

# **Mobile Live Video Broadcasting Publishing 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

## Publishing

### iOS

Publishing from Camera

Publishing from Screen

### Android

Publishing from Camera

Publishing from Screen

### Web

### Flutter

Publishing from Camera

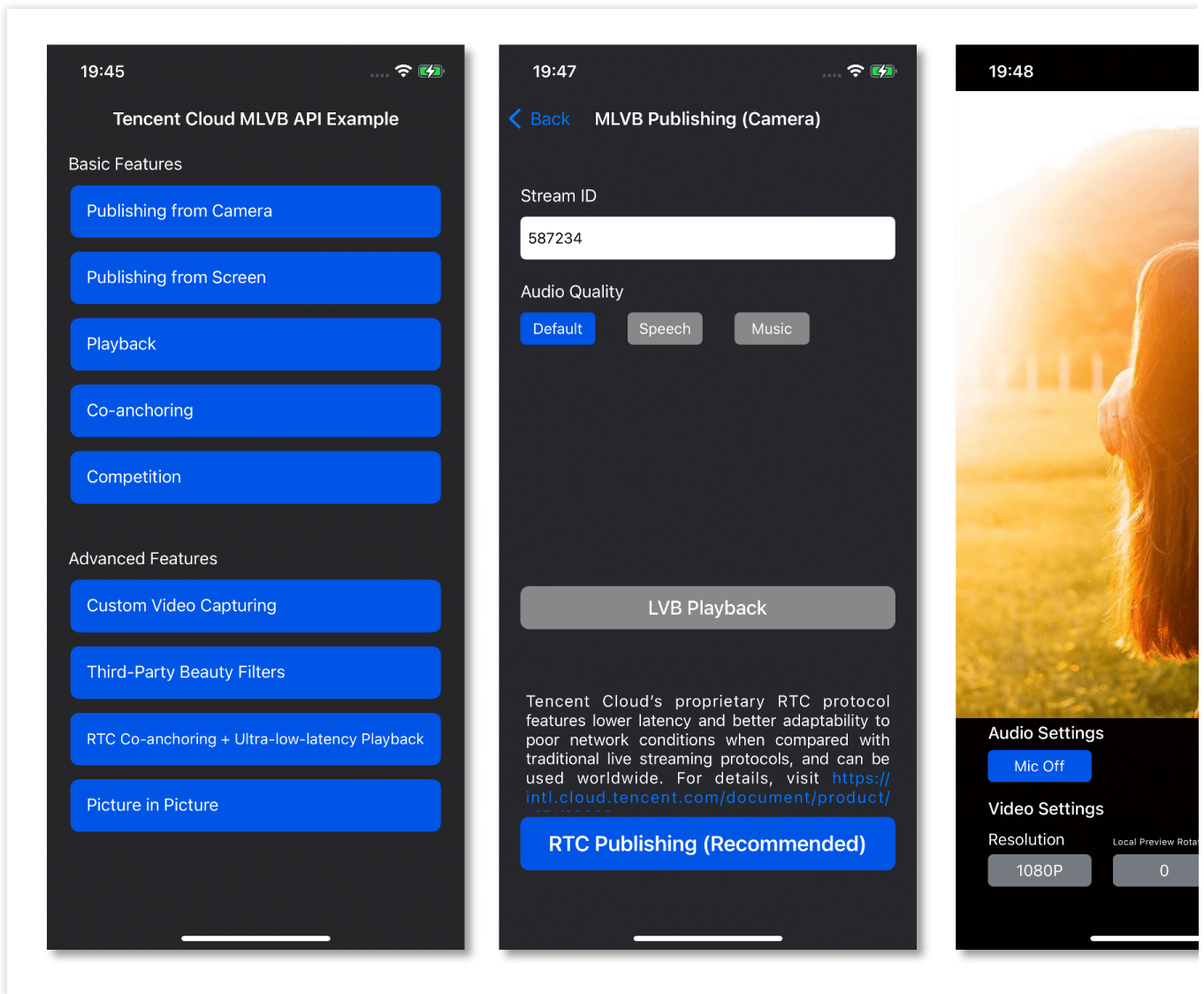
Publishing from Screen

# Publishing iOS Publishing from Camera

Last updated : 2024-01-13 15:49:41

## Overview

Publishing from camera refers to the process of collecting video and audio data from the mobile phone's camera and mic, encoding the data, and publishing it to cloud-based live streaming platforms. Tencent Cloud's LiteAVSDK provides the camera publishing capability via `V2TXLivePusher`, the following is the relevant operation interface of the demo camera in the simple version of LiteAVSDK:



## Notes

**About running projects on x86 emulators:** The SDK uses a lot of audio and video APIs of the iOS system, most of which cannot be used on the x86 emulator built into macOS. Therefore, we recommend that you test your project on a real device.

## Sample Code

Platform	GitHub Address	Key Class
iOS	<a href="#">Github</a>	CameraPushViewController.m

Android	<a href="#">Github</a>	CameraPushMainActivity.java
Flutter	<a href="#">Github</a>	live_camera_push.dart

## Integration

### 1. Download the SDK

[Download](#) the SDK and follow the instructions in [SDK Integration](#) to integrate the SDK into your application.

### 2. Configure License Authorization for SDK

#### 1. Obtain license authorization :

If you have obtained the relevant license authorization, Need to Get License URL and License Key in [Cloud Live Console](#)

**Create Official License**

Price Overview

Create

An official license is valid for a year. Click [Create] to purchase one. Please make sure that the bundle ID and package name entered are correct as the information cannot be modified after submission.

Official License

Live

Application Name

Package Name

Bundle Id

Key

LicenseUrl

Start Date

End Date

com

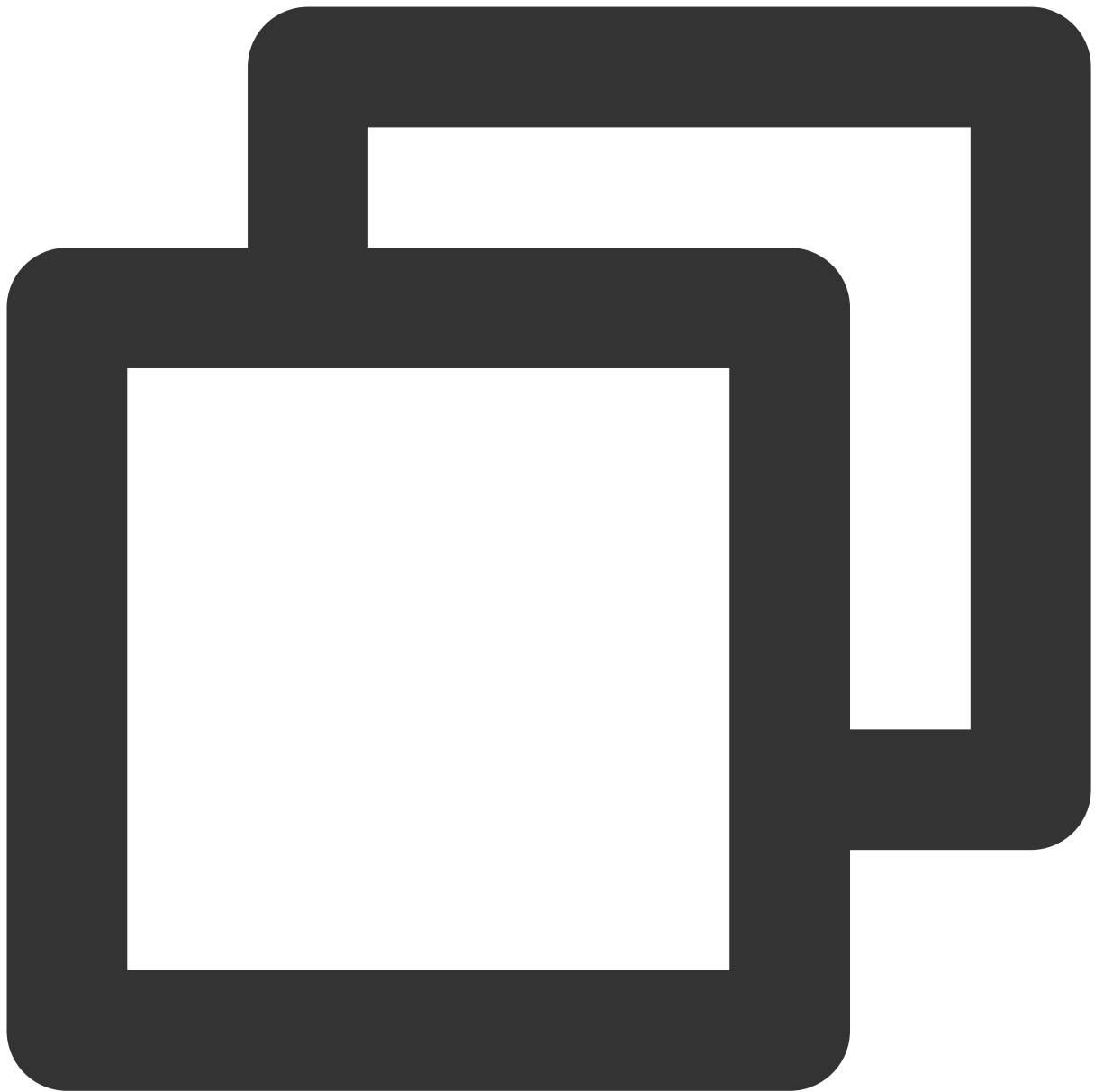
com

2021-12-03

2022-12-03

If you have not yet obtained the license authorization, Please reference [Adding and Renewing Licenses](#) to make an application.

2. Before your App calls SDK-related functions (it is recommended in the `- [AppDelegate application:didFinishLaunchingWithOptions:]` ), set the following settings:



```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    NSString * const licenceURL = @"<your licenseUrl>";  
    NSString * const licenceKey = @"<your key>";  
  
    // V2TXLivePremier is located in "V2TXLivePremier.h"  
    [V2TXLivePremier setEnvironment:@"GDPR"]; // set environment  
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];  
    [V2TXLivePremier setObserver:self];  
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);  
    return YES;  
}
```

```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

**Note:**

The BundleId configured in the license must be the same as the application itself, otherwise the streaming will fail.

### 3. Initialize the `V2TXLivePusher` component

Create a `V2TXLivePusher` object and specify `V2TXLiveMode` .

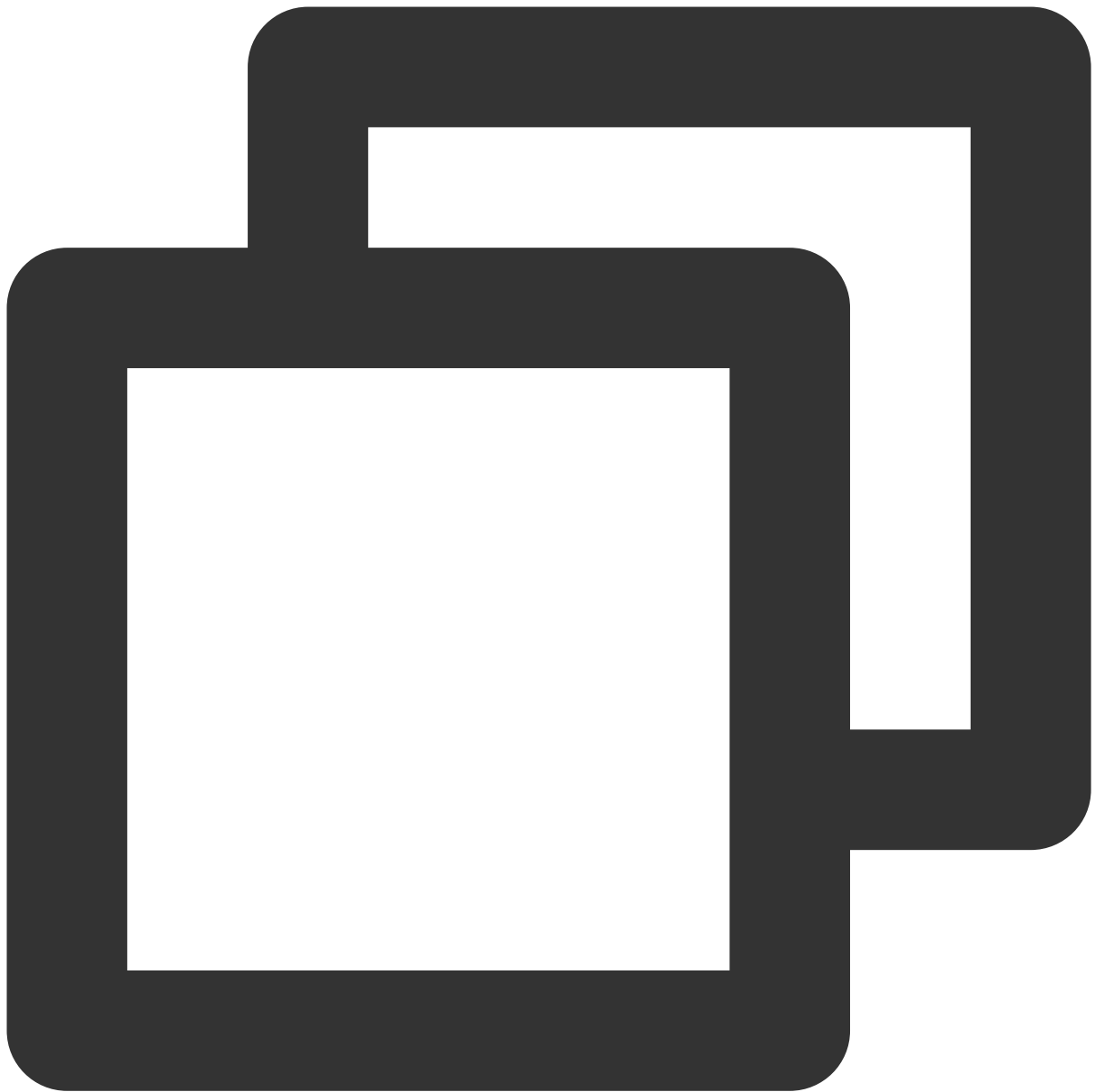




```
// Specify the corresponding live broadcast protocol as RTMP, which does not support  
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP
```

#### 4. Enable camera preview

Call `setRenderView` in [V2TXLivePusher](#) to configure a view object for displaying video images, and then call `startCamera` to enable camera preview for your mobile phone.

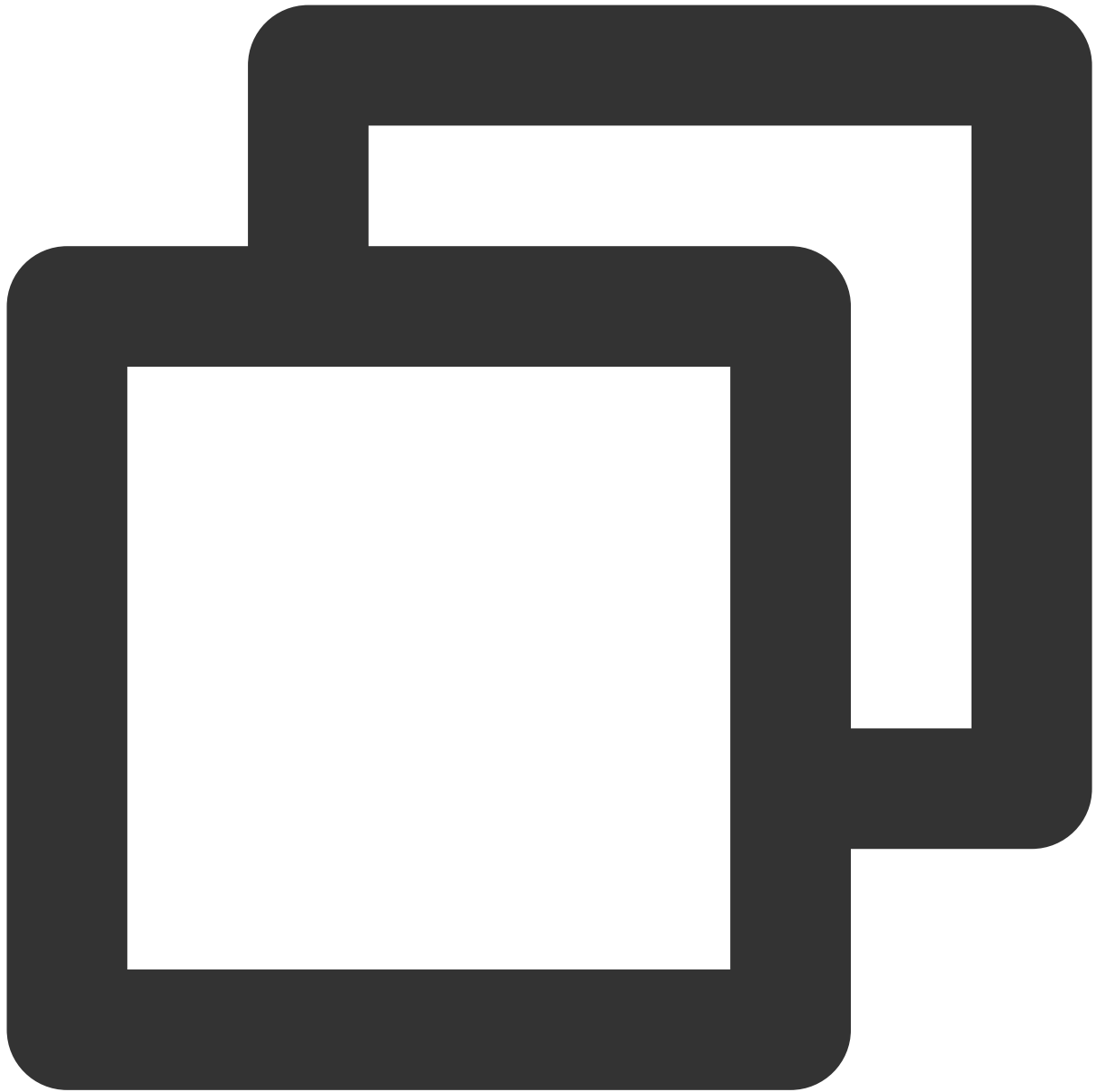


```
// Create a view object and insert it into the UI
UIView *_localView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view insertSubview:_localView atIndex:0];
_localView.center = self.view.center;

// Enable preview for the local camera
[_pusher setRenderView:_localView];
[_pusher startCamera:YES];
[_pusher startMicrophone];
```

**Note:**

To add animated effects to the view, modify its `transform` attribute rather than `frame` attribute.



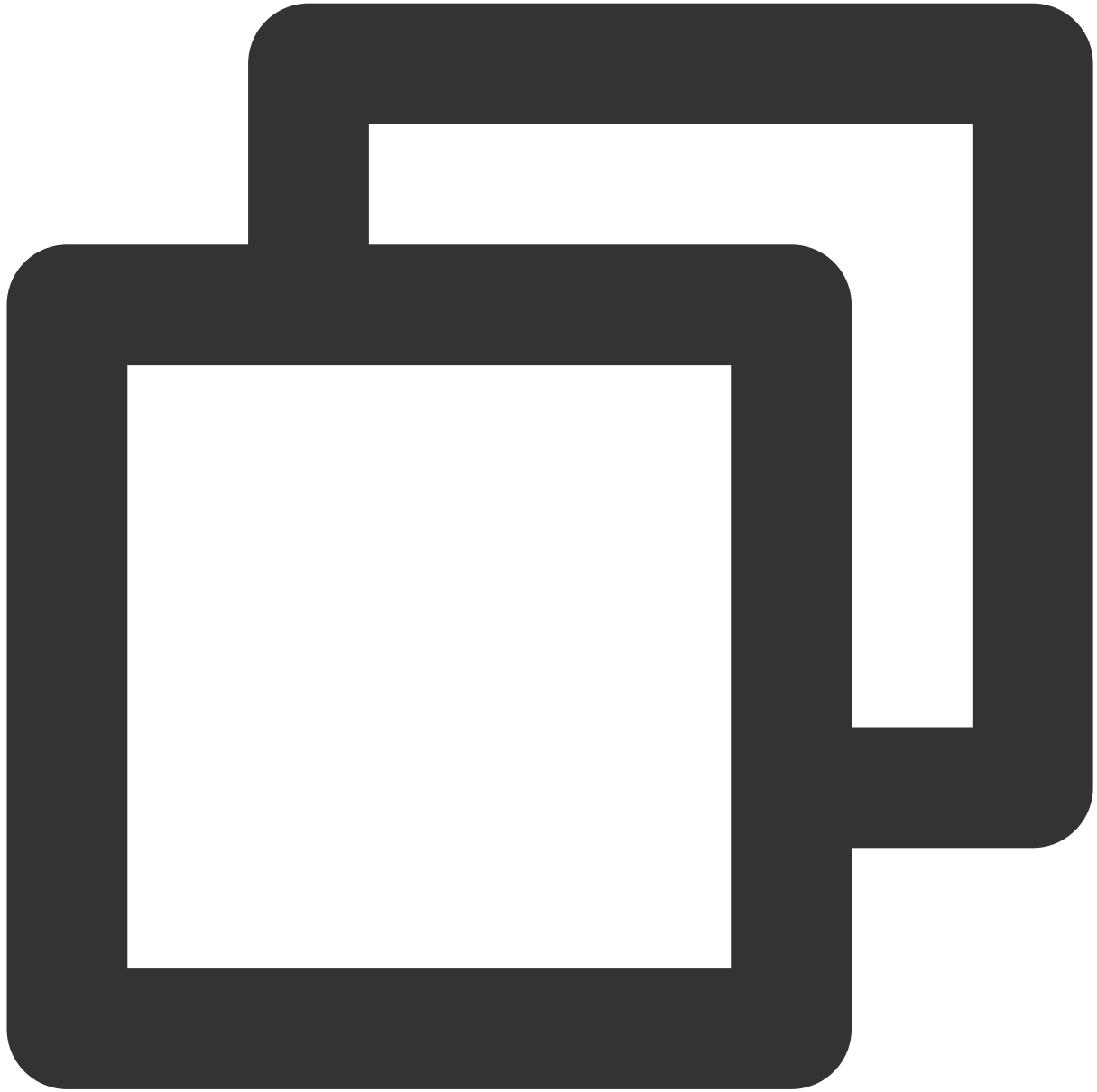
```
[UIView animateWithDuration:0.5 animations:^(
    _localView.transform = CGAffineTransformMakeScale(0.3, 0.3); // Shrink by 1/3
)];
```

## 5. Start and stop publishing

After calling `startCamera` to enable camera preview, you can call the `startPush` API in `V2TXLivePusher` to start publishing.

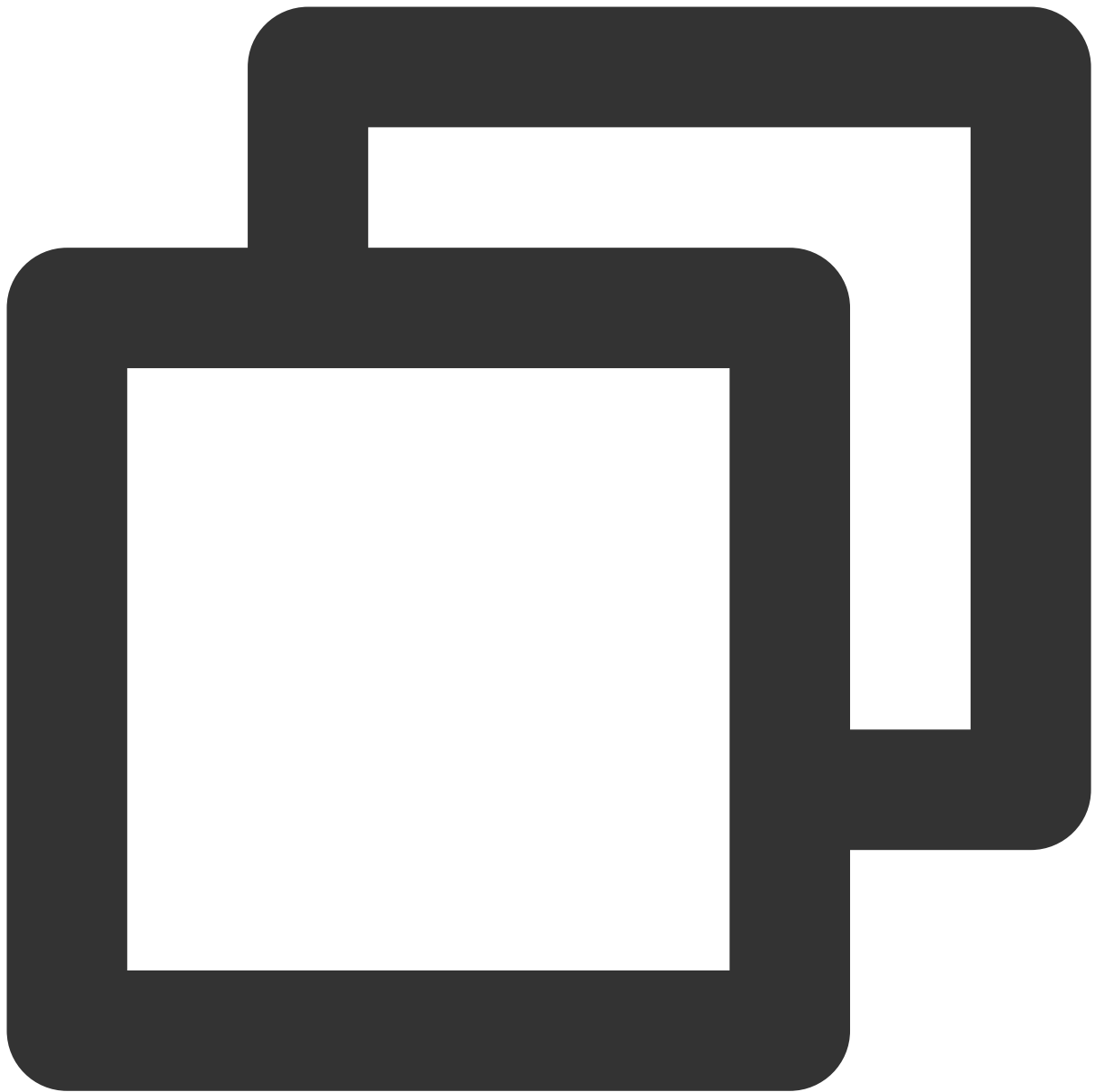
**Note:**

If the `RTMP` protocol is selected in [Step 3](#) to push the stream, please refer to the generation of the push stream URL [RTMP URL](#).



