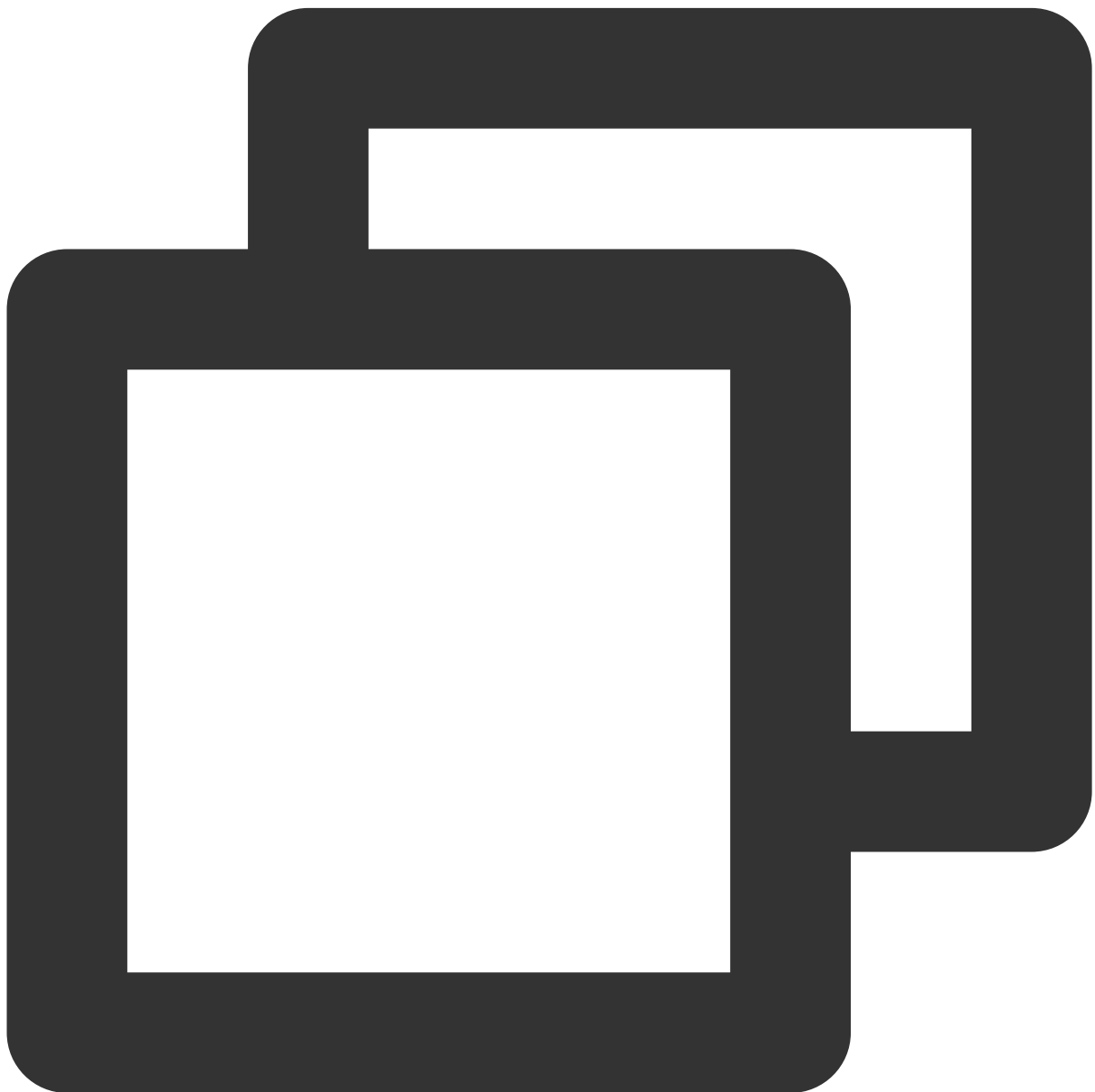
## Deploying a signature service

Because the signature expires after a given time, and to prevent the token from being leaked, you need to deploy a signature generation service.

### Note:

If the token is leaked, your identity will be stolen, which will lead to your resources also being leaked.

If the signature generation method is implemented on the frontend, the token may be leaked. Therefore, to safeguard your security, we recommend that you do not implement signature generation on the frontend.



```
// Taking the `express` backend as an example,  
// Signature algorithm: sha256(timestamp+token+appid+timestamp)
```

```
const { createHash } = require('crypto');
const config = {
  appid: 'Your Tencent Cloud `APPID`',
  token: 'Your token',
}
const sha256 = function(str) {
  return createHash('sha256')
    .update(str)
    .digest('hex');
}

const genSignature = function() {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(timestamp + config.token + config.appid + timestamp).t
  return { signature, timestamp };
}

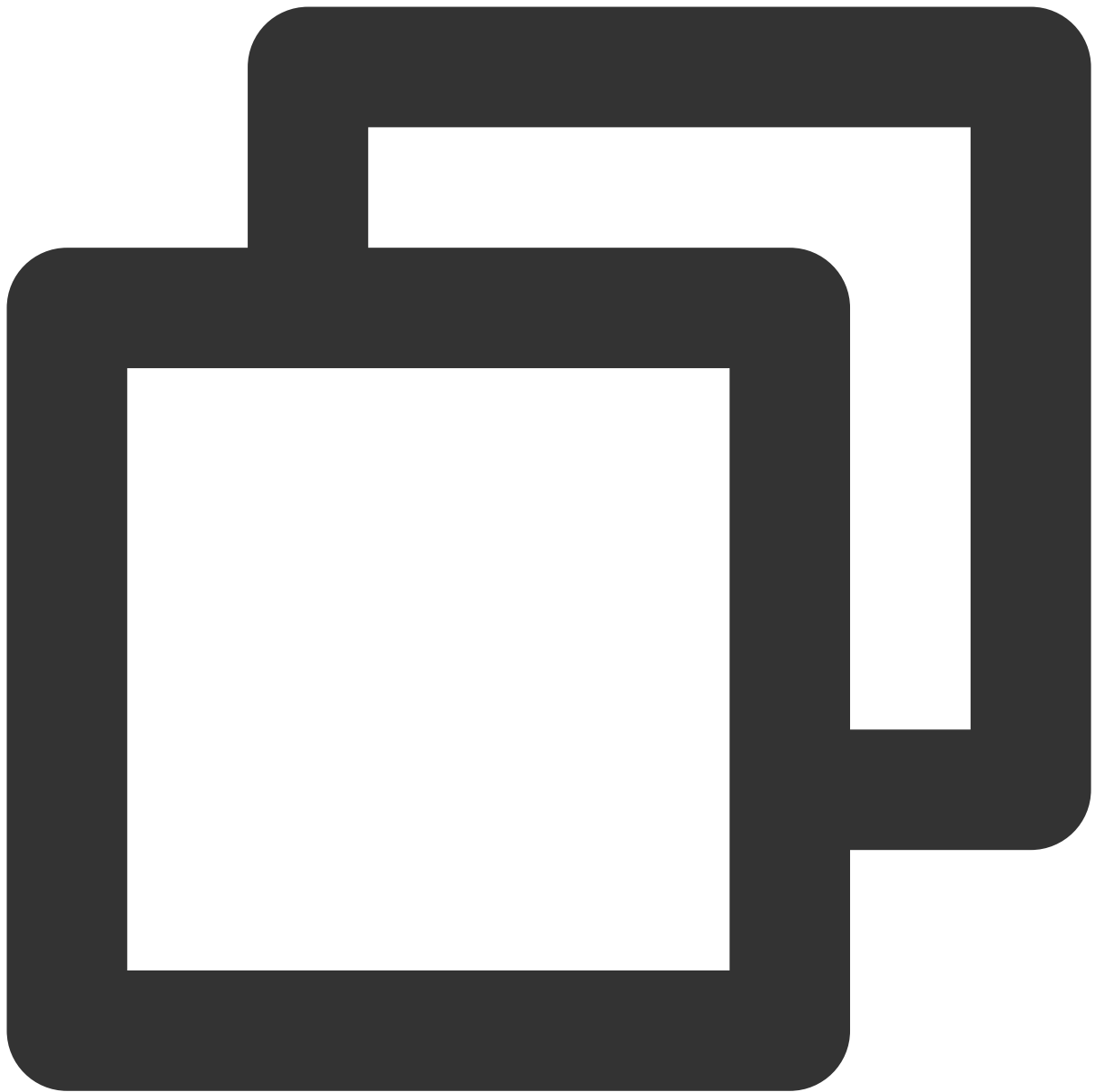
app.get("/get-ar-sign", (req, res) => {
  const sign = genSignature();
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'GET, OPTIONS');
  res.send(sign);
})
```

## Calling the signature service on the frontend

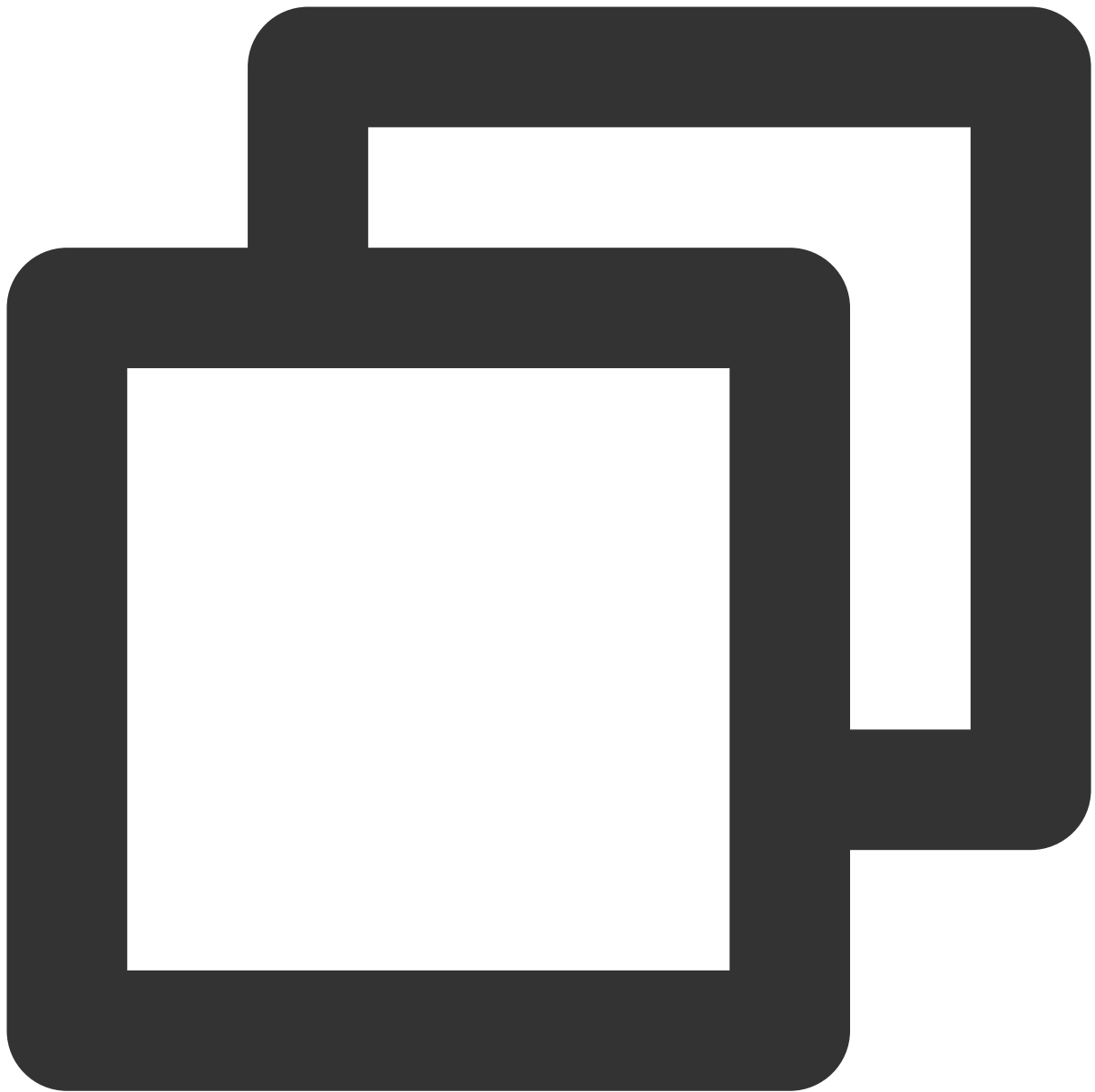
After deploying the signature service, add a signature acquisition method to your webpage for the SDK to connect to and call.

Web

Mini program



```
async function getSignature() {  
  const res = await fetch('Your domain/get-ar-sign')  
  const authdata = await res.json()  
  console.log('authdata',authdata)  
  return authdata  
}
```



```
async function getSignature() {  
  return new Promise((resolve, reject) => {  
    wx.request({  
      url: 'Your domain/get-ar-sign',  
      method: 'GET',  
      success(res) {  
        console.log('getSignature ok', res)  
        resolve(res.data);  
      },  
      fail(e) {  
        console.log('getSignature error', e)  
      }  
    })  
  })  
}
```



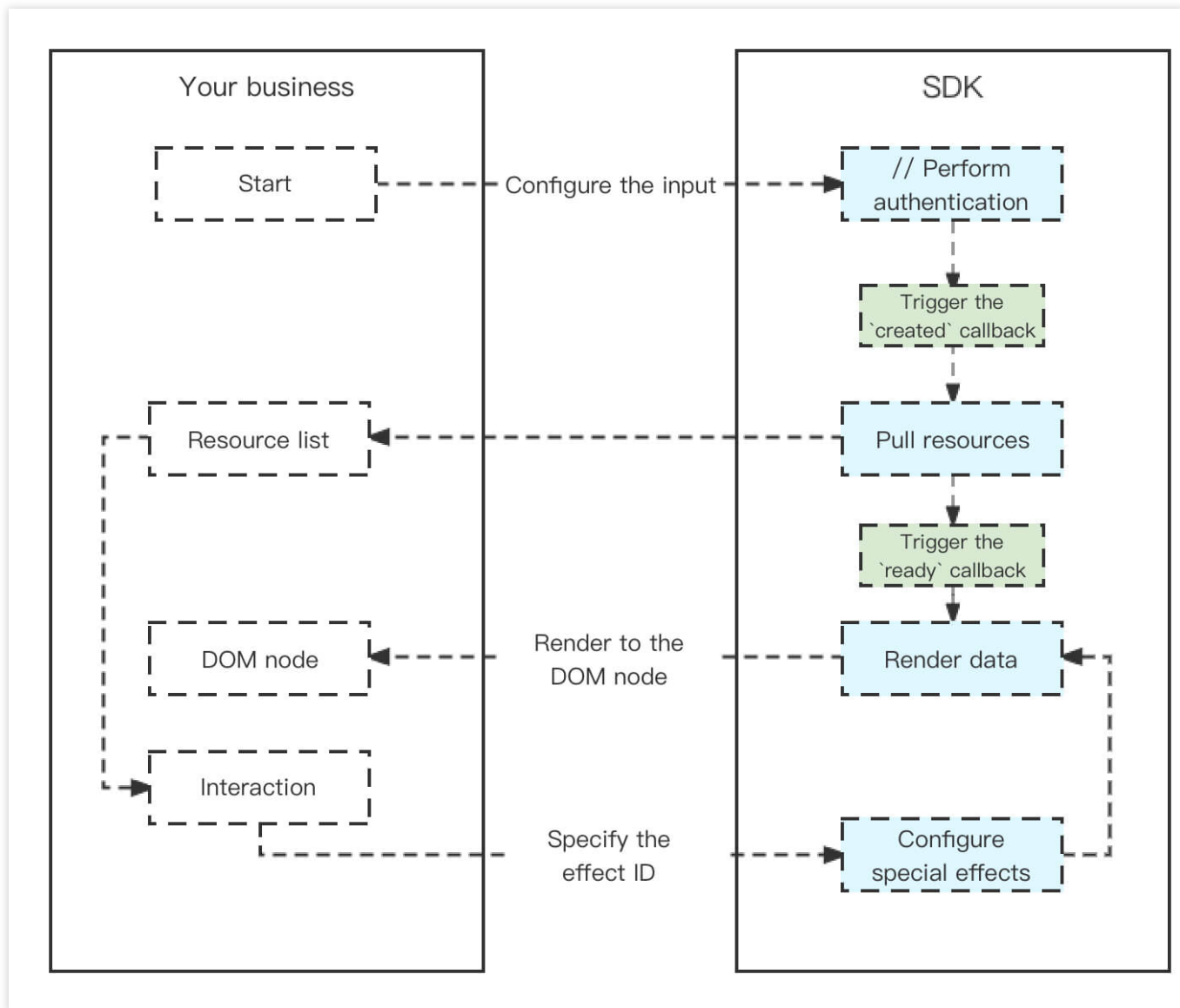
```
    }  
  })  
})  
}
```

## SDK Integration

After completing the above preparations, follow the process below to connect to and use the SDK.

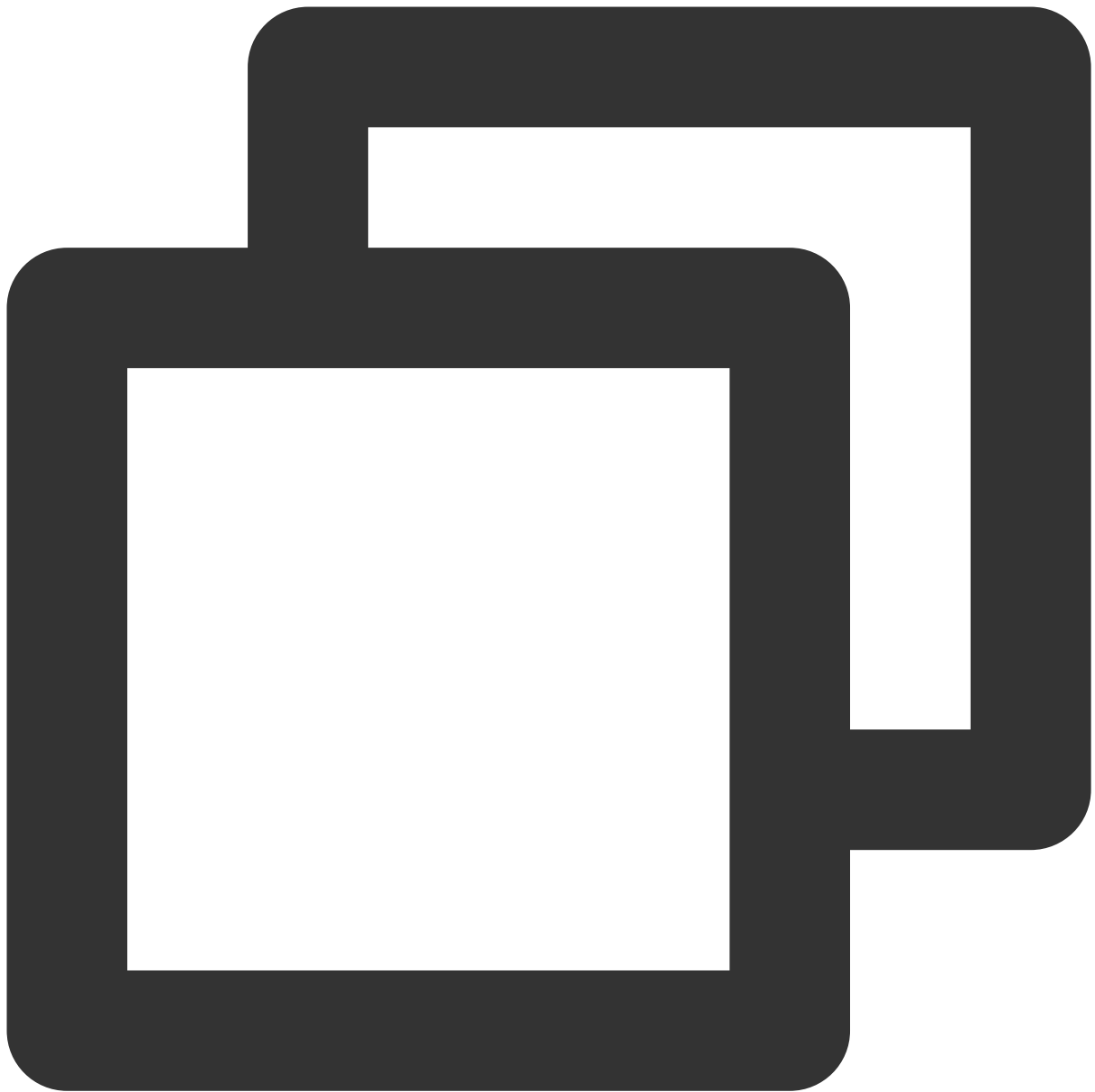
### Process description

The **Tencent Effect web SDK** offers simple and minimally invasive APIs. To integrate it and use its features, you only need to initialize an instance and add the render node to your webpage.



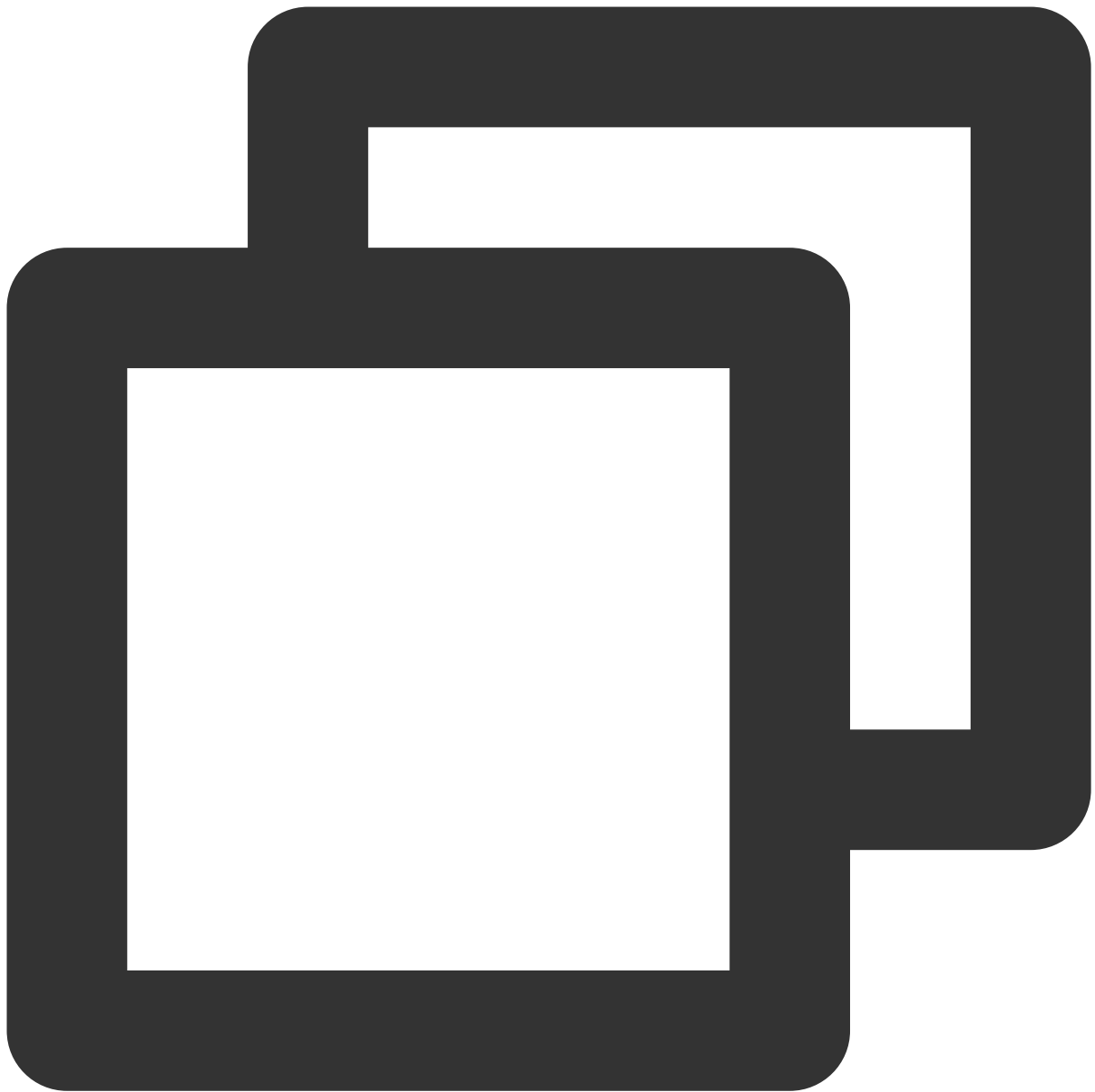
## Installing the SDK

The SDK is offered as an npm package.



```
npm install tencentcloud-webar
```

In addition, you can also use it for your project by importing JS.



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

## Initializing the SDK

For web integration, we offer two initialization modes for the SDK.

**Built-in camera and player:** The device's built-in camera and player are used. API calls are easy and fast, with rich interactive features.

**Custom streams:** You can use this mode if you want to apply effects to your own streams or want greater flexibility and control.

## Using the SDK

### Configuring beauty filters and special effects

For more information, see [Configuring Filters and Effects](#).

### Segmentation

The keying feature allows you to segment and change the background in the image. For details, see [Configuring Segmentation](#).

### 3D effects

For more information, see [Configuring Filters and Effects](#).

### Animojis and virtual avatars

This capability relies on a WebGL2 environment. For more information, see [Configuring Animojis and Virtual Avatars](#).

## Parameters and APIs

See [Parameters and APIs](#).

## Sample Code

See [Quick Start](#).

# Integrating Beauty AR Web

Last updated : 2023-04-11 16:12:04

## Separate integration

The SDK can be integrated into a webpage as instructed in [Overview](#).

The SDK can be integrated into a mini program as instructed in [Integration into Mini Program](#).

## Scenario-specific integration

The SDK can be integrated with WebRTC as instructed in the following documents:

[Publishing over WebRTC](#)

[Publishing over WebRTC \(Preinitialization Scheme\)](#)

[Publishing with TRTC](#)

# Web Integration

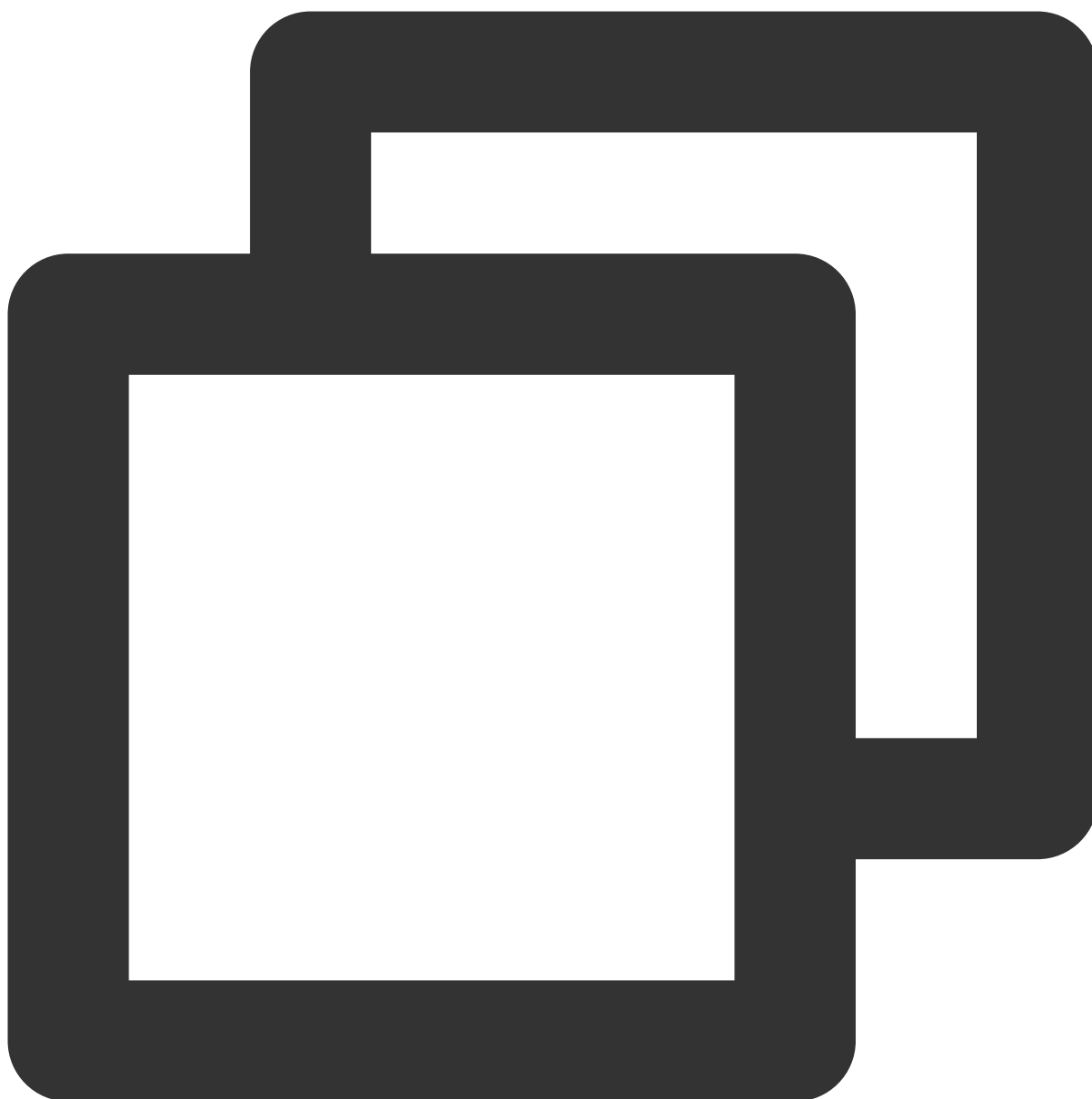
## Built-in Camera

Last updated : 2024-05-08 18:30:33

You can choose this integration mode if you want to use the SDK with a device's built-in camera or if your business scenario involves interaction with the built-in camera.

### Step 1. Import the SDK

use npm package:



```
npm install tencentcloud-webar
```





```
import { ArSdk } from 'tencentcloud-webar';
```

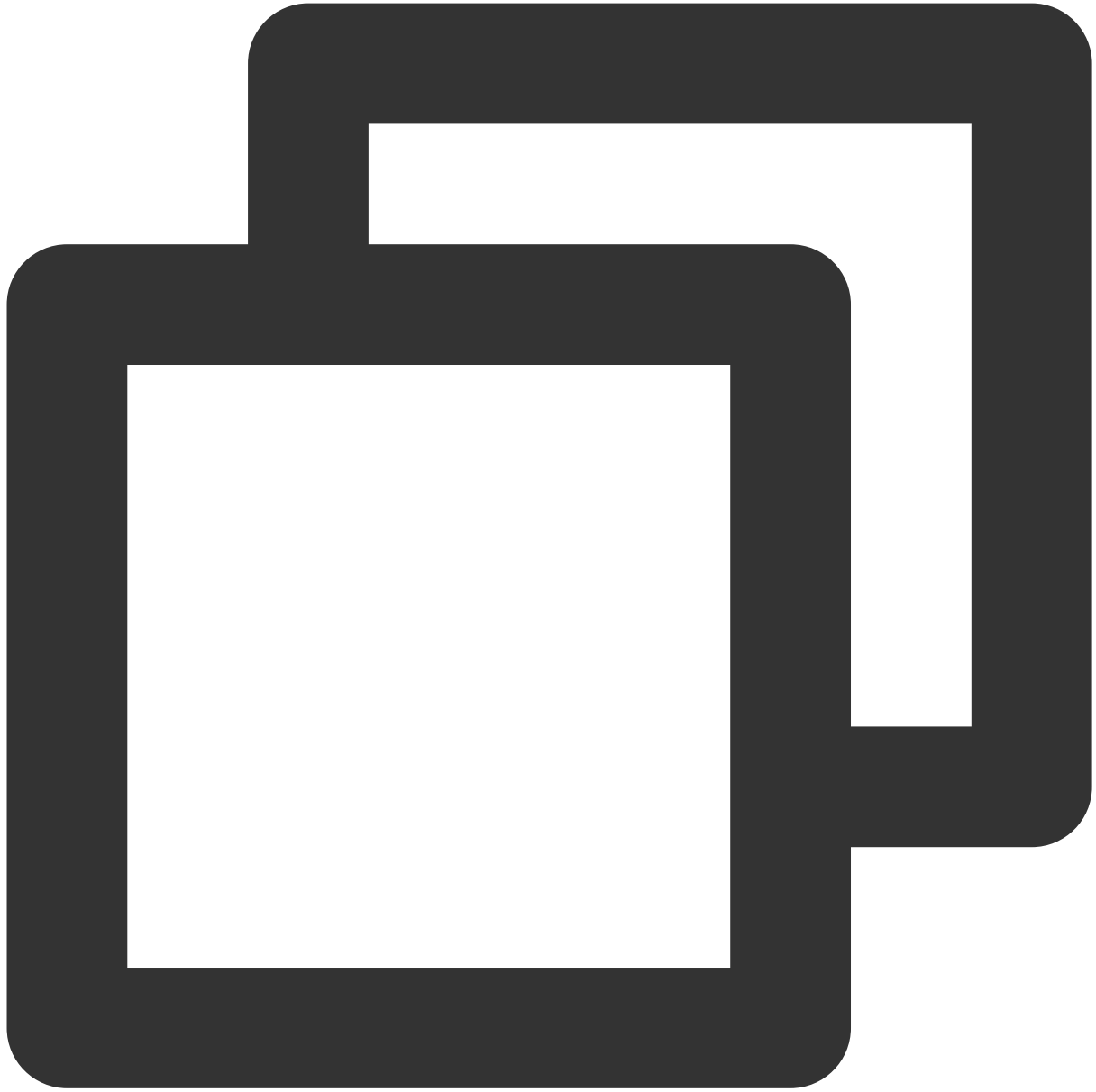
you can also import the SDK using the following method:



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
<script>
  // Receive ArSdk class from window.AR
  const { ArSdk } = window.AR;
  .....
</script>
```

## Step 2. Initialize an Instance

Initialize an SDK instance.