```
//This URL does not support co-anchoring. The stream is published to a live stream  
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxxx&txTime=xxxxxxxxx";  
[_pusher startPush:url];
```

Call [stopPush](#) in `V2TXLivePusher` to stop publishing streams.

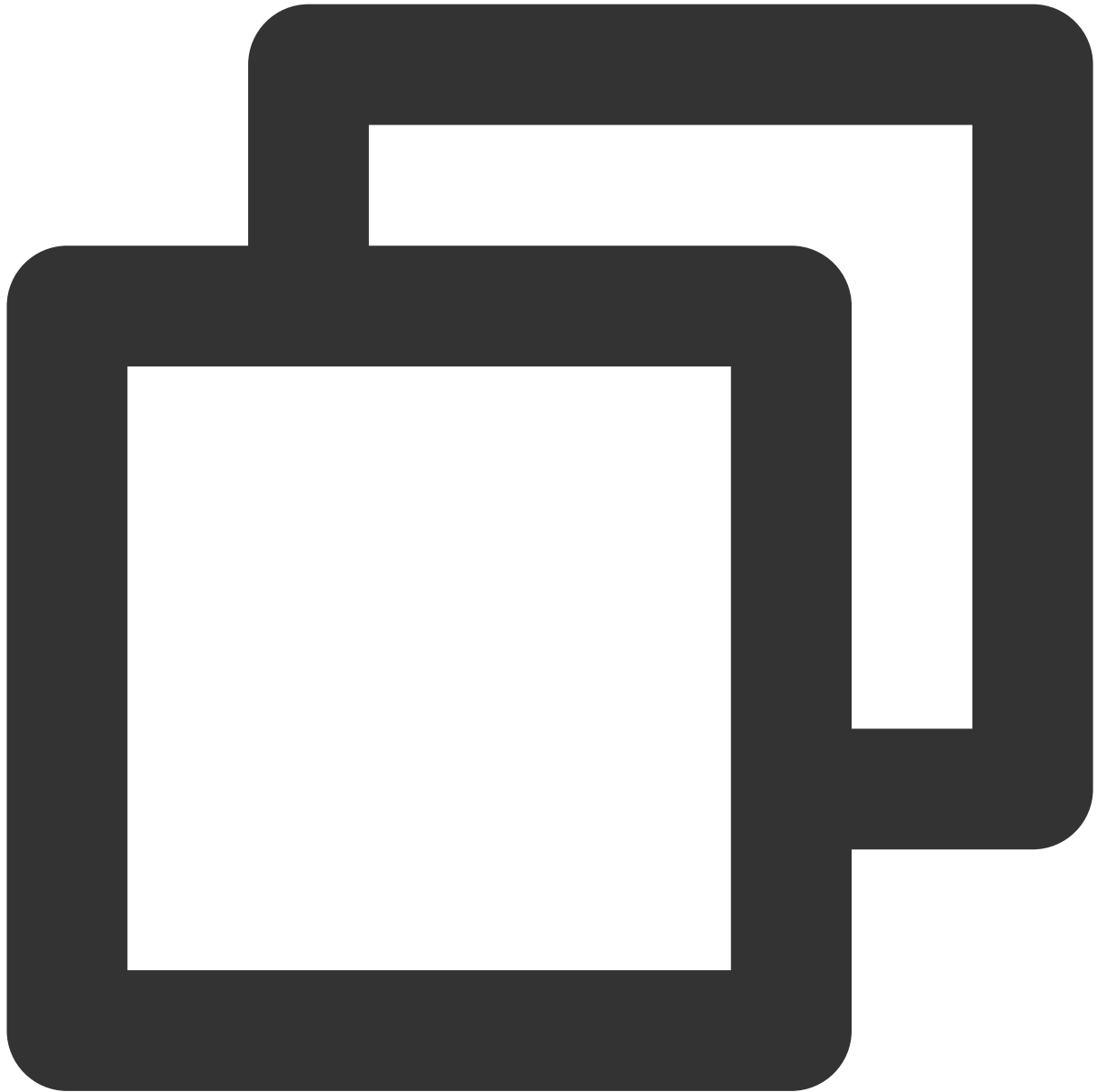


```
//Stop publishing  
[_pusher stopPush];
```

**Why is `V2TXLIVE_ERROR_INVALID_LICENSE` returned?** If the `startPush` API returns `V2TXLIVE_ERROR_INVALID_LICENSE`, it means your license verification failed. Please check your configuration against [Step 2. Configure License Authorization for SDK](#).

## 6. Publish audio-only streams

If your live streaming scenarios involve audio only, you can skip [Step 4](#) or do not call `startCamera` before `startPush` .



```
[_pusher startMicrophone];  
//This URL does not support co-anchoring. The stream is published to a live streami  
NSString* url = @"rtmp://test.com/live/streamid?txSecret=xxxxxx&txTime=xxxxxxxxx";  
[_pusher startPush:url];
```

**Note:**

If you publish audio-only streams but no streams can be pulled from an RTMP, FLV, or HLS playback URL, there is a problem with your line configuration, please [submit a ticket](#) for help.

## 7. Set video quality

Call [setVideoQuality](#) in `V2TXLivePusher` to set the quality of videos watched by audience. The encoding parameters set determine the quality of videos presented to audience. The local video watched by the host is the original HD version that has not been encoded or compressed, and is therefore not affected by the settings. For details, please see [Setting Video Quality](#).

## 8. Set the beauty filter style and skin brightening and rosy skin effects

Call [getBeautyManager](#) in `V2TXLivePusher` to get a `TXBeautyManager` instance to set beauty filters.

### Beauty filter style

The SDK has three built-in beauty filter algorithms, each corresponding to a beauty filter style. Choose one that best fits your product positioning. For details, please see the [TXBeautyManager.h](#) file.

Beauty Filter Style	Description
<code>TXBeautyStyleSmooth</code>	The smooth style, which features more obvious skin smoothing effects and is suitable for live showrooms
<code>TXBeautyStyleNature</code>	The natural style, which retains more facial details and is more natural
<code>TXBeautyStylePitu</code>	The Pitu style, which uses the beauty filter algorithm developed by YouTu Lab. Its effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and delivers more obvious skin smoothing effects than the natural style.

You can call the [setBeautyStyle](#) API in `TXBeautyManager` to set the beauty filter style.

Item	Configuration	Description
Beauty filter strength	Via the <code>setBeautyLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Skin brightening filter strength	Via the <code>setWhitenessLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Rosy skin filter strength	Via the <code>setRuddyLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.

## 9. Set color filters

Call `getBeautyManager` in `V2TXLivePusher` to get a `TXBeautyManager` instance to set color filters.

Call the `setFilter` API in `TXBeautyManager` to set color filters. Color filters are a technology that adjusts the color tone of sections of an image. For example, it may lighten the yellow sections of an image to achieve the effect of skin brightening, or add warm tones to a video to give it a refreshing and soft boost.

Call the `setFilterStrength` API in `TXBeautyManager` to set the strength of a color filter. The higher the strength, the more obvious the effect.

Based on our experience of operating Mobile QQ and Now Live, it's not enough to use only the `setBeautyStyle` API in `TXBeautyManager` to set the beauty filter style. The `setBeautyStyle` API must be used together with `setFilter` to produce richer effects. Given this, our designers have developed 17 built-in color filters for you to choose from.





```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"  
path = [path stringByAppendingPathComponent:lookupFileName];
```

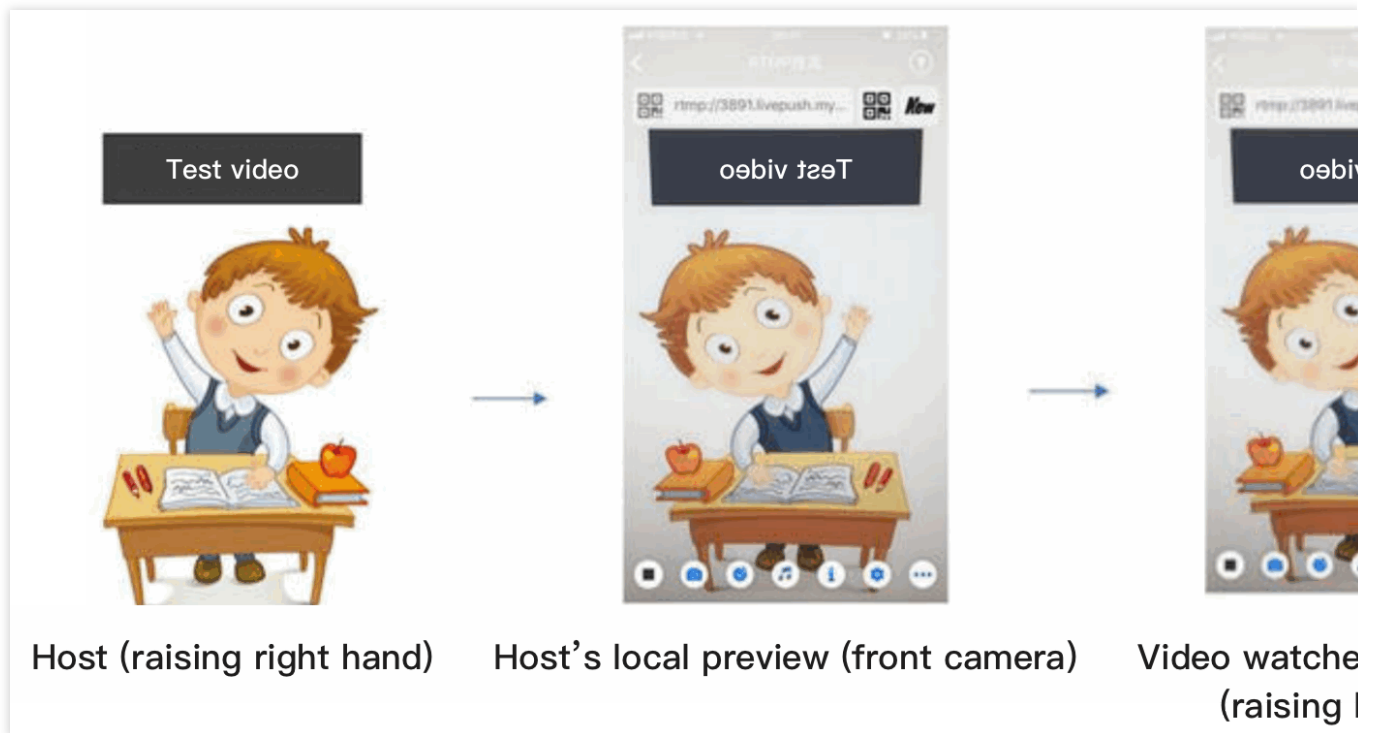
```
UIImage *image = [UIImage imageWithContentsOfFile:path];  
[[_pusher getBeautyManager] setFilter:image];  
[[_pusher getBeautyManager] setFilterStrength:0.5f];
```

## 10. Manage devices

`V2TXLivePusher` provides a series of APIs for the control of devices. You can call `getDeviceManager` to get a `TXDeviceManager` instance for device management. For detailed instructions, please see [TXDeviceManager API](#).

## 11. Set the video mirroring effect for audience

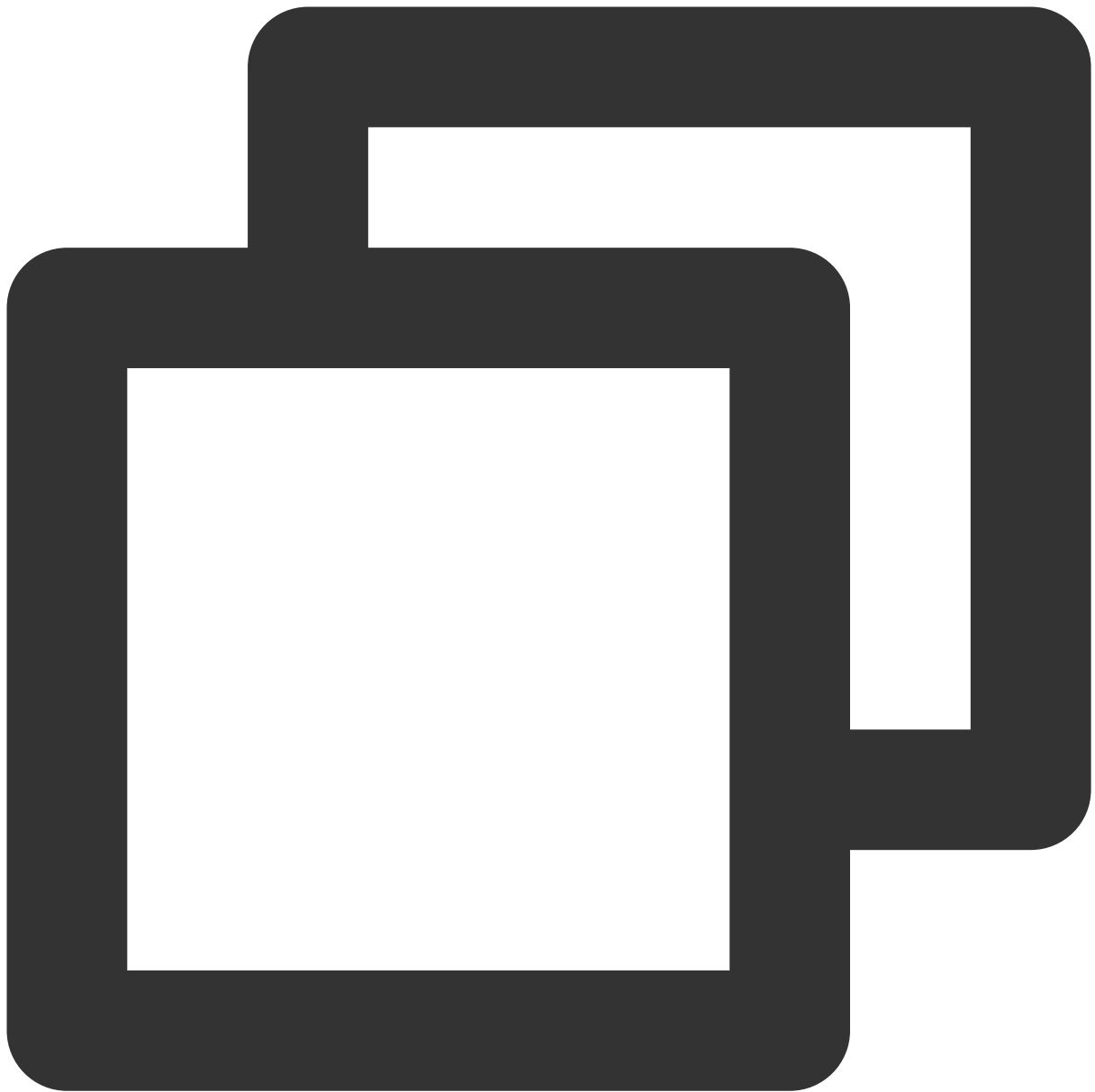
Call `setEncoderMirror` in `V2TXLivePusher` to set the camera mirror mode, which affects the way video images are presented to audience. By default, the local image seen by the host is flipped when the front camera is used.



## 12. Publish streams in landscape mode

In most cases, hosts stream while holding their phones vertically, and audience watch videos in portrait resolutions (e.g., 540 × 960). However, there are also cases where hosts hold phones horizontally, and ideally, audience should watch videos in landscape resolutions (960 × 540).

By default, `V2TXLivePusher` outputs videos in portrait resolutions. You can publish landscape-mode videos to audience by modifying a parameter of the `setVideoQuality` API.



```
[_pusher setVideoQuality:videoQuality  
        resolutionMode:isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2
```

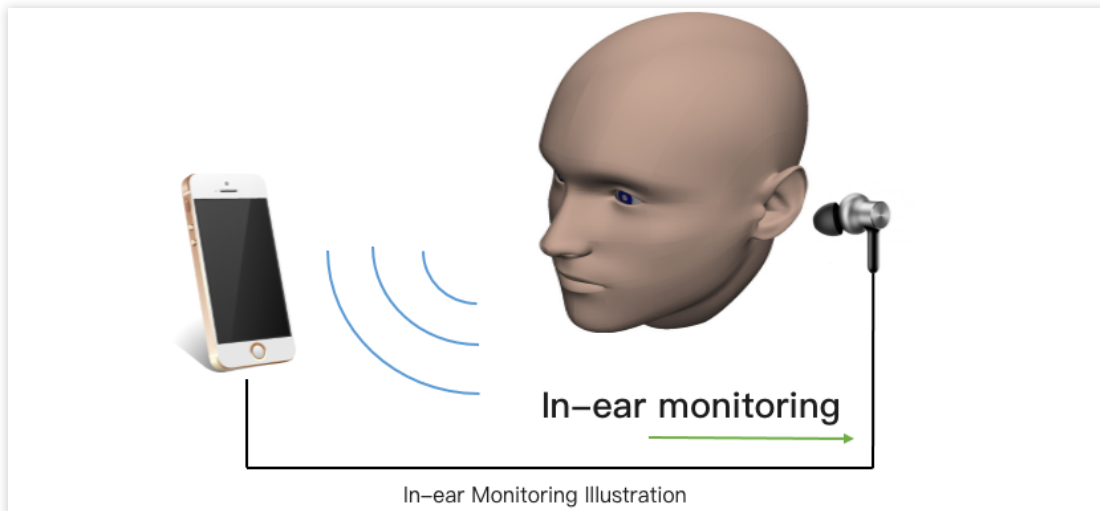
### 13. Set audio effects

Call [getAudioEffectManager](#) in `V2TXLivePusher` to get a `TXAudioEffectManager` instance, which can be used to mix background music and set in-ear monitoring, reverb, and other audio effects. Background music mixing means mixing into the published stream the music played by the host's phone so that audience can also hear the music.

Call the `enableVoiceEarMonitor` API in `TXAudioEffectManager` to enable in-ear monitoring, which allows hosts to hear their vocals in earphones when they sing.

Call the `setVoiceReverbType` API in `TXAudioEffectManager` to add reverb effects such as karaoke, hall, husky, and metal. The effects are applied to the videos watched by audience.

Call the `setVoiceChangerType` API in `TXAudioEffectManager` to add voice changing effects such as little girl and middle-aged man to enrich host-audience interaction. The effects are applied to the videos watched by audience.



**Note:**

For detailed instructions, please see [TXAudioEffectManager API](#).

## 14. Set watermarks

Call `setWatermark` in `V2TXLivePusher` to add a watermark to videos output by the SDK. The position of the watermark is determined by the `(x, y, scale)` parameter passed in.

The watermark image must be in PNG rather than JPG format. The former carries opacity information, which allows the SDK to better address the image aliasing issue (changing the extension of a JPG image to PNG won't work).

The `(x, y, scale)` parameter specifies the normalized coordinates of the watermark relative to the resolution of the published video. For example, if the resolution of the published video is 540 x 960, and `(x, y, scale)` is set to `(0.1, 0.1, 0.1)`, the actual pixel coordinates of the watermark will be (540 x 0.1, 960 x 0.1). The width of the watermark will be the video width x 0.1, and the height will be scaled automatically.



```
// Set a video watermark  
[_pusher setWatermark:[UIImage imageNamed:@"watermark"] x:0.03 y:0.015 scale:1];
```

## 15. Inform hosts of poor network conditions

Connecting phones to Wi-Fi does not necessarily guarantee network conditions. In case of poor Wi-Fi signal or limited bandwidth, the network speed of a Wi-Fi connected phone may be slower than that of a phone using 4G. Hosts should be informed when their network conditions are bad and be prompted to switch to a different network.



You can capture the **V2TXLIVE\_WARNING\_NETWORK\_BUSY** event using `onWarning` in `V2TXLivePusherObserver`. The event indicates poor network conditions for hosts, which result in stuttering for audience. When this event occurs, you can send a UI message about poor network conditions to hosts, as shown above.



```
- (void)onWarning:(V2TXLiveCode)code
    message:(NSString *)msg
    extraInfo:(NSDictionary *)extraInfo {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (code == V2TXLIVE_WARNING_NETWORK_BUSY) {
            [_notification displayNotificationWithMessage:
                @"Your network conditions are poor. Please switch to a different ne
            ]
        }
    });
}
```

## 16. Send SEI messages

Call the `sendSeiMessage` API in `V2TXLivePusher` to send SEI messages. SEI refers to the supplementary enhancement information of encoded video. It is not used most of the time, but you can insert custom information into SEI messages. The information will be forwarded to audience by live streaming CDNs. The applications for SEI messages include:

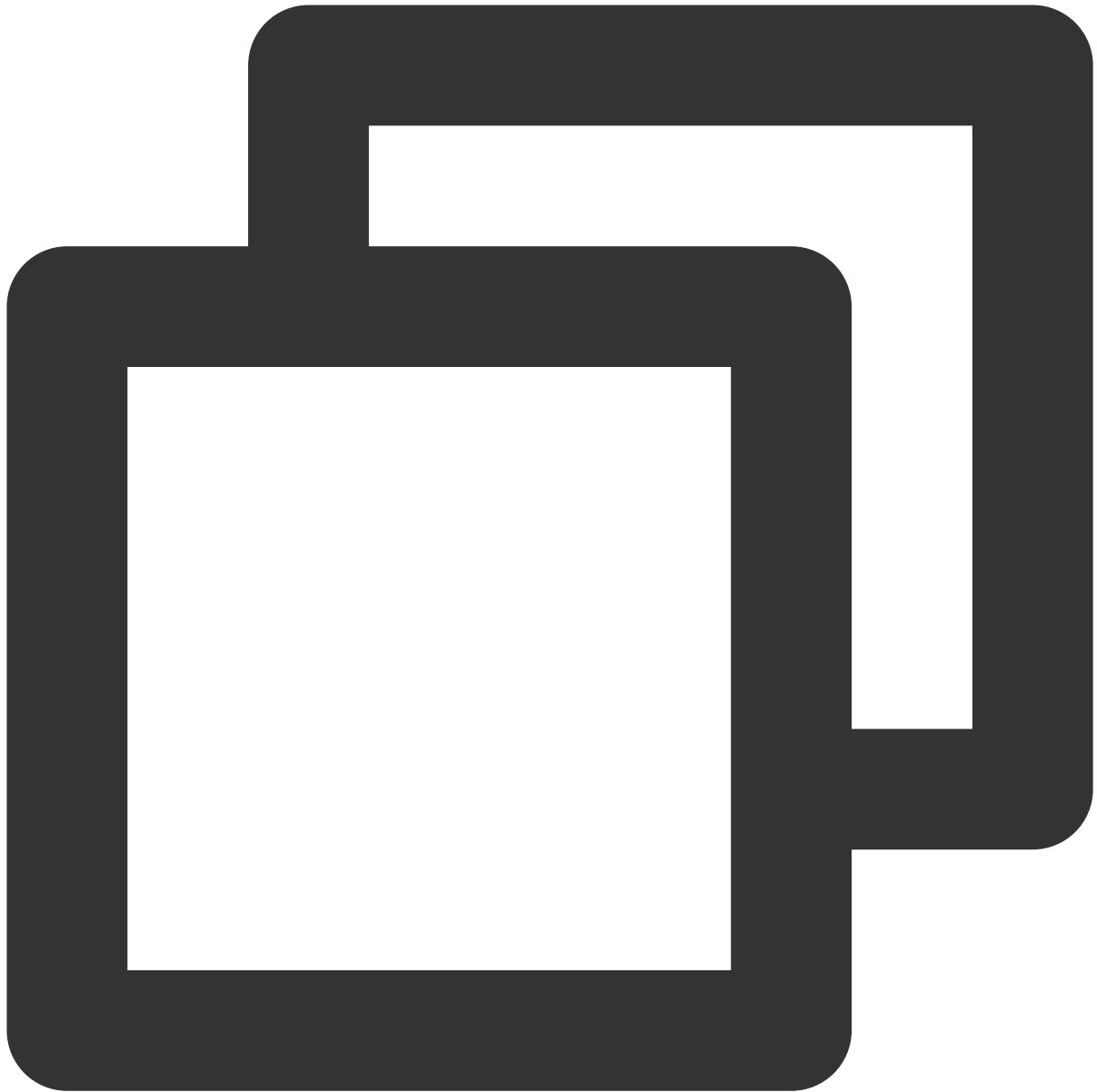
Live quiz: The publisher can use SEI messages to send questions to the audience. SEI can ensure synchronization among audio, video, and the questions.

Live showroom: The publisher can use SEI messages to display lyrics to the audience in real time. The effects are not affected by reduction in video encoding quality.

Online education: The publisher can use SEI messages to display pointers and sketches on slides to the audience in real time.

Custom data is inserted directly into video data and therefore cannot be too large in size (preferably several bytes). It's common to insert information such as custom timestamps.



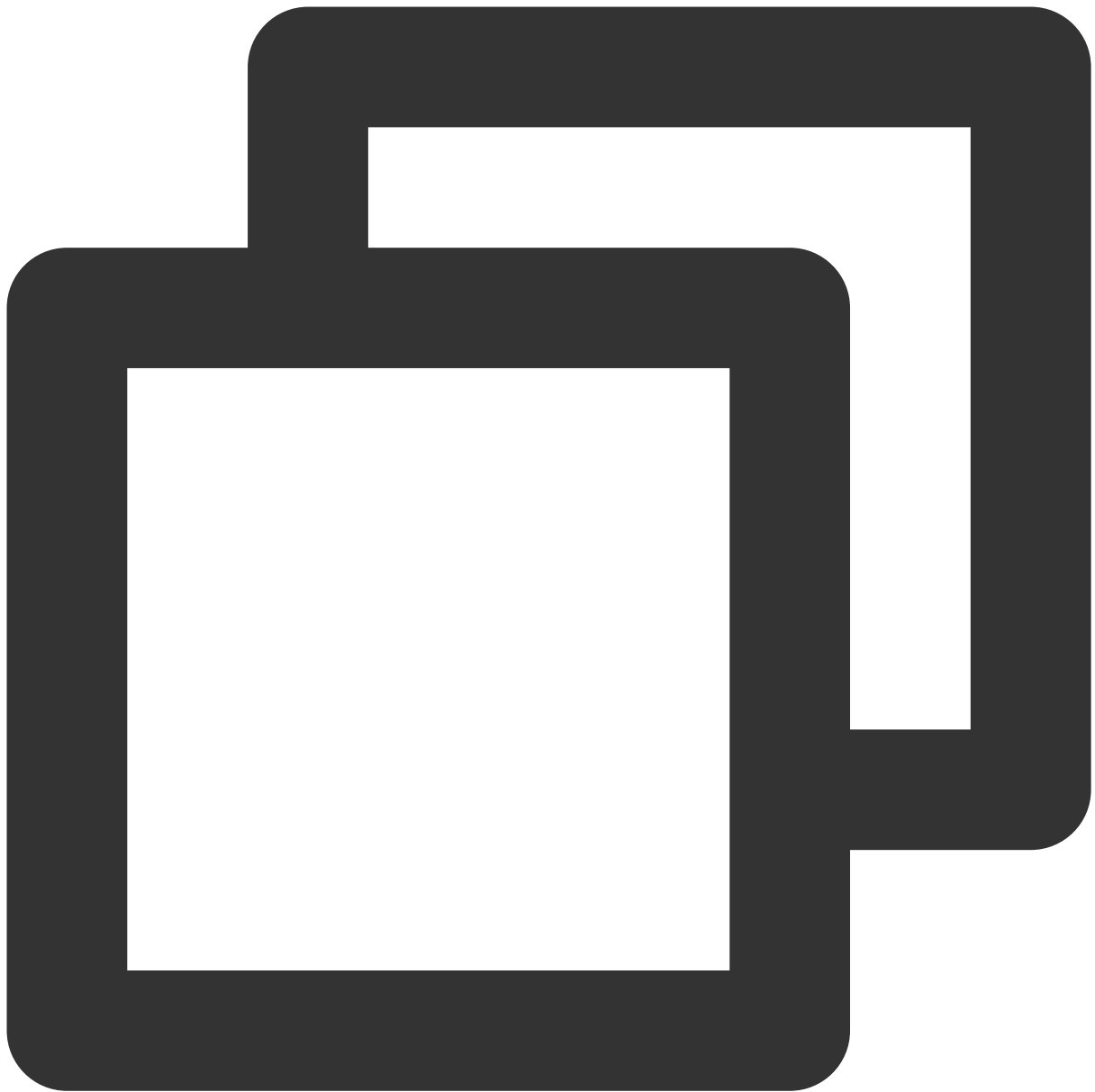


```
int payloadType = 5;
NSString* msg = @"test";
[_pusher sendSeiMessage:payloadType data:[msg dataUsingEncoding:NSUTF8StringEncoding]
```

Common open-source players or web players are incapable of parsing SEI messages. You must use

`V2TXLivePlayer`, the built-in player of LiteAVSDK.

#### 1. Configuration:



```
int payloadType = 5;  
[_player enableReceiveSeiMessage:YES payloadType:payloadType];
```

2. If the video streams played by `V2TXLivePlayer` contain SEI messages, you will receive the messages via the `onReceiveSeiMessage` callback in `V2TXLivePlayerObserver`.

## Event Handling

## Listening for events

The SDK listens for publishing events and errors via the [V2TXLivePusherObserver](#) delegate. See [V2TXLiveCode](#) for a detailed list of events and error codes.

## Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common error not yet classified
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Latency during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile

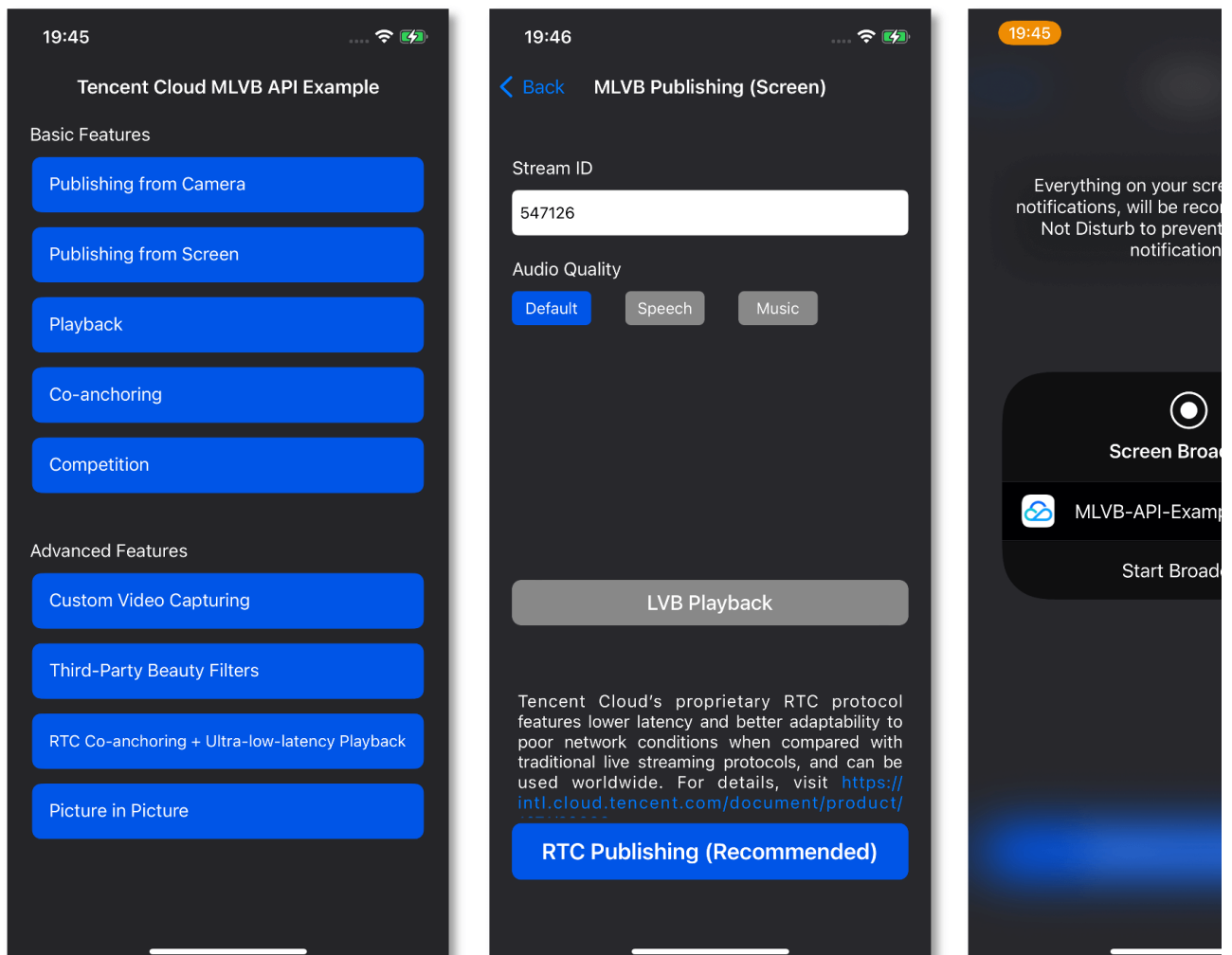
		devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the user is having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system.

# Publishing from Screen

Last updated : 2024-01-13 15:49:41

## Overview

Screen recording allows a host to live stream the image on their phone screen. It can be combined with local camera preview and is used in scenarios such as game streaming and mobile app demos. The Tencent Cloud LiteAV SDK offers screen recording capabilities via `V2TXLivePusher`. The UIs for screen recording operations in the `SDK` `API-Example` project are as shown below:



## Restrictions

The screen recording feature is available only on iOS 11.0 or later. This document describes how to use ReplayKit 2 on iOS 11 to push streams from the screen. The parts about the use of the SDK also apply to other custom stream push scenarios. For more information, see the code sample in the `TXReplayKit_Screen` folder of the [demo](#). Screen recording is a new feature on iOS 10. In addition to using ReplayKit to record video from the screen, which is possible on iOS 9, with iOS 10, users can also stream live video from the screen. For more information, see [Go Live with ReplayKit](#). On iOS 11, Apple made ReplayKit more usable and more universally applicable and launched [ReplayKit 2](#), going from supporting ReplayKit alone to allowing the recording of the entire screen. Therefore, we recommend you use ReplayKit 2 on iOS 11 to enable the screen recording feature. Screen recording relies on extensions, which operate as independent processes. However, to ensure system smoothness, iOS allocates limited resources to extensions and may kill extensions with high memory usage. Given this, Tencent Cloud has further reduced the memory usage of the LiteAV SDK while retaining its high streaming quality and low latency to ensure the stability of extensions.

## Sample Code

Tencent Cloud offers an easy-to-understand API example project to help you quickly learn how to use different APIs.

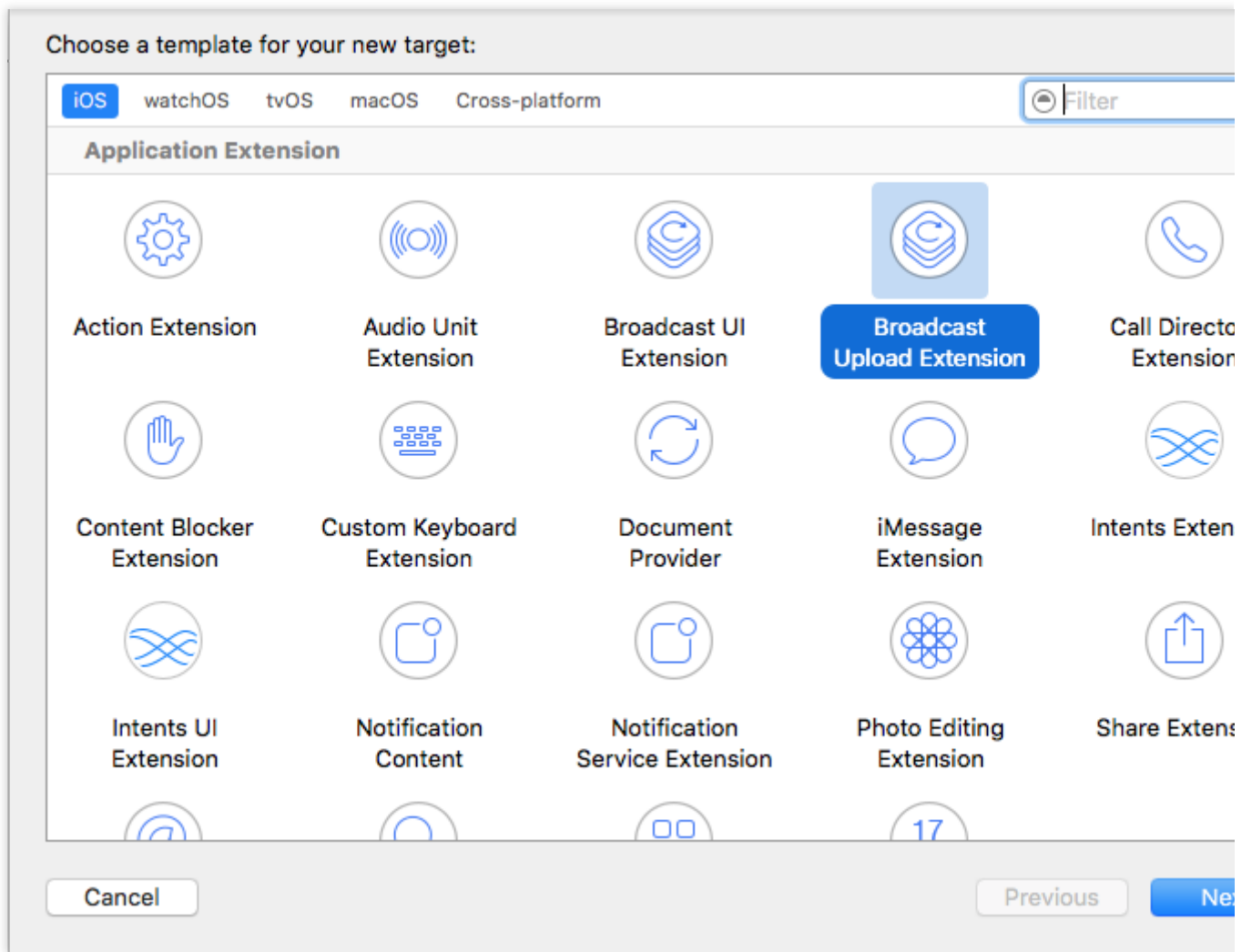
Platform	GitHub Address
iOS	<a href="#">LivePushScreenViewController.m</a>
Android	<a href="#">LivePushScreenActivity.java</a>
Flutter	<a href="#">live_screen_push.dart</a>

### Xcode

Xcode 9 or above is required, and your iPhone must be updated to iOS 11 or above. Screen recording is not supported on emulators.

### Create a broadcast upload extension

Open your project with Xcode and select **New > Target... > Broadcast Upload Extension**, as shown below.



Enter a product name and click **Finish**. A new directory with the product name entered will appear in your project. Under the directory, there is an automatically generated `SampleHandler` class, which is responsible for screen recording operations.

## SDK Integration

### 1. Download the SDK

[Download](#) the SDK and follow the instructions in [SDK Integration]

(<https://intl.cloud.tencent.com/document/product/1071/38155> to integrate the SDK into your application.

### 2. Configure a license for the SDK

1. Get the license:

If you have the required license, get the license URL and key in the [CSS console](#).

**Create Official License**

An official license is valid for a year. Click [Create] to purchase one. Please make sure that the bundle ID and package name entered are correct as the information cannot be modified after submission.

[Price Overview](#)[Create](#)**▼ Official License**

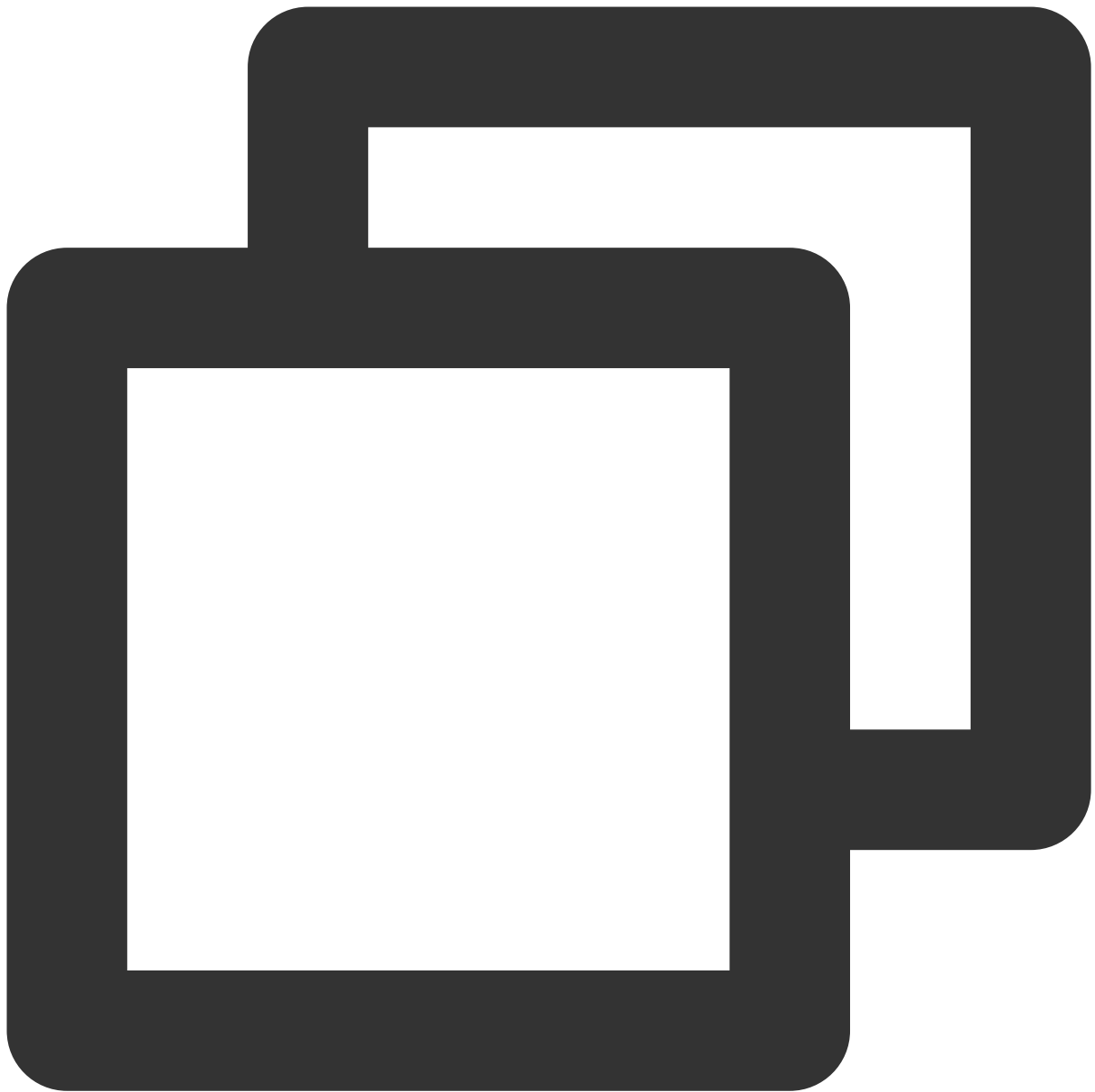
	Live
Application Name	
Package Name	com
Bundle Id	com
Key	
LicenseUrl	.licence
Start Date	2021-12-03
End Date	2022-12-03

If you don't have the required license, apply for a license as instructed in [New License and Renewal](#).

2. Before your application calls features of LiteAVSDK, complete the following configuration (preferably in `-`

```
[AppDelegate application:didFinishLaunchingWithOptions:] ):
```





```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {  
    NSString * const licenceURL = @"<The license URL obtained>";  
    NSString * const licenceKey = @"<The key obtained>";  
  
    // `V2TXLivePremier` is in the `V2TXLivePremier.h` header file.  
    [V2TXLivePremier setLicence:licenceURL key:licenceKey];  
    [V2TXLivePremier setObserver:self];  
    NSLog(@"SDK Version = %@", [V2TXLivePremier getSDKVersionStr]);  
    return YES;  
}
```

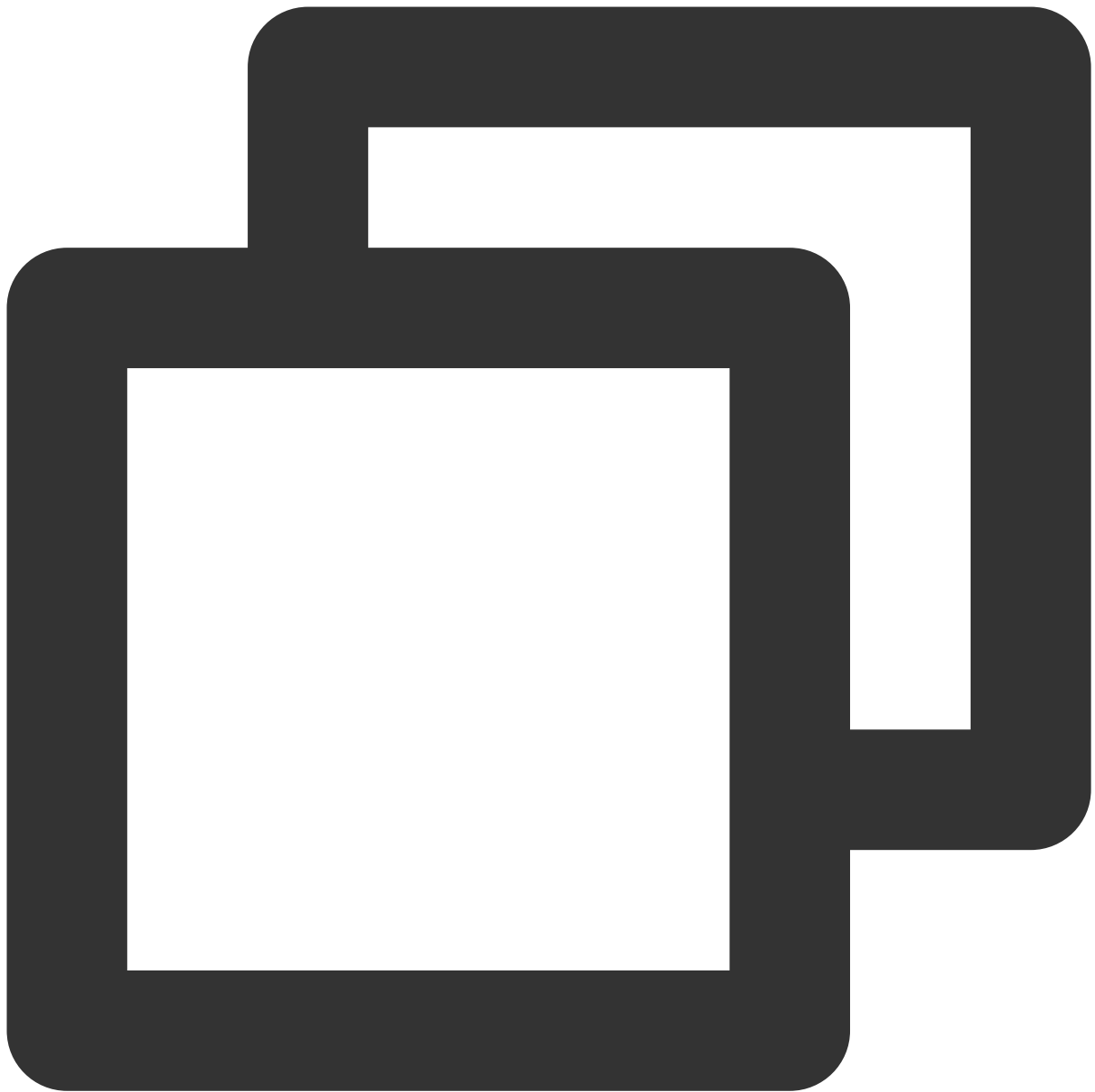
```
#pragma mark - V2TXLivePremierObserver
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
```

**Note:**

**BundleId** configured in the license must be the same as that of the application; otherwise, stream push will fail.

### 3. Initialize the **V2TXLivePusher** component

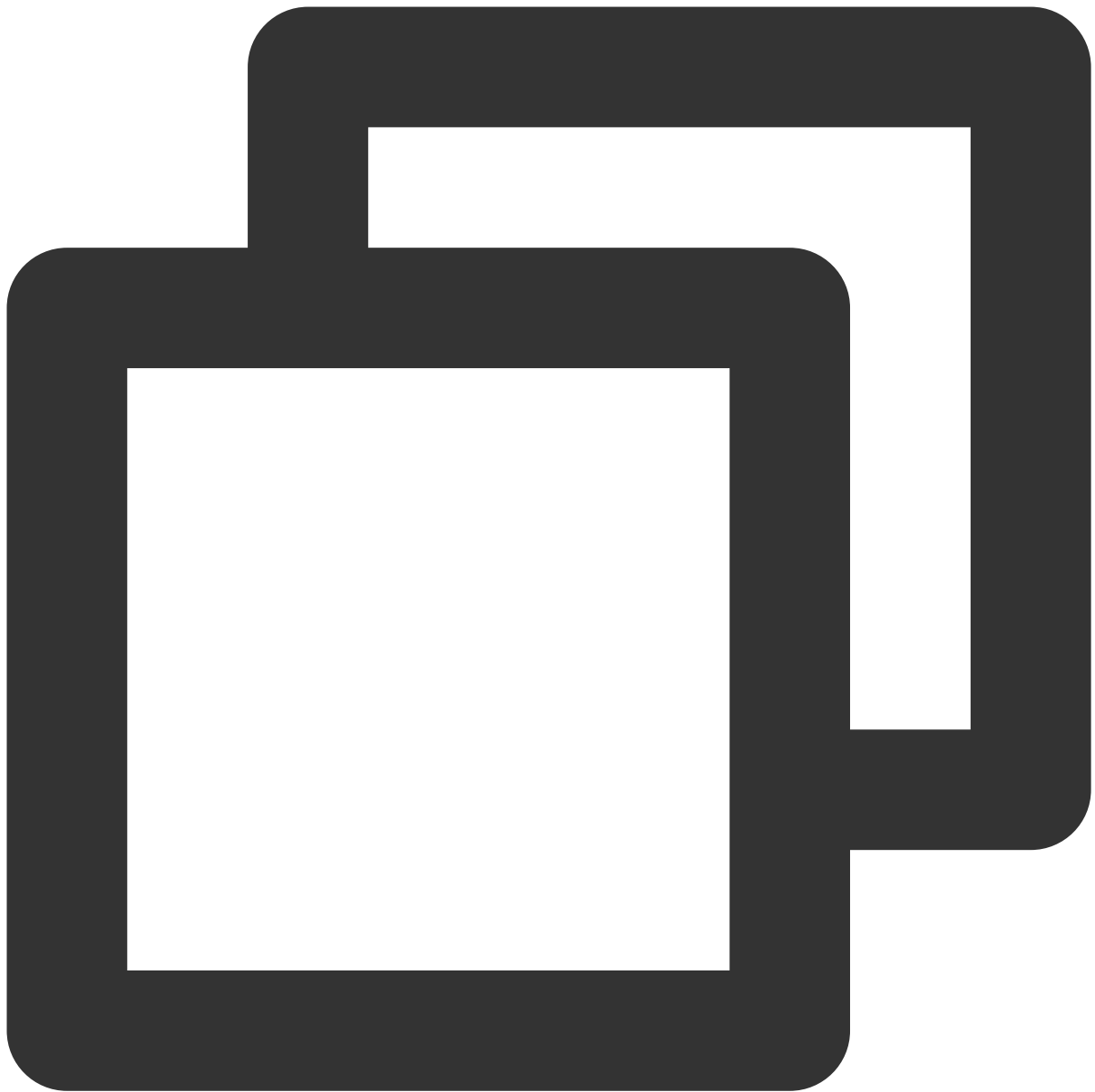
Create a **V2TXLivePusher** object and specify **V2TXLiveMode** .