```
// Get the authentication information
const authData = {
  licenseKey: 'xxxxxxxxxx',
  appId: 'xxx',
  authFunc: authFunc // For details, see "Configuring and Using a License - Signa
};

const config = {
  module: {
```

```
    beautify: true, // Whether to enable the effect module, which offers beauti
    segmentation: true // Whether to enable the keying module, which allows you
  },
  auth: authData, // The authentication information
  camera: { // Pass in the camera parameters
    width: 1280,
    height: 720
  },
  beautify: { // The effect parameters for initialization (optional)
    whiten: 0.1,
    dermabrasion: 0.3,
    eye: 0.2,
    chin: 0,
    lift: 0.1,
    shave: 0.2,
    .....
  }
}
const sdk = new ArSdk(
  // Pass in a config object to initialize the SDK
  config
)

let effectList = [];
let filterList = [];

sdk.on('created', () => {
  // You can display the effect and filter list in the `created` callback. For de
  sdk.getEffectList({
    Type: 'Preset',
    Label: 'Makeup',
  }).then(res => {
    effectList = res
  });
  sdk.getCommonFilter().then(res => {
    filterList = res
  })
})

// Call `setBeautify`, `setEffect`, or `setFilter` in the `ready` callback
// For details, see “SDK Integration - Configuring Effects”
sdk.on('ready', () => {
  // Configure beautification effects
  sdk.setBeautify({
    whiten: 0.2
  });
  // Configure special effects
```

```
    sdk.setEffect({
      id: effectList[0].EffectId,
      intensity: 0.7
    });
    // Configure filters
    sdk.setFilter(filterList[0].EffectId, 0.5)
  })
```

**Note:**

The loading of the effect and keying modules takes time and consumes resources. You can enable only the module you need during initialization. A module not enabled will not be loaded or initialized.

If you specify the `camera` parameter of `config`, the video data the SDK captures from the device's camera will be used as the input. We also offer some basic device management APIs. For details, see [Step 6. Control Devices](#).

## Step 3. Play the Stream

The SDK offers players for quick preview of effects on your webpage. The players you get in different callbacks vary slightly. Choose the one that fits your needs.

If you want to display a video image as quickly as possible, get the player in the `cameraReady` callback. Because the SDK hasn't loaded the resources or completed initialization at this point, the player can only play the original video.



```
sdk.on('cameraReady', async () => {  
  // Initialize a player of the SDK. `my-dom-id` is the ID of the player's container  
  const player = await sdk.initLocalPlayer('my-dom-id')  
  // Play the video  
  await player.play()  
})
```

If you want to play the video after the SDK is initialized and effects are applied, get the player in the `ready` playback.



```
sdk.on('ready', async () => {  
  // Initialize a player of the SDK. `my-dom-id` is the ID of the player's container  
  const player = await sdk.initLocalPlayer('my-dom-id')  
  // Play the video  
  await player.play()  
})
```

**Note:**

The player obtained by `initLocalPlayer` is muted by default. If you unmute it, there may be echoes.

The player obtained is integrated with the `sdk.getOutput()` API.

The player object obtained by `initLocalPlayer` is integrated with the following APIs:

API	Description	Request Parameter	Return Value
<code>play</code>	Plays the video.	-	Promise;
<code>pause</code>	Pauses the video. This does not stop the stream. You can resume the playback.	-	-
<code>stop</code>	Stops the video. This stops the stream.	-	-
<code>mute</code>	Mutes the video.	-	-
<code>unmute</code>	Unmutes the video.	-	-
<code>setMirror</code>	Sets whether to mirror the video.	true false	-
<code>getVideoElement</code>	Gets the built-in video object.	-	HTMLVideoElement
<code>destroy</code>	Terminates the player.	-	-

**Note:**

The player's behaviors are affected by [camera settings](#). Camera settings prevail over the settings of `LocalPlayer`.

For example, after you call `camera.muteVideo` to disable video, playback will not start even if you call `play`.

After you call `camera.unmuteVideo` to enable video, the player will play the video automatically.

Therefore, if you specify `camera`, you don't need to manually configure `localPlayer`.

## Step 4. Get the Output

If you need to publish the stream, call `getOutput` to get the output stream.

After getting the `MediaStream`, you can use a live streaming SDK (for example, TRTC web SDK or LEB web SDK) to publish the stream.





```
const output = await sdk.getOutput()
```

**Note:**

If you use the built-in camera, the type of all media returned by `getOutput` is `MediaStream`.

The video track of the output stream is processed in real time by the Tencent Effect SDK. The audio track (if any) is kept.

`getOutput` is an async API. The output will be returned only after the SDK is initialized and has generated a stream.

You can pass an `FPS` parameter to `getOutput` to specify the output frame rate (for example, 15). If you do not pass this parameter, the original frame rate will be kept.

To learn more about how to publish the processed streams, see [Publishing Using TRTC](#) and [Publishing over WebRTC](#).

## Step 5. Configuring Effects

For detailed directions, see [Configuring Effects](#).

## Step 6. Control Devices

You can use an `sdk.camera` instance to enable and disable the camera or perform other camera operations.



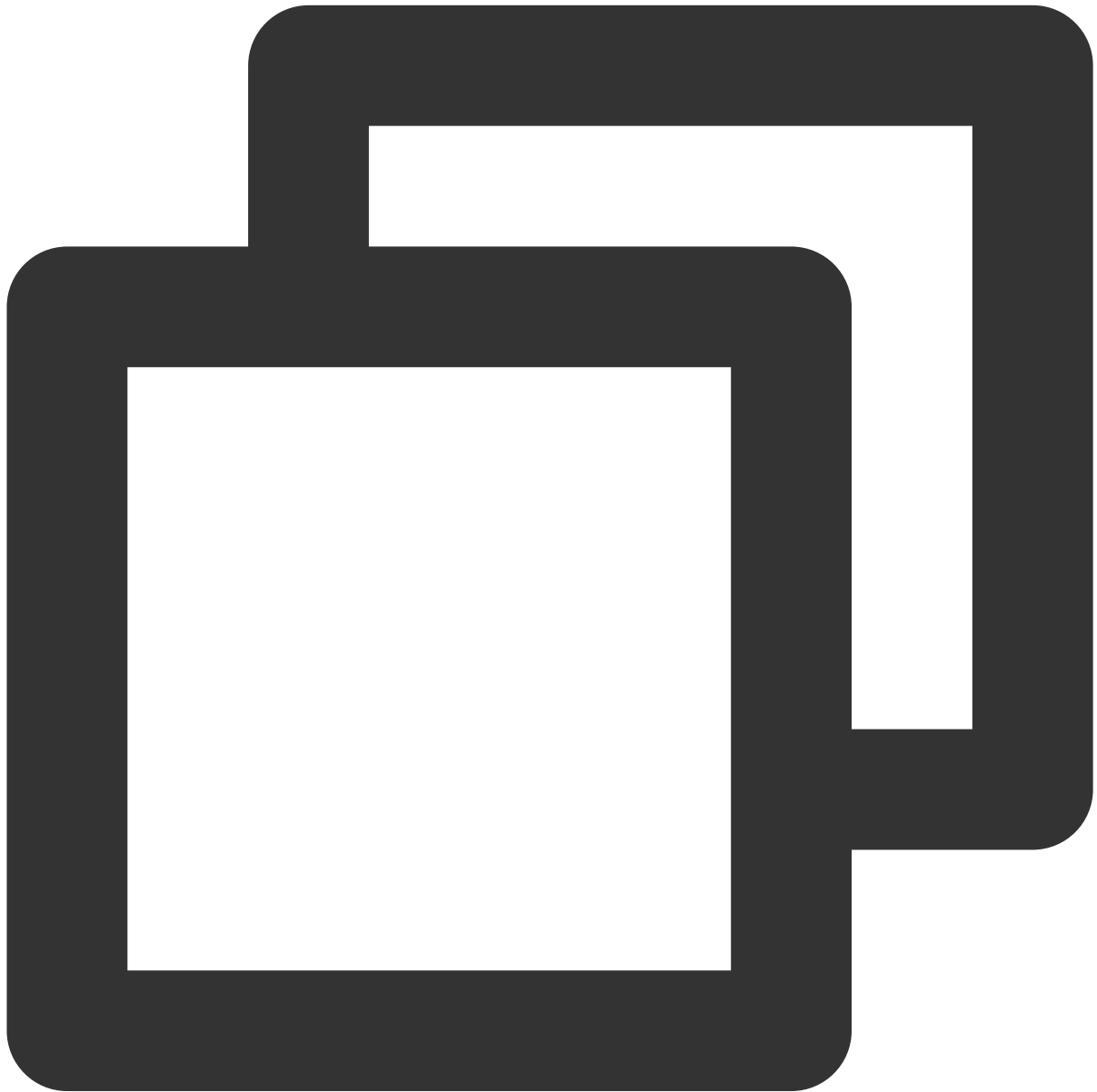
```
const output = await sdk.getOutput()
// Your business logic
// ...

// `sdk.camera` will have been initialized after `getOutput`. You can get an instance
const cameraApi = sdk.camera

// Get the device list
const devices = await cameraApi.getDevices()
console.log(devices)
// Disable the video track
```

```
// cameraApi.muteVideo()  
// Enable the video track  
// cameraApi.unmuteVideo()  
// Change to a different camera by specifying the device ID (if there are multiple  
// await cameraApi.switchDevices('video', 'your-device-id')
```

If you want to get an `sdk.camera` instance as soon as possible, you can get it in the `cameraReady` callback.



```
// Initialization parameters  
// ...  
const sdk = new ArSdk(  

```

```

    config
  )

  let cameraApi;
  sdk.on('cameraReady', async () => {
    cameraApi = sdk.camera
    // Get the device list
    const devices = await cameraApi.getDevices()
    console.log(devices)
    // Disable the video track
    // cameraApi.muteVideo()
    // Enable the video track
    // cameraApi.unmuteVideo()
    // Change to a different camera by specifying the device ID (if there are multi
    // await cameraApi.switchDevices('video', 'your-device-id')
  })

```

You can use the following APIs of `camera` to control the built-in camera.

API	Description	Request Parameter	Return Value
<code>getDevices</code>	Gets all devices.	-	<code>Promise&lt;Array&lt;MediaDeviceInfo&gt;&gt;</code>
<code>switchDevice</code>	Switches the device.	<code>type:string,</code> <code>deviceId:string</code>	<code>Promise</code>
<code>muteAudio</code>	Mutes the current stream.	-	-
<code>unmuteAudio</code>	Unmutes the current stream.	-	-
<code>muteVideo</code>	Disables the video track of the camera stream. This does not stop the stream.	-	-
<code>unmuteVideo</code>	Enables the video of the camera stream.	-	-
<code>stopVideo</code>	Disables the camera. This stops the video stream, but the audio stream is not affected.	-	-
<code>restartVideo</code>	Enables the camera. This API can only be called after <code>stopVideo</code> .	-	<code>Promise</code>
<code>stop</code>	Disables the current camera and audio device.	-	-



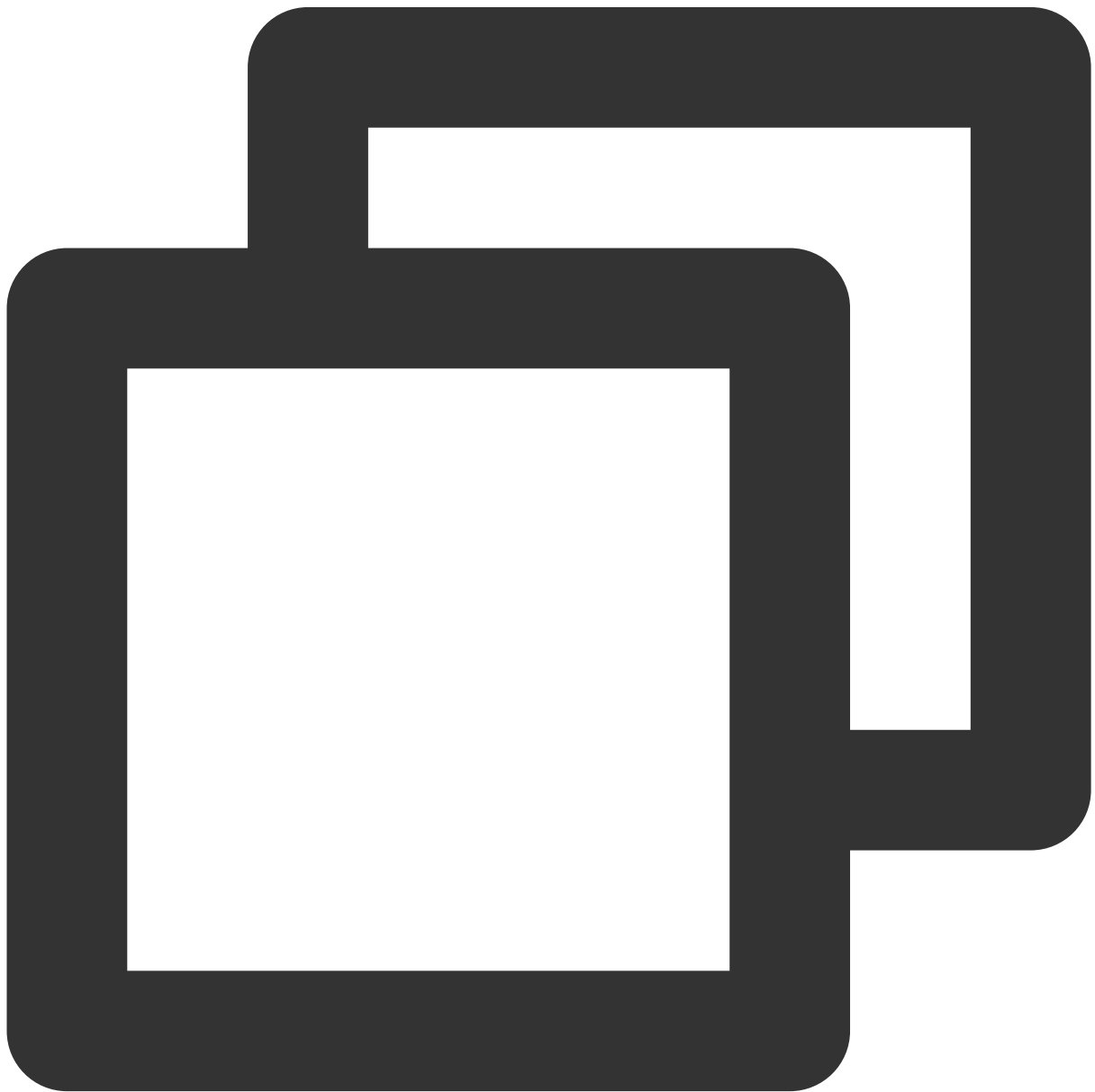
# Custom Stream

Last updated : 2024-05-08 18:30:33

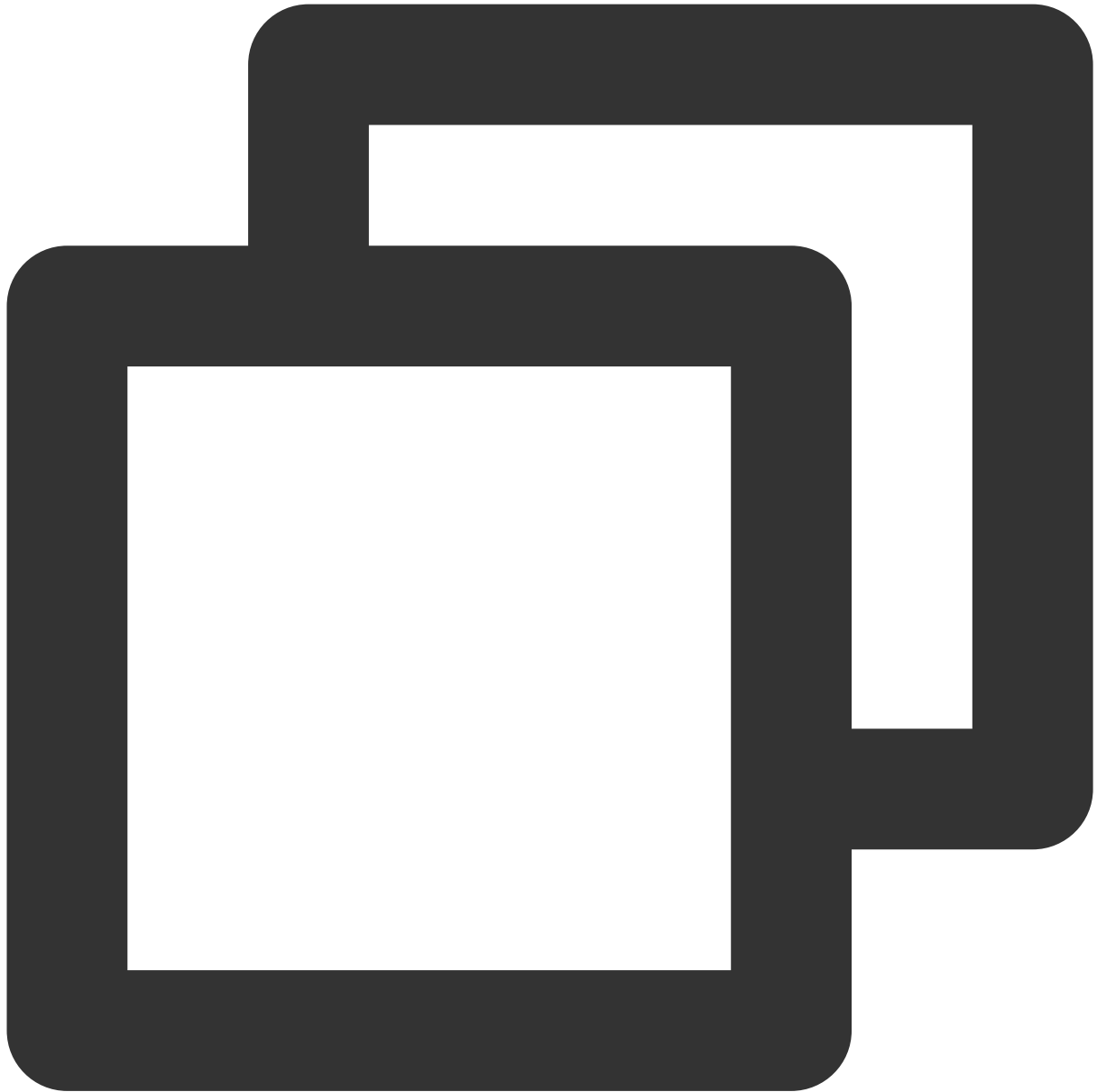
You can use this integration mode if you want to apply effects to your own streams or want greater flexibility and control.

## Step 1. Import the SDK

use npm package:



```
npm install tencentcloud-webar
```



```
import { ArSdk } from 'tencentcloud-webar';
```

you can also import the SDK using the following method:





```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
<script>
  // Receive T ArSdk class from window.AR
  const { ArSdk } = window.AR;
  .....
</script>
```

## Step 2. Initialize an instance

1. Initialize an SDK instance.



```
// Get the authentication information
const authData = {
  licenseKey: 'xxxxxxxxxx',
  appId: 'xxx',
  authFunc: authFunc // For details, see "Configuring and Using a License - Signatur
};
// The input stream
const stream = await navigator.mediaDevices.getUserMedia({
  audio: true,
  video: { width: w, height: h }
})
```

```
const config = {
  module: {
    beautify: true, // Whether to enable the effect module, which offers beautiful
    segmentation: true // Whether to enable the keying module, which allows you to
  },
  auth: authData, // The authentication information
  input: stream, // The input stream
  beautify: { // The effect parameters for initialization (optional)
    whiten: 0.1,
    dermabrasion: 0.3,
    eye: 0.2,
    chin: 0,
    lift: 0.1,
    shave: 0.2
  }
}
const sdk = new ArSdk(
  // Pass in a config object to initialize the SDK
  config
)
```

**Note:**

The loading of the effect and segmentation modules takes time and consumes resources. You can enable only the module you need during initialization. A module not enabled will not be loaded or initialized.

2. For `input`, you can also pass in `string|HTMLImageElement` to process images.



```
const config = {
  auth: authData, // The authentication information
  input: 'https://xxx.png', // The input stream
}
const sdk = new ArSdk(
  // Pass in a config object to initialize the SDK
  config
)

// You can display the effect and filter list in the `created` callback. For detail
sdk.on('created', () => {
```

```
// Get the built-in makeup effects
sdk.getEffectList({
  Type: 'Preset',
  Label: 'Makeup',
}).then(res => {
  effectList = res
});
// Get the built-in filters
sdk.getCommonFilter().then(res => {
  filterList = res
})
})

// Call `setBeautify`, `setEffect`, or `setFilter` in the `ready` callback
// For details, see “SDK Integration - Configuring Effects”
sdk.on('ready', () => {
  // Configure beautification effects
  sdk.setBeautify({
    whiten: 0.2
  });
  // Configure special effects
  sdk.setEffect({
    id: effectList[0].EffectId,
    intensity: 0.7
  });
  // Configure filters
  sdk.setFilter(filterList[0].EffectId, 0.5)
})
```

### Step 3. Play the stream

Call `ArSdk.prototype.getOutput` to get the output stream.

The output streams you get in different callbacks vary slightly. Choose the one that fits your needs.

If you want to display a video image as quickly as possible, get and play the stream in the `cameraReady` callback.

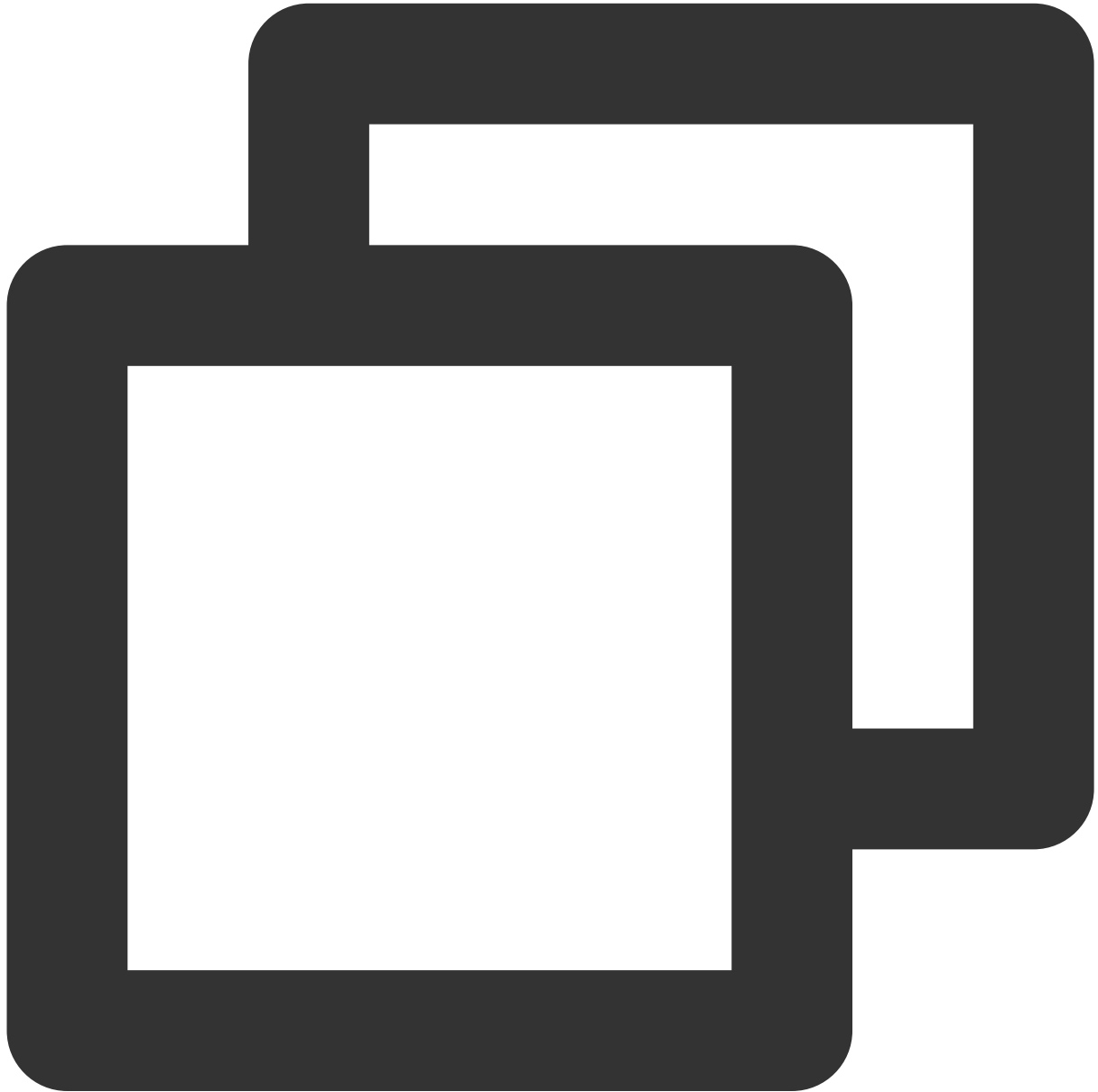
Because the SDK hasn't loaded the resources or completed initialization at this point, the original video will be played.



```
sdk.on('cameraReady', async () => {  
  // By getting the output stream in the `cameraReady` callback, you can display a  
  // You can choose this method if you want to display a video image as soon as pos  
  // You don't need to update the stream after the effects start to work.  
  const output = await ar.getOutput();  
  // Use `video` to preview the output stream  
  const video = document.createElement('video')  
  video.setAttribute('playsinline', '');  
  video.setAttribute('autoplay', '');  
  video.srcObject = output  
  document.body.appendChild(video)
```

```
video.play()  
})
```

If you want to play the video after the SDK is initialized and effects are applied, get and play the output stream in the `ready` playback.



```
sdk.on('ready', async () => {  
  // If you get the output stream in the `ready` callback, because the initializati  
  // The `ready` callback occurs later than `cameraReady`. You can get the output s  
  const output = await ar.getOutput();  
  // Use `video` to preview the output stream
```

```
const video = document.createElement('video')
video.setAttribute('playsinline', '');
video.setAttribute('autoplay', '');
video.srcObject = output
document.body.appendChild(video)
video.play()
})
```

#### Step 4. Get the output

After getting the `MediaStream`, you can use a live streaming SDK (for example, TRTC web SDK or LEB web SDK) to publish the stream.





```
const output = await sdk.getOutput()
```

To learn more about how to publish the processed streams, see [Publishing Using TRTC](#) and [Publishing over WebRTC](#).

**Note:**

If the input passed in is an image, a string-type data URL will be returned. Otherwise, `MediaStream` will be returned.

The video track of the output stream is processed in real time by the Tencent Effect SDK. The audio track (if any) is kept.

`getOutput` is an async API. The output will be returned only after the SDK is initialized and has generated a stream.

You can pass an `FPS` parameter to `getOutput` to specify the output frame rate (for example, 15). If you do not pass this parameter, the original frame rate will be kept.

You can call `getOutput` multiple times to generate streams of different frame rates for different scenarios (for example, you can use a high frame rate for preview and a low frame rate for stream publishing).

## Step 5. Configuring effects

For detailed directions, see [Configuring Effects](#).

## Updating the Input Stream

If you want to feed a new input stream to the SDK after changing the device or enabling/disabling the camera, you don't need to initialize the SDK again. Just call `sdk.updateInputStream` to update the input stream.

The following code shows you how to use `updateInputStream` to update the input stream after switching from the computer's default camera to an external camera.



```
async function getVideoDeviceList(){
  const devices = await navigator.mediaDevices.enumerateDevices()
  const videoDevices = []
  devices.forEach((device)=>{
    if(device.kind === 'videoinput'){
      videoDevices.push({
        label: device.label,
        id: device.deviceId
      })
    }
  })
}
```

```
    })
    return videoDevices
  }

  async function initDom(){
    const videoDeviceList = await getVideoDeviceList()
    let dom = ''
    videoDeviceList.forEach(device=>{
      dom = `${dom}
      <button id=${device.id} onclick='toggleVideo("${device.id}")'>${device.labe
      `

    })
    const div = document.createElement('div');
    div.id = 'container';
    div.innerHTML = dom;
    document.body.appendChild(div);
  }

  async function toggleVideo(deviceId){
    const stream = await navigator.mediaDevices.getUserMedia({
      video: {
        deviceId,
        width: 1280,
        height: 720,
      }
    })
    // Call an API provided by the SDK to change the input stream. The SDK will sto
    // After the input stream is updated, you don't need to call `getOutput` again.
    sdk.updateInputStream(stream)
  }

  initDom()
```

## Pausing and Resuming Detection

You can call `disable` and `enable` to manually pause and resume detection. Pausing detection can reduce CPU usage.



```
<button id="disable">Disable detection</button>  
<button id="enable">Enable detection</button>
```



```
// Disable detection and output the original stream
disableButton.onClick = () => {
  sdk.disable()
}
// Enable detection and output a processed stream
enableButton.onClick = () => {
  sdk.enable()
}
```

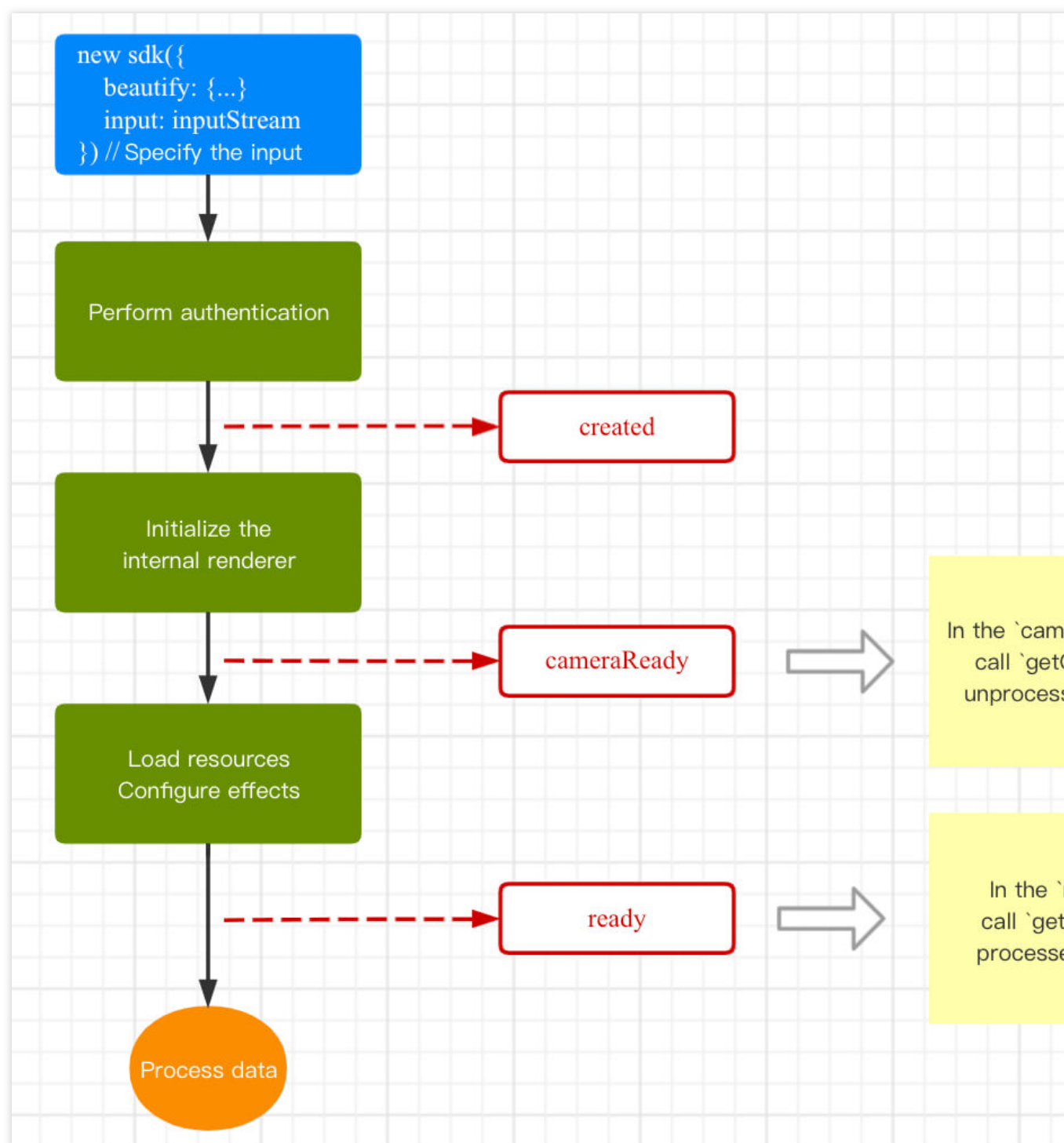
# Loading Optimization

Last updated : 2024-05-08 16:30:10

## Regular Mode

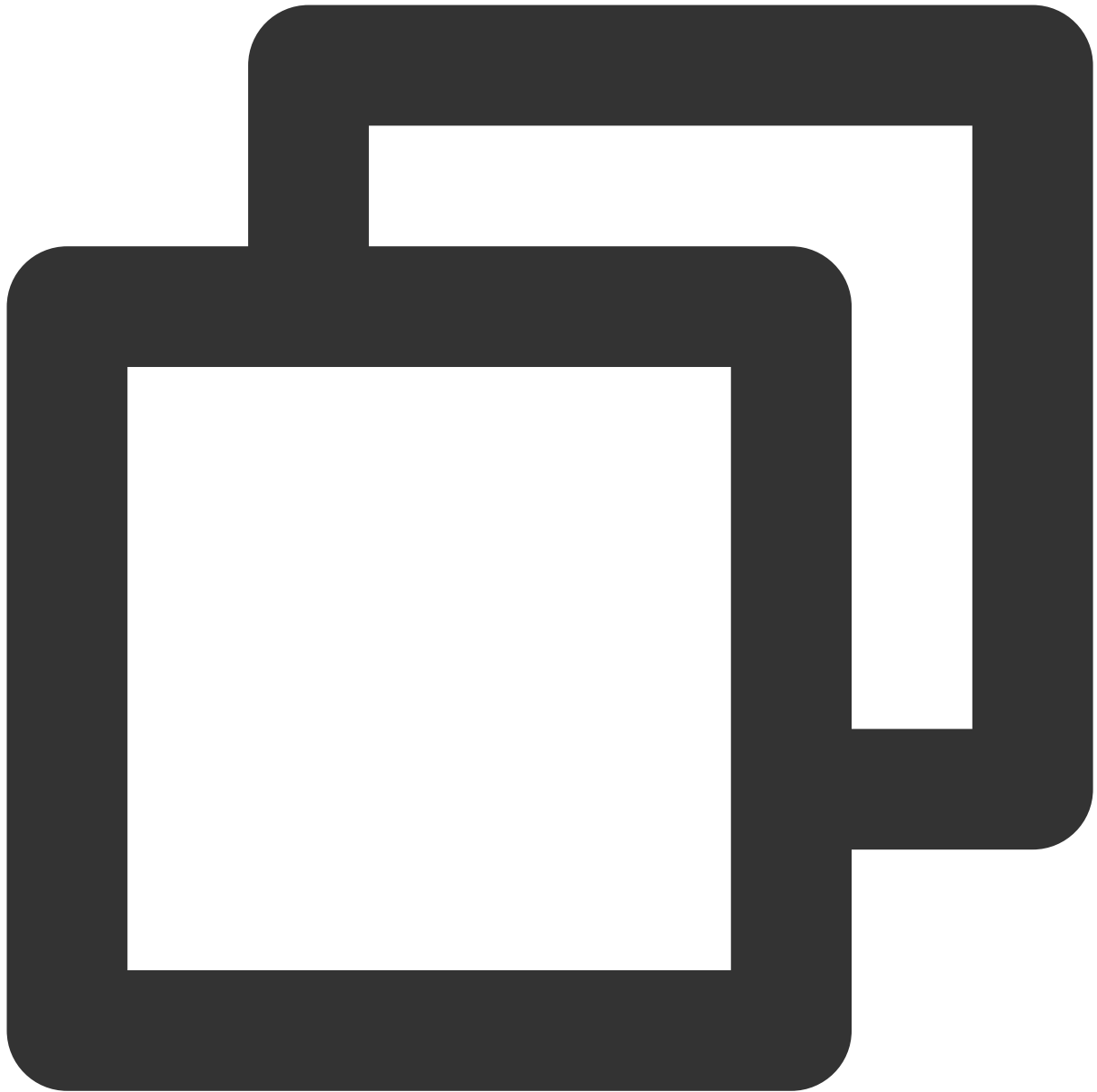
In regular mode, if you call `getOutput` to get the output stream in the `cameraReady` callback, because the effects are not working yet at this point, the stream obtained will be the same as the input stream fed into the SDK. After the effects start to work, the SDK will apply them to the output stream automatically. You don't need to call `getOutput` again. You can use this method if you want to display a video image as soon as possible but do not need to apply effects to the video the moment it is played.

In contrast, if you call `getOutput` in the `ready` callback, the output stream obtained will have been processed. Because the `ready` callback occurs later than `cameraReady`, you can call `getOutput` in the `ready` callback if you want to apply effects to the video the moment it is displayed, but do not expect the video to be played as soon as possible.



Sample code:





```
// Get the authentication information
const authData = {
  licenseKey: 'xxxxxxxxxx',
  appId: 'xxx',
  authFunc: authFunc // For details, see "Configuring and Using a License - Signa
};

// When initializing the SDK in regular mode, pass in the input or camera parameter
const config = {
  auth: authData, // The authentication information
  beautify: { // The effect parameters
```

```
        whiten: 0.1,
        dermabrasion: 0.5,
        lift: 0,
        shave: 0,
        eye: 0.2,
        chin: 0,
    },
    input: inputStream // Prepare the stream data fed into the SDK as the input. For
}
const sdk = new ArSdk(
    // Pass in a config object to initialize the SDK
    config
)
// After authentication succeeds, the SDK will trigger the `created` callback immediately
sdk.on('created', () => {
    // Pull and display the filter and effect list in the `created` callback
    sdk.getEffectList({
        Type: 'Preset',
        Label: 'Makeup',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})

// The data you get by calling `getOutput` in different callbacks vary slightly. Check
sdk.on('cameraReady', async () => {
    const output = await sdk.getOutput() // The effect parameters have not taken effect
    // Play the stream
    ...
})
sdk.on('ready', async () => {
    const output = await sdk.getOutput() // The effect parameters have taken effect
    // Play the stream
    ...
    // Call `setBeautify`, `setEffect`, or `setFilter` in the `ready` callback
})
```

## Pre-Initialization

When the SDK is loaded for the first time, it needs to download static resources in order to initialize the detection module. As a result, the loading of the SDK is affected by network conditions. Given that in some scenarios, you may want to display a video image as soon as possible, we offer a pre-initialization plan that loads static resources in advance.

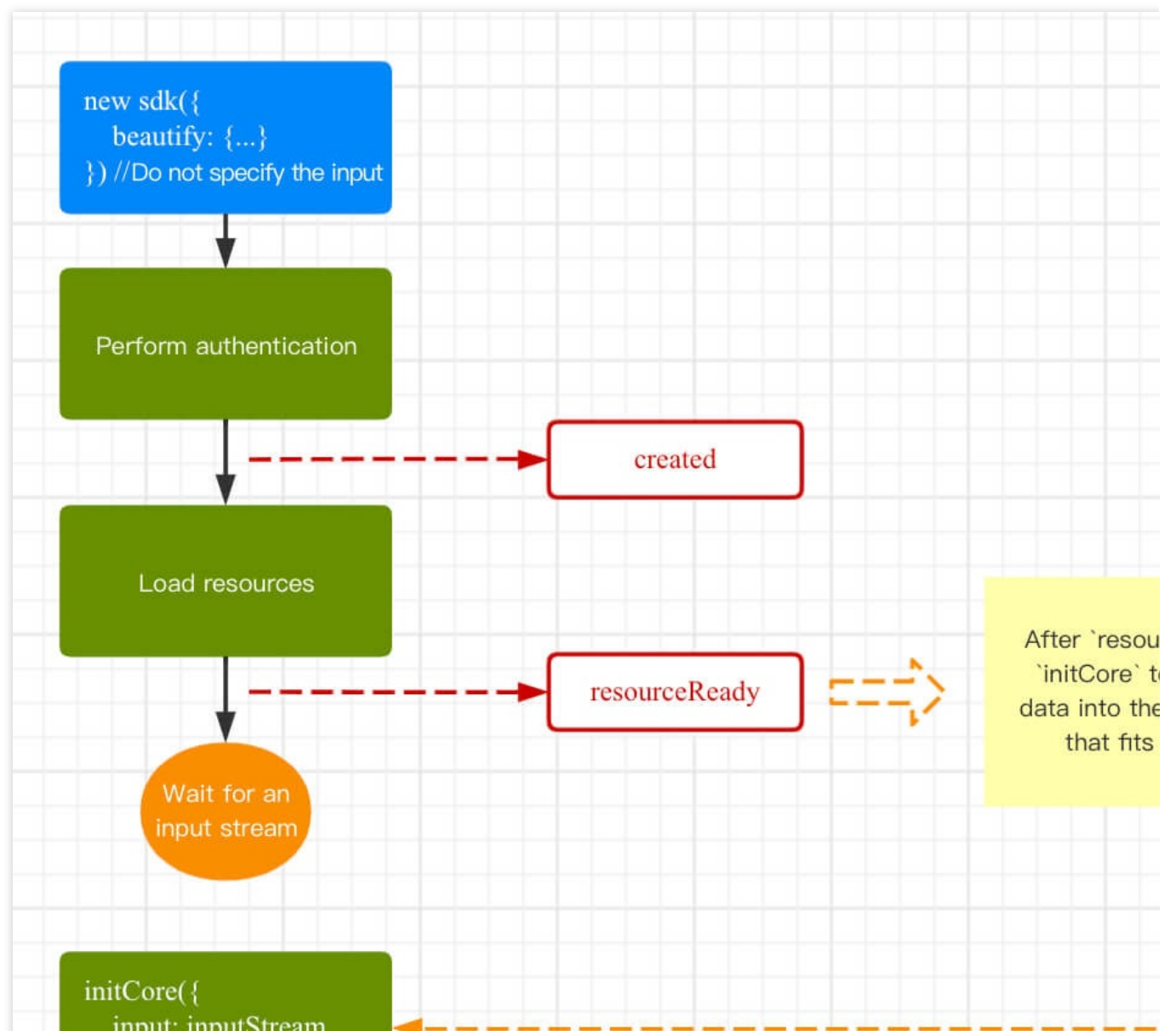
### Use cases for pre-initialization

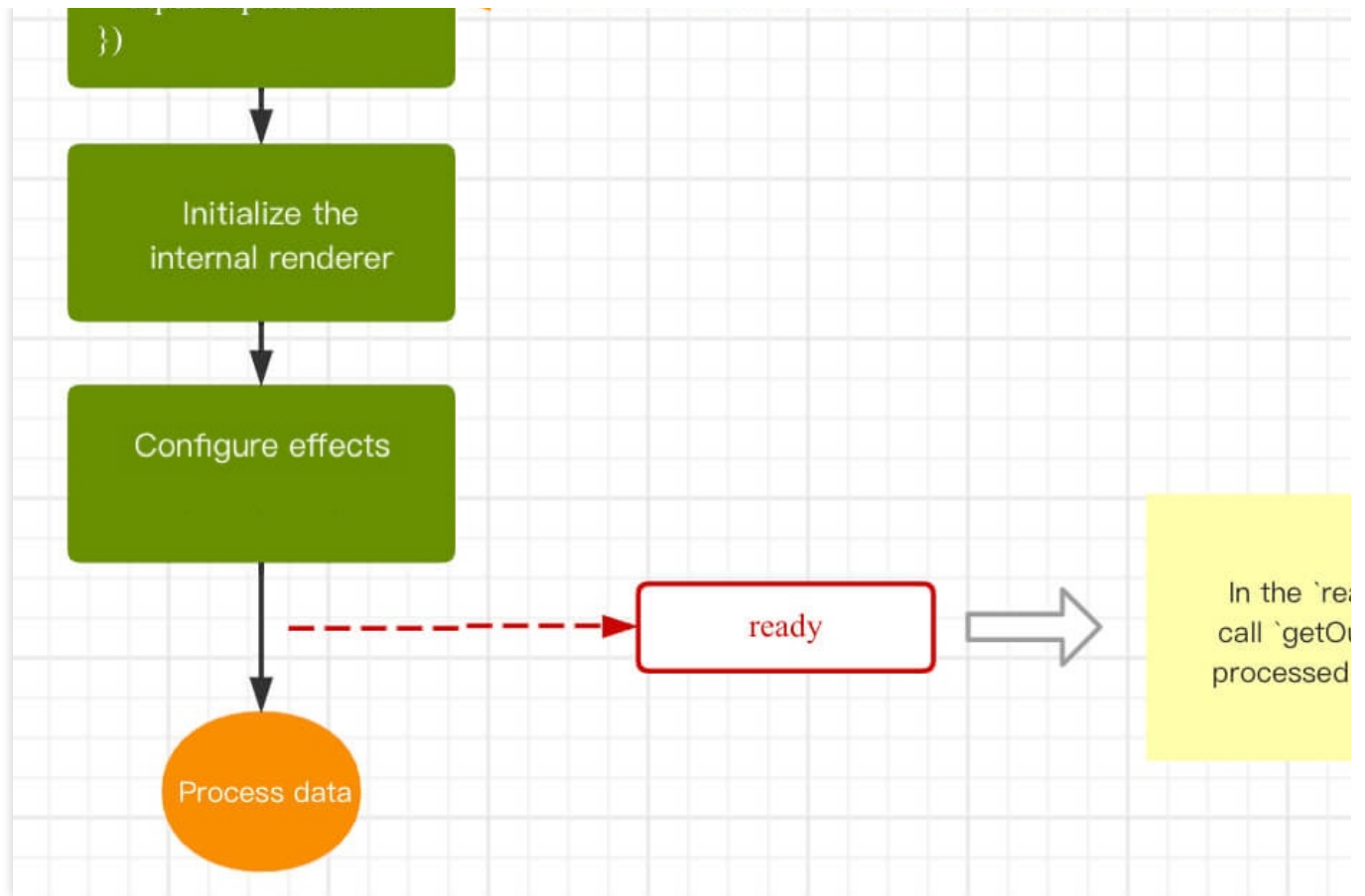
Case 1: The effect SDK is not needed when the webpage is initialized. A video is displayed only after the user performs certain operation.

Case 2: Effects are needed for page B, and page B is directed from page A.

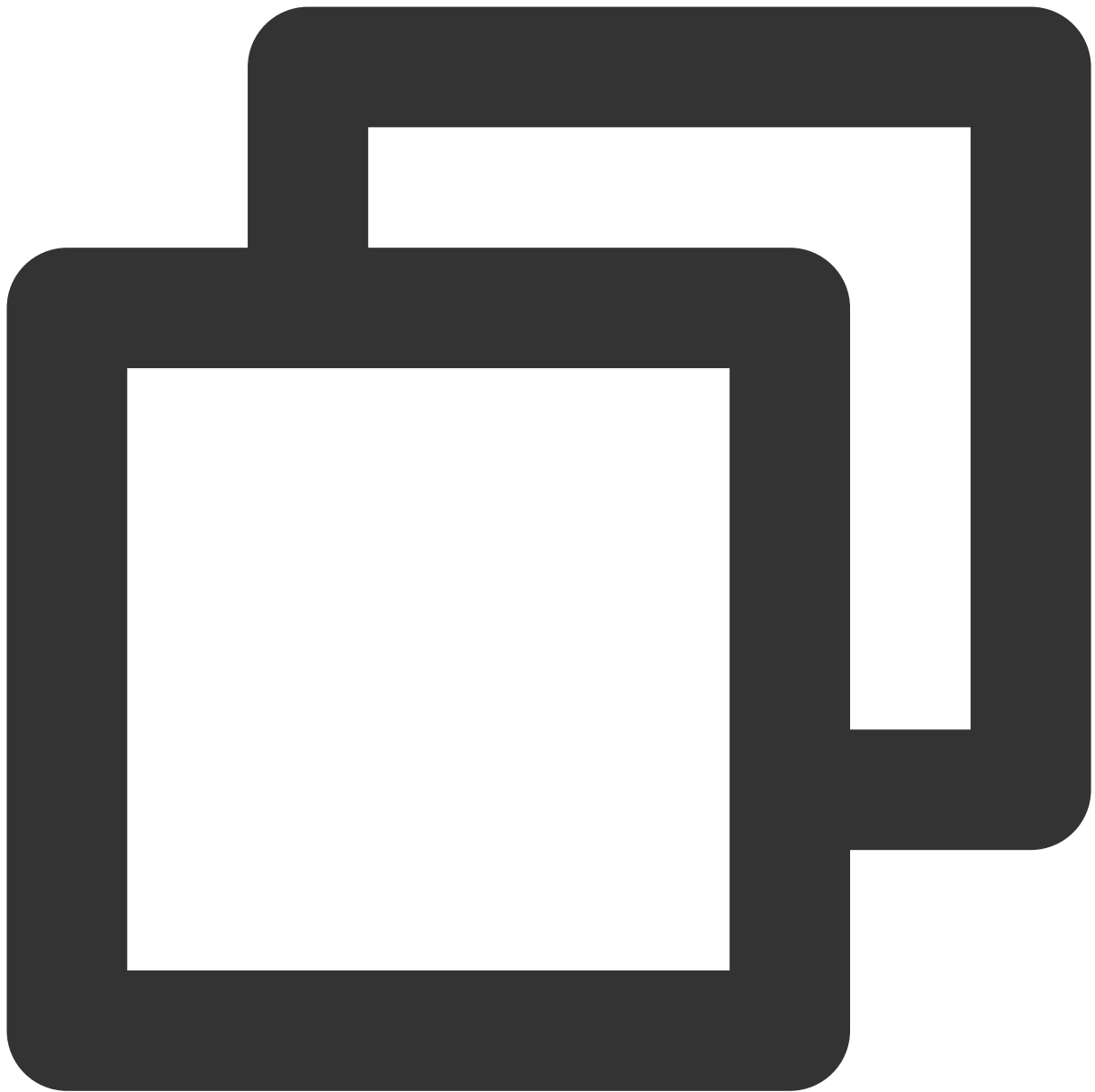
In such cases, you can load resources in advance (as early as possible), feed an input stream to the SDK when necessary, and then get a processed output stream.

For example, in case 1, you can load resources when the webpage is initialized; in case 2, you can get an SDK instance on page A and pass it to page B.

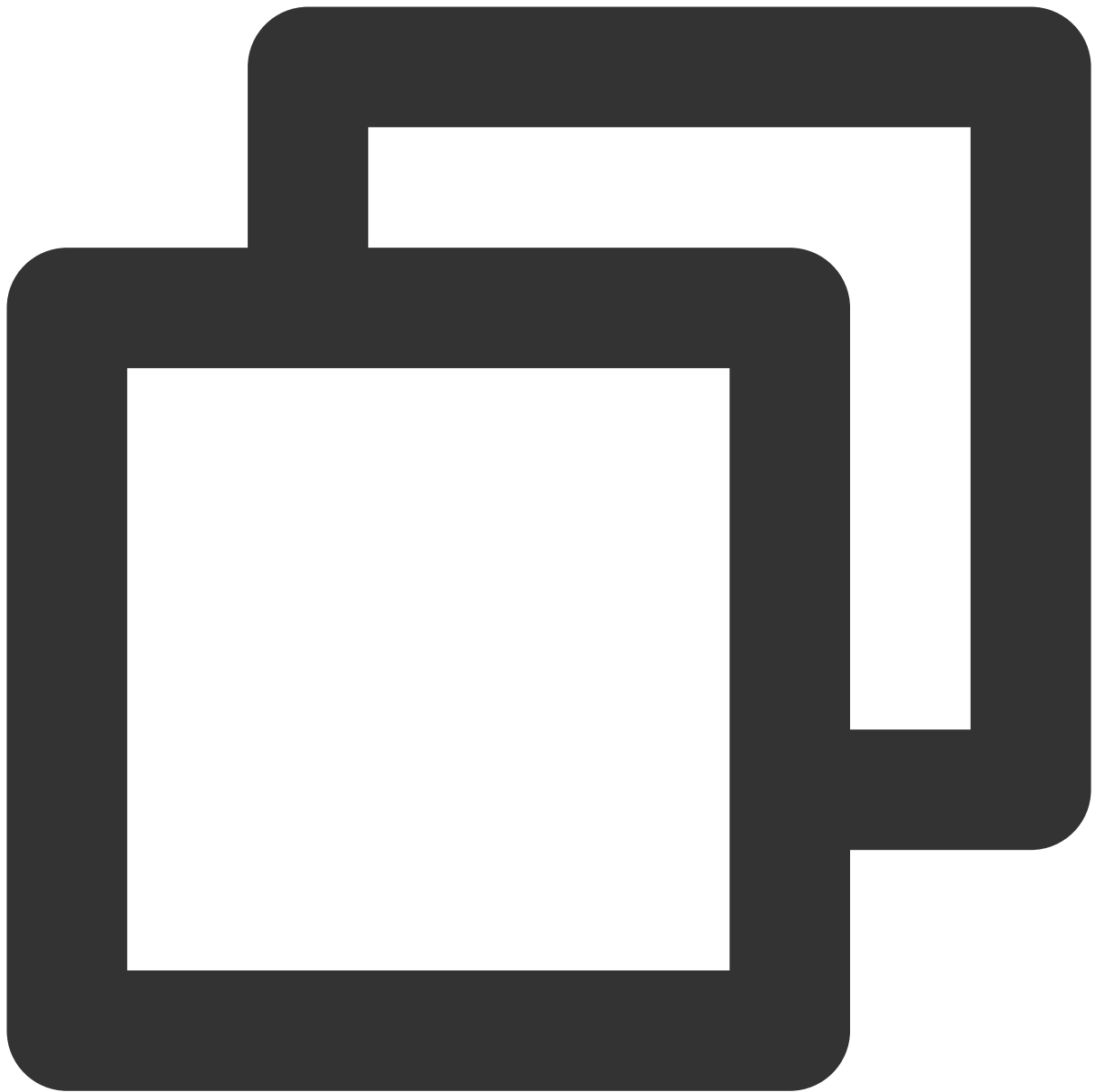




The code below works for case 1



```
<button id="start">Enable the camera</button>
```



```
// Get the authentication information
const authData = {
  licenseKey: 'xxxxxxxxxx',
  appId: 'xxx',
  authFunc: authFunc // For details, see "Configuring and Using a License - Signa
};

// Do not pass in the input or camera parameters when initializing the SDK. After a
const config = {
  auth: authData, // The authentication information
  beautify: { // The effect parameters
```

```
        whiten: 0.1,
        dermabrasion: 0.5,
        lift: 0,
        shave: 0,
        eye: 0.2,
        chin: 0,
    },
}
const sdk = new ArSdk(
    // Pass in a config object to initialize the SDK
    config
)
// After authentication succeeds, the SDK will trigger the `created` callback immedi
sdk.on('created', () => {
    // Pull and display the filter and effect list in the `created` callback
    sdk.getEffectList({
        Type: 'Preset',
        Label: 'Makeup',
    }).then(res => {
        effectList = res
    });
    sdk.getCommonFilter().then(res => {
        filterList = res
    })
})
// `resourceReady` indicates that the necessary resources are ready. After receivin
sdk.on('resourceReady', () => {
})
// In this mode, the SDK will trigger the `ready` callback only after `initCore` is
sdk.on('ready', async () => {
    const output = await sdk.getOutput() // The effects have been applied.
    // Play the stream
    ...
    // Call `setBeautify`, `setEffect`, or `setFilter` in the `ready` callback
})
// Feed stream data to the SDK when the user turns the camera on
document.getElementById('start').onclick = async function(){
    const devices = await navigator.mediaDevices.enumerateDevices()
    const cameraDevice = devices.find(d=>d.kind === 'videoinput')
    navigator.mediaDevices.getUserMedia({
        audio: false,
        video: {
            deviceId: cameraDevice.deviceId
            ... // Other configuration
        }
    }).then(mediaStream => {
        // In this mode, make sure you call `initCore` after the `resourceReady` ca
```

```
    sdk.initCore({  
      input: mediaStream // Prepare the stream data fed into the SDK as the i  
    })  
  })  
}
```



# Configuring Effects

Last updated : 2024-05-08 16:30:10

The Tencent Effect SDK offers **beautification effects**, **filters**, and **special effects**. For filters and special effects, you need to get the material list first and configure them in the SDK by specifying the effect ID.

## Beautification

You can pass in beautification parameters during initialization. You can also call `setBeautify` of the SDK to set beautification effects.

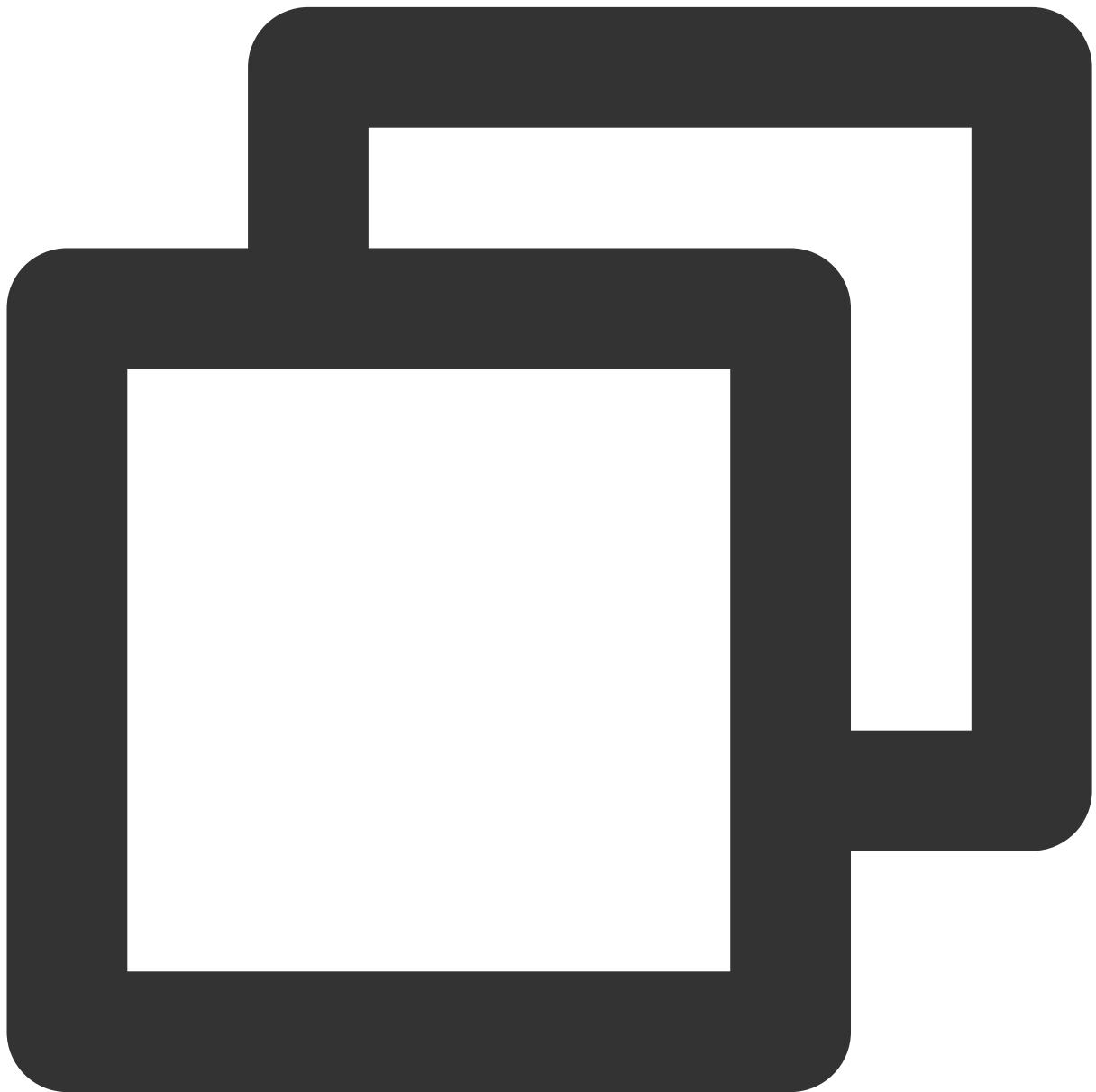
Currently, the SDK supports the following beautification effects:



```
type BeautifyOptions = {
  whiten?: number, // The brightening effect. Value range: 0-1.
  dermabrasion?: number // The smooth skin effect. Value range: 0-1.
  lift?: number // The slim face effect. Value range: 0-1.
  shave?: number // The face width. Value range: 0-1.
  eye?: number // The big eyes effect. Value range: 0-1.
  chin?: number // The chin effect. Value range: 0-1.
  // The following parameters are only available in version 1.0.11 and above
  darkCircle?: number; // The dark circle effect. Value range: 0-1.
  nasolabialFolds?: number; // The nasolabial folds effect. Value range: 0-1.
  cheekbone?: number; // The cheek bone effect. Value range: 0-1.
```

```
head?: number; // The head effect. Value range: 0-1.  
eyeBrightness?: number; // The eye brightness effect. Value range: 0-1.  
lip?: number; // The lip effect. Value range: 0-1.  
forehead?: number; // The forehead effect. Value range: 0-1.  
nose?: number; // The nose effect. Value range: 0-1.  
usm?: number; // The distinct effect. Value range: 0-1.  
}
```

Call `setBeautify` of the SDK to set beautification effects:



```
sdk.setBeautify({
```

```
whiten: 0.2,  
dermabrasion: 0,  
lift: 0.3,  
shave:0.1,  
eye: 0.9,  
chin: 0,  
.....  
})
```

## Filters

Given the relatively high cost of filter design, we offer some built-in filters which you can use directly.

1. Get the built-in filter list:



```
const filterList = await sdk.getCommonFilter()
```

2. Set the filter:



```
sdk.setFilter(filterList[0].EffectId, 0.5)
```

## Special Effects

You can use the SDK's built-in effects or effects you customized in the [RT-Cube console](#). To learn about how to customize effects, please [contact us](#).



```
// Get the built-in effects
// By default, both makeup effects and stickers are returned. You can also use the
const presetEffectList = await sdk.getEffectList({
  Type: 'Preset'
  // Label: ['Sticker'] (Return only stickers)
})

// Get the custom effects
const customEffectList = await sdk.getEffectList({
  Type: 'Custom'
})
```

```
// Pass in the request parameters of `getEffectList`
const lipList = await sdk.getEffectList({
  PageNumber: 0,
  PageSize: 10,
  Name: '',
  Label: ['Lip makeup'], // Specify the specific type of materials to return
  Type: 'Custom'
})
const eyeList = await sdk.getEffectList({
  PageNumber: 0,
  PageSize: 10,
  Name: '',
  Label: ['Eye makeup'], // Specify the category of resources to return
  Type: 'Custom'
})
// Use an effect
sdk.setEffect([lipList[0].EffectId, eyeList[0].EffectId])

// Specify the effect to use and the strength of the effect
sdk.setEffect([
  {
    id: lipList[0].EffectId,
    intensity: 0.5
  },
  {
    id: eyeList[0].EffectId,
    intensity: 0.7
  }
])
// Set only the filter strength
sdk.setEffect([
  {
    id: lipList[0].EffectId,
    intensity: 0.5,
    filterIntensity: 0
  },
  {
    id: eyeList[0].EffectId,
    intensity: 0.7,
    filterIntensity: 1
  }
])
```

**Note:**

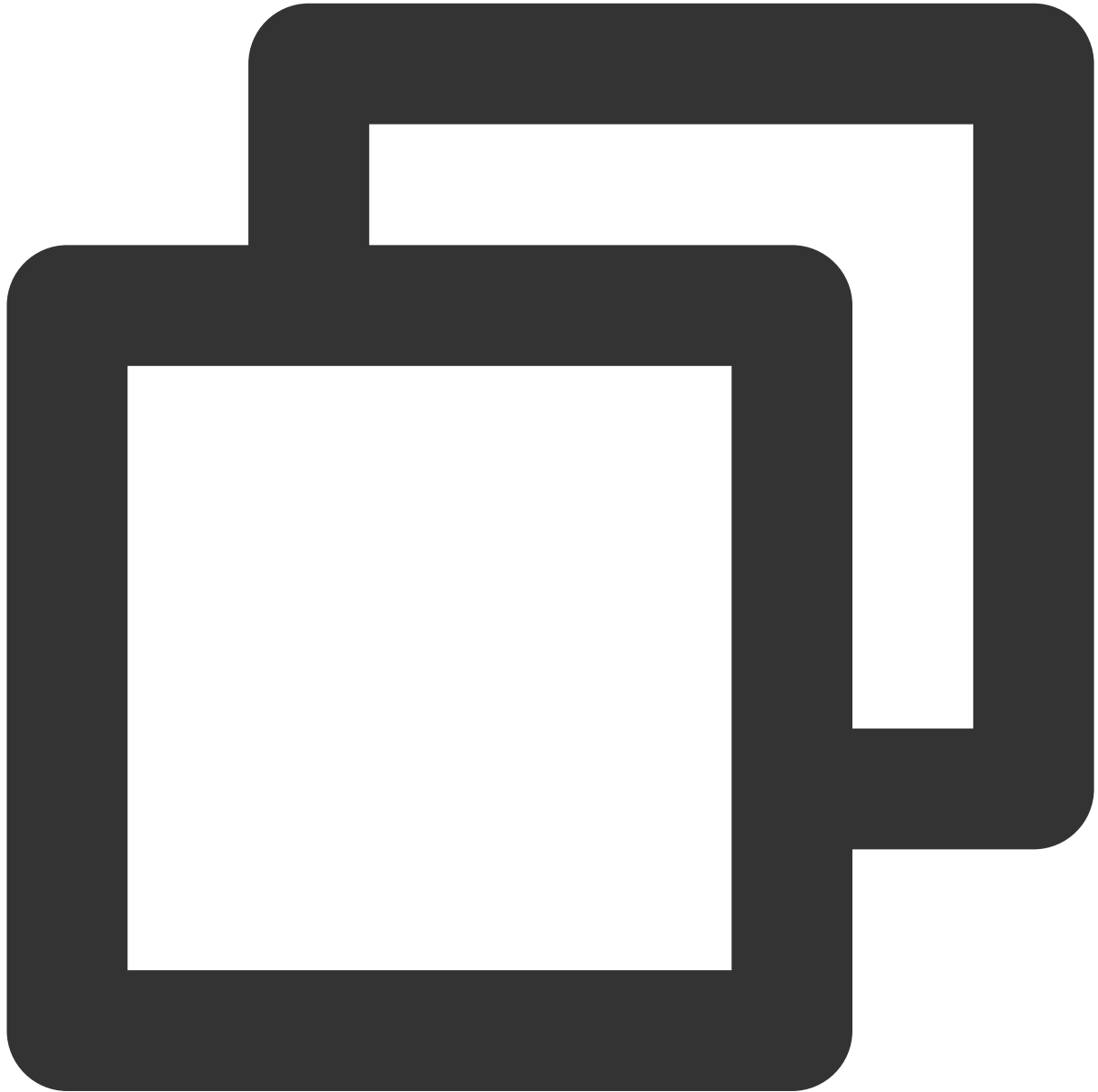
When customizing a special effect, you can add a filter to the effect. In such cases, you can use

`filterIntensity` to adjust the strength of the filter separately.



## Disabling Detection

The performance overhead of AI detection is high. To reduce resource consumption, the SDK will automatically disable AI detection when no effects are used and enable AI detection again when effects are applied.



```
// Clear all effect settings to disable AI detection
sdk.setBeautify({
  whiten: 0,
  dermabrasion: 0,
  lift: 0,
```

```
    shave:0,  
    eye: 0,  
    chin: 0,  
    ..... // reset all parameters to 0  
  })  
  sdk.setEffect('')
```

# Configuring Segmentation

Last updated : 2024-05-08 16:34:16

The segmentation module needs to be enabled during initialization to implement virtual backgrounds. For more information, see [Custom Stream or Image](#) and [Built-in Camera](#).

## Setting the Background

The SDK allows you to blur the background or set an image as the background. You can pass in keying parameters during initialization.



```
const config = {
  module: {
    beautify: true, // Whether to enable the effect module, which offers beautifi
    segmentation: true // Whether to enable the segmentation module, which allows
    segmentationLevel: 0 // Switch background segmentation models supported since
  },
  auth: authData, // The authentication information
  input: stream, // The input stream
  beautify: { // The effect parameters for initialization (optional)
    whiten: 0.1,
    dermabrasion: 0.3,
```

```
    eye: 0.2,  
    chin: 0,  
    lift: 0.1,  
    shave: 0.2  
  },  
  background: {  
    type: 'blur' // Blur the background  
  }  
}  
const sdk = new ArSdk(  
  // Pass in a config object to initialize the SDK  
  config  
)
```

You can also dynamically change the background.