```
// Set the live streaming protocol to RTMP, which doesn't support mic connect.  
V2TXLivePusher *pusher = [[V2TXLivePusher alloc] initWithLiveMode:V2TXLiveMode_RTMP
```

#### 4. Configure `RPBroadcastSampleHandler`

To use screen recording, you need to use a subclass of the system API `RPBroadcastSampleHandler` to get the screen audio/video data. Here, add the following code to the custom subclass [SampleHandler.m](#) to implement screen recording:



```
#import "SampleHandler.h"
#import TXLiteAVSDK_ReplayKitExt;

@implementation SampleHandler
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupI
    // The application group ID. The ID must match `startScreenCapture` of `V2TXLiv
    [[TXReplayKitExt sharedInstance] setupWithAppGroup:APPGROUP delegate:self];
}

- (void)broadcastPaused {
    // User has requested to pause the broadcast. Samples will stop being delivered
```

```

    [[TXReplayKitExt sharedInstance] broadcastPaused];
}

- (void)broadcastResumed {
    // User has requested to resume the broadcast. Samples delivery will resume.
    [[TXReplayKitExt sharedInstance] broadcastResumed];
}

- (void)broadcastFinished {
    // User has requested to finish the broadcast.
    [[TXReplayKitExt sharedInstance] broadcastFinished];
}

- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sampleType {
    [[TXReplayKitExt sharedInstance] sendSampleBuffer:sampleBuffer withType:sampleType];
}

#pragma mark - TXReplayKitExtDelegate
- (void)broadcastFinished:(TXReplayKitExt *)broadcast reason:(TXReplayKitExtReason)reason {
    {
        NSString *tip = @"";
        switch (reason) {
            case TXReplayKitExtReasonRequestedByMain:
                tip = NSLocalizedString(@"MLVB-API-Example.liveStop", "");
                break;
            case TXReplayKitExtReasonDisconnected:
                tip = NSLocalizedString(@"MLVB-API-Example.appReset", "");
                break;
            case TXReplayKitExtReasonVersionMismatch:
                tip = NSLocalizedString(@"MLVB-API-Example.sdkError", "");
                break;
        }

        NSError *error = [NSError errorWithDomain:NSStringFromClass(self.class)
                                         code:0
                                         userInfo:@{
                                             NSLocalizedFailureReasonErrorKey:tip
                                         }];

        [self finishBroadcastWithError:error];
    }
}

@end

```

In the implementation class of `RPBroadcastSampleHandler`, you need to call the corresponding method in `TXReplayKitExt` to set the screen recording information and handle screen recording events.

The **broadcastFinished** callback can be used to get the reason that screen recording is stopped and can be used to prompt the user to take action.

For details about communication between the extension and host app, see [Communication and Data Transfer Between Extensions and Host Apps](#).

## 5. Enable screen recording and stream push

Call [startScreenCapture](#) to start screen recording and then call the [startPush](#) API in `V2TXLivePusher` to start pushing the stream.

### Note:

If you select the `RTMP` protocol for stream push in [step 3](#), you can generate a stream push address as instructed in [Quick URL Generation](#).



```
// The application group ID for data sharing between the host application and Broad
[livePusher startScreenCapture:@"group.com.xxx"];
[livePusher startMicrophone];

// Pass in the corresponding URL based on the stream push protocol to start stream
NSString * const url = @"rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxx";
V2TXLiveCode code = [livePusher startPush:url];
if (code != V2TXLIVE_OK) {
    // Check the error code
}
```

### Why is `V2TXLIVE\_ERROR\_INVALID\_LICENSE` returned?

If the startPush API returns V2TXLIVE\_ERROR\_INVALID\_LICENSE, it means your license verification failed. Check the error code and error message in the onLicenceLoaded callback in [Step 2. Configure a license for the SDK](#).

On iOS 11 or later, you can enable screen recording only by pulling down the status bar and pressing and holding the screen recording button.

On iOS 12 or later, you can use `RPSystemBroadcastPickerView` to pop up the screen recording selection page as instructed in [TRTCBroadcastExtensionLauncher.m](#).

## 6. Set landscape stream push and the resolution

Call [setVideoQuality](#) to set the resolution (you can select from a number of resolutions) and orientation for stream publishing. Below is an example:



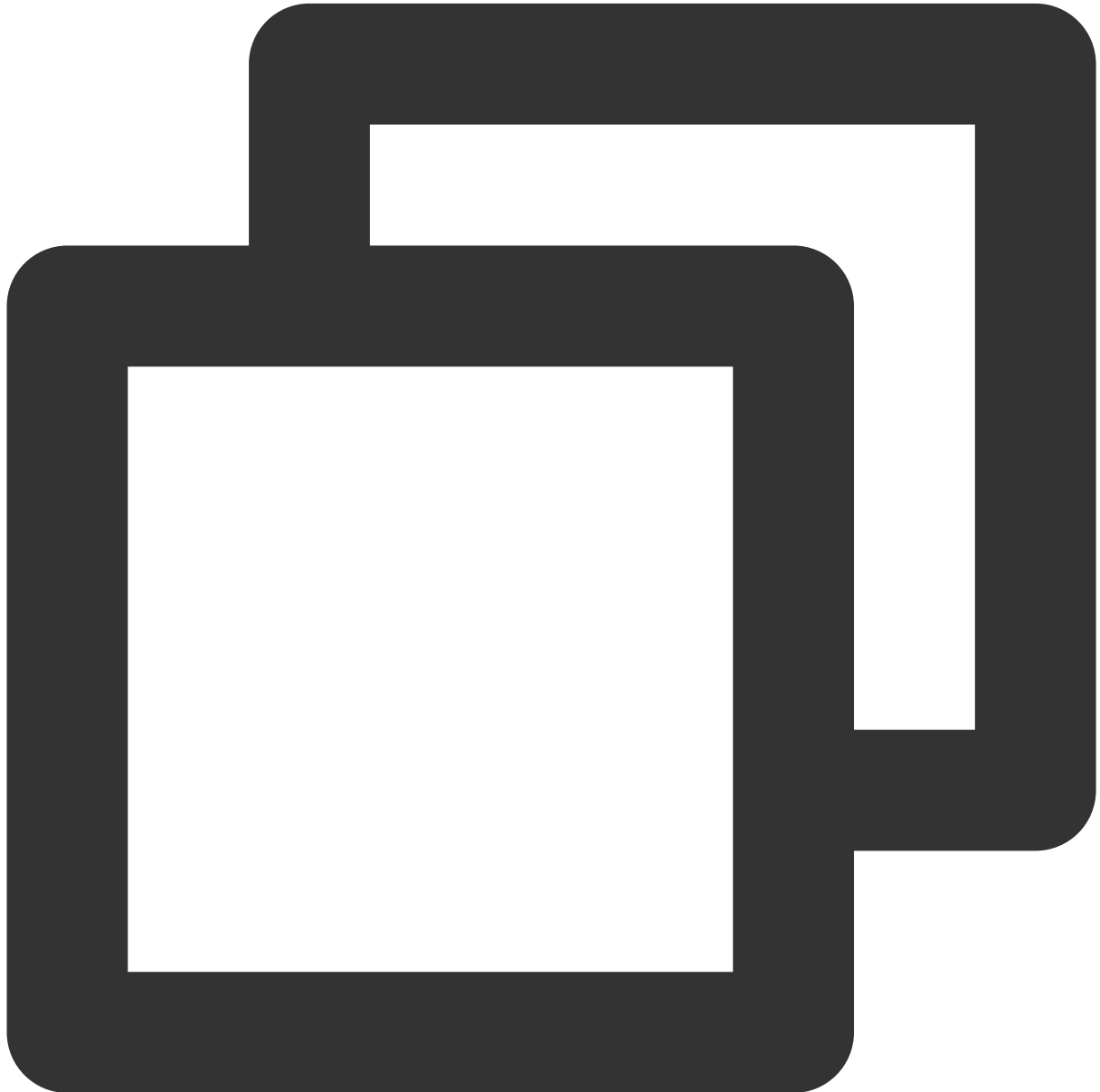


```
BOOL landscape; // `YES`: Landscape; `NO`: Portrait.  
V2TXLiveVideoEncoderParam *videoParam = [[V2TXLiveVideoEncoderParam alloc] init  
videoParam.videoResolution = V2TXLiveVideoResolution960x540;  
videoParam.videoResolutionMode = landscape ? V2TXLiveVideoResolutionModeLandscape  
[livePusher setVideoQuality:videoParam];
```

## 7. Set the logo watermark

Call [setWatermark](#) in `V2TXLivePusher` to add a watermark to videos output by the SDK. The position of the watermark is determined by the `(x, y, scale)` parameter passed in.

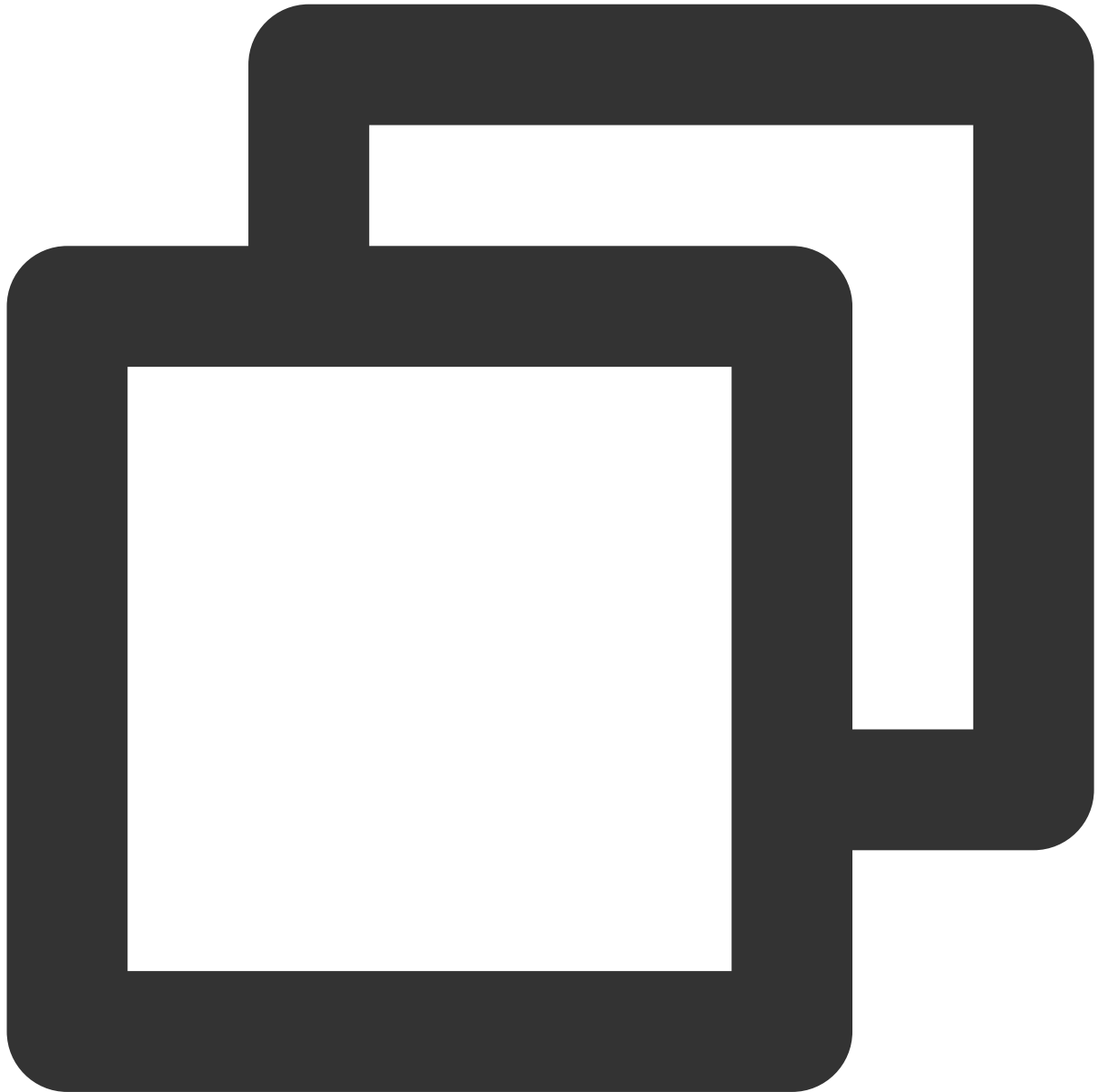
The watermark image must be in PNG rather than JPG format. The former carries opacity information, which allows the SDK to better address the image aliasing issue (changing the extension of a JPG image to PNG won't work). The `(x, y, scale)` parameter specifies the normalized coordinates of the watermark relative to the resolution of the published video. For example, if the resolution of the published video is 540 x 960, and `(x, y, scale)` is set to `(0.1, 0.1, 0.1)`, the actual pixel coordinates of the watermark will be (540 x 0.1, 960 x 0.1). The width of the watermark will be the video width x 0.1, and the height will be scaled automatically.



```
// Set the video watermark  
[livePusher setWatermark:image x:0 y:0 scale:1];
```

## 8. Stop stream push

As there can be only one `V2TXLivePusher` object running at a time, make sure that you release all the resources when stopping publishing.



```
// Stop stream push
[livePusher stopScreenCapture];
[livePusher stopPush];
```

# Event Handling

## 1. Listening for events

The SDK listens for publishing events and errors via the [V2TXLivePusherObserver](#) delegate. See [V2TXLiveCode](#) for a detailed list of events and error codes.

## 2. Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common unclassified error occurred.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## 3. Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: Data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Stuttering during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.

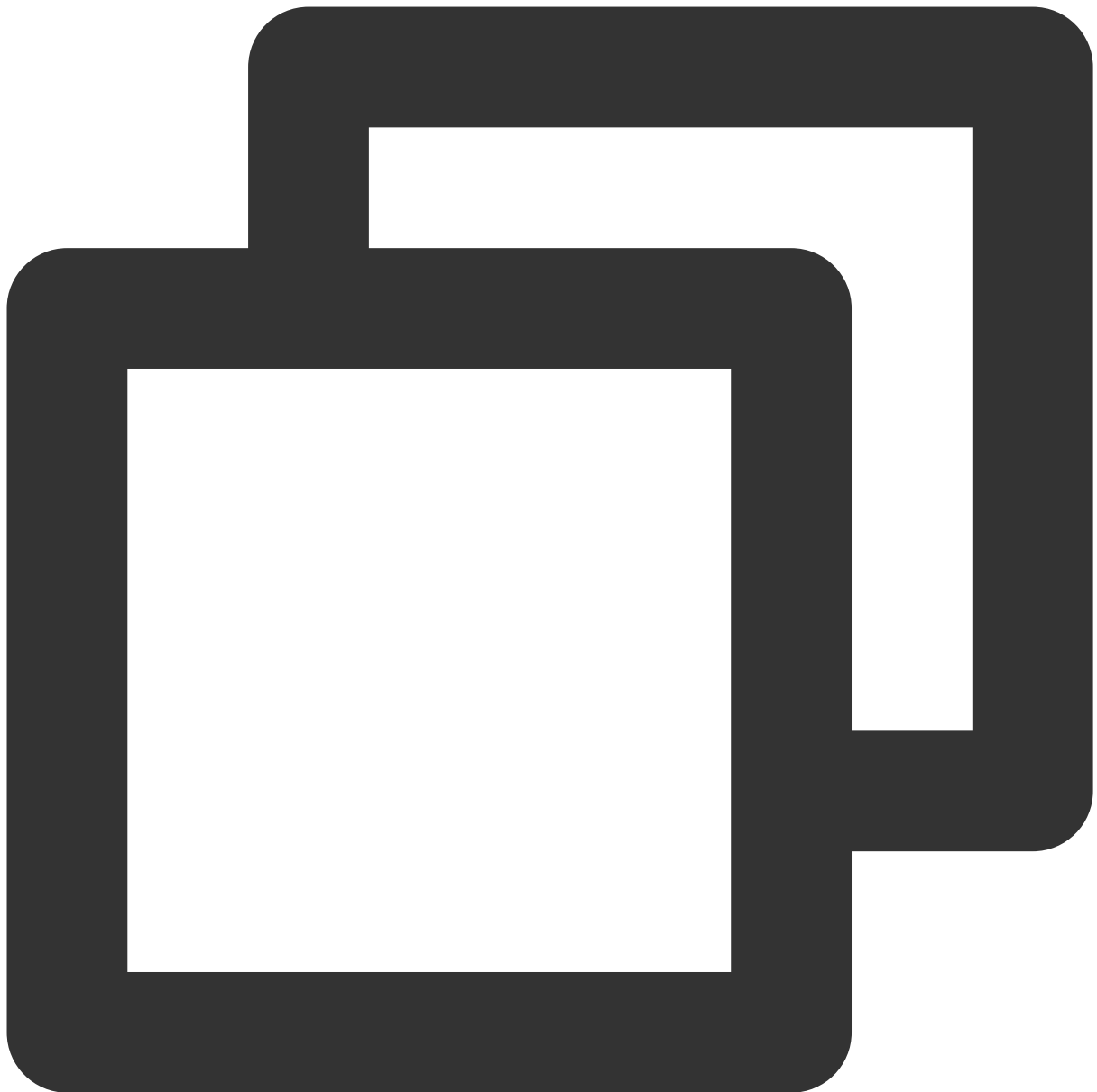
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the user is currently having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system

## Appendix: Communication and Data Transfer Between Extensions and Host Applications

ReplayKit2 invokes only the broadcast upload extension during screen sharing. The extension does not support UI operations and cannot implement complicated business logic. Therefore, the host app is often responsible for implementing business logic such as authentication, while the extension focuses on recording the screen and publishing the audio and video data captured. This makes it necessary to communicate and transfer data between the extension and host app.

### 1. Sending local notifications

Users should be informed of the status of the extension. For example, in cases where the host app is not started, you can send a local notification asking users to interact with the host app. Below is an example of sending a notification to users when the broadcast upload extension is started.



```
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupI
    [self sendLocalNotificationToHostAppWithTitle:@"Tencent Cloud Screen Sharing" m
}

- (void)sendLocalNotificationToHostAppWithTitle:(NSString*)title msg:(NSString*)msg
{
    UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificatio

    UNMutableNotificationContent* content = [[UNMutableNotificationContent alloc] i
    content.title = [NSString localizedUserNotificationStringForKey:title arguments
    content.body = [NSString localizedUserNotificationStringForKey:msg arguments:n
```

```
content.sound = [UNNotificationSound defaultSound];
content.userInfo = userInfo;

// Schedule a local notification to be sent at the specified time
UNTimeIntervalNotificationTrigger* trigger = [UNTimeIntervalNotificationTrigger
                                              triggerWithTimeInterval:0.1f repea

UNNotificationRequest* request = [UNNotificationRequest requestWithIdentifier:@
                                content:c

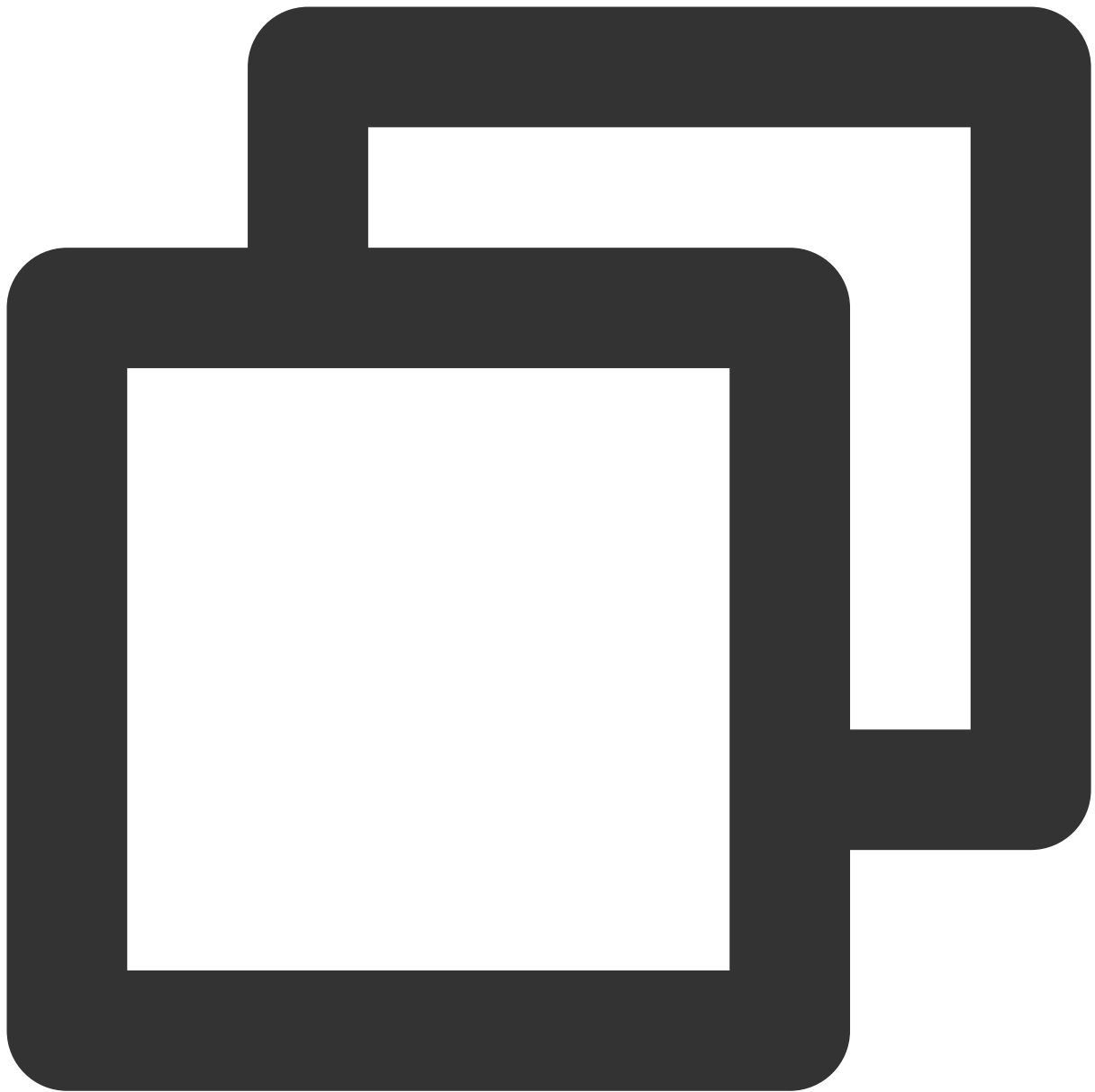
// Add operation after the notification is sent
[center addNotificationRequest:request withCompletionHandler:^(NSError * _Nulla

}];
}
```

You can use this notification to ask the user to return to the host app to configure publishing information and start stream publishing.

## 2. Sending notifications between processes via CFNotificationCenter

The extension and host app may also need to interact with each other in real time, which cannot be achieved through local notifications because with local notifications, code is triggered only after users tap the banner. Neither can it be implemented via NSNotificationCenter because NSNotificationCenter does not allow communication between processes. To send notifications between processes, you will need CFNotificationCenter, but instead of using the `userInfo` field for data transfer, you must configure an app group and use `NSUserDefaults` for data transfer. For example, after getting the publishing URL, the host app can notify the broadcast upload extension via CFNotificationCenter that stream publishing can start. You may also use the clipboard, but delayed rendering is needed as the clipboard sometimes fails to transfer data between processes in real time.



```
CFNotificationCenterPostNotification(CFNotificationCenterGetDarwinNotifyCenter(),  
                                     kDarvinNotificationNamePushStart,  
                                     NULL,  
                                     nil,  
                                     YES);
```

The extension can start publishing streams after receiving this notification. As the notification is at the CF layer, to facilitate operations, it needs to be sent to the Cocoa layer via NSNotificationCenter.





```
CFNotificationCenterAddObserver(CFNotificationCenterGetDarwinNotifyCenter(),
                                (__bridge const void *) (self),
                                onDarwinReplayKit2PushStart,
                                kDarwinNotificationNamePushStart,
                                NULL,
                                CFNotificationSuspensionBehaviorDeliverImmediately);
```

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handlePushStart) object:nil];
```

```
static void onDarwinReplayKit2PushStart(CFNotificationCenterRef center,
```

```
void *observer, CFStringRef name,
const void *object, CFDictionaryRef
userInfo)

{
// Send to the Cocoa layer
[[NSNotificationCenter defaultCenter] postNotificationName:@"Cocoa_ReplayKit2_P
}

- (void)handleReplayKit2PushStartNotification:(NSNotification*)noti
{
// Get the data to be transferred by the host app via NSUserDefaults or the clipboar
//   NSUserDefaults *defaults = [[NSUserDefaults alloc] initWithSuiteName:kReplayK

UIPasteboard* pb = [UIPasteboard generalPasteboard];
NSDictionary* defaults = [self jsonData2Dictionary:pb.string];

s_rtmpUrl = [defaults objectForKey:kReplayKit2PushUrlKey];
s_resolution = [defaults objectForKey:kReplayKit2ResolutionKey];
if (s_resolution.length < 1) {
    s_resolution = kResolutionHD;
}
NSString* rotate = [defaults objectForKey:kReplayKit2RotateKey];
if ([rotate isEqualToString:kReplayKit2Portrait]) {
    s_landScape = NO;
}
else {
    s_landScape = YES;
}
[self start];
}
```

## FAQs

ReplayKit2 is a new framework introduced by Apple in iOS 11, for which relatively few official documents have been released. The framework is still being improved, and problems have been found. See below for some common questions you may have when using ReplayKit2.

### 1. When does screen recording stop automatically?

Screen recording stops automatically when the screen locks or there is an incoming call. At such times, the `broadcastFinished` function in `SampleHandler` will be invoked, and you can send a notification to users about the interruption.

### 2. Why does screen recording stop sometimes during screen sharing?

The problem usually occurs after landscape/portrait mode switch if the resolution for stream publishing is set high. The

broadcast upload extension is allocated a memory of only 50 MB and will be killed if its memory usage exceeds the limit. Given this, we recommend that you set the resolution to 720p or lower.

### 3. Why are images streamed from the screen of iPhone X distorted?

iPhone X has a notch at the top of the screen, so video captured from the screen is not in the aspect ratio of 16:9. If you set the output resolution for stream publishing to 16:9, for example, to HD (960 × 540), the images published will be slightly distorted because their original aspect ratio is not 16:9. We recommend that you set the resolution according to your screen size. Besides, if you play video streamed from the screen of iPhone X in aspect fit mode, the video may have black bars, and if you play it in aspect fill mode, the video may be cropped.

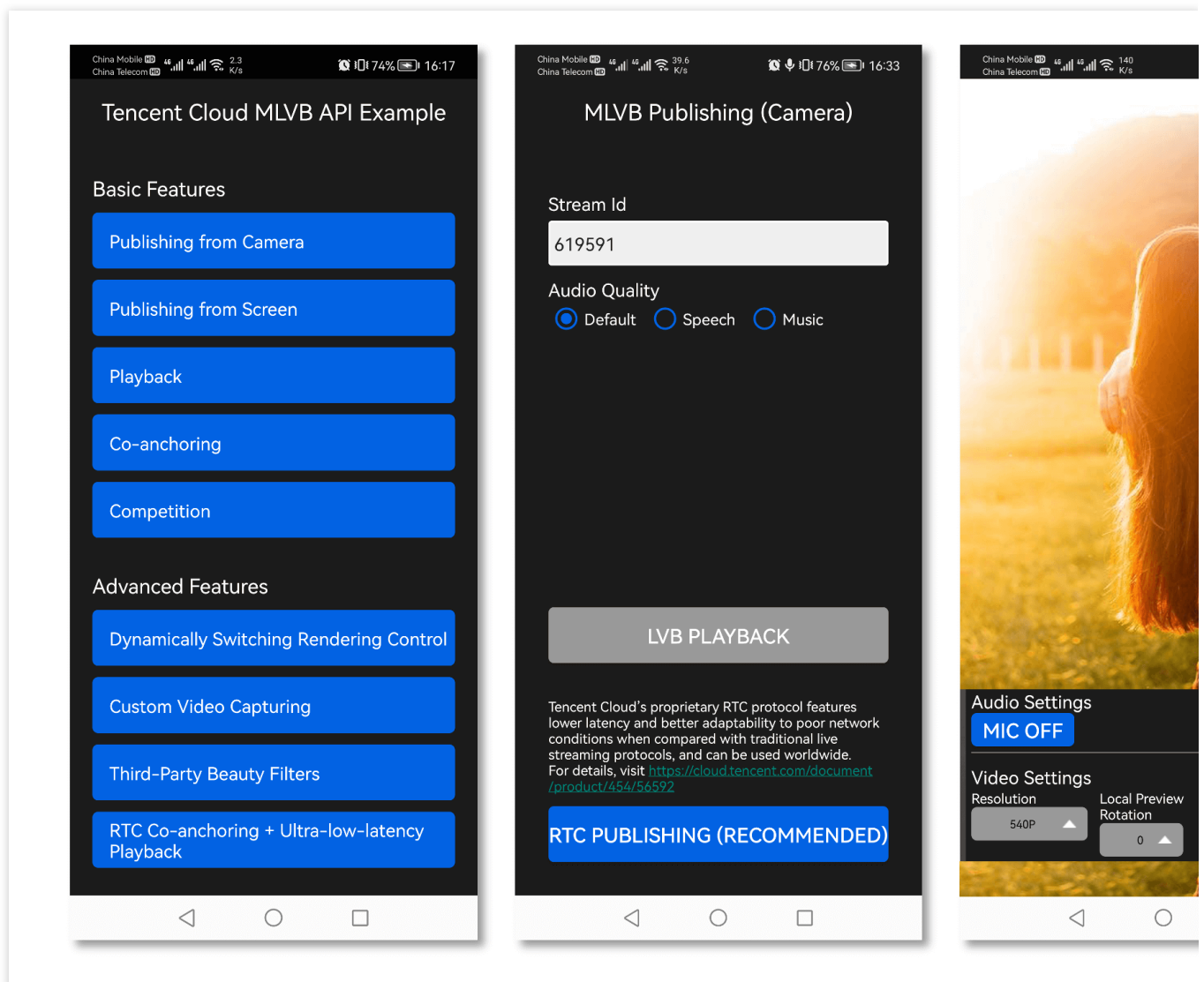
# Android

## Publishing from Camera

Last updated : 2024-01-13 15:49:41

### Overview

Publishing from camera refers to the process of collecting video and audio data from the mobile phone's camera and mic, encoding the data, and publishing it to cloud-based live streaming platforms. Tencent Cloud's LiteAVSDK provides the camera publishing capability via `V2TXLivePusher`. The following is the relevant GUI that demonstrating camera push stream in the SDK API-Example project:



**Testing on real devices:** The SDK uses a lot of audio and video APIs of the Android system, most of which cannot be used on emulators. Therefore, we recommend that you test your project on a real device.

Platform	GitHub Address	Key Class
iOS	<a href="#">Github</a>	LivePushCameraViewController.m
Android	<a href="#">Github</a>	LivePushCameraActivity.java
Flutter	<a href="#">Github</a>	live_camera_push.dart

## 1. Download the SDK

[Download](#) the SDK and follow the instructions in [SDK Integration](#) to integrate the SDK into your application.

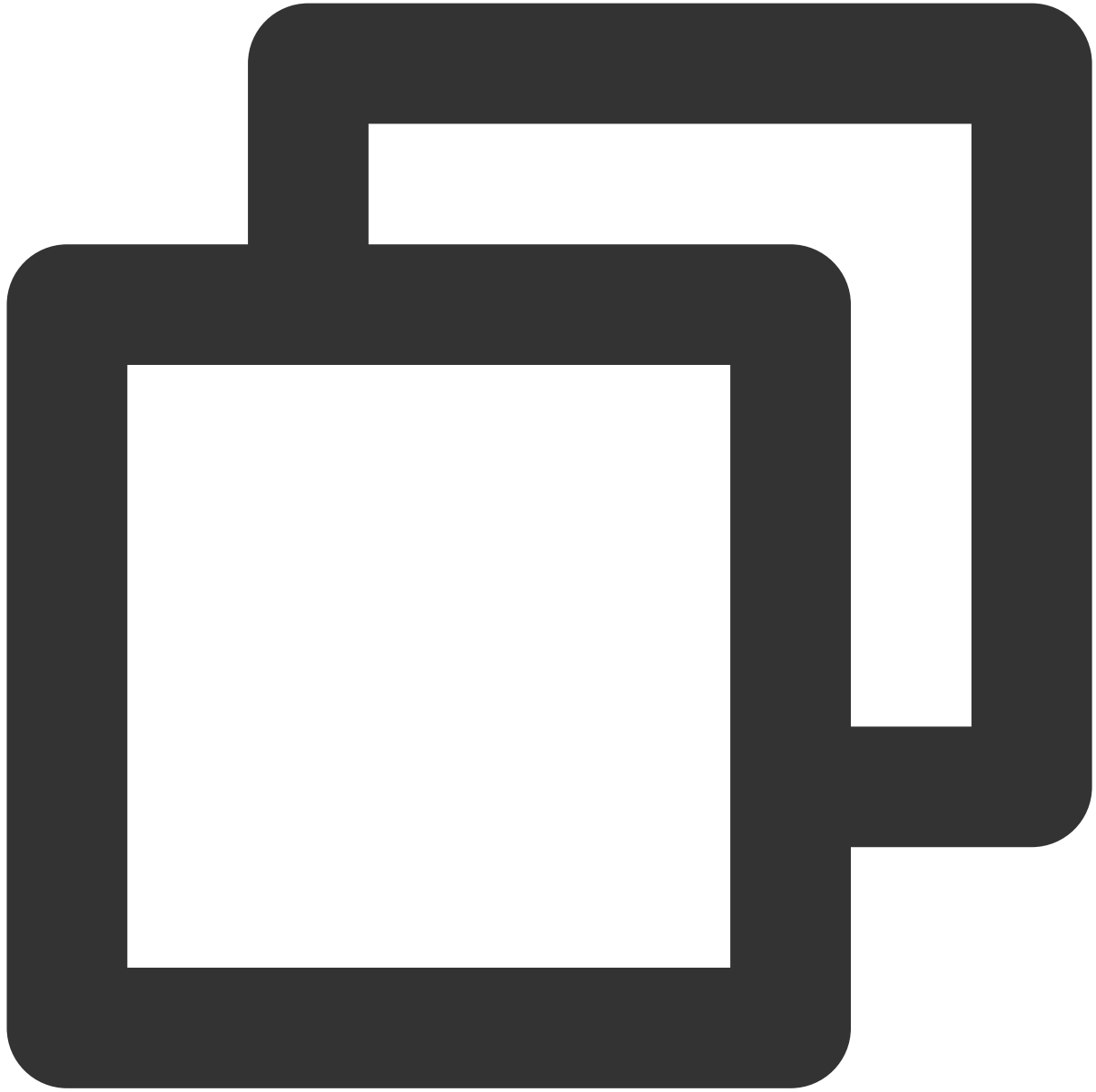
## 2. Configure License Authorization for SDK

1. Obtain license authorization :

If you have obtained the relevant license authorization, need to Get License URL and License Key in [Cloud Live Console](#)

If you have not yet obtained the license authorization, please reference [Adding and Renewing Licenses](#) to make an application.

2. Before your App calls SDK-related functions (it is recommended in the Application class), set the following settings:



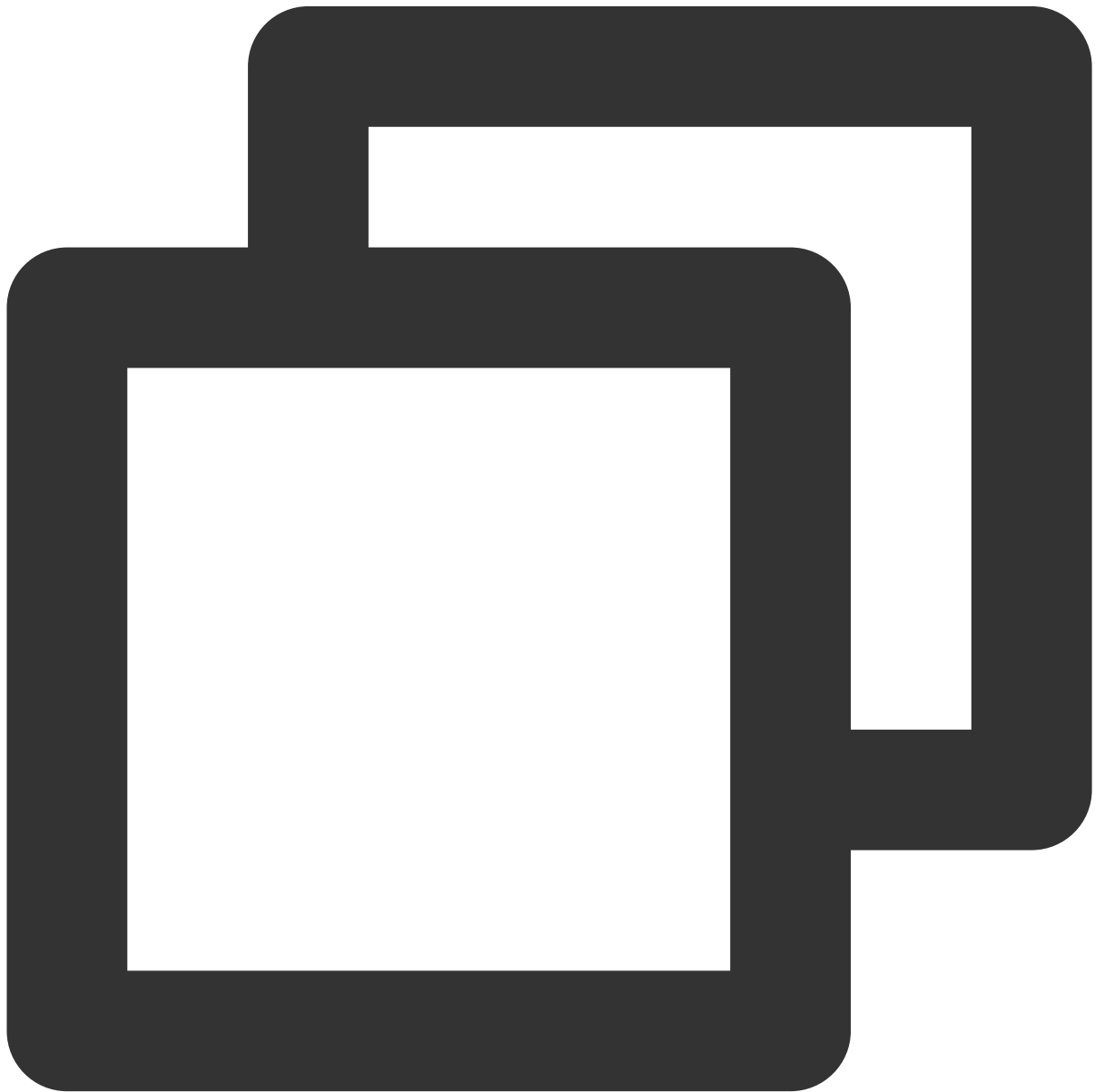
```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // your licence url
        String licenceKey = ""; // your licence key
        V2TXLivePremier.setEnvironment("GDPR"); // set your environment
        V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
    }
}
```

```
V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {  
    @Override  
    public void onLicenceLoaded(int result, String reason) {  
        Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);  
    }  
});  
}
```

### 3. Initialize the `V2TXLivePusher` component

Create a `V2TXLivePusher` object, which will be responsible for publishing operations.

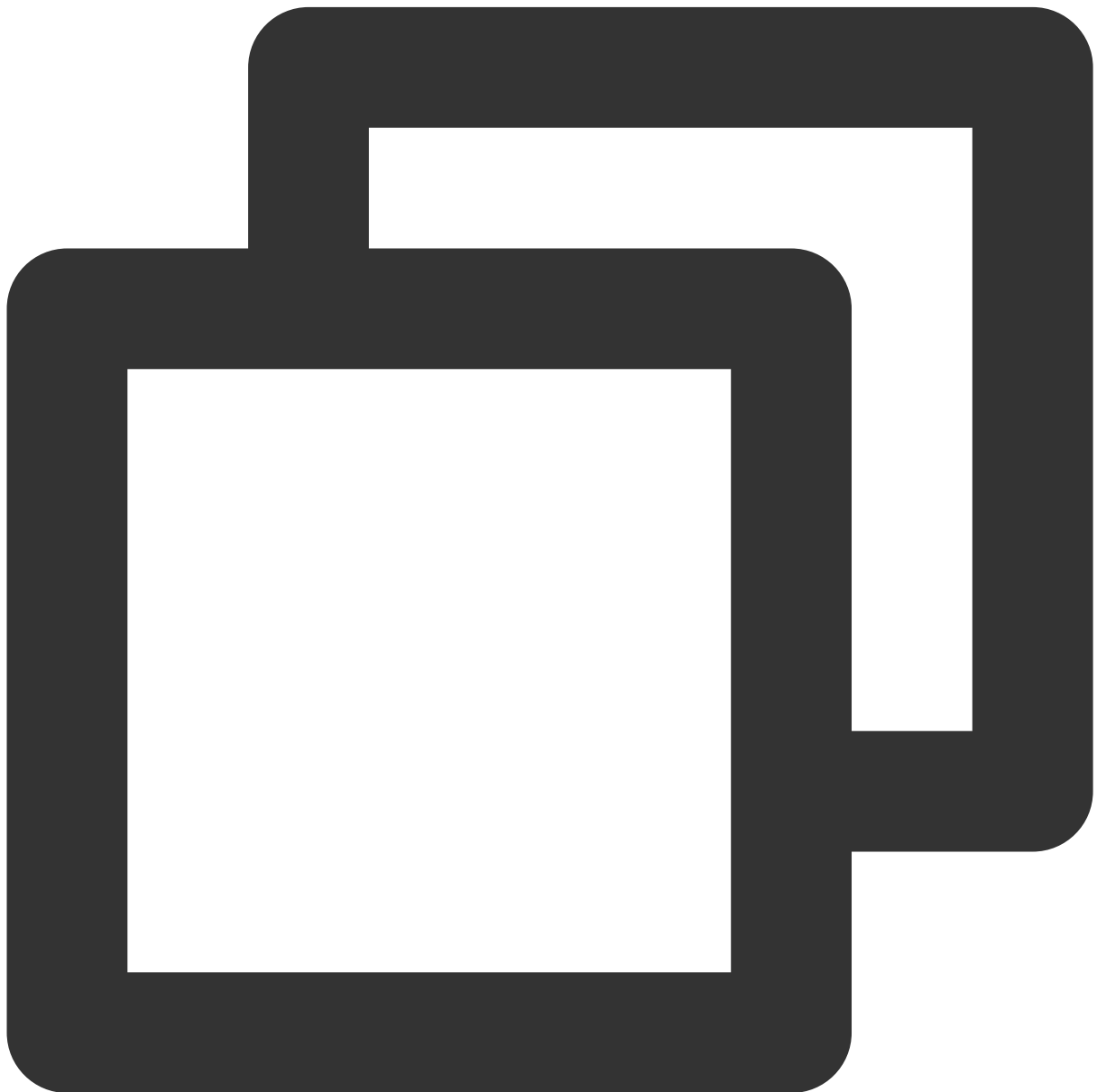


```
// Specify the corresponding live broadcast protocol as RTMP, which does not support  
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.
```

#### 4. Enable camera preview

Before enabling camera preview, you must first provide the SDK with a `TXCloudVideoView` object to display video images. Given that `TXCloudVideoView` is inherited from `FrameLayout` in Android, you can:

1. Add a video rendering control in the XML file:

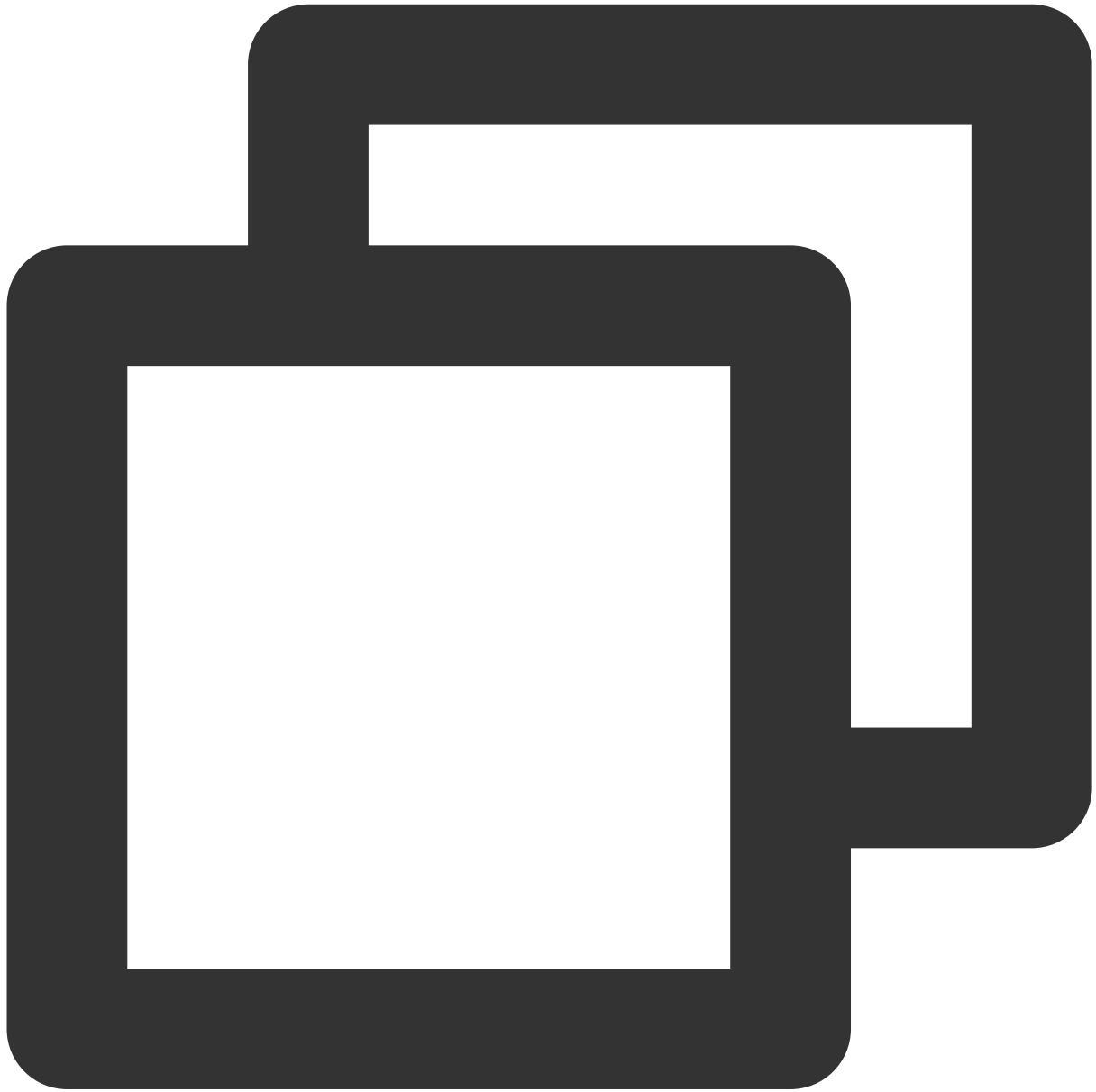


```
<com.tencent.rtmp.ui.TXCloudVideoView
```



```
android:id="@+id/pusher_tx_cloud_view"  
android:layout_width="match_parent"  
android:layout_height="match_parent" />
```

2. Call `startCamera` in `V2TXLivePusher` to enable camera preview for your mobile phone.



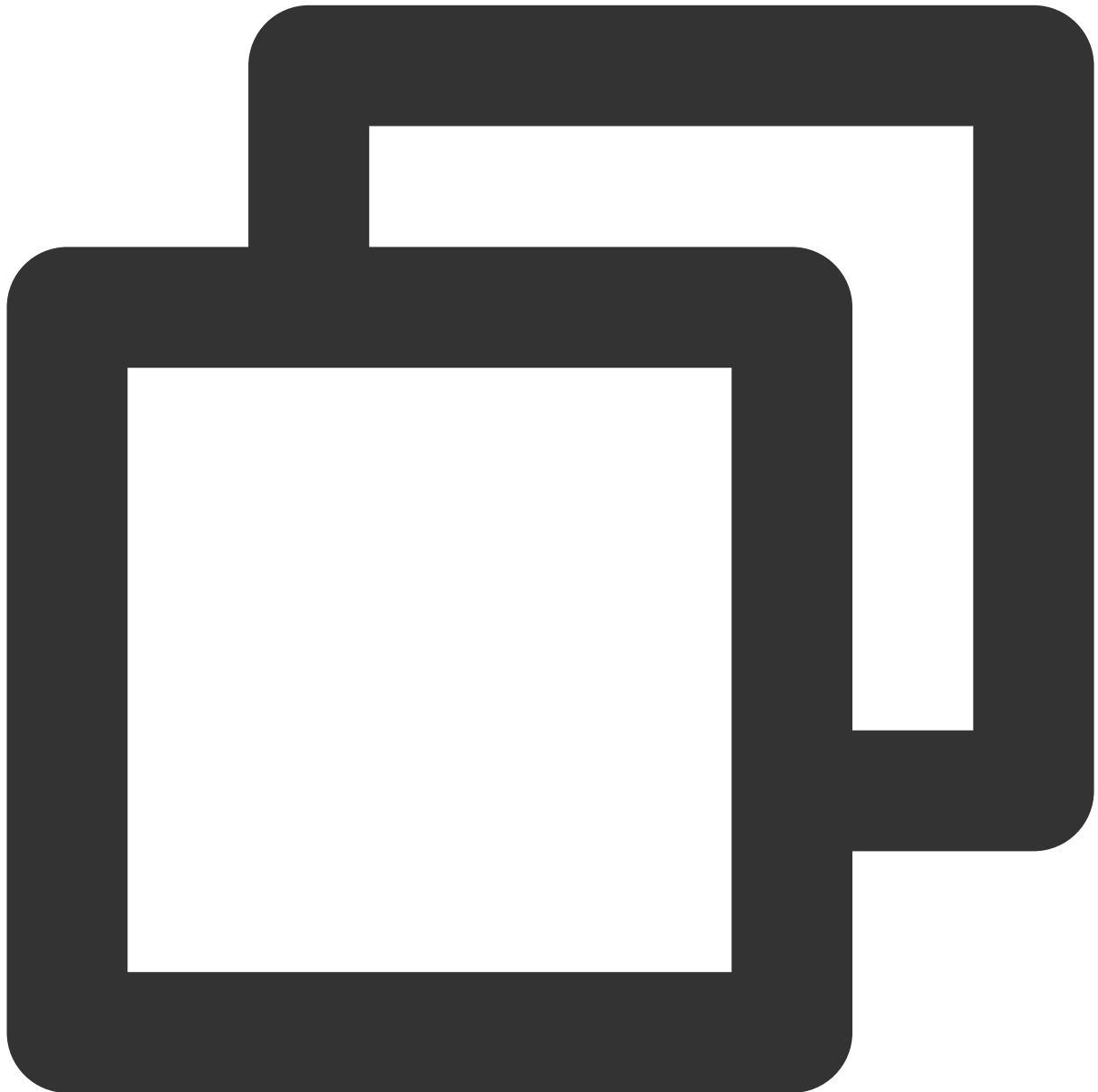
```
// Enable preview for the local camera  
TXCloudVideoView mPusherView = (TXCloudVideoView) findViewById(R.id.pusher_tx_cloud  
mLivePusher.setRenderView(mPusherView);  
mLivePusher.startCamera(true);  
mLivePusher.startMicrophone();
```

## 5. Start and stop publishing

1. After calling `startCamera` to enable camera preview, you can call the `startPush` API in `V2TXLivePusher` to start publishing.