```
sdk.setBackground({  
  type: 'image', // The background image  
  src: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg'  
})
```

## Using Transparent Backgrounds

The SDK supports transparent backgrounds on some browsers.



```
sdk.setBackground({  
  type: 'transparent'  
})
```

**Note:**

Keep the following in mind:

Segmentation is supported on both mobile and desktop browsers.

Because WebRTC does not support alpha channels, you can only use transparent backgrounds locally. Background transparency will not work after publishing.

Background transparency is supported on desktop Chrome and Firefox, but not supported on desktop or mobile Safari.

Starting from version 1.0.19 and above, switching background segmentation models is supported, with parameters: 0, 1, 2.

Level 0 has the best performance but relatively average segmentation results.

Level 1 has a moderate performance and effect.

Level 2 has the best segmentation results and the longest inference time, suitable for applications with high requirements for segmentation effects and low performance requirements.



# Using Animojis and Virtual Avatars

Last updated : 2024-05-08 16:34:16

The Tencent Effect SDK supports animojis and VR virtual avatars starting from v0.3.0.

## Checking Support

Animojis and VR virtual avatars rely on a WebGL2 environment. The SDK offers a static method for you to check whether a browser supports the capability.



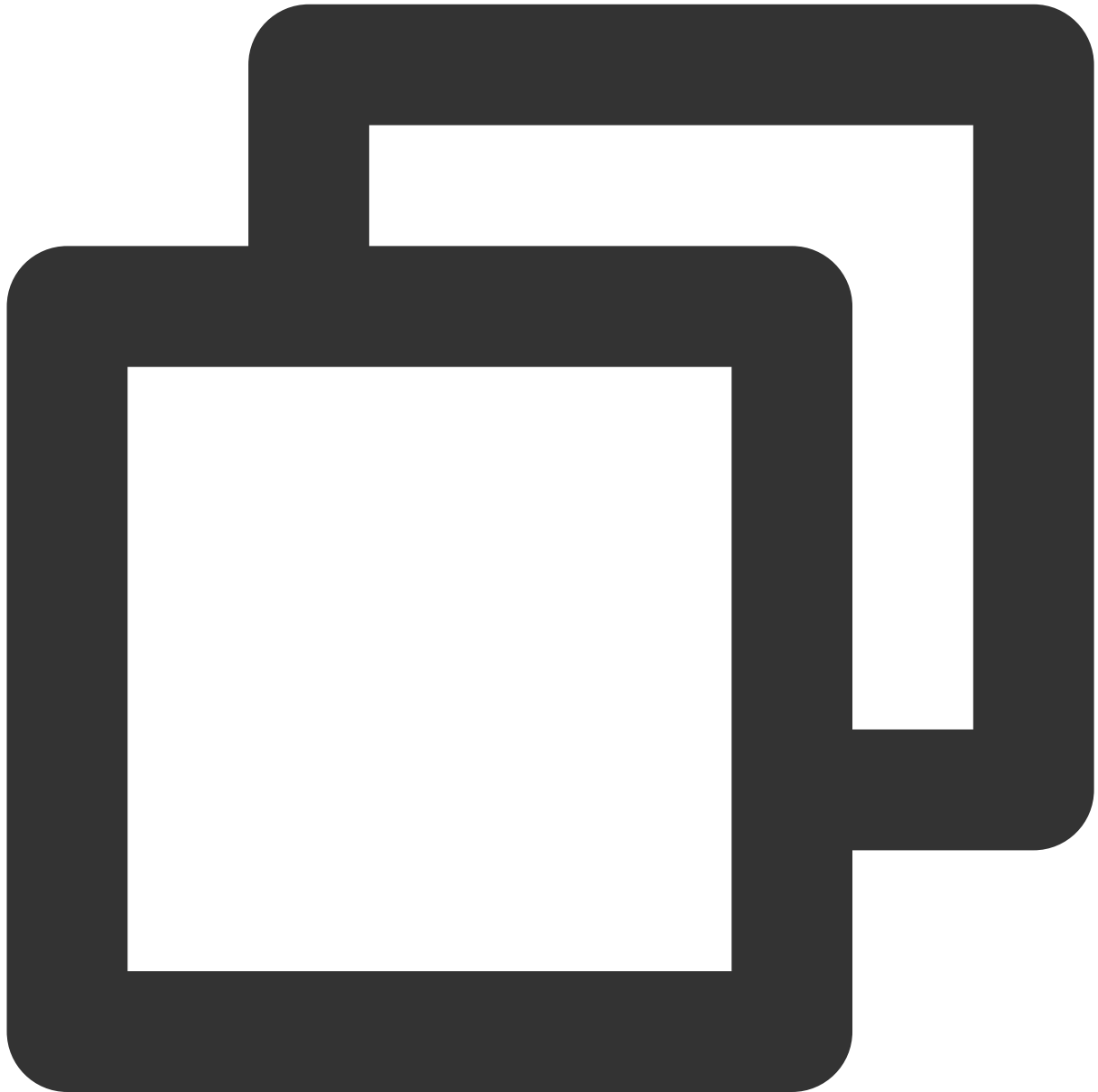
```
import {ArSdk} from 'tencentcloud-webar'
if (ArSdk.isAvatarSupported()) {
  // Initialize the feature

} else {
  alert('This browser does not support virtual avatars')
  // Hide the feature
}
```

# Animojis

## Getting models

After initialization, you can get the built-in models. Currently, the SDK offers four built-in animoji models.



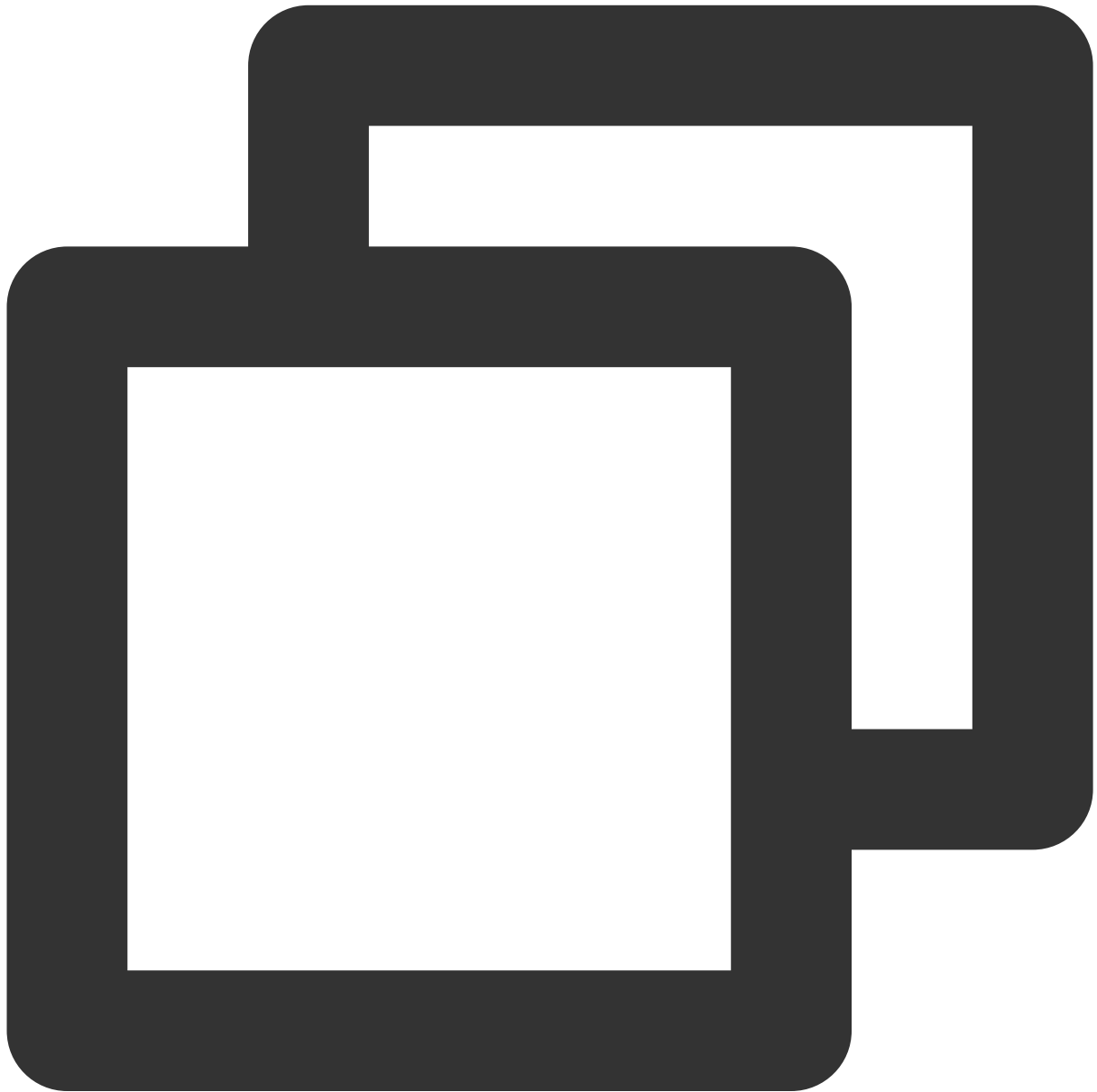
```
const avatarARList = await sdk.getAvatarList('AR')
```

### Note:

Configuring animojis and virtual avatars will automatically remove other effects such as makeup and stickers, and vice versa.

## Setting a model

After you get the list of built-in models, you can select one by specifying the `EffectId` parameter.



```
ar.setAvatar({
  mode: 'AR', // Set the mode to `VR`
  effectId: avatarARList[0].EffectId// Pass in the built-in ID
}, () => {
```

```
// success callback  
  
});
```

## Customizing a model

If you need to customize a model, feel free to [contact us](#).

# VR Virtual Avatars

## Getting models

The list of built-in models can be obtained after the SDK is initialized. Currently, the SDK offers 10 virtual avatars.



```
const avatarVRList = await sdk.getAvatarList('VR')
```

## Setting a scene



```
ar.setAvatar({
  mode: 'VR', // Set `mode` to `VR`
  effectId: avatarVRList[0].EffectId, // Pass in the built-in ID
  backgroundUrl: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg',
}, () => {
  // success callback
});
```

**Note:**

To set a VR scene, you need to set the background image URL, or the black background will be used by default.

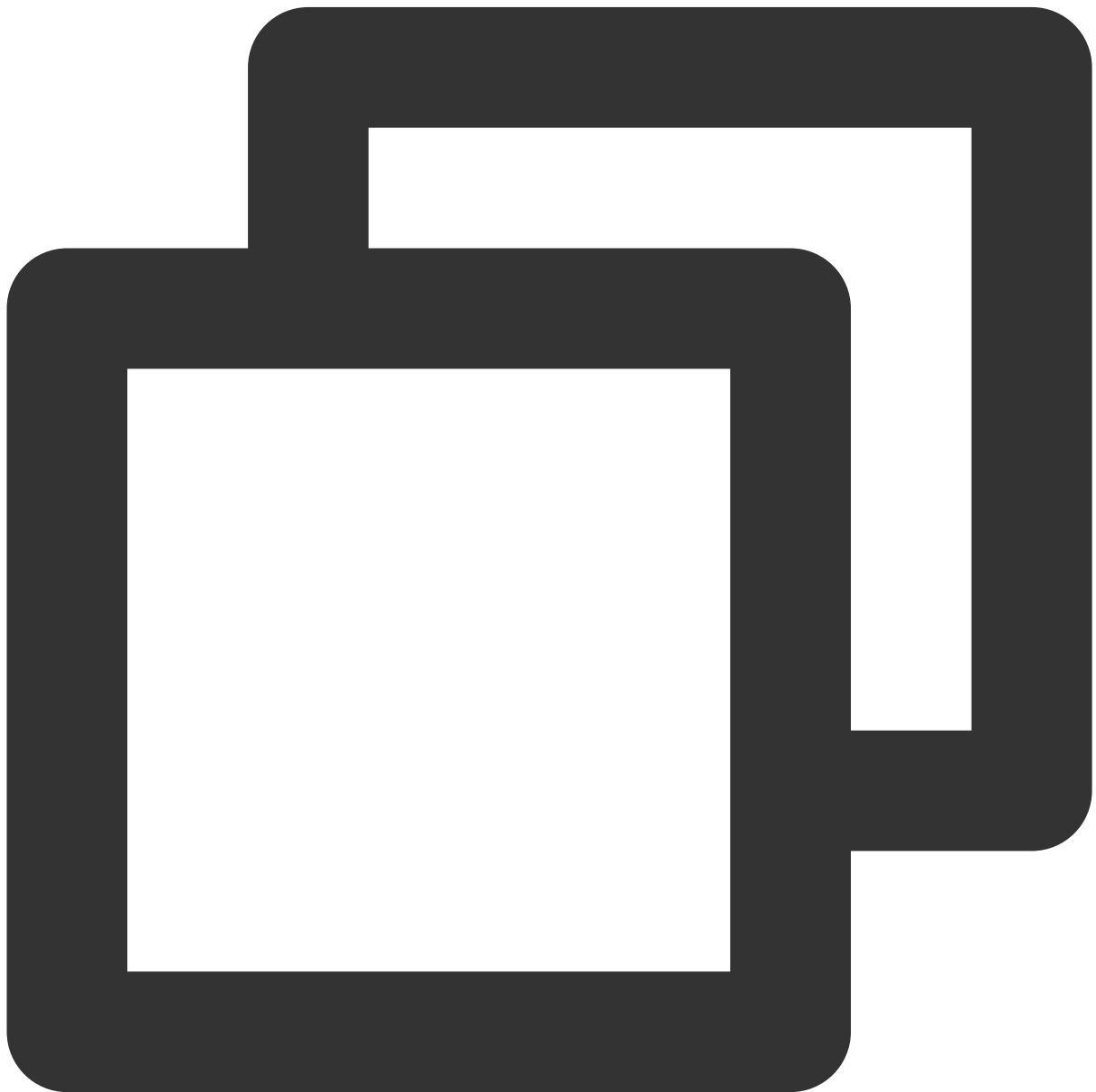
**Customizing a model**

You can quickly customize a virtual avatar in two ways and directly use it in the SDK.

Option 1. `readyplayer.me`

Option 2. [Vroid](#)

With either option, you need to upload the exported model to CDN and use the URL to set the SDK.



```
ar.setAvatar({
```



```
mode: 'VR', // Set `mode` to `VR`  
url: 'https://xxxx.glb', // Pass in the built-in ID  
backgroundUrl: 'https://webar-static.tencent-cloud.com/assets/background/1.jpg',  
, () => {  
    // success callback  
  
});
```

Currently, a custom model can be either in GLB or VRM format.

# Integration into Mini Program

Last updated : 2023-04-21 15:36:13

For how to develop a Mini Program, see [Weixin Mini Program documentation](#).

## Integrating the SDK

### Step 1. Configure a domain allowlist

As the SDK will request the backend to perform authentication and load resources, you need to configure a domain allowlist on the Mini Program backend.

1. Open the [Mini Program backend](#) and go to **Development > Development Management > Development Settings > Server Domain Name**.

2. Click **Modify**, configure the following domain names and save them.

Request domain names:



```
https://webar.qcloud.com;  
https://webar-static.tencent-cloud.com;  
https://aegis.qq.com;  
The URL of the authentication signature API (`get-ar-sign`)
```

downloadFile domain name:



`https://webar-static.tencent-cloud.com`

## Step 2. Install and build the npm package

### 1. Install:



```
npm install tencentcloud-webar
```

2. Open Weixin DevTools and select **Tools > Build npm** on the topbar.
3. Configure the path of `workers` in `app.json` :



```
"workers": "miniprogram_npm/tencentcloud-webar/worker"
```

### Step 3. Import the files

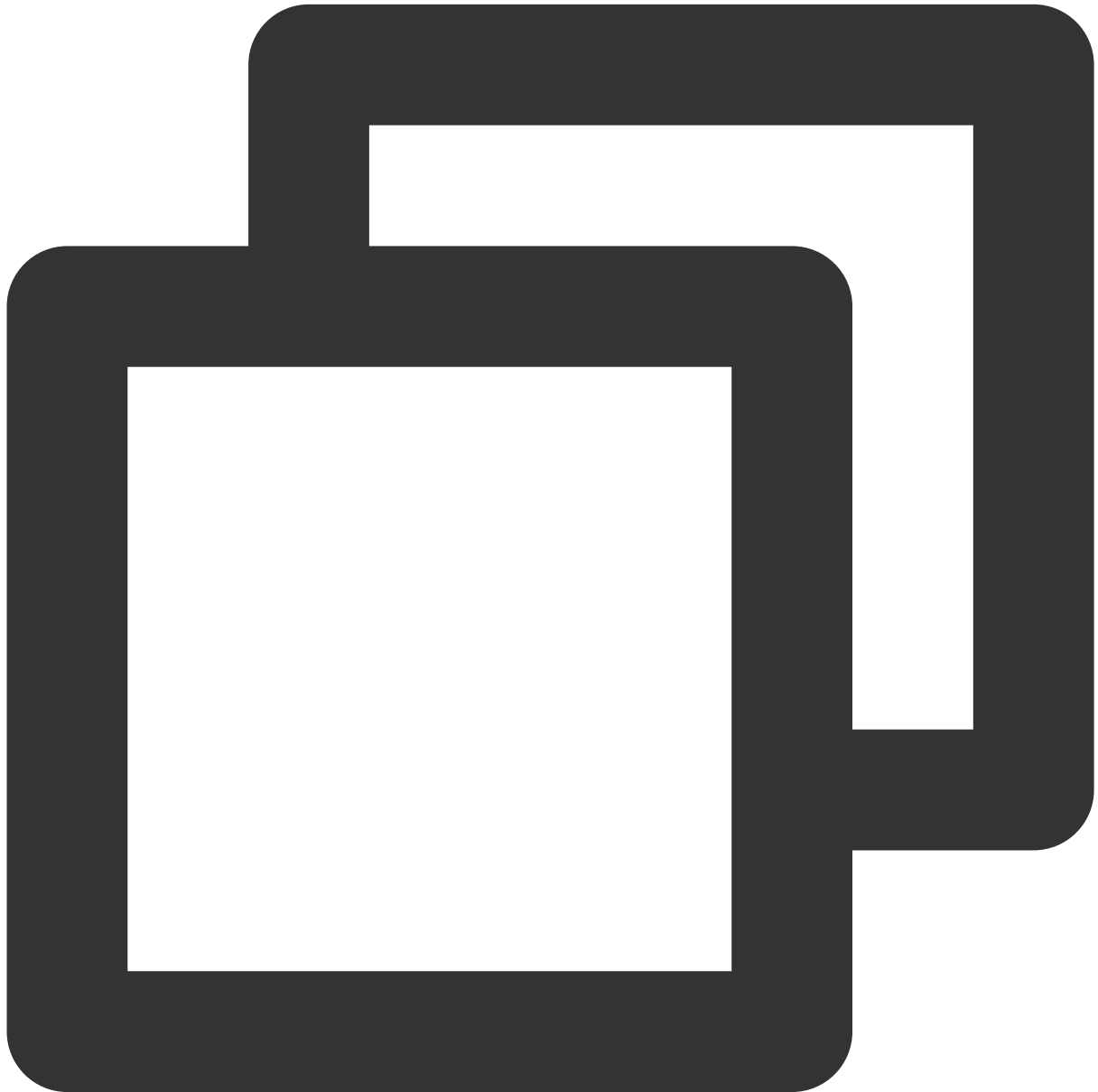


```
// The import method for versions earlier than 0.3.0 (one file)
// import "../../miniprogram_npm/tencentcloud-webar/lib.js";
// The import method for v0.3.0 or later (two files and the 3D module (optional))
import '../../miniprogram_npm/tencentcloud-webar/lib.js';
import '../../miniprogram_npm/tencentcloud-webar/core.js';
// Initialize the 3D plugin (optional)
import '../../miniprogram_npm/tencentcloud-webar/lib-3d.js';
import { plugin3d } from '../../miniprogram_npm/tencentcloud-webar/plugin-3d'
// Import `ArSdk`
import { ArSdk } from "../../miniprogram_npm/tencentcloud-webar/index.js";
```

**Note**

Because Mini Program has a 500 KB limit for file size, the SDK is provided as multiple JS files.

Starting from v0.3.0, the SDK is further split to support 3D. The 3D module can be loaded as needed. Before import, check your SDK version and use the corresponding import method.

**Step 4. Initialize an instance**

```
// wxml
//Open the camera
<camera
```



```

    device-position="{{'front'}}"
    frame-size="large" flash="off" resolution="medium"
    style="width: 750rpx; height: 134rpx;position:absolute;top:-9999px;"
  />
  // The SDK outputs the processed image to the canvas in real time.
  <canvas
    type="webgl"
    canvas-id="main-canvas"
    id="main-canvas"
    style="width: 750rpx; height: 1334rpx;">
  </canvas>
  // Take a photo and draw the `ImageData` object onto the canvas
  <canvas
    type="2d"
    canvas-id="photo-canvas"
    id="photo-canvas"
    style="position:absolute;width:720px;height:1280px;top:-9999px;left:-9999px;">
  </canvas>
  // js
  Component({
    methods: {
      async getCanvasNode(id) {
        return new Promise((resolve) => {
          this.createSelectorQuery()
            .select(`#${id}`)
            .node()
            .exec((res) => {
              const canvasNode = res[0].node;
              resolve(canvasNode);
            });
        });
      },
    },

    // Initialize the camera type
    async initSdkCamera() {
      // Get the onscreen canvas. The SDK will output the processed image to
      const outputCanvas = await this.getCanvasNode("main-canvas");
      const sdk = new ArSdk({
        camera: {
          width:720,
          height:1280,
        },
        output: outputCanvas,
        loading: {
          enable: false,
        },
        auth: {

```

```
        licenseKey: LICENSE_KEY,
        appId: APP_ID,
        authFunc: authFunc
    },
    plugin3d: plugin3d // You can ignore this parameter (only available
});
this.sdk = sdk
sdk.setBeautify({
    whiten: 0.2
});
sdk.on('created', () => {
    // You can add your business logic in this callback. For details, s
})
}
}
})
```

### Note

Before initializing the SDK, you need to configure the Mini Program `APPID` in the RT-Cube console

Unlike web, for Mini Programs, the `input` parameter must be an image URL string.

Camera configuration is the same as that for web. Pass in the camera parameters instead of `input`. Make sure you have already inserted a camera tag in your page.

Mini Program does not support `getOutput`, so you need to pass in an onscreen WebGL canvas. The SDK will output images onto this canvas.

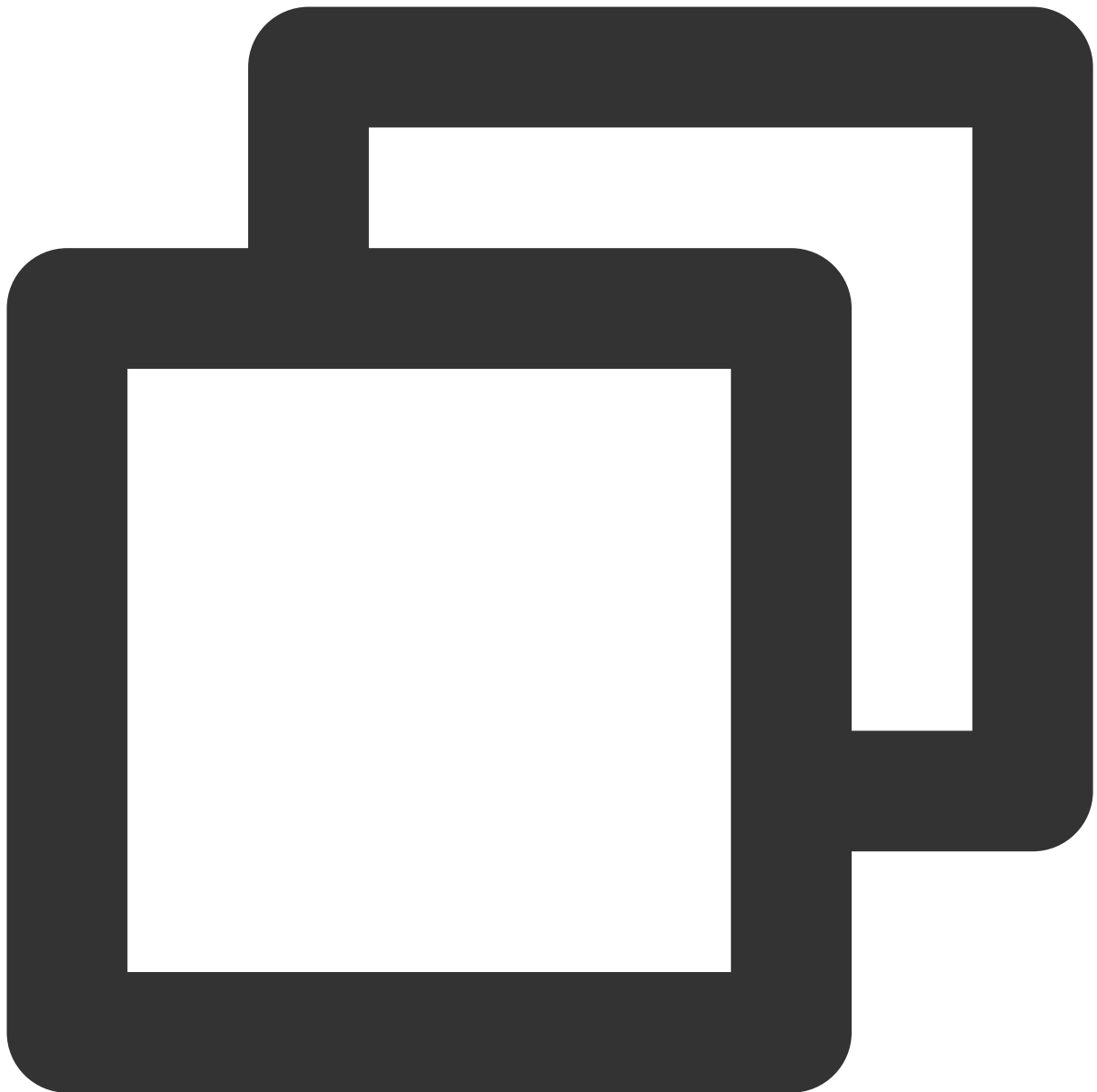
## Photo Taking and Shooting

The SDK works for both photo taking and shooting in a Mini Program.

### Photo

### Shooting

The SDK will return an object containing the width, height, and buffer data. You can draw the data onto the 2D canvas on your page and export it as an image file.



```
Component ({
  ...
  async takePhoto() {
    const {uint8ArrayData, width, height} = this.sdk.takePhoto(); // The `takeP
    const photoCanvasNode = await this.getCanvasNode('photo-canvas');
    photoCanvasNode.width = parseInt(width);
    photoCanvasNode.height = parseInt(height);
    const ctx = photoCanvasNode.getContext('2d');
    // Create an `ImageData` object with the data returned by the SDK
    const imageData = photoCanvasNode.createImageData(uint8ArrayData, width, he
    // Draw the `ImageData` object onto the canvas
```

```
ctx.putImageData(imageData, 0, 0, 0, 0, width, height);  
// Save the canvas as a local image  
wx.canvasToTempFilePath({  
  canvas: photoCanvasNode,  
  x: 0,  
  y: 0,  
  width: width,  
  height: height,  
  destWidth: width,  
  destHeight: height,  
  success: (res) => {  
    // Save the photo  
    wx.saveImageToPhotosAlbum({  
      filePath: res.tempFilePath  
    });  
  }  
})  
}  
})  
})
```

**Note**

When the Mini Program is switched to the background or the screen is locked, call `stopRecord` to stop shooting. If you don't do this, an error may occur.



```
Component({
  methods: {
    // Start shooting
    startRecord() {
      this.sdk.startRecord()
    }
    // Stop shooting
    async stopRecord() {
      const res = await this.sdk.stopRecord();
      // Save the video
      wx.saveVideoToPhotosAlbum({
```

```
        filePath: res.tempFilePath
    })
}
})
```

# Release Notes

Last updated : 2024-05-08 16:30:10

## **SDK Version 1.0.21 @ 2023-12-2**

Optimized performance issues.

Fixed other issues.

## **SDK Version 1.0.19 @ 2023-11-20**

Supported Segmentation level.

Optimized performance issues.

Fixed other issues.

## **SDK Version 1.0.16 @ 2023-10-24**

Optimized performance issues.

Fixed other issues.

## **SDK Version 1.0.11 @ 2023-07-17**

Add hairline, clarity and other beauty parameters.

Optimized performance issues.

Fixed other issues.

## **SDK Version 1.0.0 @ 2023-02-27**

Launched commercial editions.

## **SDK Version 0.3.0 @ 2022-10-14**

Added animojis and virtual avatars for web.

Fixed a memory leak issue.

Fixed other issues.

## **SDK Version 0.2.5 @ 2022-08-01**

Fixed the forced quit of the Weixin browser on Android.

Fixed the bug where the background did not disappear after being disabled.

Fixed other issues.

## **SDK Version 0.2.3 @ 2022-07-20**

Fixed the adaptation failure when the input source was switched.

Added support for starting and stopping face detection.

Fixed other issues.

### **SDK Version 0.2.0 @ 2022-06-29**

Added support for keying on web.

Optimized the integration experience.

### **SDK Version 0.1.18 @ 2022-06-17**

Optimized the built-in camera configuration, improved camera management capabilities, and added the local player.

Added the configuration of delayed initialization.

Optimized the filter strength settings and overlay effects.

### **SDK Version 0.1.12 @ 2022-05-11**

Optimized the SDK performance.

Added the FPS attribute, which can be used to set the rendering frame rate.

### **SDK Version 0.1.9 @ 2022-04-28**

Added built-in stickers.

Fixed the black screen issue which occurred while switching to the background on mobile web.

### **SDK Version 0.1.1 @ 2022-04-13**

Officially released the SDK.



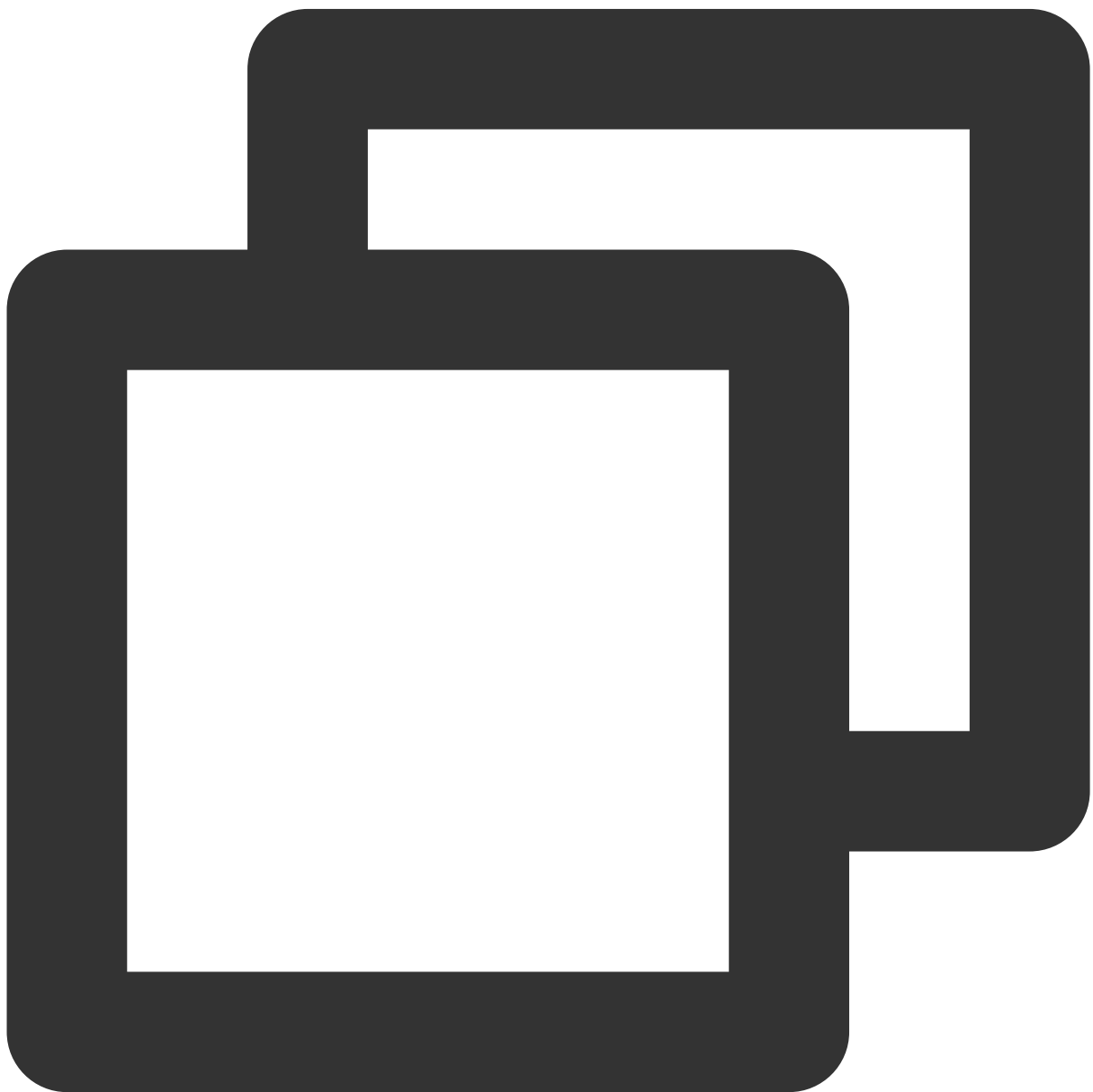
# API Error Codes

Last updated : 2024-05-08 16:34:16

## Built-in API Error Codes

Error codes of APIs such as `https://webar.qcloud.com/sdk/xxx` .

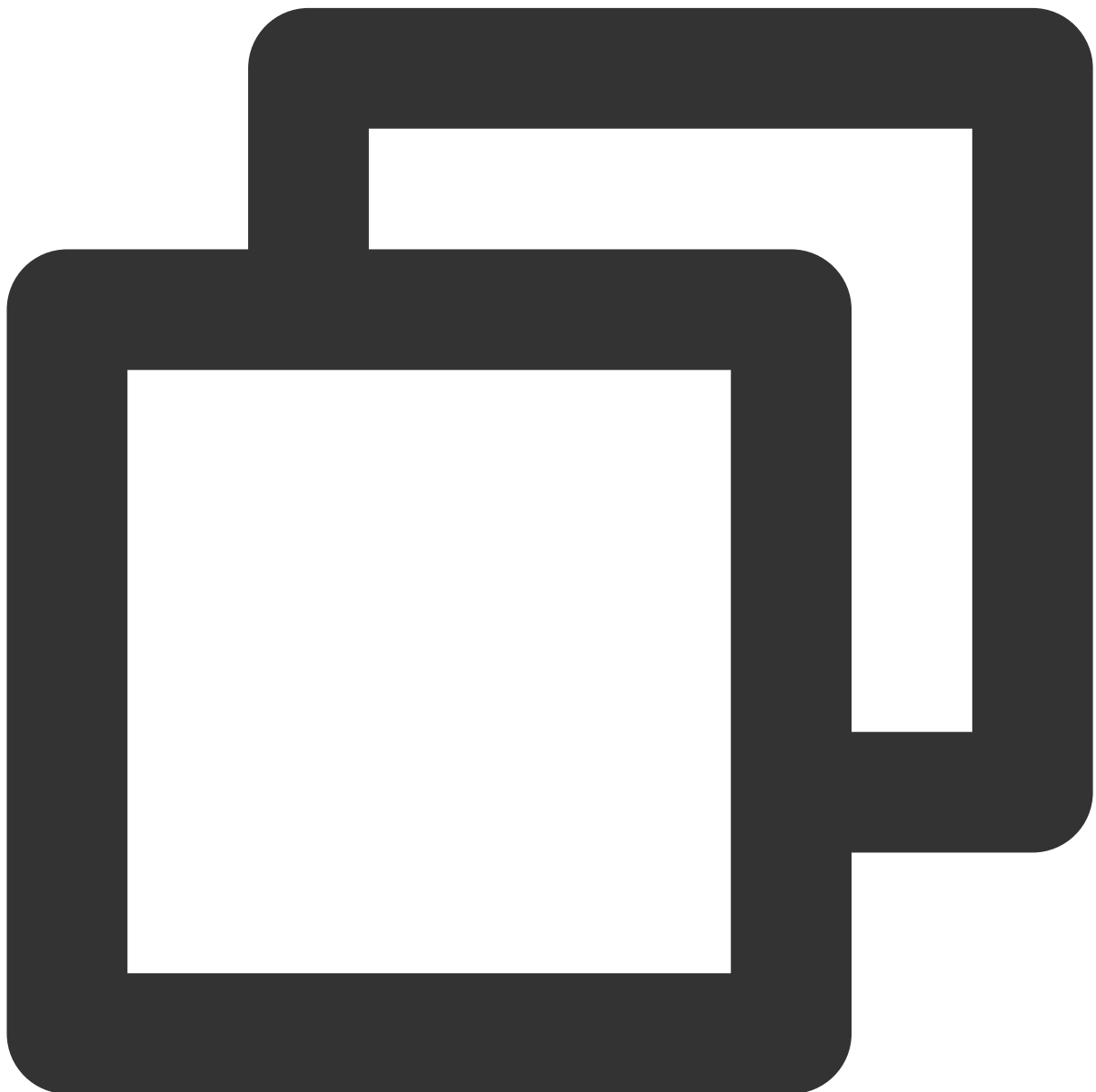
The response structure is as follows:



```
{  
  "Code": xxx,  
  "Message": "xxx"  
}
```

## Successful requests

If `Code` is 0, a request is successful, and information will be returned in `Data` .

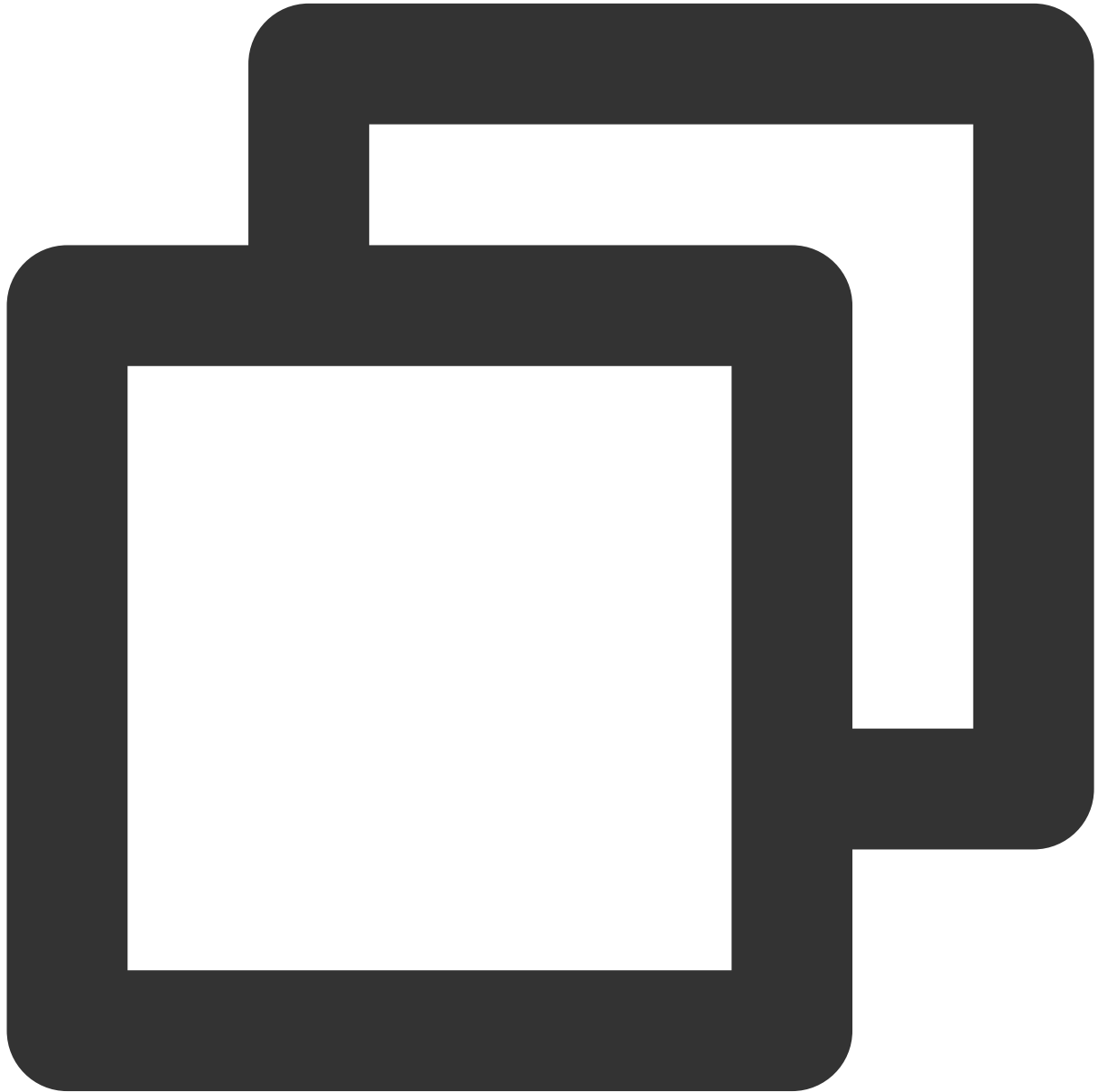


```
{
```

```
Code: 0,  
Data:{...}  
}
```

## Authentication errors

If `Code` is a value from 100 to 104, an authentication error occurred. The response status code is `401`.



```
{  
  "100":{  
    en: "Missing authentication parameter."  
  }  
}
```

```
{,
"101":{
  en: "Signature timeout."
},
"102":{
  en: "Unable to find the user."
},
"103":{
  en: "Signature error."
},
"104":{
  en: "Mismatch of referer or WeChatAppId."
}
}
```

## Request parameter errors

If `Code` is -2, a request parameter error occurred. For details, see the value of `Message` .



```
{
  "Code": -2,
  "Message": "LicenseKey must be a string."
}
```

## Business verification errors

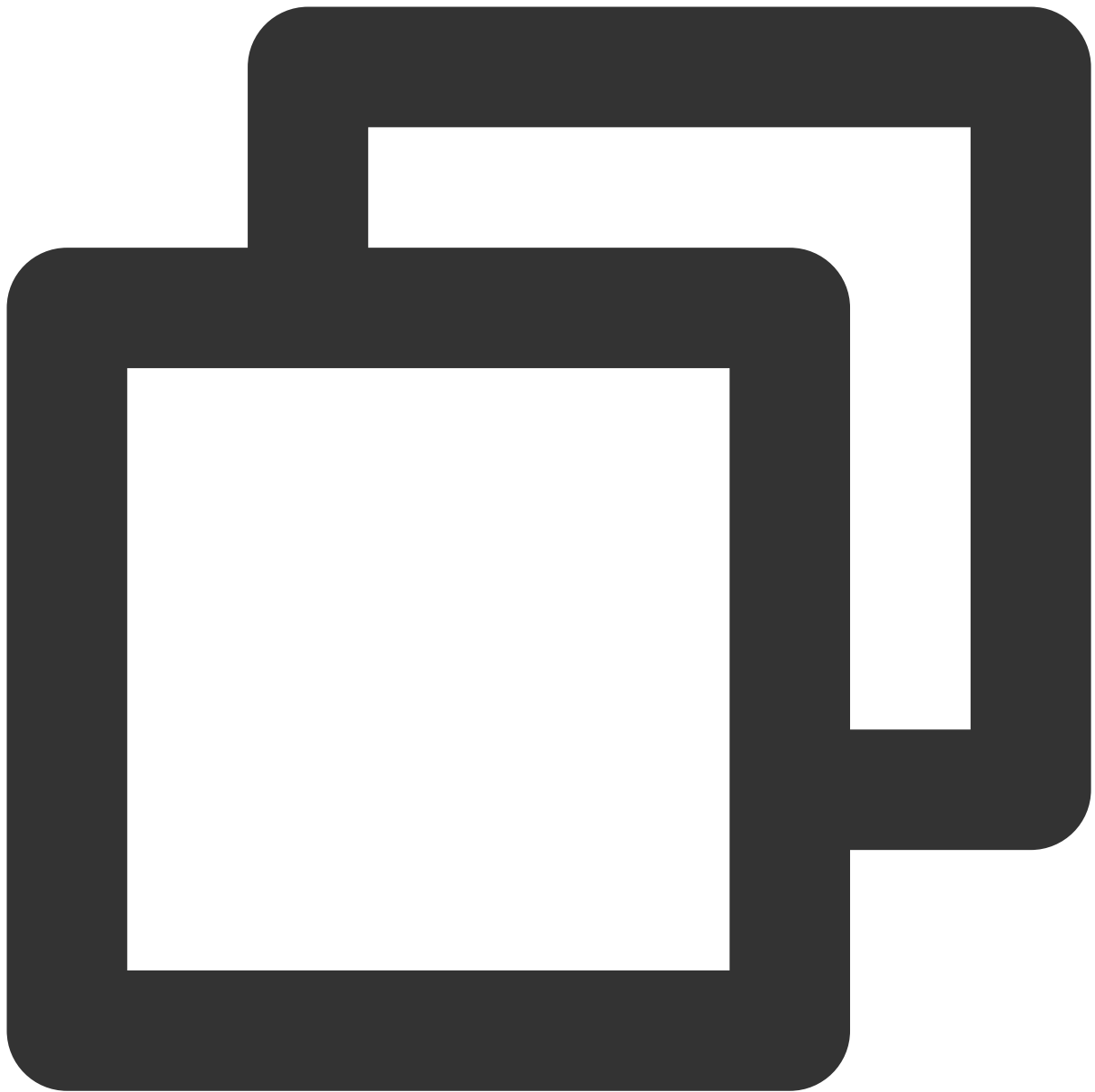
If `Code` is greater than 1000, a business verification error occurred. For details, see the value of `Message` .



```
{  
  "Code": 2007,  
  "Message": "The project does not exist."  
}
```

## Unknown errors

If `Code` is -1, an unknown error occurred.



```
{  
  "Code": -1,  
  "Message": "Unknown error."  
}
```

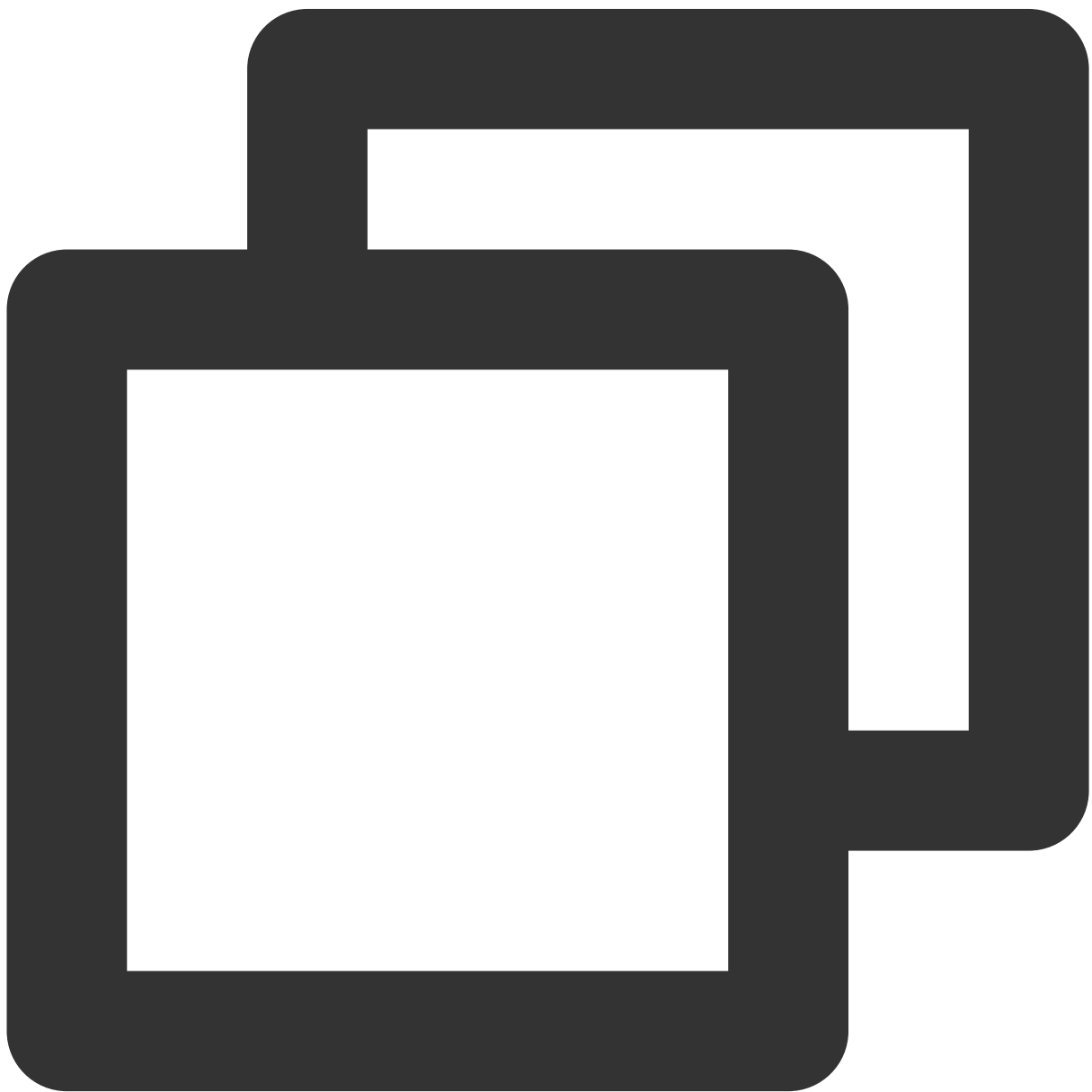
# Parameters and APIs

Last updated : 2024-05-08 18:34:14

This document describes the core parameters and methods of the Beauty AR Web SDK.

**Note:**

The Beauty AR Web SDK relies on hardware acceleration to achieve smooth rendering. The SDK allows you to check whether a browser supports hardware acceleration. You can block the browser if it does not support hardware acceleration.

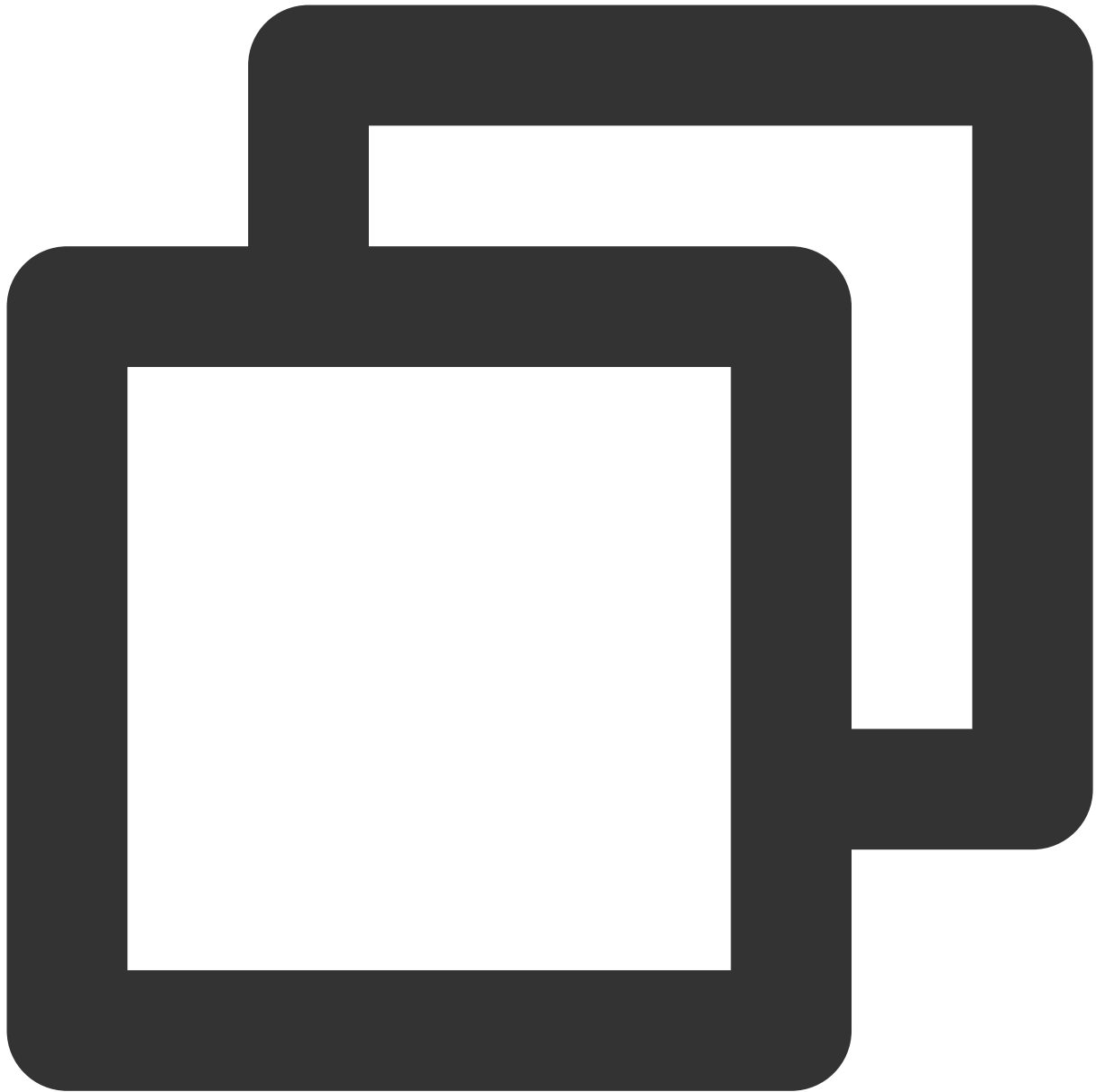




```
import {ArSdk, isWebGLSupported} from 'tencentcloud-webar'

if(isWebGLSupported()) {
  const sdk = new ArSdk({
    ...
  })
} else {
  // The browser blocking logic
}
```

## Initialization Parameters

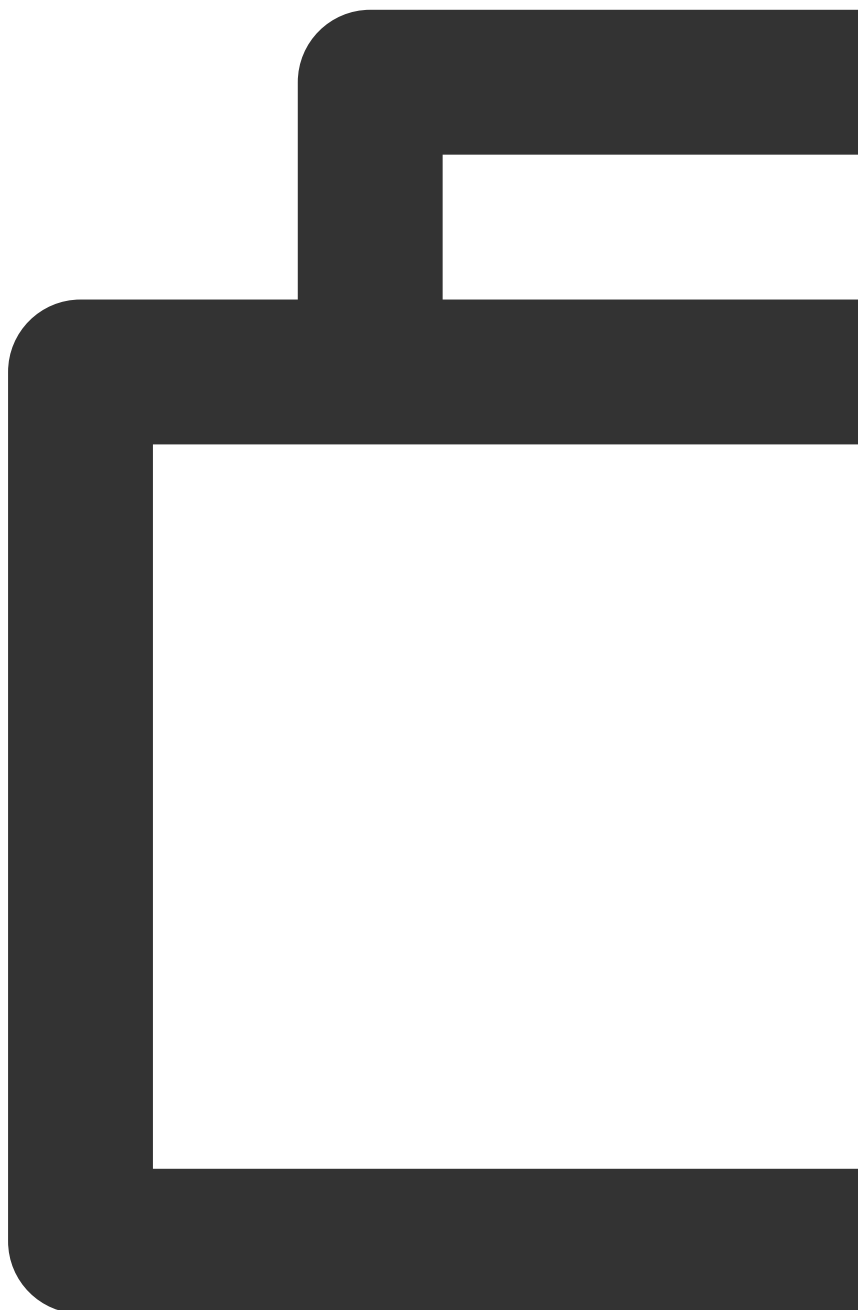


```
import { ArSdk } from 'tencentcloud-webar'  
// Initialize the SDK  
const sdk = new ArSdk({  
  ...  
})
```

**Config** of the SDK supports the following initialization parameters:

Parameters	Description	Type
module	The module	

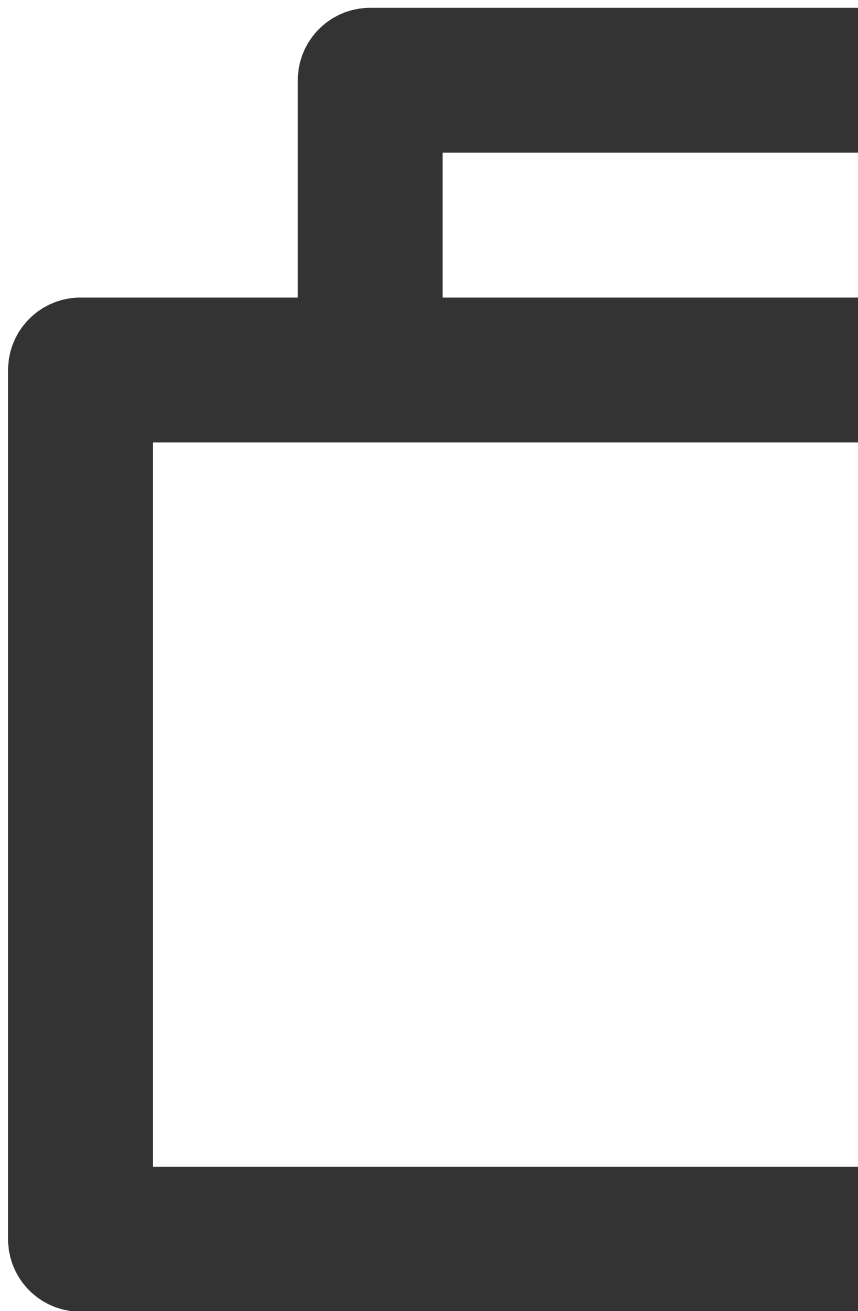
configuration



```
type SegmentationLevel = 0 | 1 | 2 // since version 1.0.
type ModuleConfig = {
  beautify: boolean // The default is `true`.
  segmentation: boolean // The default is `false`
  segmentationLevel: SegmentationLevel // The default is
}
```

auth

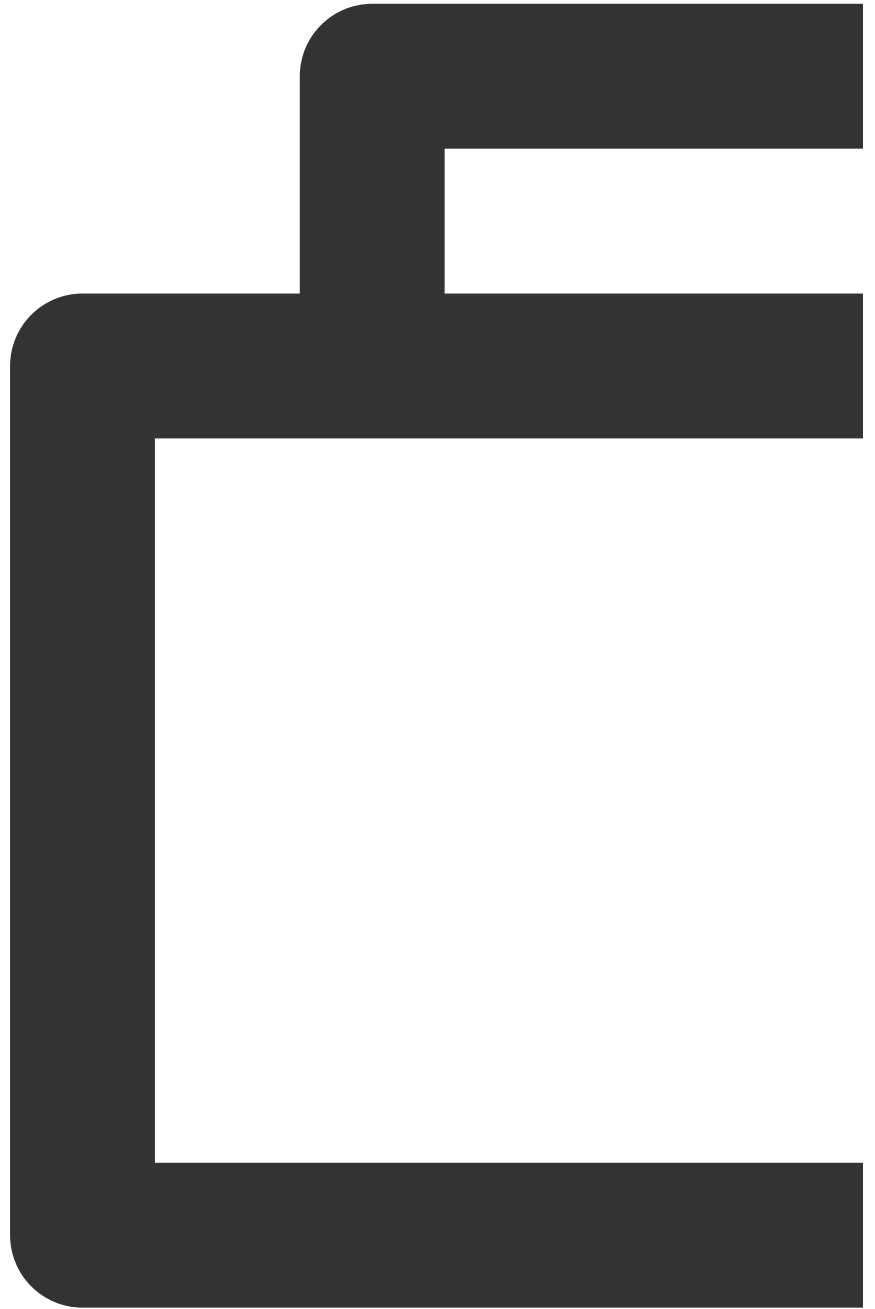
The authentication parameter