### Note

if you choose `RTMP` protocol to push in [Step3](#), the generate of the push URL, please refer to [RTMP URL](#).



```
//This URL does not support co-anchoring. The stream is published to a live stream
String url = "rtmp://test.com/live/streamid?txSecret=xxxxx&txTime=xxxxxxxx";
int ret = mLivePusher.startPush(url);
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {
```

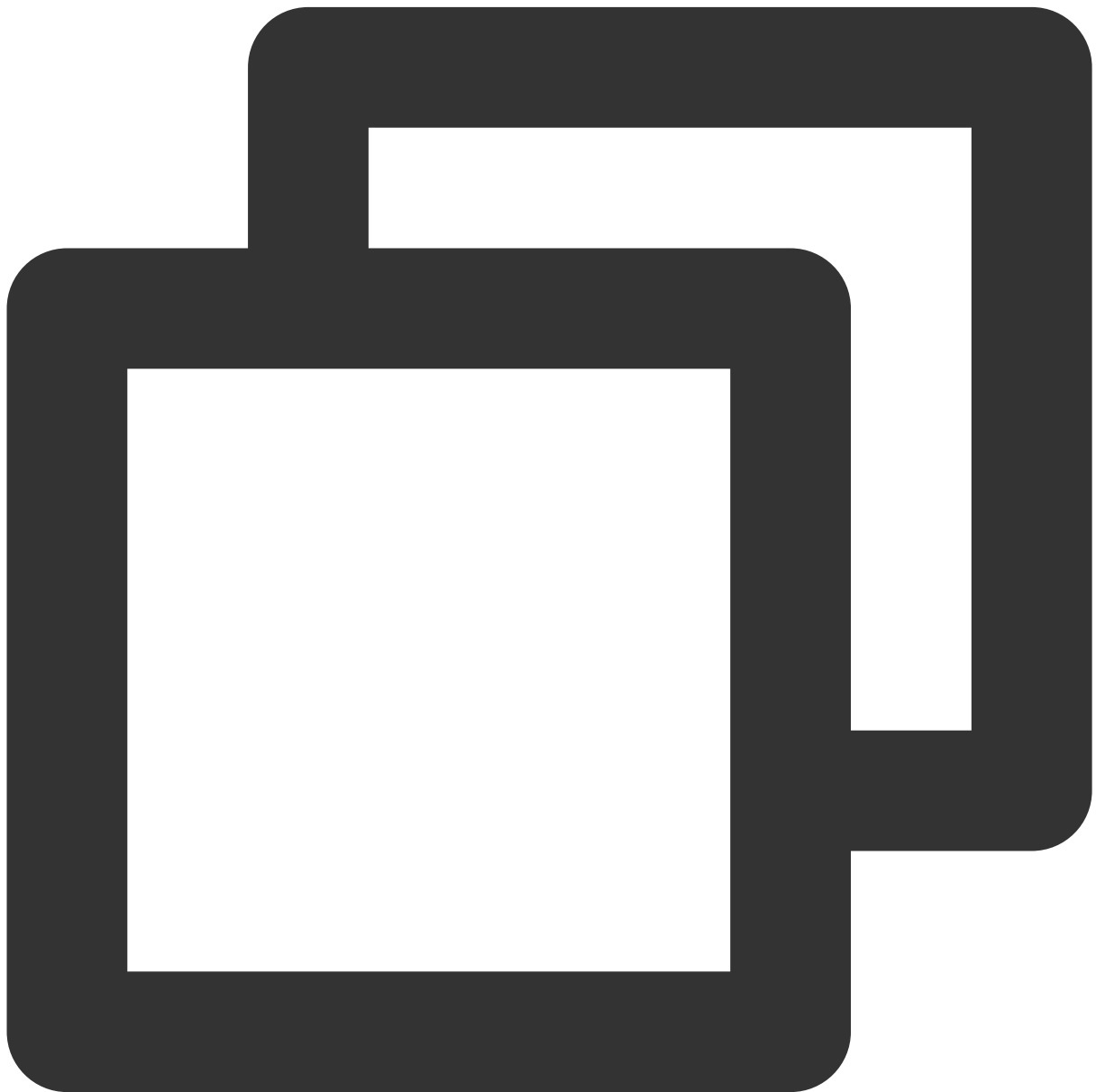
```
Log.i(TAG, "startRTMPPush: license verification failed");  
}
```

### Note

#### Reason for returning V2TXLIVE\_ERROR\_INVALID\_LICENSE?

If the startPush interface returns V2TXLIVE\_ERROR\_INVALID\_LICENSE, it means that your license verification failed, please check the url and key set in [Step 2: Configure the SDK for license authorization](#).

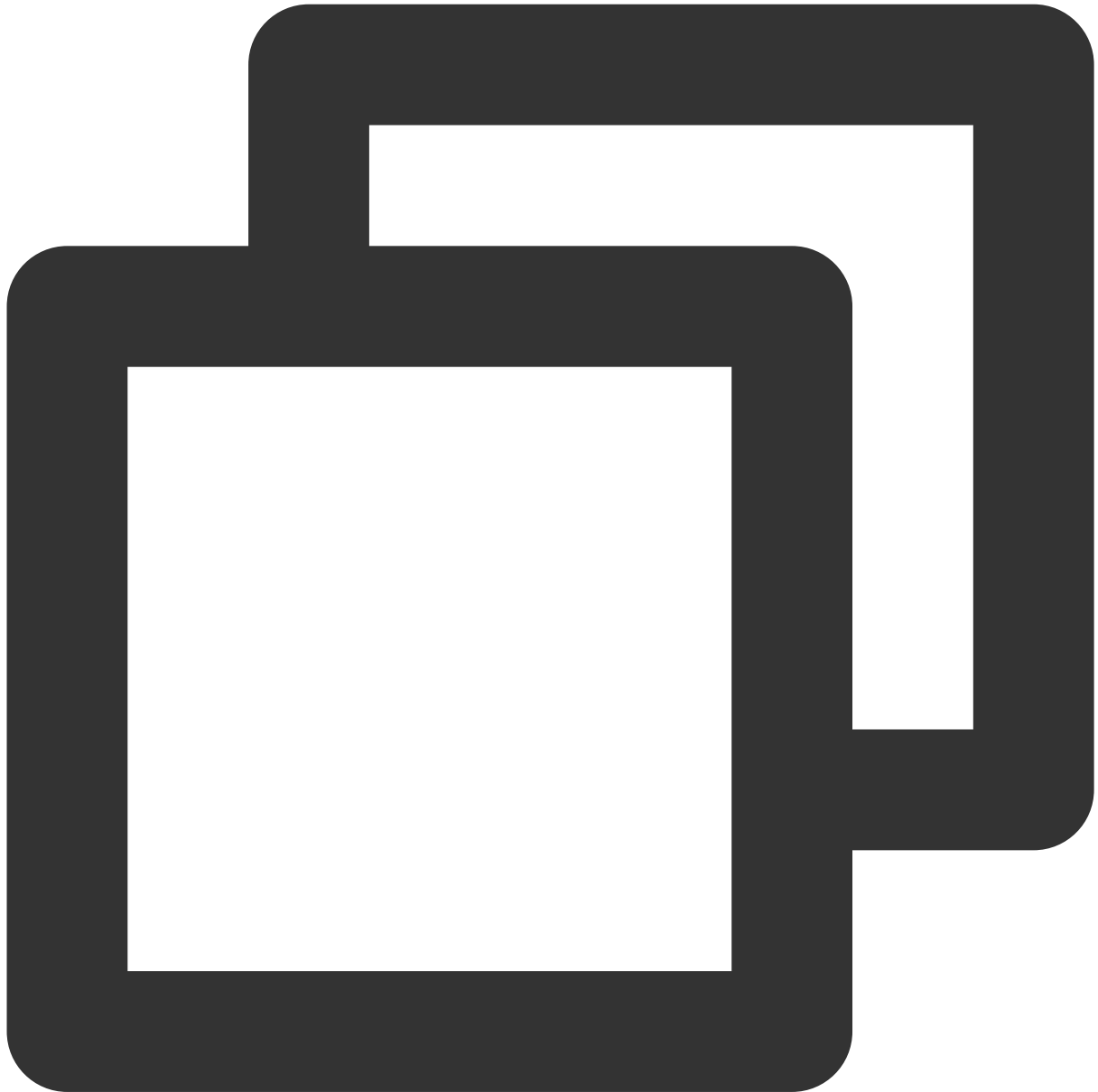
2. Call [stopPush](#) in `V2TXLivePusher` to stop publishing streams



```
//Stop publishing  
mLivePusher.stopPush();
```

## 6. Publish audio-only streams

If your live streaming scenarios involve audio only, you can skip [Step 4](#) or call `stopCamera` before `startPush` .



```
mLivePusher.startMicrophone();  
// The push stream can be started by passing in the corresponding URL according to  
String url = "rtmp://test.com/live/streamid?txSecret=xxxxxx&txTime=xxxxxxxxx";  
int ret = mLivePusher.startPush(url);
```

### Note

If you publish audio-only streams but no streams can be pulled from an RTMP, FLV, or HLS playback URL, there is a problem with your line configuration, please [submit a ticket](#) for help.

## 7. Set video quality

Call `setVideoQuality` in `V2TXLivePusher` to set the quality of videos watched by audience. The encoding parameters set determine the quality of videos presented to audience. The local video watched by the host is the original HD version that has not been encoded or compressed, and is therefore not affected by the settings. For details, please see [Setting Video Quality](#).

## 8. Set the beauty filter style and skin brightening and rosy skin effects

Call `getBeautyManager` in `V2TXLivePusher` to get a `TXBeautyManager` instance to set beauty filters.

### Beauty filter style

The SDK has three built-in beauty filter algorithms, each corresponding to a beauty filter style. Choose one that best fits your product positioning. For the definitions, see `TXLiveConstants.java`.

Beauty Filter Style	Description
BEAUTY_STYLE_SMOOTH	The smooth style, which features more obvious skin smoothing effects and is suitable for live showrooms
BEAUTY_STYLE_NATURE	The natural style, which retains more facial details and is more natural
BEAUTY_STYLE_PITU	The Pitu style, which uses the beauty filter algorithm developed by YouTu Lab. Its effect is between the smooth style and the natural style, that is, it retains more skin details than the smooth style and delivers more obvious skin smoothing effects than the natural style.

You can call the `setBeautyStyle` API of `TXBeautyManager` to set the beauty filter style.

Item	Configuration	Description
Beauty filter strength	Via the <code>setBeautyLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Skin brightening filter strength	Via the <code>setWhitenessLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Rosy skin filter strength	Via the <code>setRuddyLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.

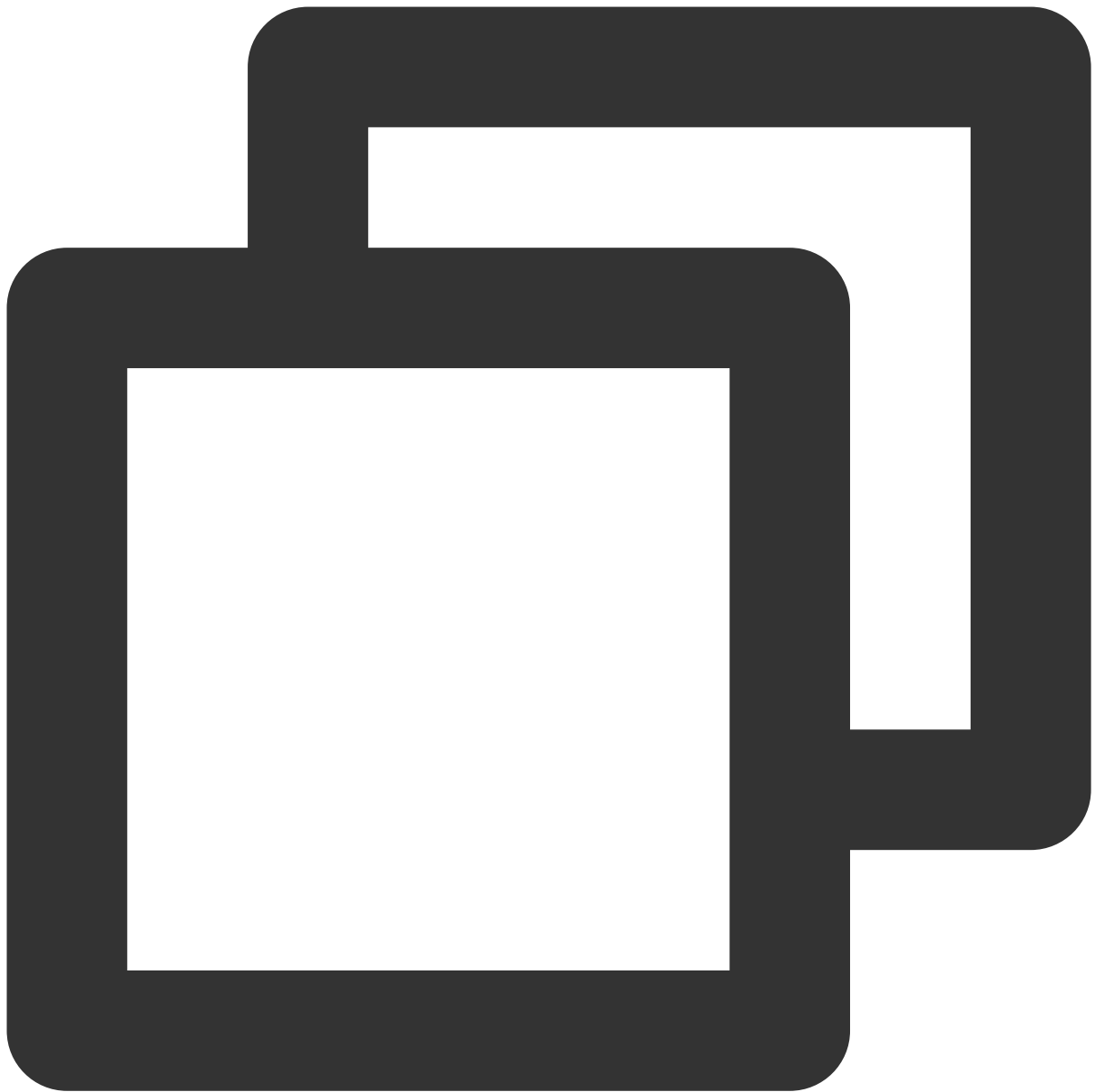
## 9. Set color filters

Call `getBeautyManager` in `V2TXLivePusher` to get a `TXBeautyManager` instance to set color filters.

Call the `setFilter` API in `TXBeautyManager` to set color filters. Color filters are a technology that adjusts the color tone of sections of an image. For example, it may lighten the yellow sections of an image to achieve the effect of skin brightening, or add warm tones to a video to give it a refreshing and soft boost.

Call the `setFilterStrength` API in `TXBeautyManager` to set the strength of a color filter. The higher the strength, the more obvious the effect.

Based on our experience of operating Mobile QQ and Now Live, it's not enough to use only the `setBeautyStyle` API in `TXBeautyManager` to set the beauty filter style. The `setBeautyStyle` API must be used together with `setFilter` to produce richer effects. Given this, our designers have developed 17 built-in color filters for you to choose from.



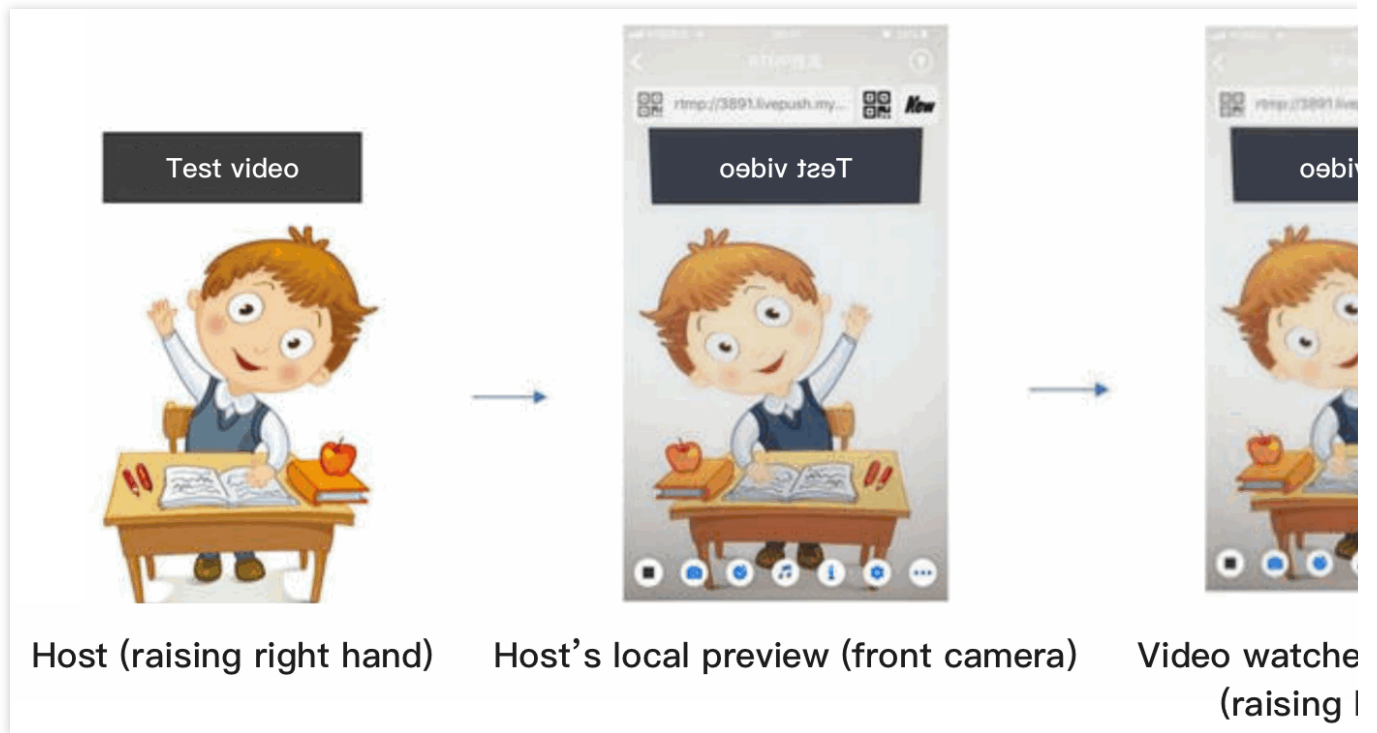
```
// Select the desired color filter file. The filter file can be obtained from the r
Bitmap filterBmp = decodeResource(getResources(), R.drawable.tuibeauty_filter_biaoz
mLivePusher.getBeautyManager().setFilter(filterBmp);
mLivePusher.getBeautyManager().setFilterStrength(0.5f);
```

## 10. Manage devices

`V2TXLivePusher` provides a series of APIs for the control of devices. You can call `getDeviceManager` to get a `TXDeviceManager` instance for device management. For detailed instructions, please see [TXDeviceManager API](#).

## 11. Set the video mirroring effect for audience

Call `setEncoderMirror` in `V2TXLivePusher` to set the camera mirror mode, which affects the way video images are presented to audience. By default, the local video watched by the host is flipped when the front camera is used, which creates the same effect as a mirror does. The video watched by audience is the same as that watched by the host, as shown below.

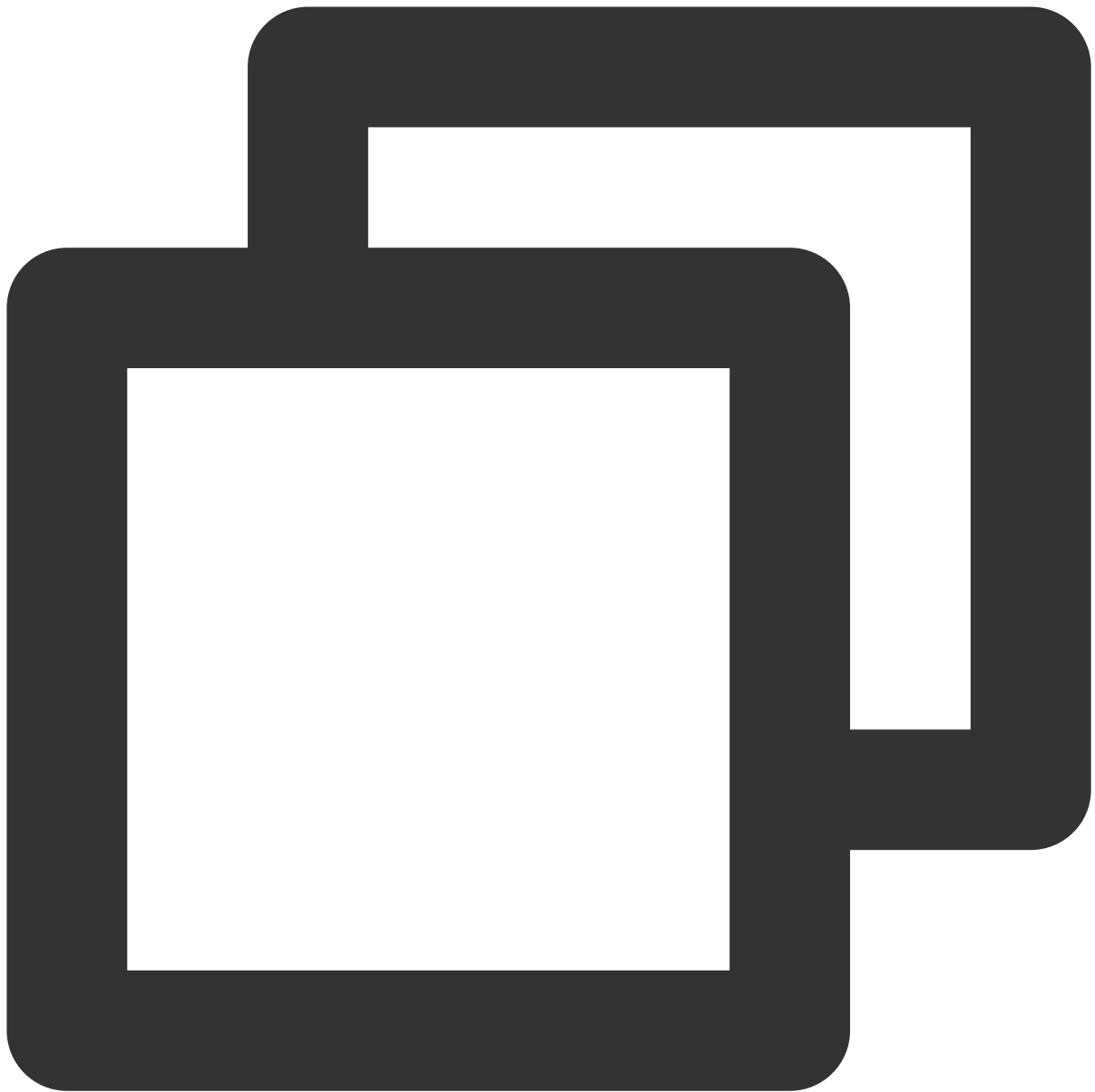


## 12. Publish streams in landscape mode

In most cases, hosts stream while holding their phones vertically, and audience watch videos in portrait resolutions (e.g., 540 × 960). However, there are also cases where hosts hold phones horizontally, and ideally, audience should watch videos in landscape resolutions (960 × 540).

By default, `V2TXLivePusher` outputs videos in portrait resolutions. You can publish landscape-mode videos to audience by modifying a parameter of the `setVideoQuality` API.





```
V2TXLiveDef.V2TXLiveVideoEncoderParam param = new V2TXLiveDef.V2TXLiveVideoEncoderParam();
param.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionModeLandscape : V2TXLiveVideoResolutionModePortrait;
mLivePusher.setVideoQuality(param);
```

### 13. Set audio effects

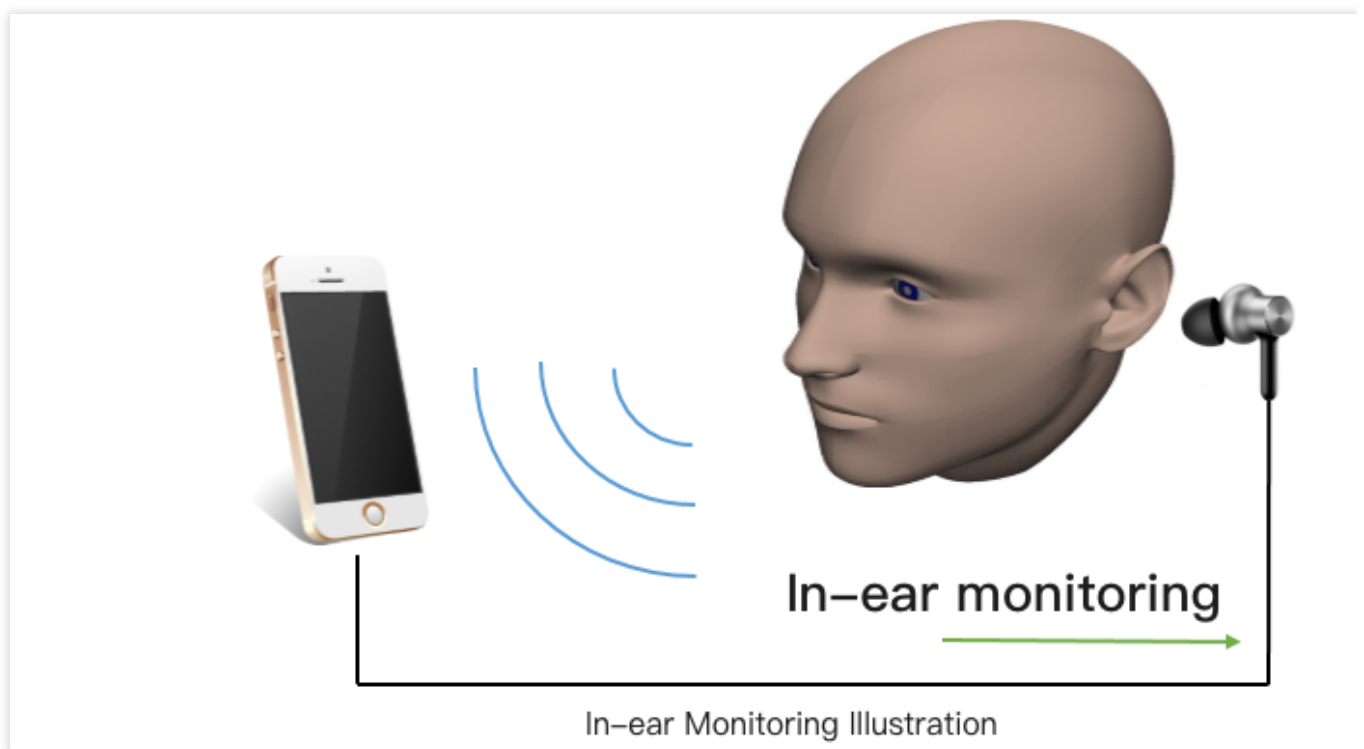
Call `getAudioEffectManager` in `V2TXLivePusher` to get a `TXAudioEffectManager` instance, which can be used to mix background music and set in-ear monitoring, reverb, and other audio effects. Background music

mixing means mixing into the published stream the music played by the host's phone so that audience can also hear the music.

Call the `enableVoiceEarMonitor` API in `TXAudioEffectManager` to enable in-ear monitoring, which allows hosts to hear their vocals in earphones when they sing.

Call the `setVoiceReverbType` API in `TXAudioEffectManager` to add reverb effects such as karaoke, hall, husky, and metal. The effects are applied to the videos watched by audience.

Call the `setVoiceChangerType` API in `TXAudioEffectManager` to add voice changing effects such as little girl and middle-aged man to enrich host-audience interaction. The effects are applied to the videos watched by audience.



#### Note

For detailed instructions, please see [TXAudioEffectManager API](#).

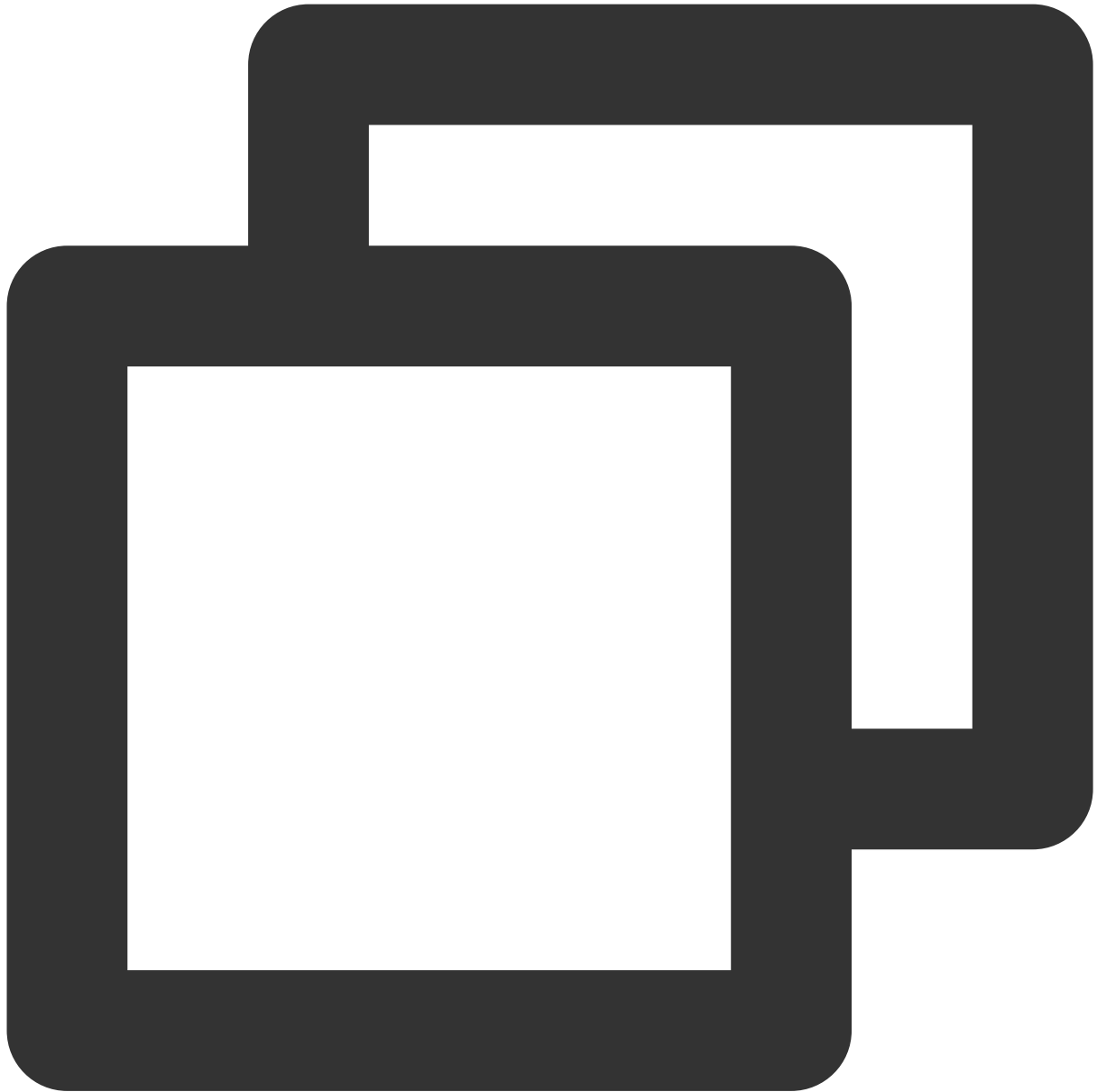
## 14. Set watermarks

Call `setWatermark` in `V2TXLivePusher` to add a watermark to videos output by the SDK. The position of the watermark is determined by the `(x, y, scale)` parameter passed in.

The watermark image must be in PNG rather than JPG format. The former carries opacity information, which allows the SDK to better address the image aliasing issue (changing the extension of a JPG image to PNG won't work).

The `(x, y, scale)` parameter specifies the normalized coordinates of the watermark relative to the resolution of the published video. For example, if the resolution of the published video is 540 x 960, and `(x, y, scale)` is set

to `(0.1, 0.1, 0.1)` , the actual pixel coordinates of the watermark will be  $(540 \times 0.1, 960 \times 0.1)$ . The width of the watermark image will be the video width  $\times 0.1$ , and the height will be scaled automatically.



```
// Set a video watermark  
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(), R.drawable.wat
```

## 15. Inform hosts of poor network conditions

Hosts should be informed when their network conditions are bad and be prompted to check their network.



You can capture the **V2TXLIVE\_WARNING\_NETWORK\_BUSY** event using `onWarning` in `V2TXLivePusherObserver`. The event indicates poor network conditions for hosts, which result in stuttering for audience. When this event occurs, you can send a UI message about poor network conditions to hosts.



```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
    if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
        showNetBusyTips(); // Show network tips
    }
}
```

## 16. Send SEI messages

Call the [sendSeiMessage](#) API in `V2TXLivePusher` to send SEI messages. SEI refers to the supplementary enhancement information of encoded video. It is not used most of the time, but you can insert custom information into SEI messages. The information will be forwarded to audience by live streaming CDNs. The applications for SEI messages include:

Live quiz: The publisher can use SEI messages to send questions to the audience. SEI can ensure synchronization among audio, video, and the questions.

Live showroom: The publisher can use SEI messages to display lyrics to the audience in real time. The effects are not affected by reduction in video encoding quality.

Online education: The publisher can use SEI messages to display pointers and sketches on slides to the audience in real time.

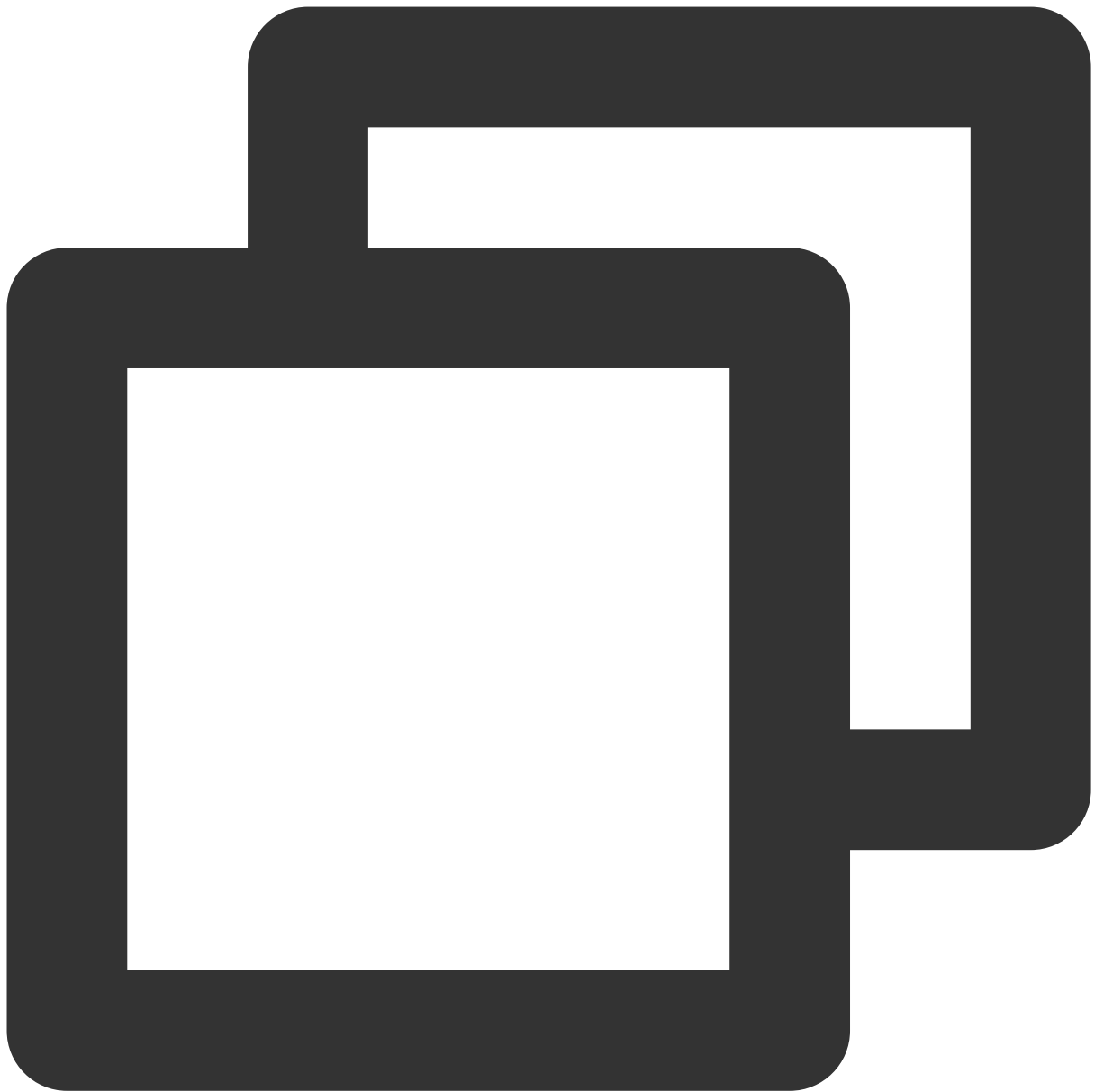
Custom data is inserted directly into video data and therefore cannot be too large in size (preferably several bytes). It's common to insert information such as custom timestamps.



```
//Sample code for Android
int payloadType = 5;
String msg = "test";
mTXLivePusher.sendSeiMessage(payloadType, msg.getBytes("UTF-8"));
```

Common open-source players or web players are incapable of parsing SEI messages. You must use `V2TXLivePlayer`, the built-in player of LiteAVSDK.

#### 1. Configuration:



```
int payloadType = 5;  
mTXLivePlayer.enableReceiveSeiMessage(true, payloadType)
```

2. If the video streams played by `V2TXLivePlayer` contain SEI messages, you will receive the messages via the `onReceiveSeiMessage` callback in `V2TXLivePlayerObserver`.

## Event Handling



## Listening for events

The SDK listens for publishing events and errors via the [V2TXLivePusherObserver](#) delegate. See [V2TXLiveCode](#) for a detailed list of events and error codes.

## Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common error not yet classified
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Stuttering during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile

		devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the user is having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system.

# Publishing from Screen

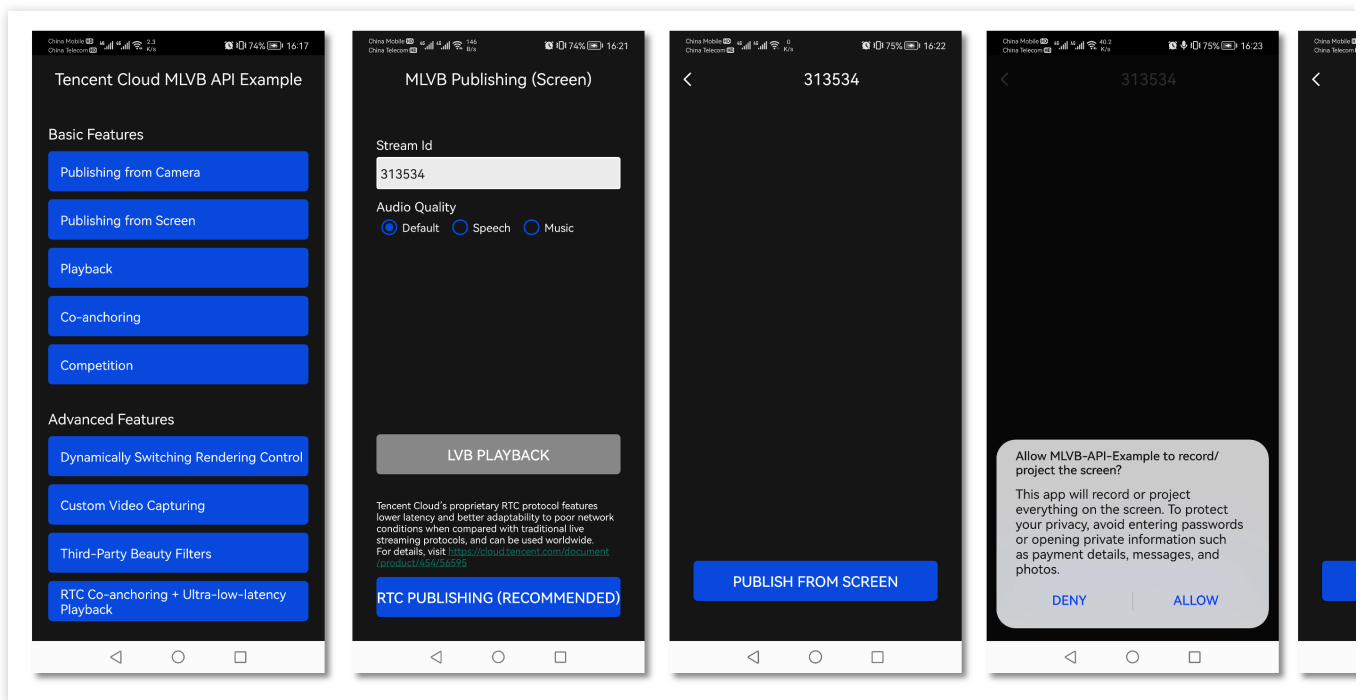
Last updated : 2024-01-13 15:49:41

## Overview

Publishing from screen means using hosts' phones as the source for live streaming. It may be combined with local camera preview and is used in scenarios such as game streaming and mobile application demonstration. Tencent Cloud's LiteAVSDK offers the screen sharing capability via `V2TXLivePusher`.

### Note:

By adding a floating window to display the image of the local camera, you can include camera preview into the streams published from the screen.



## Restrictions

Screen recording is supported in Android 5.0 and above.

Floating windows need to be enabled manually on some mobile phones and systems.

## Sample Code

Platform	GitHub Address
iOS	<a href="#">LivePushScreenViewController.m</a>
Android	<a href="#">LivePushScreenActivity.java</a>
Flutter	<a href="#">live_screen_push.dart</a>

## Integration

## Step 1. Download SDK development kit

[Download](#) the SDK and follow the instructions in [SDK Integration](#) to integrate the SDK into your application.

## Step 2. Configure License Authorization for SDK

1. Obtain license authorization :

If you have obtained the relevant license authorization, [Need to Get License URL and License Key in Cloud Live Console](#)

Create Official License

Price Overview

Create

An official license is valid for a year. Click [Create] to purchase one. Please make sure that the bundle ID and package name entered are correct as the information cannot be modified after submission.

▼ Official License

Application Name

Package Name

Bundle Id

Key

LicenseUrl

Start Date

End Date

Live

com

com

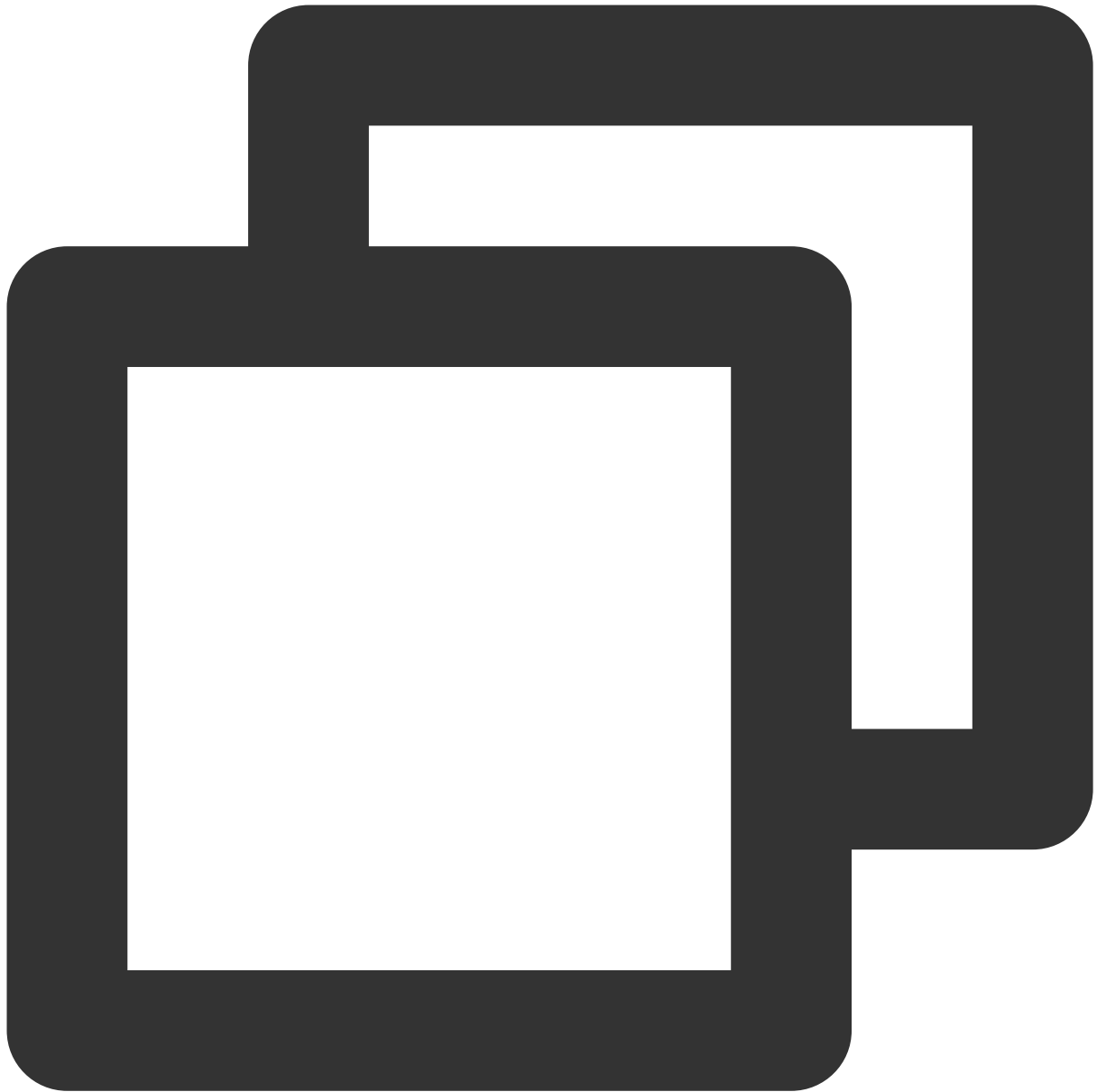
..licence

2021-12-03

2022-12-03

If you have not yet obtained the license authorization, Please reference [Adding and Renewing Licenses](#) to make an application.