```
type AuthConfig = {  
  licenseKey: string // It can be obtained on the **Web  
  appId: string // It can be viewed in **Account Info**  
  authFunc:() => Promise<{  
    signature:string,  
    timestamp:string  
  }> // Refer to the license configuration.  
}
```

input	Source	MediaStream HTMLImageElement String
camera	Built-in Camera	<div></div> <div><pre>type CameraConfig = {   width: number, // The video width.   height: number, // The video height.   mirror: boolean, // Whether to horizontally flip the   frameRate: number // The capturing frame rate.</pre></div>

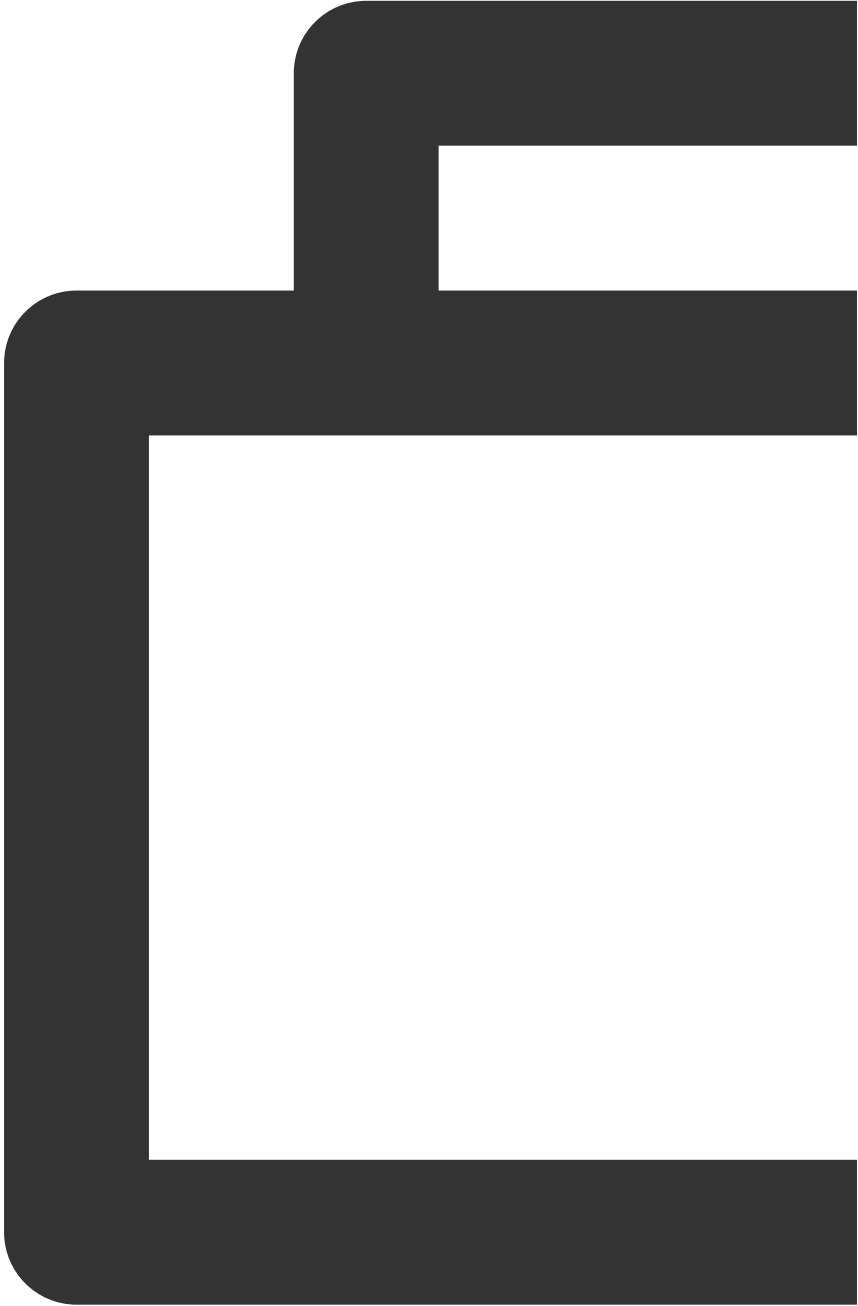
		<pre>}</pre>
mirror	Mirrored or not, mirroring of input streams is supported. (supported since 1.0.19)	Boolean
beautify	The beauty filter parameter	



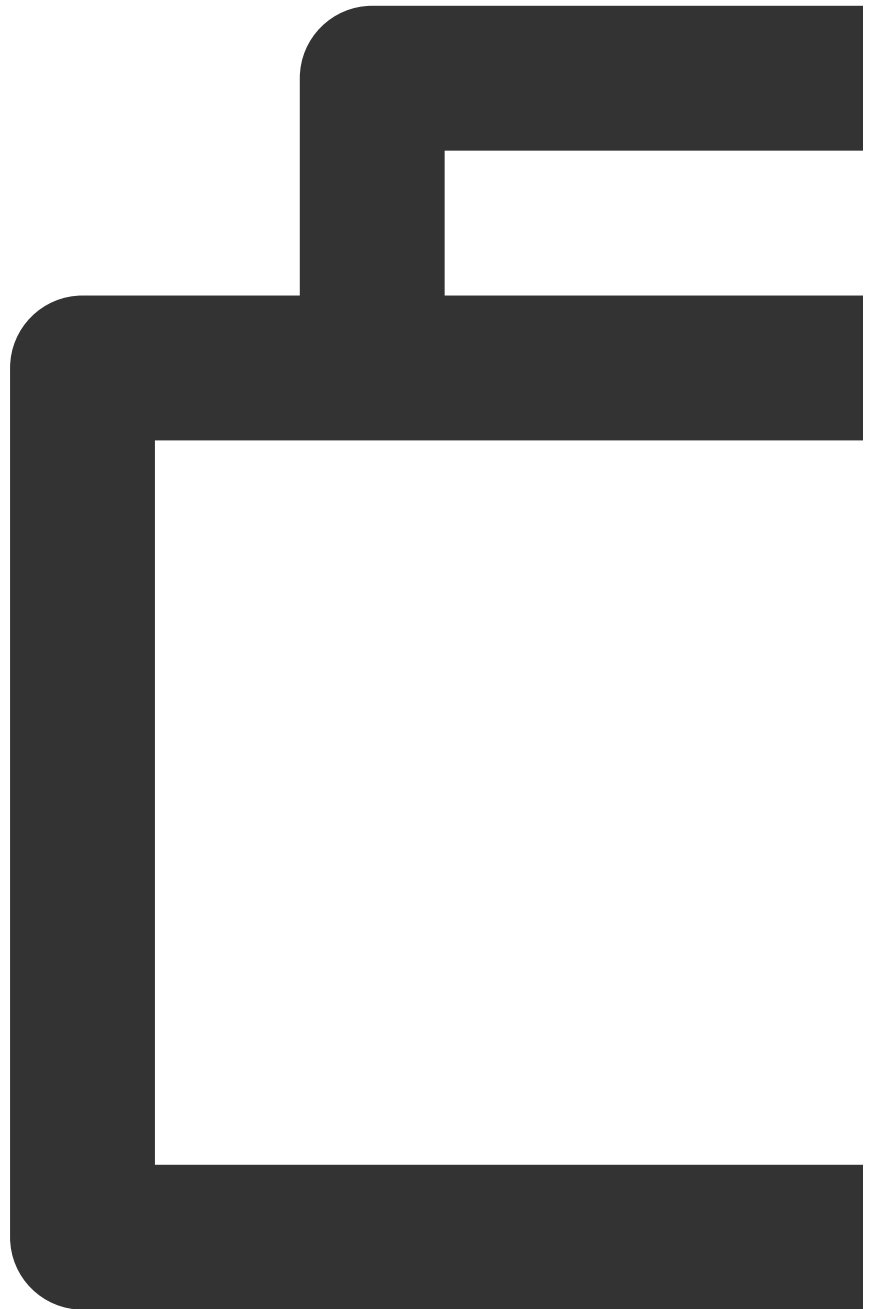
```
type BeautifyOptions = {  
    whiten?: number, // The brightening effect. Value  
    dermabrasion?: number // The smooth skin effect. V  
    lift?: number // The slim face effect. Value range  
    shave?: number // The face width. Value range: 0-1  
    eye?: number // The big eyes effect. Value range:  
    chin?: number // The chin effect. Value range: 0-1  
    // The following parameters are only available since
```

		<pre>darkCircle?: number; // The dark circle effect. Value range: 0-1. nasolabialFolds?: number; // The nasolabial folds effect. Value range: 0-1. cheekbone?: number; // The cheek bone effect. Value range: 0-1. head?: number; // The head effect. Value range: 0-1. eyeBrightness?: number; // The eye brightness effect. Value range: 0-1. lip?: number; // The lip effect. Value range: 0-1. forehead?: number; // The forehead effect. Value range: 0-1. nose?: number; // The nose effect. Value range: 0-1. usm?: number; // The distinct effect. Value range: 0-1. }</pre>
background	The background parameter	



		
		<pre>type BackgroundOptions = {   type: 'image'   'blur'   'transparent',   src?: string }</pre>
loading	The configuration	

of the built-in  
loading icon



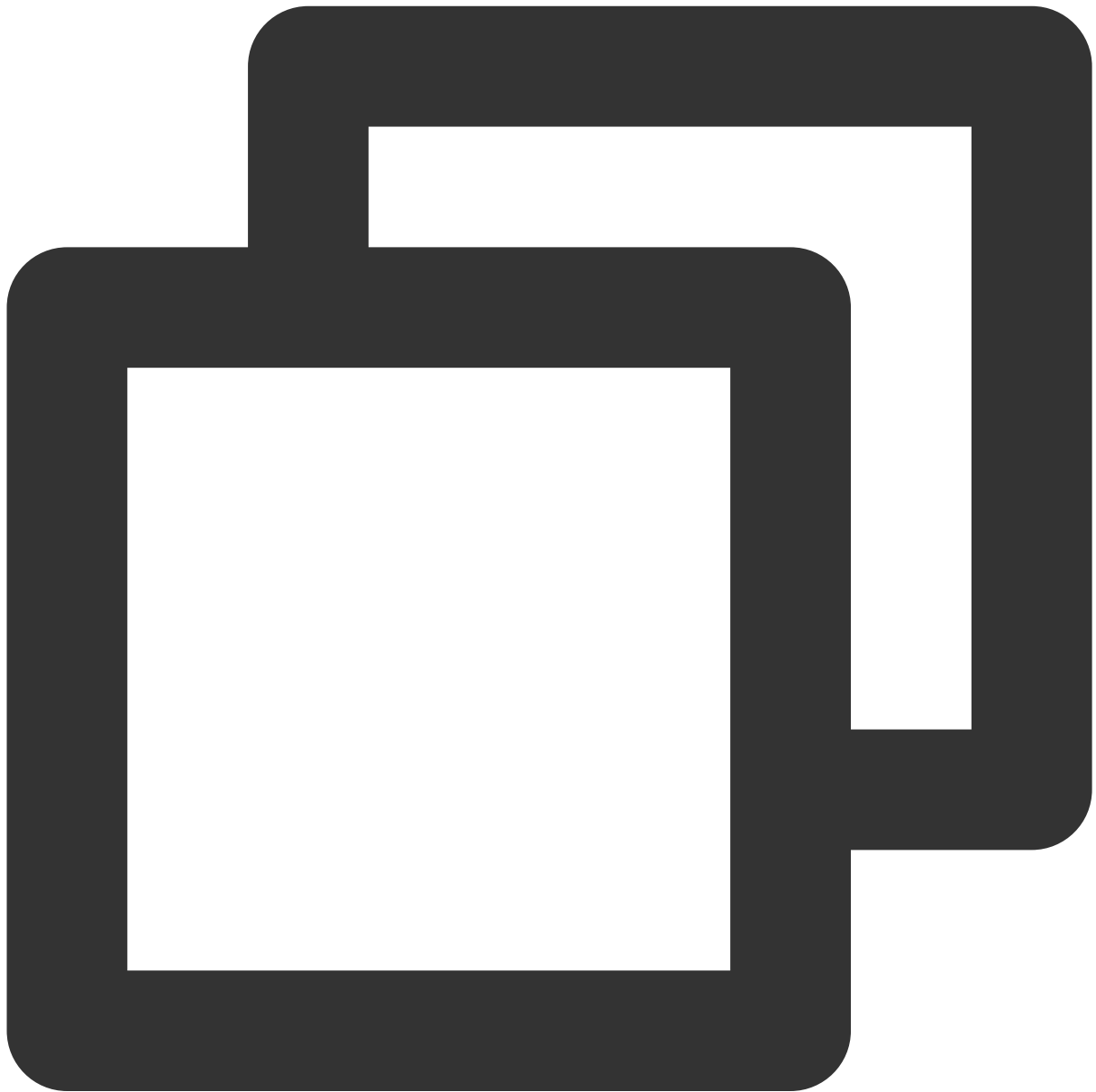
```
type loadingConfig = {  
  enable: boolean,  
  size?: number  
  lineWidth?: number  
  strokeColor?: number  
}
```

language

i18n

String: zh | en

## Callbacks



```
let effectList = [];  
let filterList = [];  
// Using the callbacks of the SDK  
sdk.on('created', () => {
```

```

// Pull and display the filter and effect list in the `created` callback
sdk.getEffectList({
  Type: 'Preset',
  Label: 'Makeup',
}).then(res => {
  effectList = res
});
sdk.getCommonFilter().then(res => {
  filterList = res
})
})
sdk.on('cameraReady', async () => {
  // By getting the output stream in the `cameraReady` callback, you can display
  // You can choose this method if you want to display a video image as soon as p
  // You don't need to update the stream after the effects start to work.
  const arStream = await ar.getOutput();
  // Play the stream locally
  // localVideo.srcObject = arStream

})
sdk.on('ready', () => {
  // Get the output stream in the `ready` callback. The initialization parameters
  // You can get the output stream in `ready` if you want your video to show effe
  // Between the two methods, choose the one that fits your needs.
  const arStream = await ar.getOutput();
  // Play the stream locally
  // localVideo.srcObject = arStream

  // Call `setBeautify`, `setEffect`, or `setFilter` in the `ready` callback
  sdk.setBeautify({
    whiten: 0.3
  });
  sdk.setEffect({
    id: effectList[0].EffectId,
    intensity: 0.7
  });
  sdk.setEffect({
    id: effectList[0].EffectId,
    intensity: 0.7,
    filterIntensity: 0.5 // In v0.1.18 and later, you can use this parameter to
  });
  sdk.setFilter(filterList[0].EffectId, 0.5)
})

```

Events	Description	Callback Parameter

created	The SDK authentication was completed and the instance was created successfully.	-
cameraReady	The SDK generated a video output (the video is not yet processed).	-
ready	Detection has been initialized. Effects are now applied to the output video. You can change the effect settings.	-
error	This callback is triggered when the SDK encounters an error.	The <code>error</code> object

## APIs

API	Parameters
async initCore()	



```
{  
  input?:MediaStream|HTMLImageElement|string; // The  
  camera?:CameraConfig; // Only for camera mode
```

	<pre>mirror?:boolean; // Mirror or not }</pre>
async getOutput(fps)	fps (optional): The output frame rate.
setBeautify(options)	



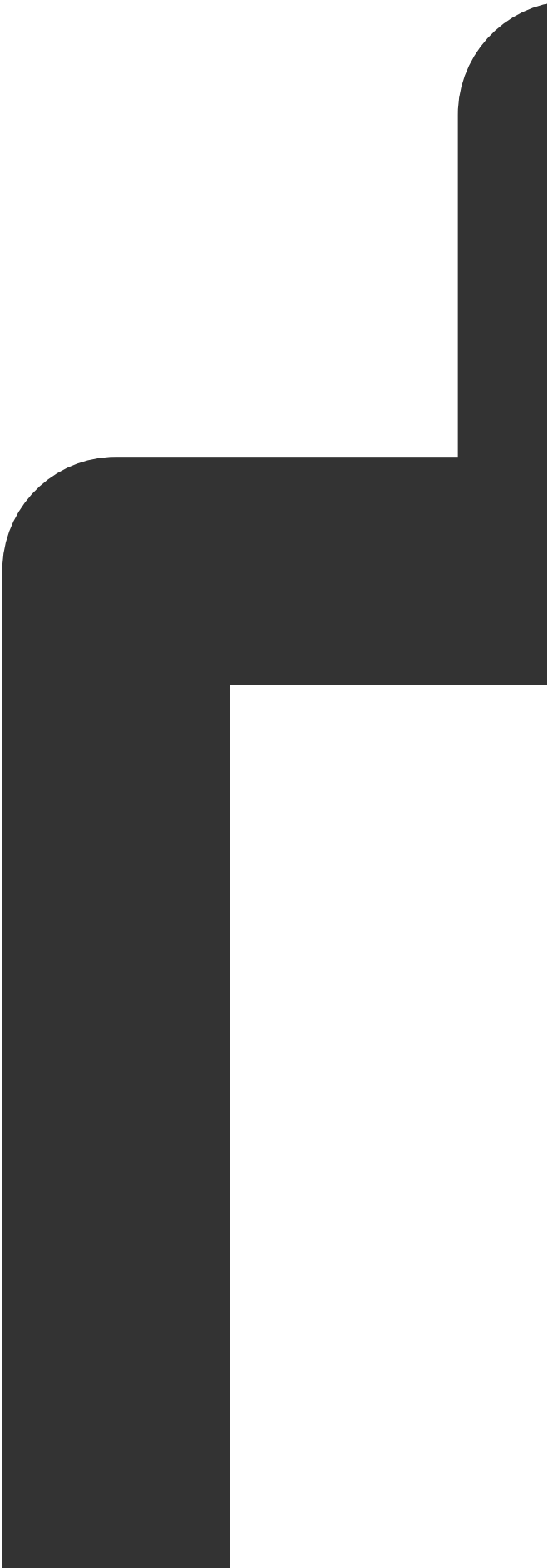


```
type BeautifyOptions = {  
  whiten?: number, // The brightening effect. Value range: 0-100  
  dermabrasion?: number // The smooth skin effect. Value range: 0-100  
  lift?: number // The slim face effect. Value range: 0-100  
  shave?: number // The face width. Value range: 0-100  
  eye?: number // The big eyes effect. Value range: 0-100  
  chin?: number // The chin shaping effect. Value range: 0-100  
  .....  
}
```

setEffect(effects, callback)

effects: Effect ID | Effect object | Effect ID / An effect array







```
effect:{
  id: string,
  intensity: number, // The effect strength. Value
  filterIntensity: number // The filter strength
}
```

callback: The callback for successful configuration

setAvatar(params)





```
{  
  mode: 'AR' | 'VR',  
  effectId?: string, // Pass through `effectId` to  
  url?: string, // Pass through `url` to use the  
  backgroundUrl?: string, // Background image URI  
}
```

setBackground(options)



```
{  
  type: 'image|blur|transparent',  
  src: string // This parameter is required only  
}
```

setSegmentationLevel(level)

level: 0 | 1 | 2

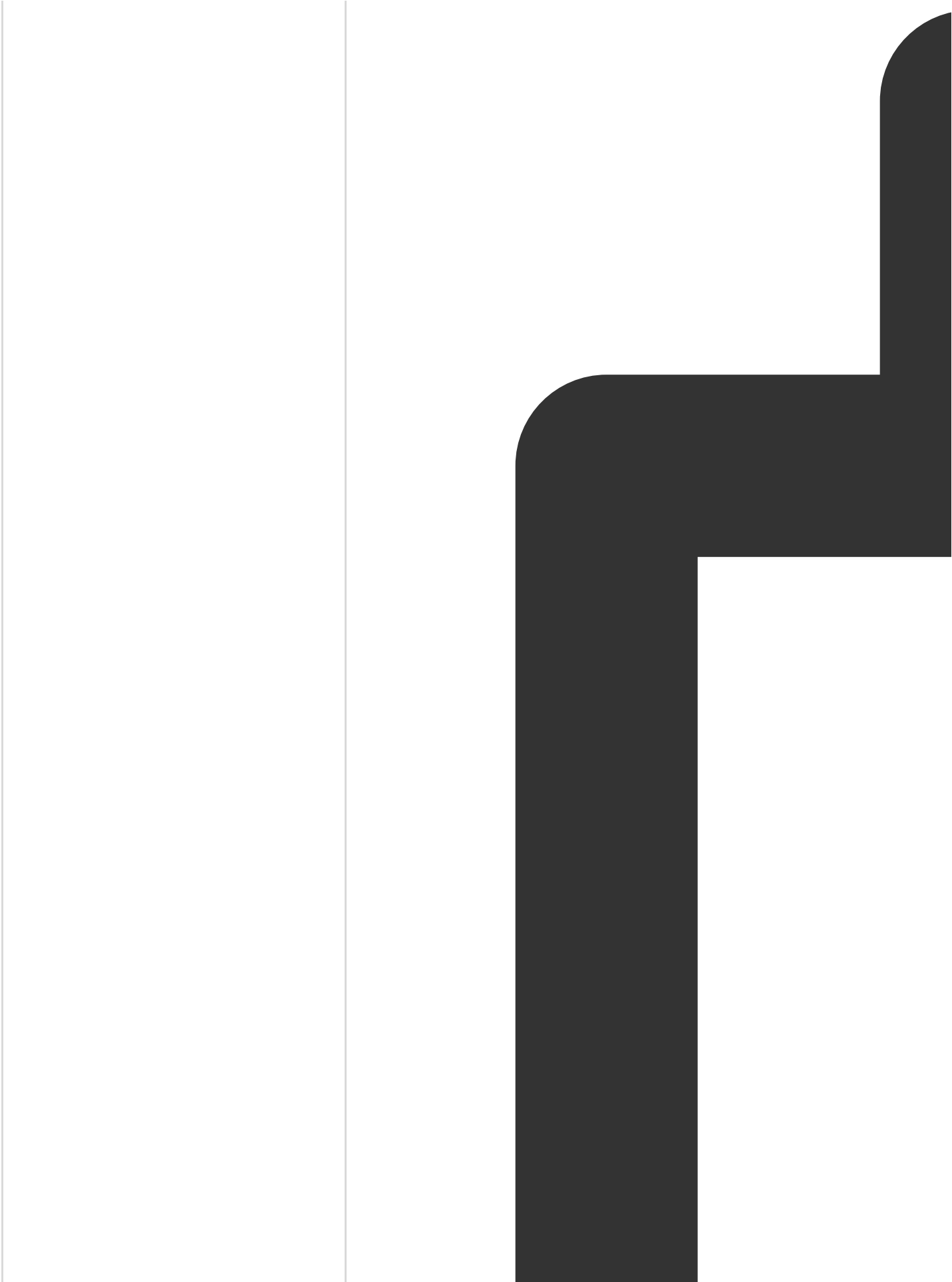
setFilter(id, intensity, callback)	id: The filter ID intensity: The filter strength. Value range: 0 - 1. callback: The callback for successful configuration.
getEffectList(params)	






```
{
  PageNumber: number,
  PageSize: number,
  Name: '',
  Label: Array,
  Type: 'Custom|Preset'
}
```

getAvatarList(type)



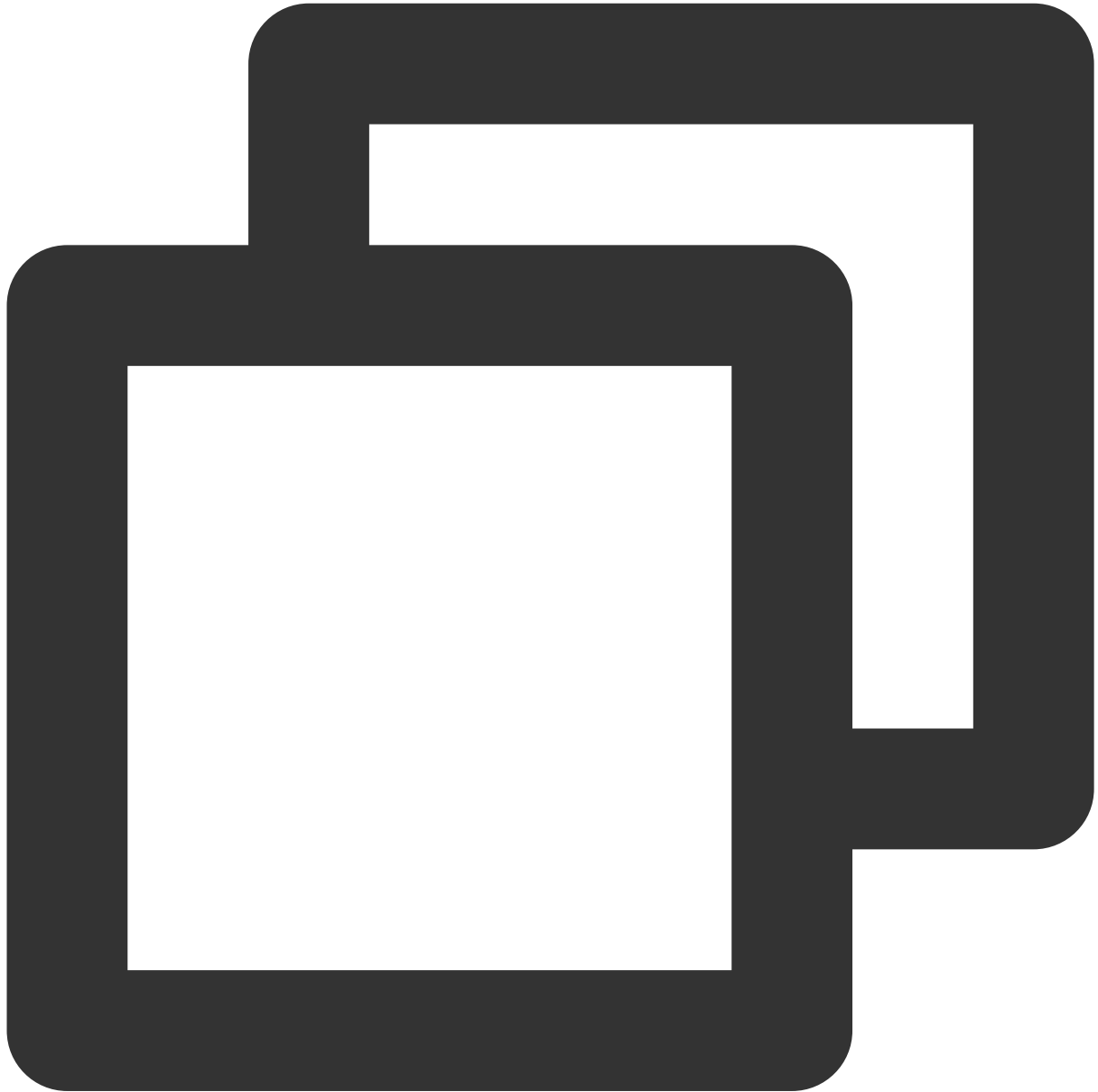


	 <pre>type = 'AR'   'VR'</pre>
<code>getEffect(effectId)</code>	effectId: The effect ID
<code>getCommonFilter()</code>	-
<code>async updateInputStream(src:MediaStream)</code>	src: New input stream ( <code>MediaStream</code> )
<code>disable()</code>	-

enable()	-
async takePhoto()	-
destroy()	-

## Error Handling

The error object returned by the error callback includes the error code and error message, which facilitate troubleshooting.



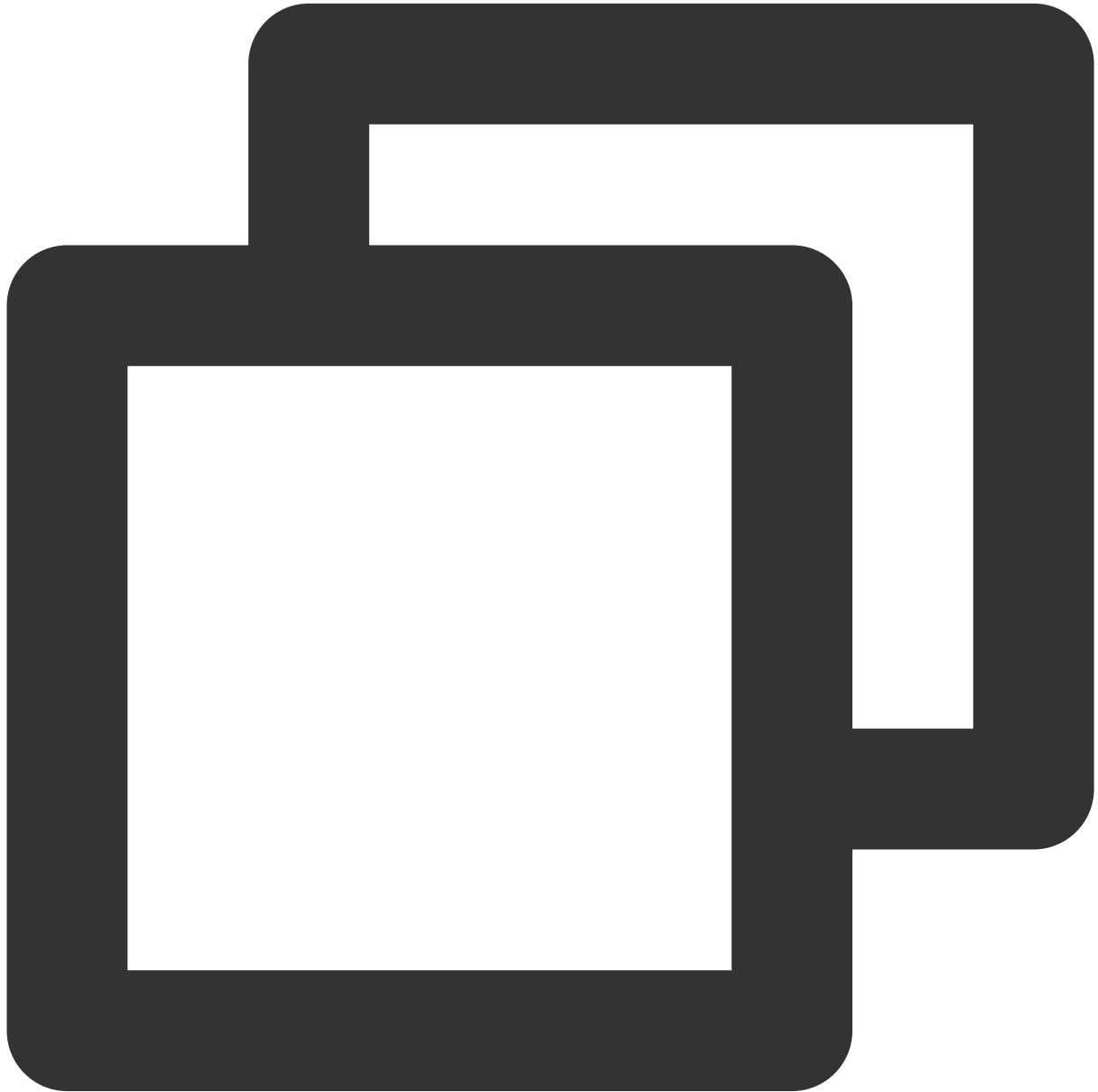
```
sdk.on('error', (error) => {  
  // Handle an error in the error callback  
  const {code, message} = error  
  ...  
})
```

Error Code	Description	Remarks
10000001	The current browser environment is incompatible.	Recommend the user to use Chrome, Firefox, Safari, or the Weixin browser.
10000002	The render context is missing.	-
10000003	The rendering is slow.	Consider reducing the video resolution or disabling the feature.
10000005	An error occurred while parsing the input source.	-
10000006	Lag may occur due to insufficient browser support.	Recommend the user to use Chrome, Firefox, Safari, or the Weixin browser.
10001101	An error occurred while configuring the effect.	-
10001102	An error occurred while configuring the filter.	-
10001103	The effect strength parameter is incorrect.	-
10001201	Failed to turn on the camera.	-
10001202	The camera stopped.	-
10001203	Failed to get the camera permission.	The user needs to enable the camera permission by going to <b>Settings &gt; Privacy &gt; Camera</b> .
20002001	The authentication parameter is missing.	-
20001001	Authentication failed	Make sure you have created a license and the signature is correct.
20001002	The API request failed.	The error callback will return the data returned by the API. For details, see <a href="#">API Error Codes</a> .
20001003	Failed to authenticate the effect configuration API.	The API is inaccessible. A Standard license cannot use the features of an Advanced license.
40000000	An uncaught exception occurred.	-
40000001	As the current SDK version is too low, certain effects cannot be correctly displayed. Upgrade the SDK version.	-

50000002	The effect is lost due to the resolution change.	The effect needs to be reconfigured.
----------	--	--------------------------------------

## Handling the missing render context error

On some computers, if the SDK is in the background for a long time, the `contextlost` error may occur. In such cases, you can call `ArSdk.prototype.resetCore(input: MediaStream)` to resume rendering.



```
sdk.on('error', async (error) => {  
  if (error.code === 10000002) {  
    const newInput = await navigator.mediaDevices.getUserMedia({...})
```

```
        await sdk.resetCore(newInput)
    }
})
```

# Console Guide

## Customizing Materials

### 3D Effects

Last updated : 2022-12-01 17:32:04

The SDK has supported 3D effects since v0.3.0. Please check your SDK version before using this capability.

Based on our standard head model, you can create more realistic 3D models using software such as Blender or 3ds Max and upload them to the console.

## Limits

To produce more applicable and realistic effects, we recommend you make 3D models based on our [standard head model](#).

Type	Limit
Format	GLB file or GLTF folder
Size	Keep the size of a model within 5 MB; otherwise it may take a long time to load.
Polygons	Keep the polygon count of a model within 100,000; otherwise the output file may be too large.
Texture maps	Use square textures in PNG format and keep the size of each map within 1024 x 1024. Commonly used dimensions include 256, 512, and 1024. Keep the size as small as possible without compromising the quality.
Materials	Make sure you use Principled BSDF materials for the surfaces; otherwise a rendering error will occur.

## Directions

### Step 1. Create a material

1. Download our standard head model and import it into your software (Blender is used as an example in this document). Create a material based on the model and export it in GLB or GLTF format.

Note :

- To ensure that the materials created fit the user's view, please do not modify the model's transform options such as location and scale.
- To keep the size of the output file small and the loading time short, we recommend you generate your model in GLB format.

## Step 2. Import the model into the console

1. Go to the material customization page of the console. A built-in 2D effect model is displayed by default. Click **3D effects** at the top to switch to a 3D model, which is the same as the standard head model mentioned above.
2. Click **Import model** on the right and select a model format (GLB is selected in this example).

Note :

- You can edit only one 3D effect scenario at a time. To import multiple 3D models, click **+** on the left.
- During editing, you can switch among different models by clicking them in the list on the left.

## Step 3. Modify parameters

Normally, because your 3D model is already built based on our standard head model, it should be ready to use after being imported into the console. If you need to modify the parameters of your model, select an editing mode at the top and drag elements in the editing area to change the settings or directly modify the parameters on the right.

Note :

If your model is not created based on our standard head model, you can adjust the tracking points to better track your model. For example, you can set nose bridge as the tracking point for a glass model and set philtrum as the tracking point for a mustache model.

## Step 4. Preview the material

You can preview your material on the built-in model. You can also switch to camera to view the effect in videos captured by the camera.

## Step 5. Export the material

Click **Export** in the top right corner. In the pop-up window, select a thumbnail and enter information such as the effect name, and click **Publish**. You can view published materials on the **Manage materials** page.



# Material Customization Guide

Last updated : 2023-03-28 16:37:22

## Panel Introduction

After binding a web license, you can start making your own effects in the console.



The material customization page is divided into two sections. There are a set of buttons at the top, and the area below is the main design area, which is further divided into (from left to right) the [effect list area](#), the [main panel](#), the [parameter configuration area](#), and the [preview area](#).

### Effect list

This area displays the effects currently used. You can use the right-click menu as well as shortcuts to perform common commands such as copy and paste. The **Group** command allows you to group multiple materials into one folder. You can click the



button next to an effect to enable or disable it. You can also drag the effect to change their sequence in the list.



## Main panel

The main panel visualizes your settings in the **Parameter configuration** area. You can zoom in/out and drag to move the panel.

For stickers, you can resize the sticker images by dragging the corners of the image frames. You can also drag to relocate the stickers.



## Parameter configuration

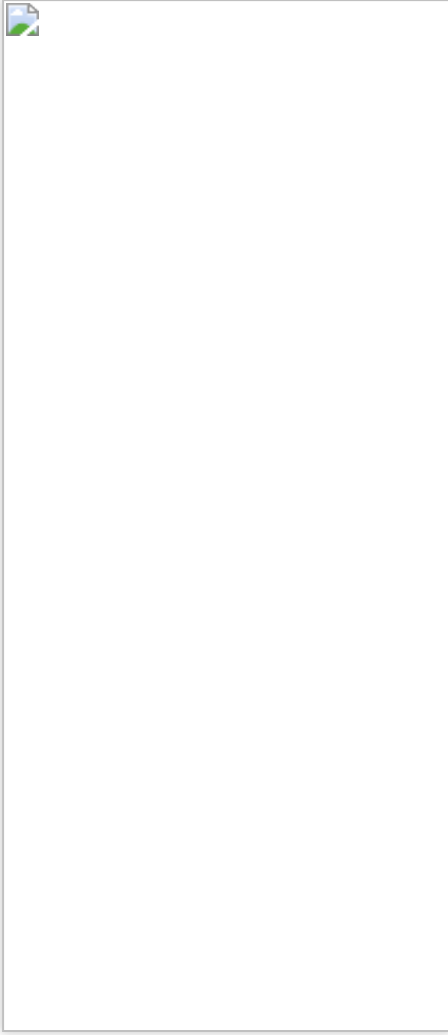
This area allows you to configure the properties of the selected material, such as transparency and the blend mode. For a sticker material, you can also adjust its location, size, and anchor point.

We offer a PSD template for each type of material. You can mouse over the icon next to **Choose material** to download the template.



## Preview

The preview area shows the effects you configure in real time. You can preview the effects either on the built-in models (man and woman) or on images captured by your camera.



## Customizing Materials

### Step 1. Download the template

1. To ensure that your effect applies well to the human face, you need to download our **human face model** and import it into Photoshop or After Effects to design your effect.
2. Click **Makeup > Eyebrows/Eyes** at the top and, in the **Parameter configuration** area, click **Download template**.

**Note**

A standard human face PSD file includes two parts – a model and a mask.

**Step 2. Design your material**

A typical effect material usually includes makeup effects, stickers, and filters.

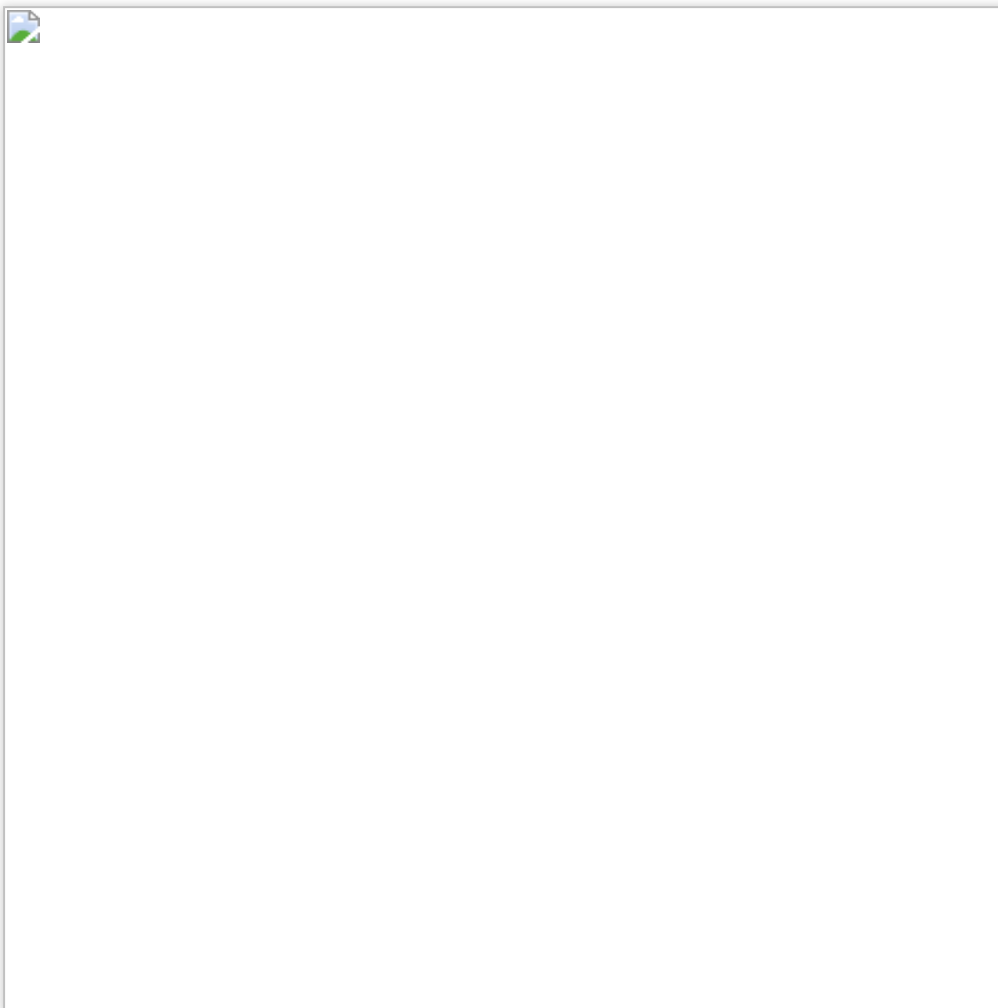


In addition to the **human face model**, we offer different PSD templates for different effect types (eye color, filters). To download the templates, just select the makeup effect you want to use at the top and then click **Download template** in the **Parameter configuration** area.



For example, after you open the PSD template with Photoshop, you can design effects for the following points of the face. The template offers different reference points. To ensure that your effect applies well to the face, you should align your material closely to the reference points.





PSD template for eye color effects:



PSD template for filters:



We do not offer a template for stickers. You can customize your own template.

#### Note

Please make sure your materials comply with the following requirements:

For makeup effects including lips, eyes, eyebrows, face mask, use an 800 x 800 PNG image.

For eye color, use an image with an aspect ratio of 1:1 (150 x 150 is recommended).

For stickers, the total frame count cannot exceed 100. An image smaller than 1000 x 1000 is recommended. Images larger than this will be compressed.

For filters, use the standard color card in the template. You can adjust the parameters of your filter using photo editing mobile apps such as VSCO, export the color card, and convert it to a standard LUT file in PNG format in your computer. If you are familiar with color adjustments, you can also make your filter directly in Photoshop. In either case, the final output file should be a 512 x 512 LUT file.

### Step 3. Import your material

1. After you are finished making your material, import it to the console.

Click **Makeup** at the top, select an effect type, and upload your material in the **Parameter configuration** area.

Click **Filter** at the top and upload your LUT file in the **Parameter configuration** area.

Click **Face sticker** at the top and upload the sequential frame images in the **Parameter configuration** area. For example, if you want to replace the default eye makeup effect, select **Eyes** in the effect list, and upload your material.



#### Note

We offer a **library** of common filters and makeup effects which you can use directly in your material.



2. After you add an effect, you will find the details of the effect in the **Parameter configuration** area. You can change the parameters of the effect here, such as the **blend mode** (the same as those in Photoshop) and strength (transparency). For a sticker effect, you can also adjust its location, size, and playback speed (fps).

3.



## Step 4. Export the material

1. After you are finished configuring your custom material, you can select a built-in model on the right or use the camera to preview the material.
2. If you are satisfied with the effect, click **Export** in the top right, complete the following settings, and click **Publish**. Upload a thumbnail image for your material.

Name your material.

Select or enter labels for your material. Labels help you categorize your materials and can be used as filter conditions.

For example, you can add labels according to a material's project type ( `Web project` , `Mini Program project` ).



# Face Stickers

Last updated : 2023-04-21 15:37:24

You can upload a PNG file or PNG sequence frames to the console to create your own face-tracking stickers.



## Limits

We do not provide a PSD template for stickers. Just make sure your materials meet the following requirements.

Requirement	Description
Format	For static stickers, upload a single PNG image. For animated stickers, upload a sequence of <b>numbered</b> PNG images, such as <code>001.PNG</code> , <code>002.PNG</code> .
Dimensions	1000 x 1000 px or smaller is recommended.



Frame count	100 frames or fewer.
Size	A single image should not exceed 1 MB.

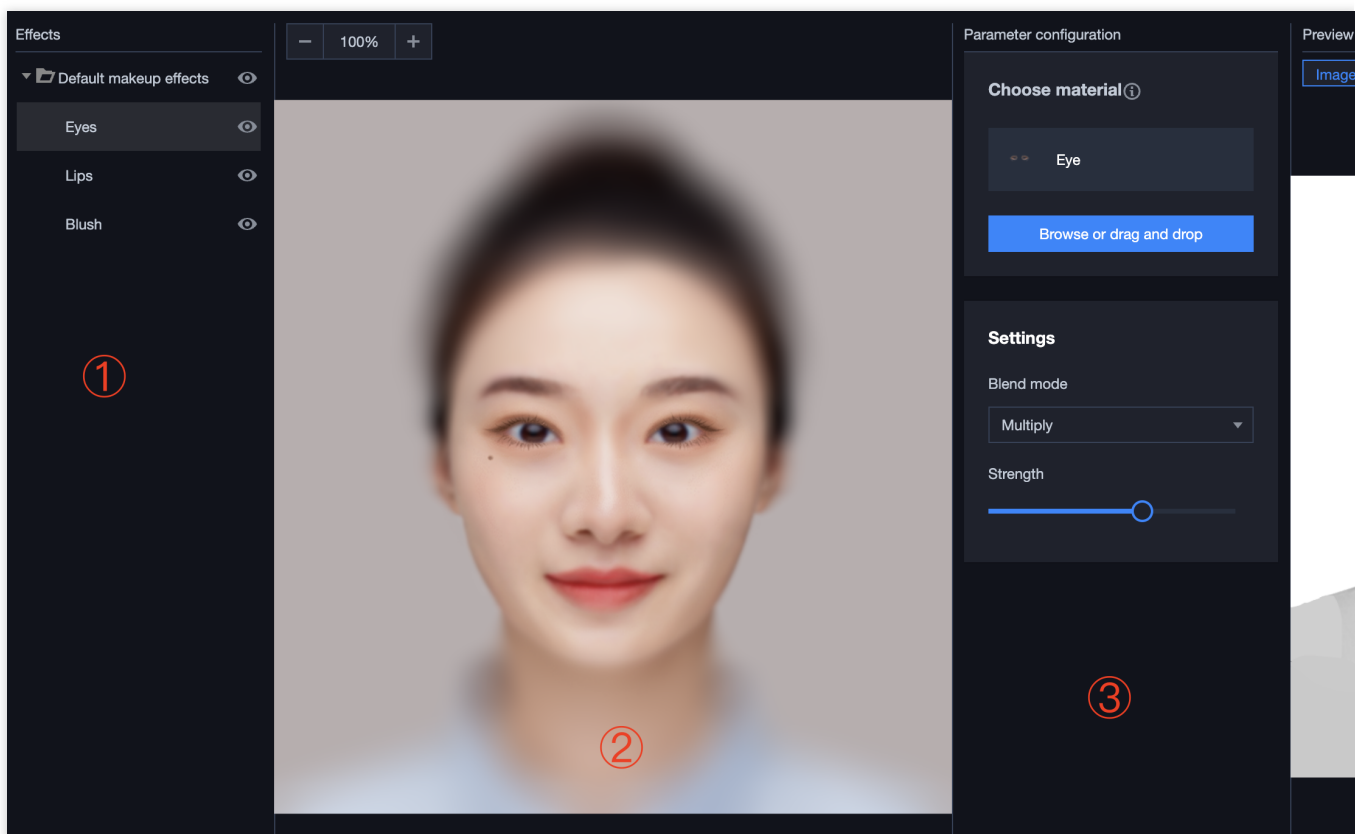
## Directions

### Step 1. Create your material

Create a PNG image or PNG sequence frames that meet the above [requirements](#).

### Step 2. Import the material to the console

1. Log in to the console and go to **Customize Materials**. Click **Face sticker** at the top to add a default sticker.
2. In the **Choose material** area of the **Parameter configuration** panel on the right, upload your material by either drag & drop or browse.



### Step 3. Modify the parameters

You can modify the parameters of your material in two ways:

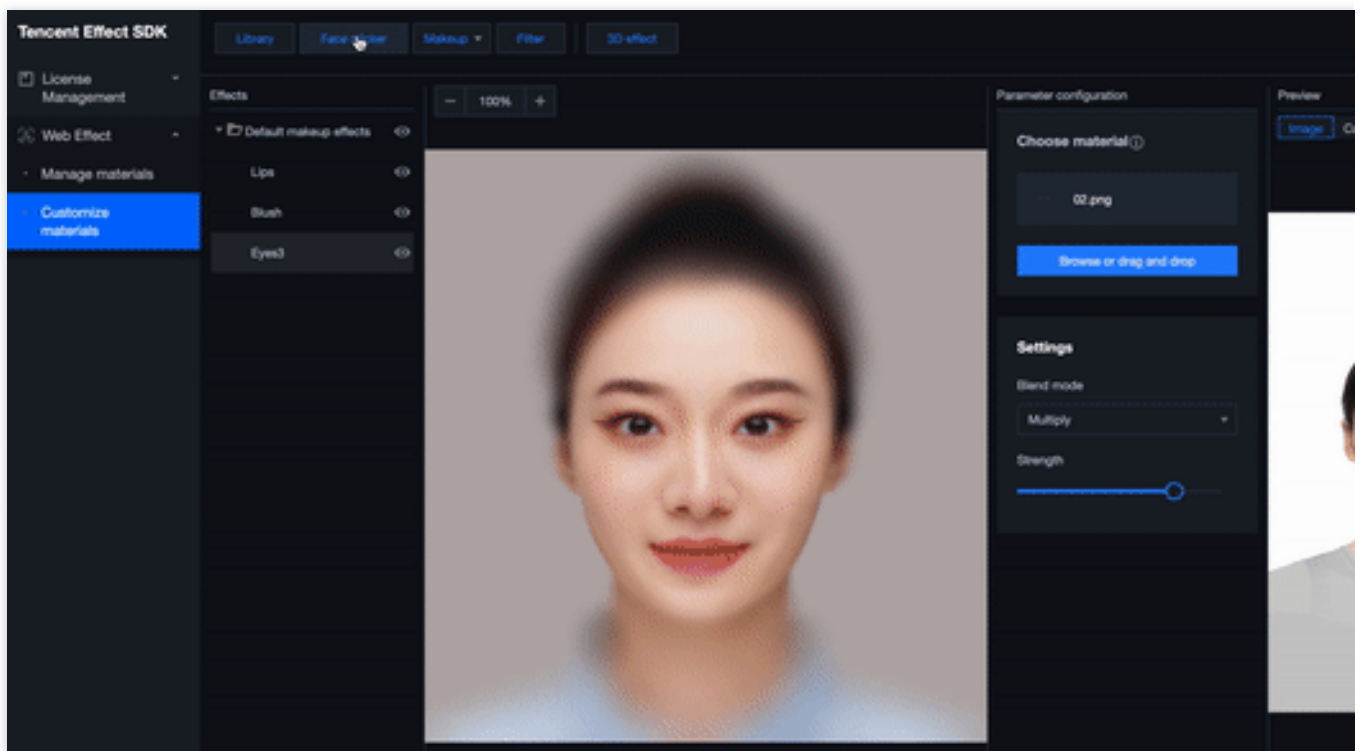
In the main panel, drag the sticker image to position it in relation to the face.

In the parameter configuration panel on the right, change settings including the **blend mode**, **strength**, **playback speed** (for sequence frames), and **anchor point**.

#### Note

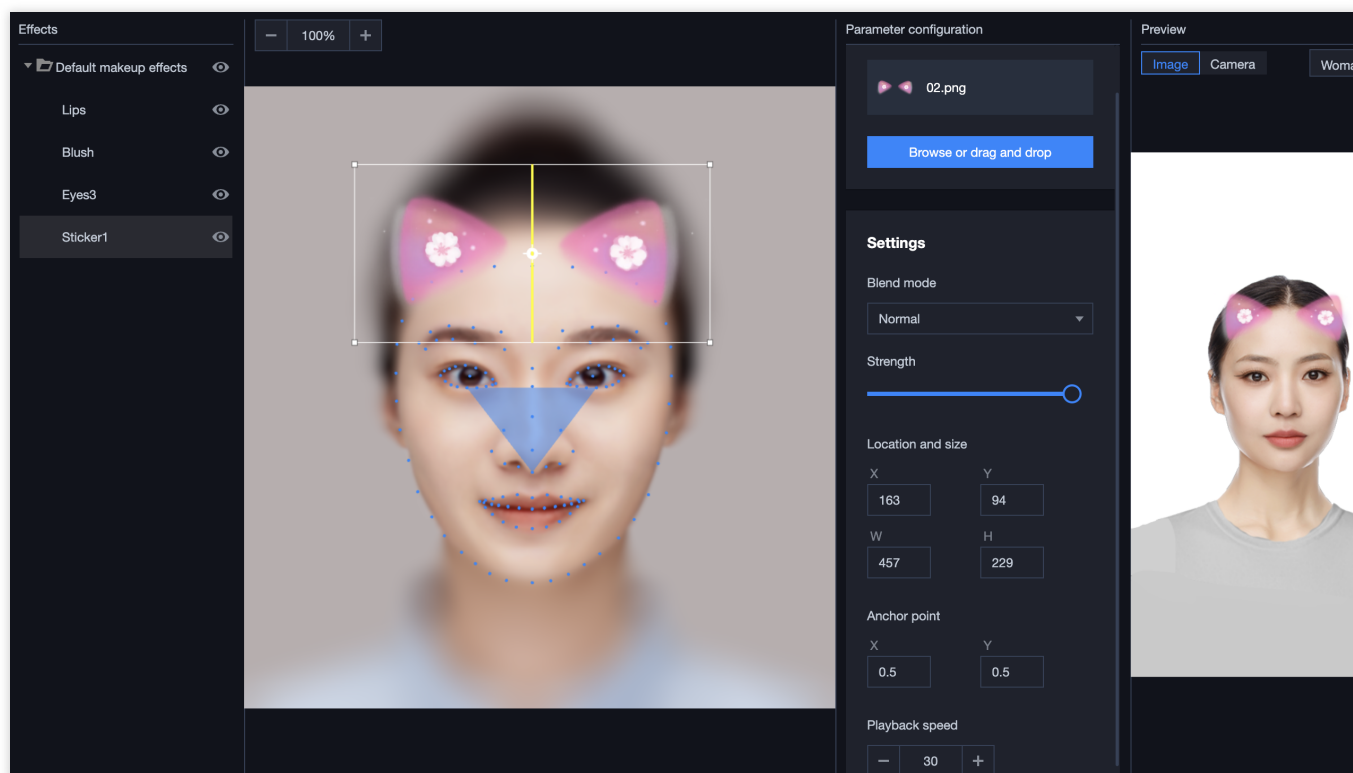
We do not recommend setting the playback speed higher than 30 fps.

**Anchor point** is the reference point for the sticker, which is the center of the sticker image by default. X and Y indicate the horizontal and vertical distances (in percentage) of the anchor point from the top left corner of the sticker image. For example, the default anchor point is (0.5, 0.5). If you want to use the top left corner of the sticker as the anchor point, set the parameter to (0, 0). A well-chosen anchor point makes for better tracking effects. For example, for headdress stickers, an anchor point in the forehead is recommended, and for stickers such as glasses, an anchor point along the nose bridge is recommended. You can keep exploring to find the best anchor point for your sticker.



#### Step 4. Preview the material

In the preview window on the right, you can preview your sticker on a built-in photo or on your device's camera.



# Material management

Last updated : 2023-06-21 16:56:21

The [Manage materials](#) page of **Beauty AR Web** in the **RT-Cube console** displays the effect materials customized in the console and allows you to view, filter, add, delete, and edit materials.

## Viewing materials

The material form allows you to preview thumbnails of materials and displays attributes such as name, label, and creation time.

## Filtering materials

The material form allows you to filter materials by attribute. You can filter materials by **name** and **label**. Fuzzy match is supported when filtering by name.

## Editing labels

You can add multiple labels to materials and edit labels of custom materials on the **Manage materials** page. When you hover over a material display card, the edit icon will be visible in the top-right corner, and you can click it to add or delete labels. Labels help you manage materials more easily, and good label settings help you better display materials and use them through the SDK more conveniently.

## Deleting a material

Custom materials can be deleted. When you hover over a material display card, the delete icon will appear in the top-right corner of the card. To delete the material, click **Delete** and then confirm the operation. Please be cautious when deleting materials, as a material cannot be recovered after you delete it.

# Demos

Last updated : 2024-05-08 16:34:16

The Beauty AR Web SDK is supported on multiple platforms such as PC web, mobile browser. You can click the link below to try out the demo.

Platform	Address	Runtime Environment
Web	<a href="#">Click here to try</a>	We recommend that you use desktop Chrome 90 or later in a PC environment. For mobile, we recommend that you use the latest version of Safari or Chrome.

# Practical Tutorial

## Publishing over WebRTC

Last updated : 2024-07-09 16:06:24

### Before You Start

Please read the [integration guide](#) for Beauty AR Web.

Please read [Getting Started](#) and [WebRTC Push](#) to complete the basic settings and learn how to publish streams over WebRTC.

### Directions

#### Step 1. Import the Beauty AR Web SDK

Add the following JavaScript script into the webpage (desktop) from which streams are to be published:



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

**Note:**

The above example uses a script tag to import the SDK. You can also [import it using an npm package](#).

**Step 2. Import the WebRTC publishing resources**

Add the following JavaScript script into the webpage (desktop) from which streams are to be published:



```
<script src="https://video.sdk.qcloudcdn.com/web/TXLivePusher-2.0.0.min.js" charse
```

**Note:**

Make sure you add the script to the body of the HTML. Adding it to the head may cause an error.

**Step 3. Initialize the Beauty AR Web SDK**

Sample code:





```
const { ArSdk } = window.AR;

/** ----- Authentication configuration ----- */

/**
Tencent Cloud account's APPID
*
* You can view your APPID in the [Account Center] (https://console.tencentcloud.com)
*/
const APPID = ''; // Set it to your Tencent Cloud account APPID.
```

```
/**
 * Web LicenseKey
 *
 * Log in to the RT-Cube console and click [Web Licenses] (https://console.tencentcl)
 */
const LICENSE_KEY = ''; // Set it to your license key.

/**
 * The token used to calculate the signature.
 *
 * Note: This method is only suitable for debugging. In a production environment, y
 * [Signature] (https://cloud.tencent.com/document/product/616/71370#.E7.AD.BE.E5.90)
 */
const token = ''; // Set it to your token.

/** ----- */

/**
 * Get the signature
 *
 * Note: This method is only suitable for debugging. In a production environment, y
 * Example:
 * async function () {
 *   return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
  return { signature, timestamp };
};

let w = 720;
let h = 480;

// Get the input stream
const stream = await navigator.mediaDevices.getUserMedia({
  audio: true,
  video: { width: w, height: h }
});

// The basic settings for the Tencent Effect SDK
const config = {
  input: stream,
  auth: {
    licenseKey: LICENSE_KEY,
    appId: APPID,
```

```
    authFunc: getSignature
  },
  // Configure the initial effects (optional)
  beautify: {
    whiten: 0.1, // The brightening effect. Value range: 0-1.
    dermabrasion: 0.5, // The smooth skin effect. Value range: 0-1.
    lift: 0.3, // The slim face effect. Value range: 0-1.
    shave: 0, // The V shape effect. Value range: 0-1.
    eye: 0, // The big eyes effect. Value range: 0-1.
    chin: 0, // The chin effect. Value range: 0-1.
    .....
  },
  language: 'en',
  .....
}

// Pass `config` to the Tencent Effect SDK
const ar = new ArSdk(config);

// You can display the effect and filter list in the `created` callback.
ar.on('created', () => {
  // Get the built-in makeup effects and stickers
  ar.getEffectList({
    Type: 'Preset'
  }).then((res) => {
    const list = res.map(item => ({
      "name": *item.Name,
      id: item.EffectId,
      cover: item.CoverUrl,
      url: item.Url,
      label: item.Label,
      type: item.PresetType,
    }));

    const makeupList = list.filter(item=>item.label.indexOf('Makeup')>=0)
    const stickerList = list.filter(item=>item.label.indexOf('Sticker')>=0)
    // Show the makeup and sticker lists
    renderMakeupList(makeupList);
    renderStickerList(stickerList);

  }).catch((e) => {
    console.log(e);
  });
  // Get the built-in filters
  ar.getCommonFilter().then((res) => {
    const list = res.map(item => ({
      "name": *item.Name,
```

```
        id: item.EffectId,
        cover: item.CoverUrl,
        url: item.Url,
        label: item.Label,
        type: item.PresetType,
    }));

    // Show the filter list
    renderFilterList(list);

}).catch((e) => {
    console.log(e);
});

});

ar.on('ready', async (e) => {

    // After receiving the `ready` callback, you can call `setBeautify`, `setEffect`

    // For example, you can use `range input` to set the smooth skin effect:
    $('#dermabrasion_range_input').change((e) => {
        ar.setBeautify({
            dermabrasion: e.target.value, // The smooth skin effect. Value range: 0
        });
    });

    // In the `created` callback, apply the effects based on user interactions with
    $('#makeup_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });
    $('#sticker_list li').click(() => {
        ar.setEffect([{id: effect.id, intensity: 1}]);
    });

    // In the `created` callback, apply the filter based on user interactions with
    ar.setFilter(filterList[0].id, 1);
    $('#filter_list li').click(() => {
        ar.setFilter(filter.id, 1);
    });

    // Get the output stream of the Tencent Effect SDK
    const arStream = await ar.getOutput();

});

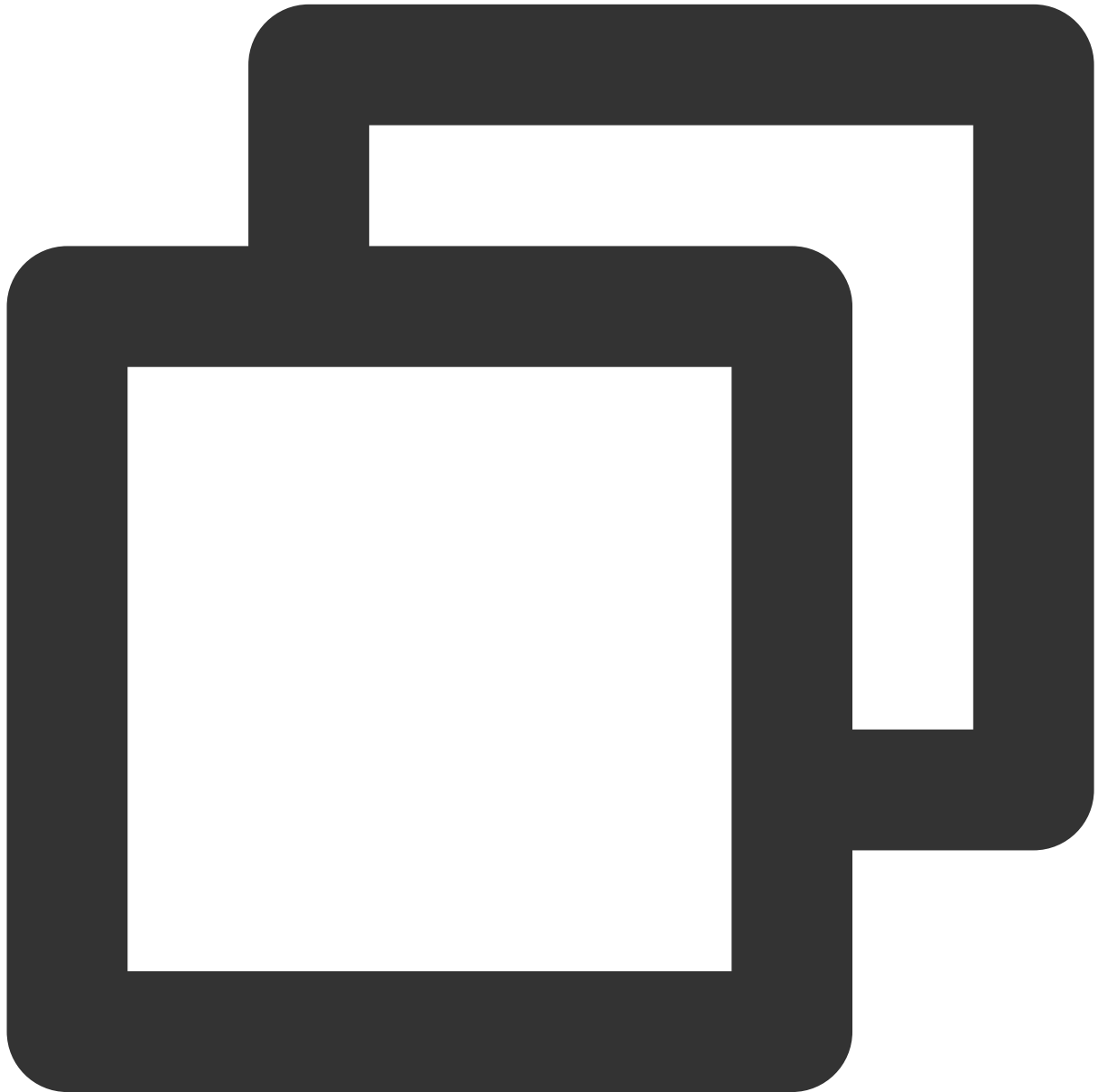
ar.on('error', (e) => {
    console.log(e);
});
```

```
});
```

To learn more about UI control, you can download our code package at the end of this document.

#### Step 4. Publish the stream

After getting the output stream in the `ready` callback of the SDK, publish it over WebRTC:



```
let livePusher = new TXLivePusher()  
// Set the basic stream publishing parameters (begin)  
let DOMAIN = 'Your push domain'
```

```
let AppName = 'Your app name'
let StreamName = 'Your stream name'
let txSecret = 'Your txSecret'
let txTime = 'Your txTime'
// Set the basic stream publishing parameters (end)

let pushUrl = `webrtc://${DOMAIN}/${AppName}/${StreamName}?txSecret=${txSecret}&txT

// Set the preview (optional)
livePusher.setRenderView('id_local_video');
// Capture the stream
livePusher.startCustomCapture(arStream);
// Publish the stream immediately (you can also use another API to control when to
livePusher.startPush(pushUrl)
```

In the above code, both `txSecret` and `txTime` need to be calculated. You can use the [address generator](#) of the **CSS console** to quickly generate the parameters and get the publishing URL. For detailed directions, see [Address Generator](#).

After the stream is successfully published (`startPush`), you should be able to see the video with effects applied.

## Step 5. Play the stream

### Note:

For the example project, you need to start the web service of your device and make sure that the HTML file can be accessed via the specified port.

If you have an available playback domain, follow the steps in [Live Playback](#) to play the stream.

If you don't have a playback domain, you can preview the stream in [Stream Management](#) of the **CSS console**.

## Sample Code Package

You can download our sample code package [here](#). The code for publishing over WebRTC is in `AR_LEB_WEB`.

# Publishing over WebRTC (Preinitialization Scheme)

Last updated : 2024-07-09 16:06:24

## Preparations

Read [Overview](#) to learn about how to use the Beauty AR Web SDK.

For more information on WebRTC publishing, see [Publishing over WebRTC](#). This document describes the differences in the code and process when the preinitialization scheme is used.

For more information on the preinitialization scheme, see [Loading Optimization](#).

## Getting Started

The preinitialization scheme differs from the general loading scheme mainly in that you don't need to specify the `input` or `camera` attribute when initializing the SDK; that is, instead of specifying the input data for the SDK during initialization, you later call the `initCore` API to specify the data at an appropriate location based on your needs. In this way, the resources depended on by the SDK are loaded in advance, and the `ready` event of the SDK will be triggered more quickly after `initCore` is later called, making it easier to get and display the output stream. Below is the key sample code:

### Initializing SDK



```
...
let resourceReady = false
// Basic configuration parameters of the Beauty AR SDK
const config = {
  // input: stream, // Do not specify `input`.
  auth: {
    licenseKey: LICENSE_KEY,
    appId: APPID,
    authFunc: getSignature
  },
  // Initial beauty effects (optional parameters)
```



```
    beautify: {
      whiten: 0.1, // The brightening effect. Value range: 0-1.
      dermabrasion: 0.5, // The smooth skin effect. Value range: 0-1.
      lift: 0.3, // The slim face effect. Value range: 0-1.
      shave: 0, // The V shape effect. Value range: 0-1.
      eye: 0, // The big eyes effect. Value range: 0-1.
      chin: 0, // The chin shaping effect. Value range: 0-1.
      .....
    },
    language: 'en'
  }

// Pass in `ar sdk` for `config`.
const ar = new ArSdk(config);
// If the `resourceReady` callback event is triggered, the resources have been comp
ar.on('resourceReady', () => {
  resourceReady = true
})
// The `ready` event will be triggered after `initCore` is called.
ar.on('ready', () => {
  // Get the output stream data of the Beauty AR SDK
  const arStream = await ar.getOutput();
  // Process the output stream
  ...
})
...
```

## Triggering the `initCore` event with a user operation



```
// The following describes how to set the input stream for the preinitialization scene
function onClickStartCamera(){
    let w = 1280;
    let h = 720;

    // Get the device's input stream
    const arInputStream = await navigator.mediaDevices.getUserMedia({
        audio: true,
        video: {
            width: w,
            height: h
        }
    });
}
```

```
    }
  });
  if(!resourceReady){ // In this mode, calling `initCore` will have no effect if
    return
  }
  // Set the input stream data of the Beauty AR SDK
  ar.initCore({
    input: arInputStream
  })
}
```

## Sample Code

You can download the [sample code](#), decompress it, and view the `AR_and_LEB_Preload.html` file in the `AR_LEB_WEB` code directory.

# Publishing Using TRTC

Last updated : 2024-07-09 16:06:24

## Before You Start

Please read the [integration guide](#) for Beauty AR Web.

Follow the steps in [Integration \(No UI\)](#) to integrate the TRTC web SDK.

Try [running a TRTC web demo](#) in your local project.

## Directions

### Step 1. Import the Beauty AR Web SDK

**You can import the Beauty AR Web SDK by making some minor changes to the import method in the TRTC web demo.**

Add the following JavaScript script to your webpage (desktop):



```
<script charset="utf-8" src="https://webar-static.tencent-cloud.com/ar-sdk/resource
```

**Note:**

The above example uses a script tag to import the SDK. You can also [import it using an npm package](#).

**Step 2. Understand the stream initialization logic of TRTC**

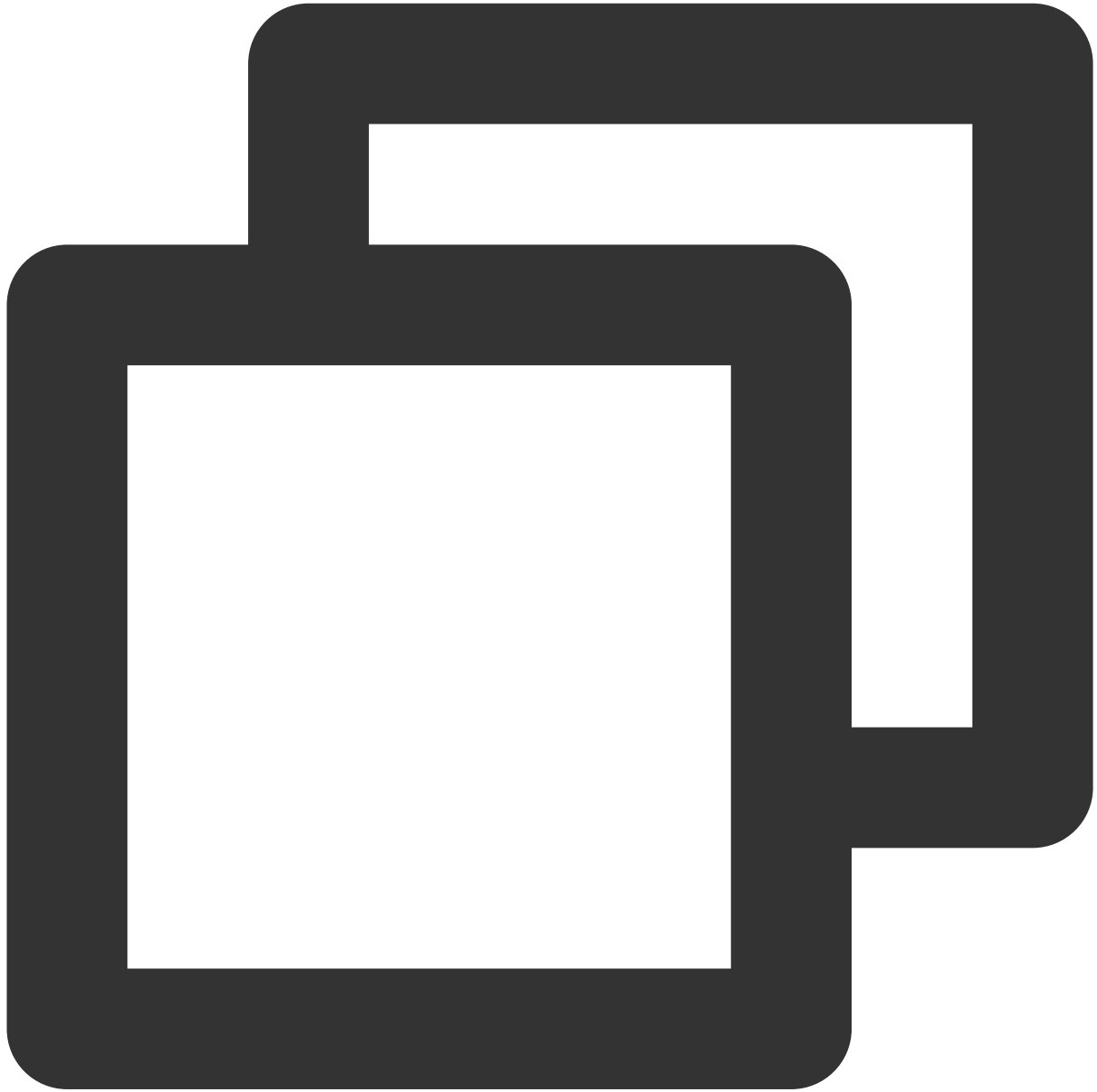
1. TRTC's demo project shows you how a local stream is initialized. The API `createStream` is used to create a stream object. In the example below, the SDK's default capturing mode is used.



```
// Capture audio and video from the mic and camera for the local stream
const localStream = TRTC.createStream({ userId, audio: true, video: true });
localStream.initialize().then(() => {
  console.log('initialize localStream success');
  // The local stream was initialized successfully. You can call `Client.publish(localStream)`.
  .catch(error => {
    console.error('failed initialize localStream ' + error);
  });
});
```

This is the most common local stream initialization method.

2. `TRTC.createStream` also allows you to use an external audio/video source for the local stream so that you can use your own custom processing on the stream (such as applying beautification effects to the video). Below is an example:



```
// Get the custom stream
const stream = await this.ar.getOutput();

// Use the audio/video source to create a local stream object
const audioTrack = stream.getAudioTracks()[0];
const videoTrack = stream.getVideoTracks()[0];
const localStream = TRTC.createStream({ userId, audioSource: audioTrack, videoSource
```

```
localStream.initialize().then(() => {  
  console.log('initialize localStream success');  
  // The local stream was initialized successfully. You can call `Client.publish(localStream)`  
  .catch(error => {  
    console.error('failed initialize localStream ' + error);  
  });  
});
```

For detailed directions on how to use `createStream`, see [TRTC SDK documentation](#).

3. To process the local stream using Tencent Effect, you need to use the second method above. Before you call `getMyStream`, [initialize the Beauty AR Web SDK](#) first.

### Step 3. Initialize the Beauty AR Web SDK

Sample code:





```
const { ArSdk } = window.AR

/** ----- Authentication configuration ----- */

/**
 * The APPID of your Tencent Cloud account.
 *
 * You can view your APPID in the [Account Center](https://console.tencentcloud.com
 */
const APPID = ''; // Set it to your Tencent Cloud account APPID.
```

```
/**
 * Web LicenseKey
 *
 * Log in to the RT-Cube console and click [Web Licenses] (https://console.tencentcloud.com/rtcube/web-licenses)
 */
const LICENSE_KEY = ''; // Set it to your license key.

/**
 * The token used to calculate the signature.
 *
 * Note: This method is only suitable for debugging. In a production environment, y
 * [Signature] (https://cloud.tencent.com/document/product/616/71370#E7.AD.BE.E5.90)
 */
const token = ''; // Set it to your token.

/** ----- */

/**
 * Get the signature
 *
 * Note: This method is only suitable for debugging. In a production environment, y
 ** Example:
 * async function () {
 *   return fetch('http://xxx.com/get-ar-sign').then(res => res.json());
 * };
 */
const getSignature = function () {
  const timestamp = Math.round(new Date().getTime() / 1000);
  const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
  return { signature, timestamp };
};

let w = 1280;
let h = 720;

// The basic settings for the Tencent Effect SDK
const config = {
  camera: { //This indicates that the SDK will capture streams from the camera.
    width: 1280,
    height: 720
  },
  auth: {
    licenseKey: LICENSE_KEY,
    appId: APPID,
    authFunc: getSignature
  },
  // Configure the initial effects (optional)
```

```
beautify: {
  whiten: 0.1, // The brightening effect. Value range: 0-1.
  dermabrasion: 0.5, // The smooth skin effect. Value range: 0-1.
  lift: 0.3, // The slim face effect. Value range: 0-1.
  shave: 0, // The V shape effect. Value range: 0-1.
  eye: 0, // The big eyes effect. Value range: 0-1.
  chin: 0, // The chin effect. Value range: 0-1.
},
language: 'en',
.....
}

// Pass `config` to the Tencent Effect SDK
const ar = new ArSdk(config);

// You can display the effect and filter list in the `created` callback.
ar.on('created', () => {
  // Get the built-in makeup effects and stickers
  ar.getEffectList({
    Type: 'Preset'
  }).then((res) => {
    const list = res.map(item => ({
      name: item.Name,
      id: item.EffectId,
      cover: item.CoverUrl,
      url: item.Url,
      label: item.Label,
      type: item.PresetType,
    }));
    const makeupList = list.filter(item=>item.label.indexOf('Makeup')>=0)
    const stickerList = list.filter(item=>item.label.indexOf('Sticker')>=0)
    // Show the makeup and sticker lists
    renderMakeupList(makeupList);
    renderStickerList(stickerList);

  }).catch((e) => {
    console.log(e);
  });
  // Get the built-in filters
  ar.getCommonFilter().then((res) => {
    const list = res.map(item => ({
      name: item.Name,
      id: item.EffectId,
      cover: item.CoverUrl,
      url: item.Url,
      label: item.Label,
      type: item.PresetType,
```

```

    }));

    // Show the filter list
    renderFilterList(list);

  }).catch((e) => {
    console.log(e);
  });
});

ar.on('ready', (e) => {

  // After receiving the `ready` callback, you can call `setBeautify`, `setEffect`

  // For example, you can use `range input` to set the smooth skin effect:
  $('#dermabrasion_range_input').change((e) => {
    ar.setBeautify({
      dermabrasion: e.target.value, // The smooth skin effect. Value range: 0
    });
  });

  // In the `created` callback, apply the effects based on user interactions with
  $('#makeup_list li').click(() => {
    ar.setEffect([{'id': effect.id, intensity: 1}]);
  });
  $('#sticker_list li').click(() => {
    ar.setEffect([{'id': effect.id, intensity: 1}]);
  });

  // In the `created` callback, apply the filter based on user interactions with
  ar.setFilter(filterList[0].id, 1);
  $('#filter_list li').click(() => {
    ar.setFilter(filter.id, 1);
  });
});

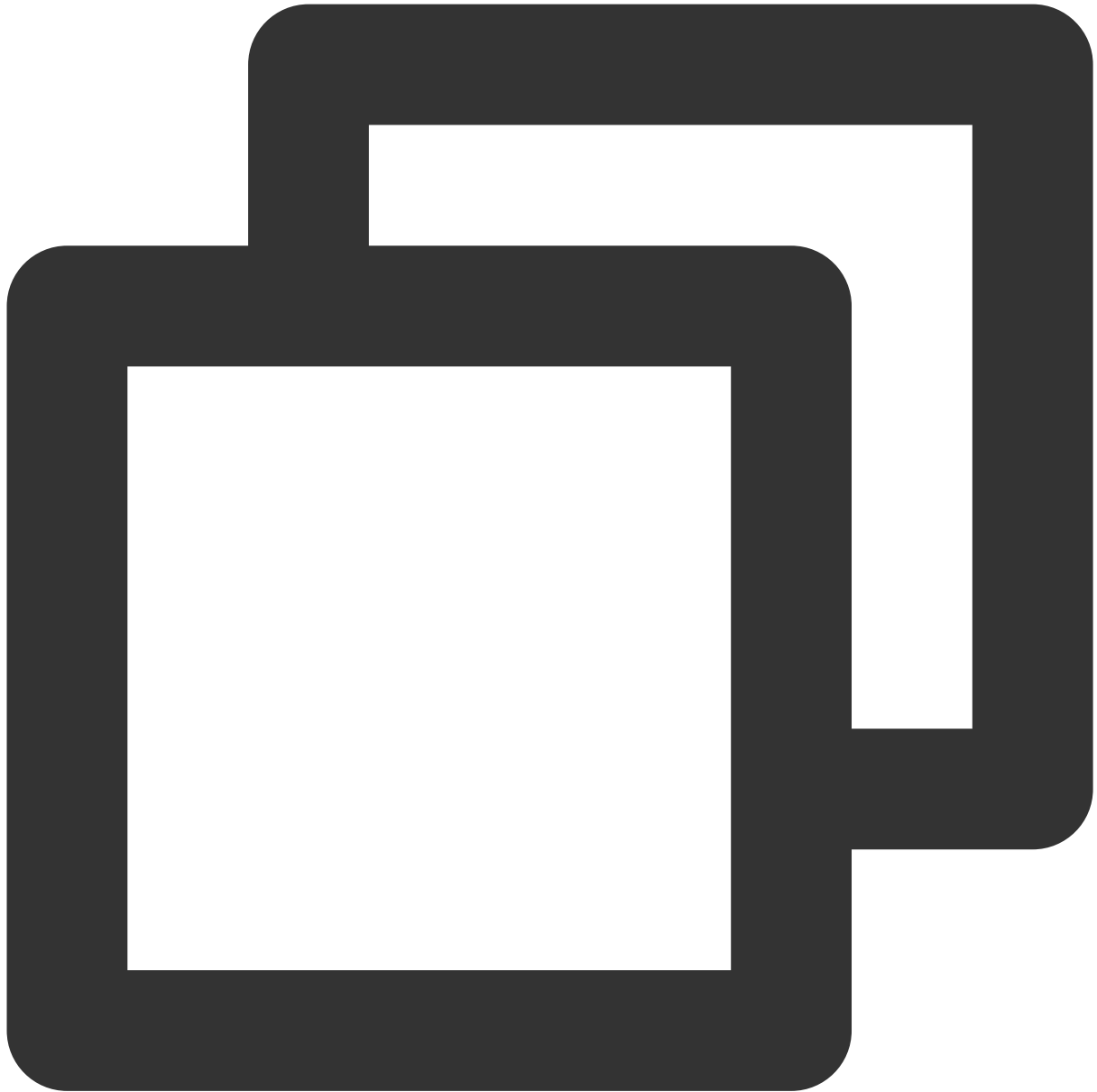
ar.on('error', (e) => {
  console.log(e);
});

```

The code above shows you how to initialize the Beauty AR Web SDK, as well as how to respond to user interactions in the `ready` callback. To learn more about UI interactions, you can download the code package at the end of this document.

#### Step 4. Initialize a TRTC stream

After initializing the Beauty AR Web SDK, you can use `getOutput` to get the stream processed by Tencent Effect and use it to initialize a TRTC stream as described in [step 2](#). Below is an example:



```
// Get the output stream of the Tencent Effect SDK
const arStream = await this.ar.getOutput();

const audioSource = arStream.getAudioTracks()[0];
const videoSource = arStream.getVideoTracks()[0];

// create a local stream with audio/video from custom source
this.localStream_ = TRTC.createStream({
```

```
    audioSource,  
    videoSource  
  });
```

## Step 5. Play the stream

### Note:

For the example project, you need to start the web service of your device and make sure that the HTML file can be accessed via the specified port.

After entering the room, you should be able to view the stream with effects applied ( `index_AR.html` in the sample code). After that, you can open a new browser tab and enter the room to simulate the entry of a new user.

## Step 6. Control devices

If you use the camera video as the input stream, the Beauty AR Web SDK also offers APIs for you to control the camera.



```
const cameraApi = this.ar.camera;
// Get the device list
const devices = await cameraApi.getDevices()
console.log(devices)
// Switch to a different camera
// await cameraApi.switchDevice('video', 'your-video-device-id')
// Disable the video track
// cameraApi.muteVideo()
// Enable the video track
// cameraApi.unmuteVideo()
// Disable the audio track
```

```
// cameraApi.muteAudio()  
// Enable the audio track  
// cameraApi.unmuteAudio()  
// Stop the camera  
// cameraApi.stop()  
// Restart the camera  
// await cameraApi.restart()
```

## Step 7. Preview effects locally

If you use the camera video as the input stream, the Beauty AR Web SDK also supports local preview.





```
// Use the built-in player of the SDK. `my-dom-id` is the ID of the player's container
const player = await sdk.initLocalPlayer('my-dom-id')
// Play the video
await player.play()
// Pause the video
player.pause()
```

## Sample Code Package

You can download our sample code package [here](#).

The main changes are in `index_AR.html` and `rtc-client-with-webar.js`. The code for the interaction logic of Tencent Effect is in `base-js/js/ar_interact.js`. The configuration of TRTC key information is in `base-js/js/debug/GenerateTestUserSig.js`.

# Using Beauty AR Web with Mini Program

Last updated : 2023-04-21 15:34:37

## Preparations

For how to get started with Mini Program development, see the [Weixin Mini Program document](#).

Read [Overview](#) to learn about how to use the Beauty AR Web SDK.

## Directions

### Step 1. Configure a domain allowlist on the Mini Program backend

As the SDK will request the backend to perform authentication and load resources, after creating your Mini Program, you need to configure a domain allowlist on the backend.

1. Open the [Mini Program backend](#) and go to **Development > Development Management > Development Settings > Server Domain Name**.

2. Click **Modify**, configure the following domain names and save them.

Request domain names:



```
https://webar.qcloud.com;  
https://webar-static.tencent-cloud.com;  
https://aegis.qq.com;  
The URL of the authentication signature API (`get-ar-sign`)
```

downloadFile domain name:



```
https://webar-static.tencent-cloud.com
```

## Step 2. Install and build the npm package

For information about Mini Program npm packages, see [Using npm in Mini Program](#).

1. Install:



```
npm install tencentcloud-webar
```

2. Build:

Open Weixin DevTools and select **Tools > Build npm** on the topbar.

3. Configure the path of `workers` in `app.json` :



```
"workers": "miniprogram_npm/tencentcloud-webar/worker"
```

### Step 3. Import the files



```
// The import method for versions earlier than 0.3.0 (one file)
// import "../../miniprogram_npm/tencentcloud-webar/lib.js";
// The import method for v0.3.0 or later (two files and the 3D module (optional))
import '../../miniprogram_npm/tencentcloud-webar/lib.js';
import '../../miniprogram_npm/tencentcloud-webar/core.js';
// Initialize the 3D plugin (optional)
import '../../miniprogram_npm/tencentcloud-webar/lib-3d.js';
import { plugin3d } from '../../miniprogram_npm/tencentcloud-webar/plugin-3d'
// Import `ArSdk`
import { ArSdk } from "../../miniprogram_npm/tencentcloud-webar/index.js";
```



**Note**

Because Mini Program has a 500 KB limit for file size, the SDK is provided as two JS files.

Starting from v0.3.0, the SDK is further split to support 3D. The 3D module can be loaded as needed. Before import, check your SDK version and use the corresponding import method.

**Step 4. Initialize the SDK****Note**

Before initializing the SDK, you need to configure the Mini Program `APPID` in the RT-Cube console as instructed in [Getting Started](#).

You need to insert the `camera` tag to open the camera and then set the camera parameters as detailed in [Overview](#).

Mini Program does not support `getOutput`, so you need to pass in an onscreen WebGL canvas. The SDK will output images onto this canvas.

Sample code:



```
// wxml
// Open the camera and hide it by setting `position`
<camera
  device-position="{{'front'}}"
  frame-size="large" flash="off" resolution="medium"
  style="width: 750rpx; height: 134rpx;position:absolute;top:-9999px;"
/>
// The SDK outputs the processed image to the canvas in real time.
<canvas
  type="webgl"
  canvas-id="main-canvas"
```

```
    id="main-canvas"
    style="width: 750rpx; height: 1334rpx;">
</canvas>
// Take a photo and draw the `ImageData` object onto the canvas
<canvas
  type="2d"
  canvas-id="photo-canvas"
  id="photo-canvas"
  style="position:absolute;width:720px;height:1280px;top:-9999px;left:-9999px;">
</canvas>
// js
/** ----- Authentication configuration ----- */

/**
 * Your Tencent Cloud account's APPID
 *
 * You can view your APPID in the [Account Center] (https://console.cloud.tencent.co)
 */
const APPID = ''; // Enter your APPID

/**
 * Web LicenseKey
 *
 * On the [Web Licenses] (https://console.cloud.tencent.com/vcube/web) page of the R
 */
const LICENSE_KEY = ''; // Enter the license key of your project

/**
 * The token used to calculate the signature
 *
 * Note: The sample code is for demo debugging only. In the production environment,
 * [Signature algorithm] (https://cloud.tencent.com/document/product/616/71370#.E7.A)
 */
const token = ''; // Enter your token

Component({
  data: {
    makeupList: [],
    stickerList: [],
    filterList: [],
    recording: false
  },
  methods: {
    async getCanvasNode(id) {
      return new Promise((resolve) => {
        this.createSelectorQuery()
          .select(`#${id}`)

```

```
        .node()
        .exec((res) => {
            const canvasNode = res[0].node;
            resolve(canvasNode);
        });
    });
},
getSignature() {
    const timestamp = Math.round(new Date().getTime() / 1000);
    const signature = sha256(timestamp + token + APPID + timestamp).toUpperCase();
    return { signature, timestamp };
},
// Initialize the camera type
async initSdkCamera() {
    // Get the onscreen canvas. The SDK will output the processed image to
    const outputCanvas = await this.getCanvasNode("main-canvas");
    // Get the authentication information
    const auth = {
        licenseKey: LICENSE_KEY,
        appId: APP_ID,
        authFunc: this.getSignature
    };
    // Construct SDK initialization parameters
    const config = {
        auth,
        camera: {
            width: 720,
            height: 1280,
        },
        output: outputCanvas,
        // Initial beauty effects (optional)
        beautify: {
            whiten: 0.1, // The brightening effect. Value range: 0-1.
            dermabrasion: 0.3, // The smooth skin effect. Value range: 0-1.
            lift: 0, // The slim face effect. Value range: 0-1.
            shave: 0, // The V shape effect. Value range: 0-1.
            eye: 0.2, // The big eyes effect. Value range: 0-1.
            chin: 0, // The chin effect. Value range: 0-1.
        }
    };
    const ar = new ArSdk(config);
    // The list of built-in effects and filters can be obtained in the `create`
    ar.on('created', () => {
        // Get the list of built-in makeup effects and stickers
        ar.getEffectList({
            Type: 'Preset'
        }).then((res) => {
```

```
const list = res.map(item => ({
  "name": *item.Name,
  id: item.EffectId,
  cover: item.CoverUrl,
  url: item.Url,
  label: item.Label,
  type: item.PresetType,
}));
const makeupList = list.filter(item=>item.label.indexOf('makeup') > -1);
const stickerList = list.filter(item=>item.label.indexOf('stick') > -1);
// Render the list of effects
this.setData({
  makeupList,
  stickerList
});
}).catch((e) => {
  console.log(e);
});
// Built-in filters
ar.getCommonFilter().then((res) => {
  const list = res.map(item => ({
    "name": *item.Name,
    id: item.EffectId,
    cover: item.CoverUrl,
    url: item.Url,
    label: item.Label,
    type: item.PresetType,
  }));
  // Render the list of filters
  this.setData({
    filterList: list
  });
}).catch((e) => {
  console.log(e);
});
});
// You can set beauty filters and effects in the `ready` callback.
ar.on('ready', (e) => {
  this._sdkReady = true
});

ar.on('error', (e) => {
  console.log(e);
});

this.ar = ar
},
```

```
// Change the beauty effects. Make sure the SDK is ready.
onChangeBeauty(val){
    if(!this._sdkReady) return
    // You can set beauty effects through `setBeautify`. Six attributes are
    this.ar.setBeautify({
        dermabrasion: val.dermabrasion, // The smooth skin effect. Value ra
    });
},
// Change the makeup style. Make sure the SDK is ready.
onChangeMakeup(id, intensity){
    if(!this._sdkReady) return
    // Use `setEffect` to configure the effect. Its input parameters can be
    this.ar.setEffect([id, intensity]);
},
// Change the sticker. Make sure the SDK is ready.
onChangeSticker(id, intensity){
    if(!this._sdkReady) return
    // Use `setEffect` to configure the effect. Its input parameters can be
    this.ar.setEffect([id, intensity]);
},
// Change the filter. Make sure the SDK is ready.
onChangeFilter(id, intensity){
    if(!this._sdkReady) return
    // Use `setFilter` to configure the filter. The second parameter indica
    this.ar.setFilter(id, 1);
}
}
})
```

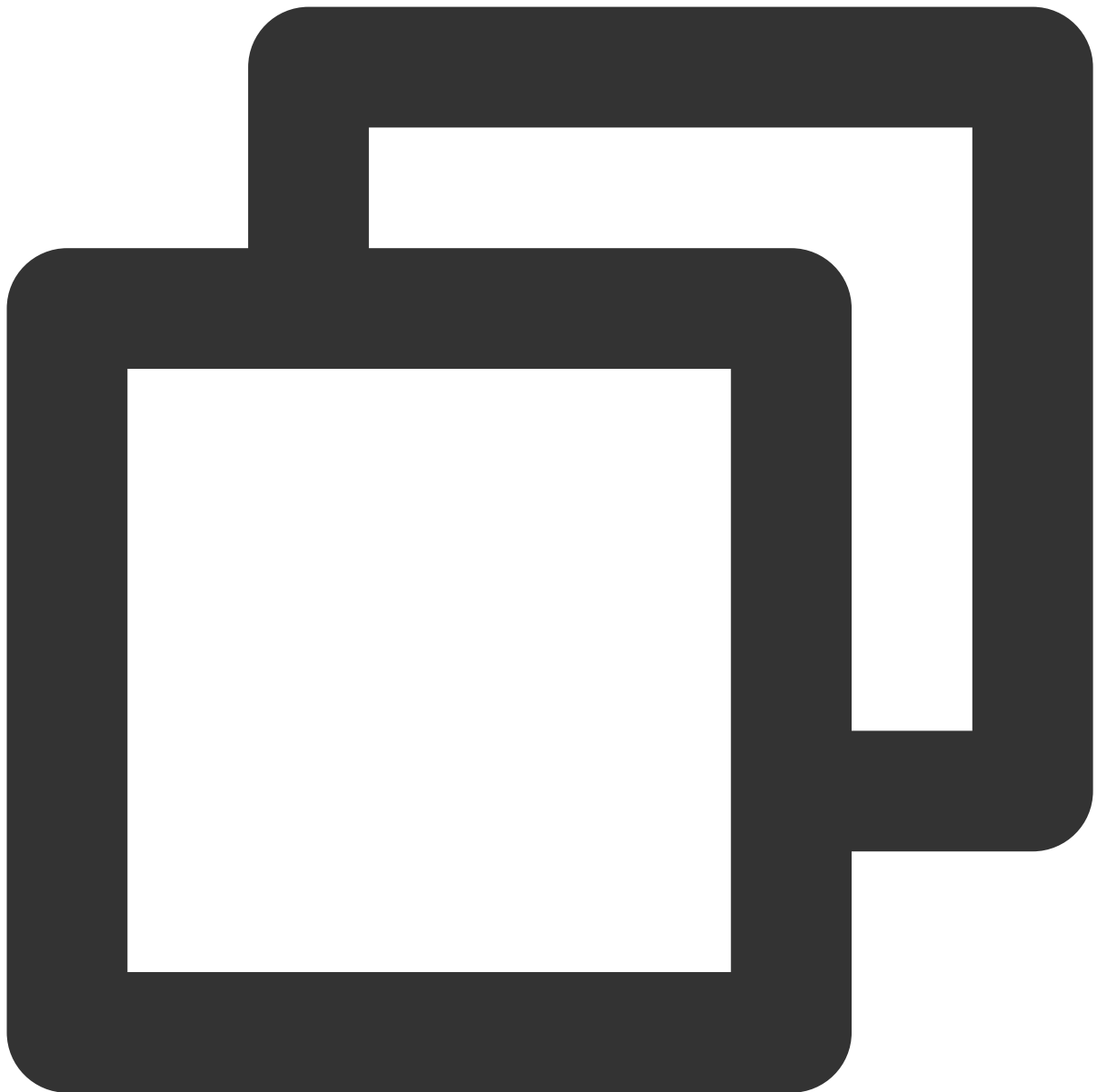
## Step 5. Implement the photo capturing and shooting features

Sample code:

Photo

Shooting

The SDK will return an object containing the width, height, and buffer data, and you can draw the data on the preset 2D canvas (in the above code, `id` is `photo-canvas` .) on your page and export it as an image file.



```
async takePhoto() {  
  const {uint8ArrayData, width, height} = this.ar.takePhoto(); // The `takePhoto`  
  const photoCanvasNode = await this.getCanvasNode('photo-canvas');  
  photoCanvasNode.width = parseInt(width);  
  photoCanvasNode.height = parseInt(height);  
  const ctx = photoCanvasNode.getContext('2d');  
  // Create an `ImageData` object with the data returned by the SDK  
  const imageData = photoCanvasNode.createImageData(uint8ArrayData, width, height);  
  // Draw the `ImageData` object onto the canvas  
  ctx.putImageData(imageData, 0, 0, 0, 0, width, height);  
  // Save the canvas as a local image
```

```
wx.canvasToTempFilePath({
  canvas: photoCanvasNode,
  x: 0,
  y: 0,
  width: width,
  height: height,
  destWidth: width,
  destHeight: height,
  success: (res) => {
    // Save the photo
    wx.saveImageToPhotosAlbum({
      filePath: res.tempFilePath
    });
  }
})
}
```





```
Component ({
  methods: {
    // Start shooting
    startRecord() {
      this.setData({
        recording: true
      });
      this.ar.startRecord()
    }
    // Stop shooting
    async stopRecord() {
```

```
        const res = await this.ar.stopRecord();  
        // Save the video  
        wx.saveVideoToPhotosAlbum({  
            filePath: res.tempFilePath  
        });  
        this.setData({  
            recording: false  
        });  
    }  
}  
})
```

When the Mini Program is switched to the background or the screen is locked, call `stopRecord` to stop shooting.

When the page is opened again, start the SDK again.



```
onShow() {  
    this.ar && this.ar.start();  
},  
onHide() {  
    this.ar && this.ar.stop();  
},  
async onUnload() {  
    try {  
        this.ar && this.ar.stop();  
        if (this.data.recording) {  
            await this.ar.stopRecord({
```

```
        destroy: true,  
      });  
    }  
  } catch (e) {  
  }  
  this.ar && this.ar.destroy();  
}
```

## Sample Code Package

You can download our sample code package [here](#). The code for Mini Program is in `ar-miniprogram` .

# FAQs

Last updated : 2024-05-08 16:34:16

This document answers questions you may encounter when using Beauty AR Web.

## What should I do if the image is upside down and lags when I run the demo in Chrome?

As the SDK uses the GPU for acceleration, you need to toggle on **Use hardware acceleration when available** in the browser's settings.

## Can I use the Beauty AR Web SDK to beautify live streams published in a web live streaming application?

Yes. The SDK can work as an intermediate rendering processor for live streaming. It supports multiple input/output sources. For information about how to easily extend your web application and quickly implement beauty filters and effects, see [Publishing over WebRTC](#) and [Publishing with TRTC](#).

## Will my signature service be frequently called?

No, because the SDK internally has a signature caching mechanism. You can also customize the return logic in your own `getSignature` method, as long as the signature algorithm is compliant.

## Will the effect displayed after the call to the SDK differ from the effect previewed in the customization tool in the console?

No. The effect that is rendered from the SDK is the same as the effect previewed in the customization tool; in other words, what you see in the preview is the same effect you will see in the actual production environment.

## How do I use the `localhost` for local development?

You can create a trial license and specify the `localhost` and port number in the domain (with a port limit). Alternatively, you can purchase an official license. During the validity period of the official license, you can use the `localhost` for local development (with no port limit).

## Why is "streamurl authentication failed" displayed in the console after LEB publishing failed?

This is usually because the signature expired. You need to generate a new signature for publishing. For details about the format of publishing URLs, see [Splicing Live Streaming URLs](#).

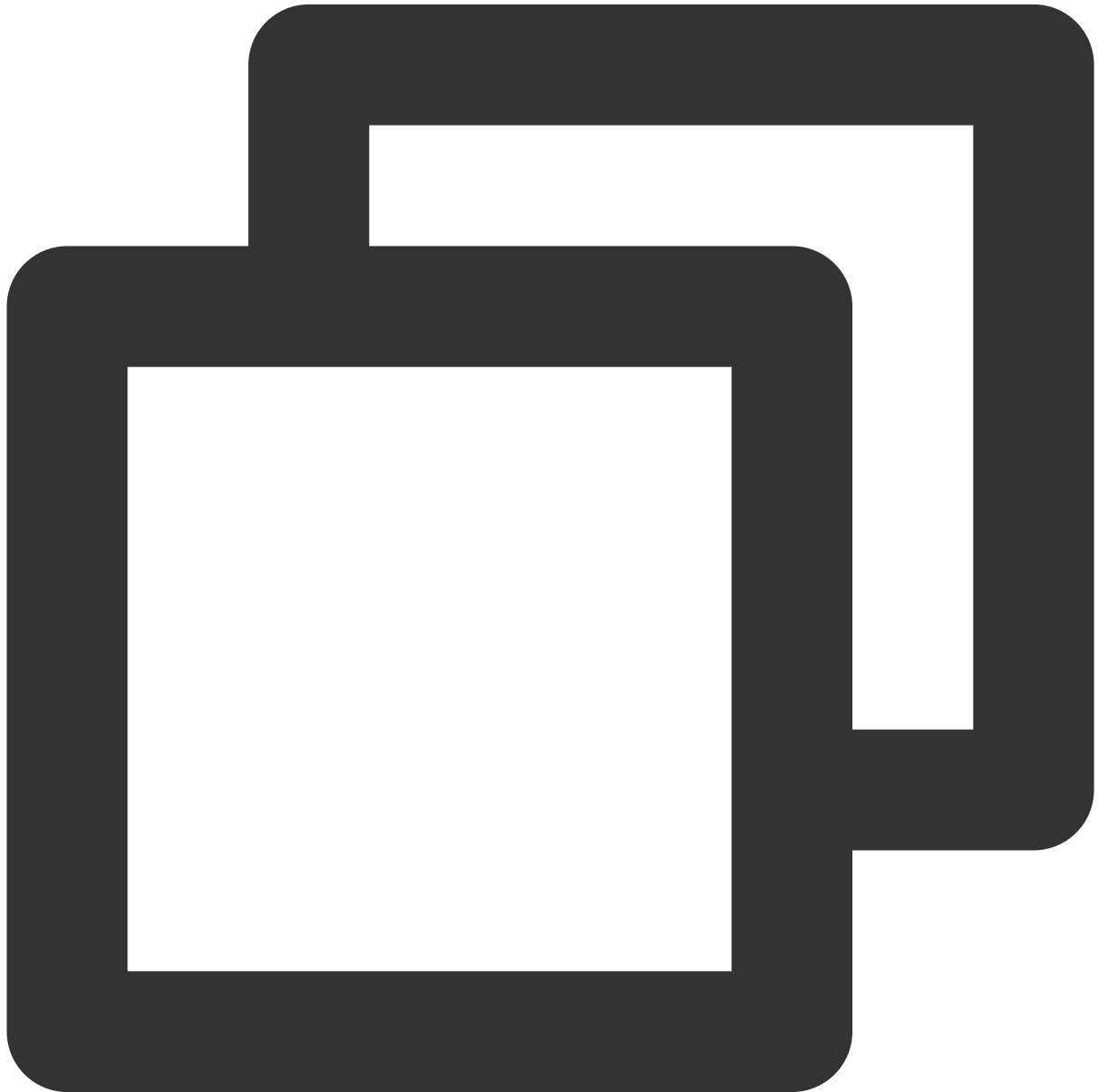
## Why is an error reported when I call `getEffectList` to pull material resources?

This is usually because the timing of the call is incorrect. Be sure to call `getEffectList` **when or after the `created` API of the SDK is called back**. At that point, the business interaction logic can be implemented based

on the material data. For the specific use cases, see [Best Practices](#).

### **Why does echo occur when I preview the video effect after web is connected to the built-in camera of the SDK?**

When audio is turned on for local preview and played back, the audio will be captured by the mic and used as the audio source of the built-in camera, which causes echo. To solve this problem, mute the preview video.



```
const output = await sdk.getOutput()  
const video = document.createElement('video')  
document.body.appendChild(video)
```

```
video.setAttribute('muted', '')  
video.volume = 0  
video.srcObject = output
```

## Why does the SDK report an authentication failure and the API returns 401?

The SDK internally requests a signature through the `getSignature` method passed through by the parameter and authentication from the backend. It will automatically retry once after the timestamp of the signature expires and will report an error and block all subsequent processes if the retry fails. You can check the logic of `getSignature` to see if the timestamp (valid for five minutes) expires or the signature generation logic is incorrect.