2. Before your App calls SDK-related functions (it is recommended in the Application class), set the following settings:



```
public class MApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        String licenceURL = ""; // your licence url
        String licenceKey = ""; // your licence key
        V2TXLivePremier.setEnvironment("GDPR"); // set your environment
        V2TXLivePremier.setLicence(this, licenceURL, licenceKey);
        V2TXLivePremier.setObserver(new V2TXLivePremierObserver() {
            @Override
```

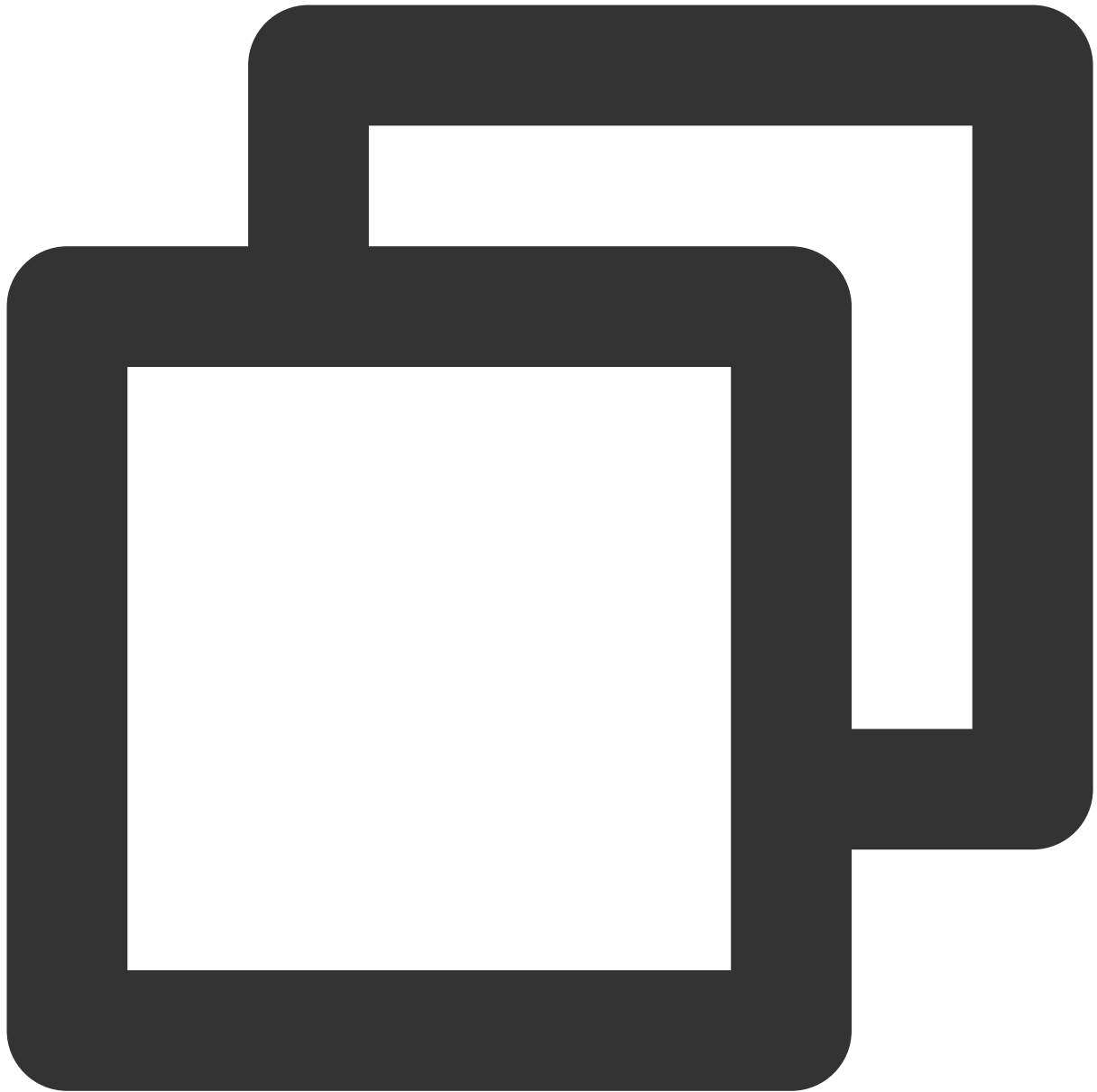
```
        public void onLicenceLoaded(int result, String reason) {  
            Log.i(TAG, "onLicenceLoaded: result:" + result + ", reason:" + reason);  
        }  
    });  
}
```

**Note:**

The `packageName` configured in the license must be the same as the application itself, otherwise the push stream will fail.

**Step 3. Add Activity**

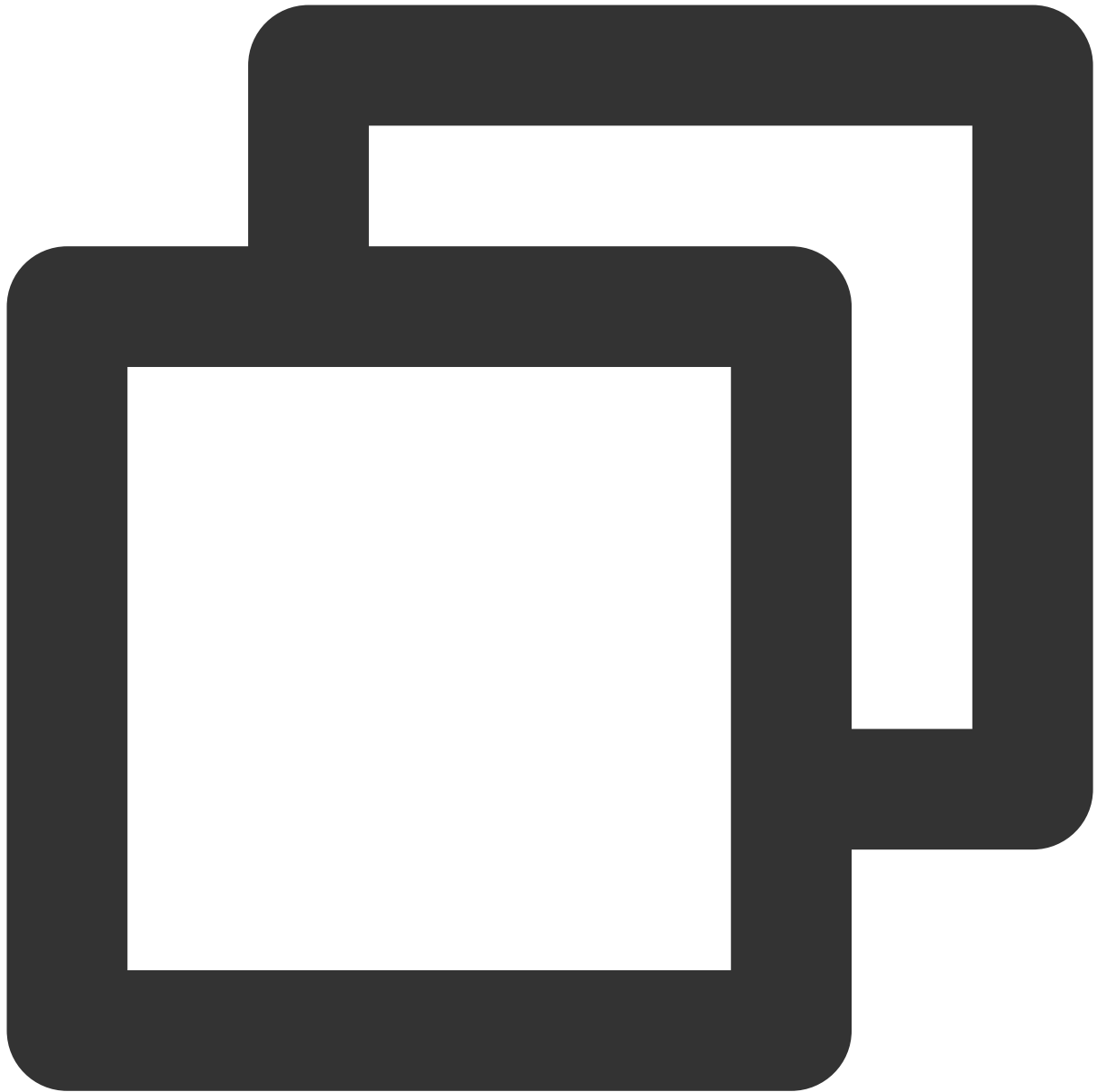
Paste the following activity in the manifest file (no need to add it if it exists in the project code).



```
<activity
    android:name="com.tencent.rtmp.video.TXScreenCapture$TXScreenCaptureAssistantAc
    android:theme="@android:style/Theme.Translucent"/>
```

#### Step 4. Create a stream publishing object

Create a **V2TXLivePusher** object, which will be responsible for publishing operations.



```
// Specify the corresponding live broadcast protocol as RTMP, which does not support  
V2TXLivePusher mLivePusher = new V2TXLivePusherImpl(this, V2TXLiveDef.V2TXLiveMode.
```

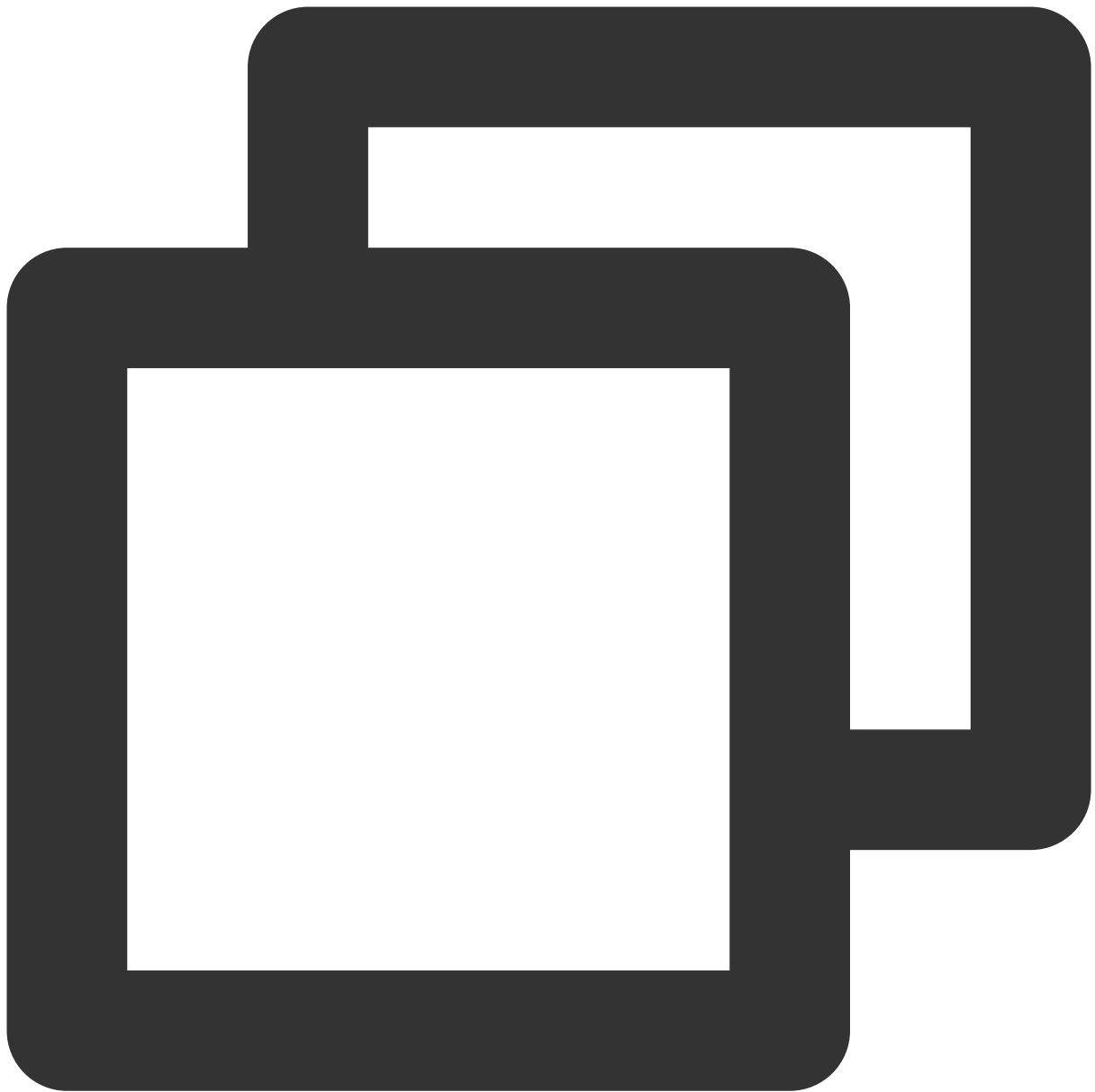
## Step 5. Start stream publishing

Use [startScreenCapture](#) to start screen recording, and use `V2TXLivePusher::startPush` to pushing

### Note:

if you choose `RTMP` protocol to push in [Step4](#), The generate of the push URL, please refer to [RTMP URL](#).





```
// The push stream can be started according to the push stream protocol. RTMP cannot  
String url = "rtmp://test.com/live/streamid?txSecret=xxxxxx&txTime=xxxxxxxxx";  
mLivePusher.startMicrophone();  
mLivePusher.startScreenCapture();  
int ret = mLivePusher.startPush(url);  
if (ret == V2TXLIVE_ERROR_INVALID_LICENSE) {  
    Log.i(TAG, "startRTMPPush: license verification failed");  
}
```

**Note:**

### Reason for returning V2TXLIVE\_ERROR\_INVALID\_LICENSE?

If the `startPush` interface returns `V2TXLIVE_ERROR_INVALID_LICENSE`, it means that your license verification failed, please check the url and key set in [Step 2: Configure the SDK for license authorization](#).

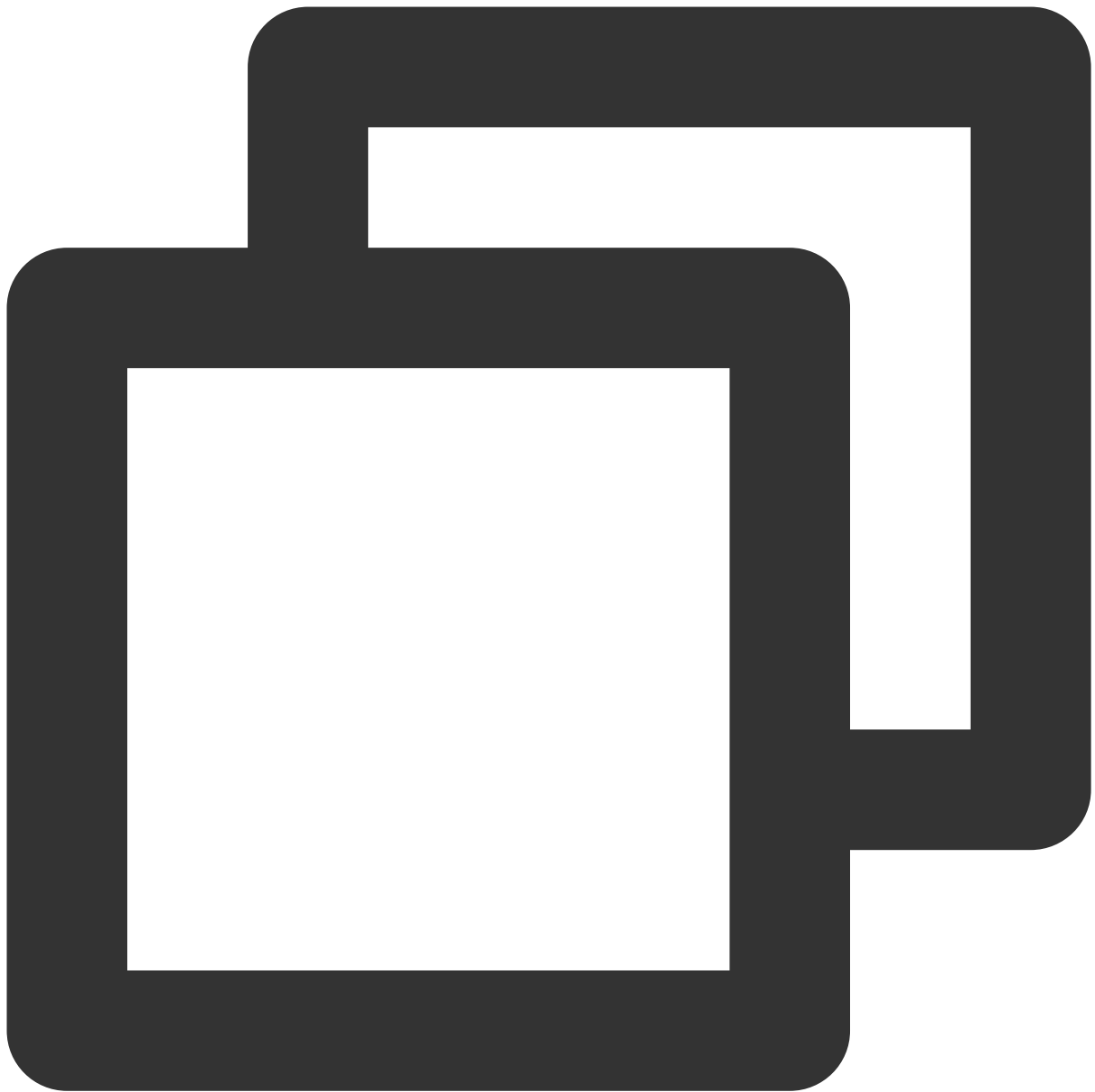
**startScreenCapture** is used to start screen recording, which is a built-in feature of the Android system. For security reasons, before screen recording starts, Android will pop up a window asking users whether to start screen recording, and you should agree.

## Step 6. Set watermarks

Call `setWatermark` in `V2TXLivePusher` to add a watermark to videos output by the SDK. The position of the watermark is determined by the `(x, y, scale)` parameter passed in.

The watermark image must be in PNG rather than JPG format. The former carries opacity information, which allows the SDK to better address the image aliasing issue (changing the extension of a JPG image to PNG won't work).

The `(x, y, scale)` parameter specifies the normalized coordinates of the watermark relative to the resolution of the published video. For example, if the resolution of the published video is 540 x 960, and `(x, y, scale)` is set to `(0.1, 0.1, 0.1)`, the actual pixel coordinates of the watermark will be (540 x 0.1, 960 x 0.1). The width of the watermark will be the video width x 0.1, and the height will be scaled automatically.



```
// Set a video watermark  
mLivePusher.setWatermark(BitmapFactory.decodeResource(getResources(), R.drawable.wat
```

## Step 7. Set video quality

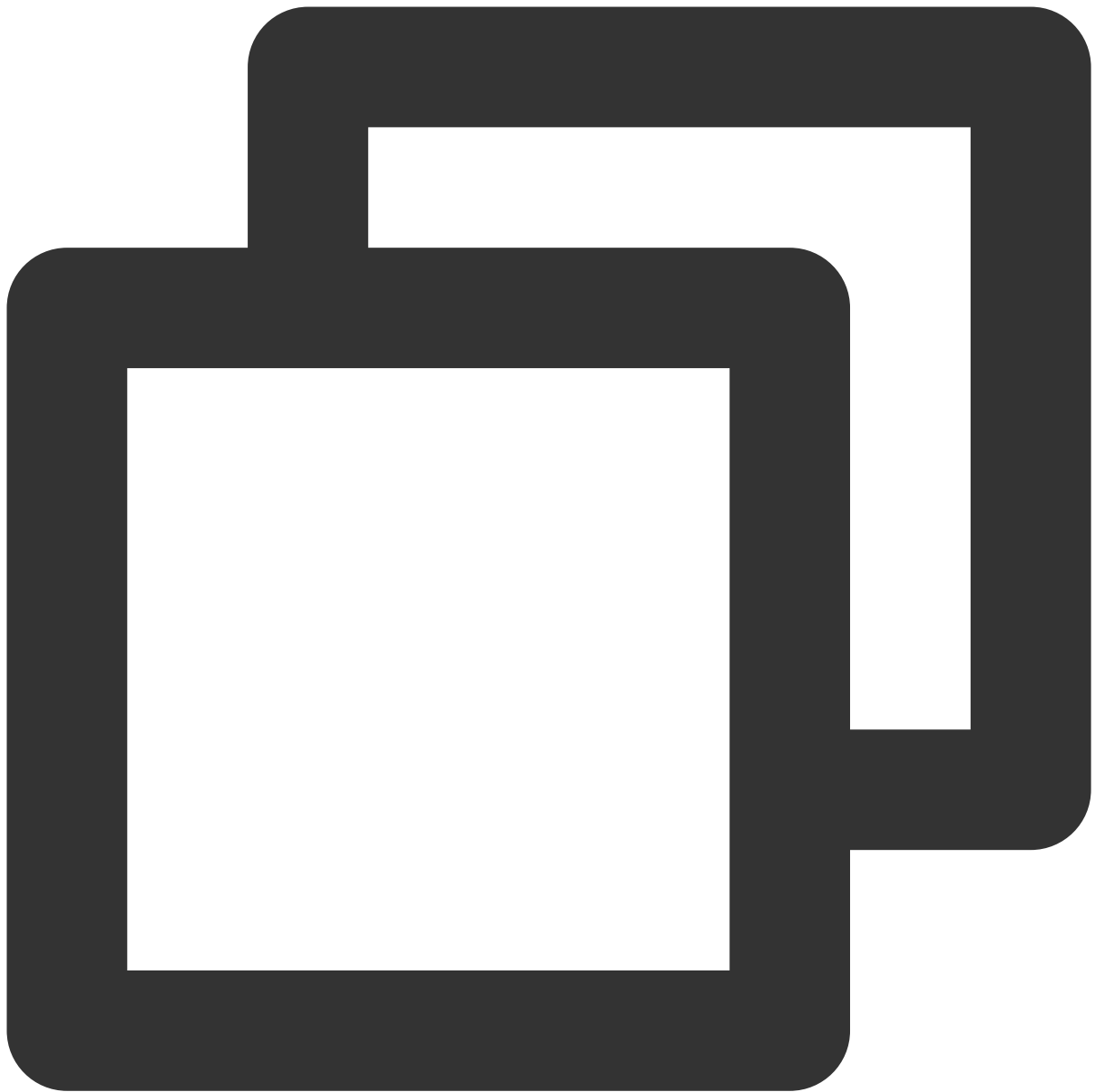
Call `setVideoQuality` in `V2TXLivePusher` to set the quality of videos watched by audience. The encoding parameters set determine the quality of videos presented to audience. The local video watched by the host is the original HD version that has not been encoded or compressed, and is therefore not affected by the settings. For details, please see [Setting Video Quality](#).

## Step 8. Inform hosts of poor network conditions

Connecting phones to Wi-Fi does not necessarily guarantee network conditions. In case of poor Wi-Fi signal or limited bandwidth, the network speed of a Wi-Fi connected phone may be slower than that of a phone using 4G. Hosts should be informed when their network conditions are bad and be prompted to switch to a different network.



You can capture the **V2TXLIVE\_WARNING\_NETWORK\_BUSY** event using [onWarning](#) in `V2TXLivePusherObserver`. The event indicates poor network conditions for hosts, which result in stuttering for audience. When this event occurs, you can send a UI message about poor network conditions to hosts, as shown above.

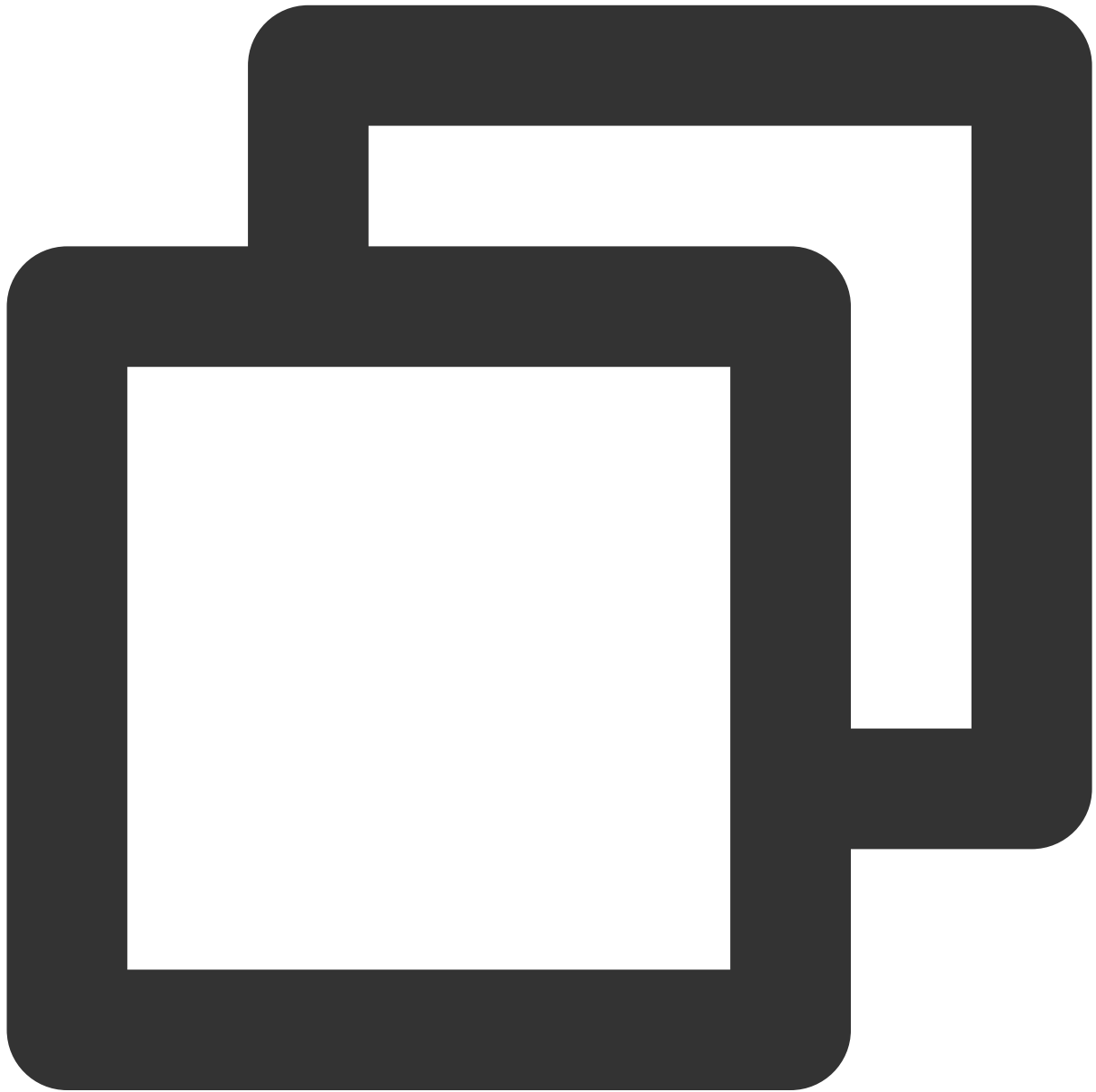


```
@Override
public void onWarning(int code, String msg, Bundle extraInfo) {
    if (code == V2TXLiveCode.V2TXLIVE_WARNING_NETWORK_BUSY) {
        showNetBusyTips(); // Show a "network busy" message
    }
}
```

## Step 9. Set the orientation

In most cases, hosts stream while holding their phones vertically, and audience watch videos in portrait resolutions (e.g., 540 × 960). However, there are also cases where hosts hold phones horizontally, and ideally, audience should watch videos in landscape resolutions (960 × 540), as shown below:

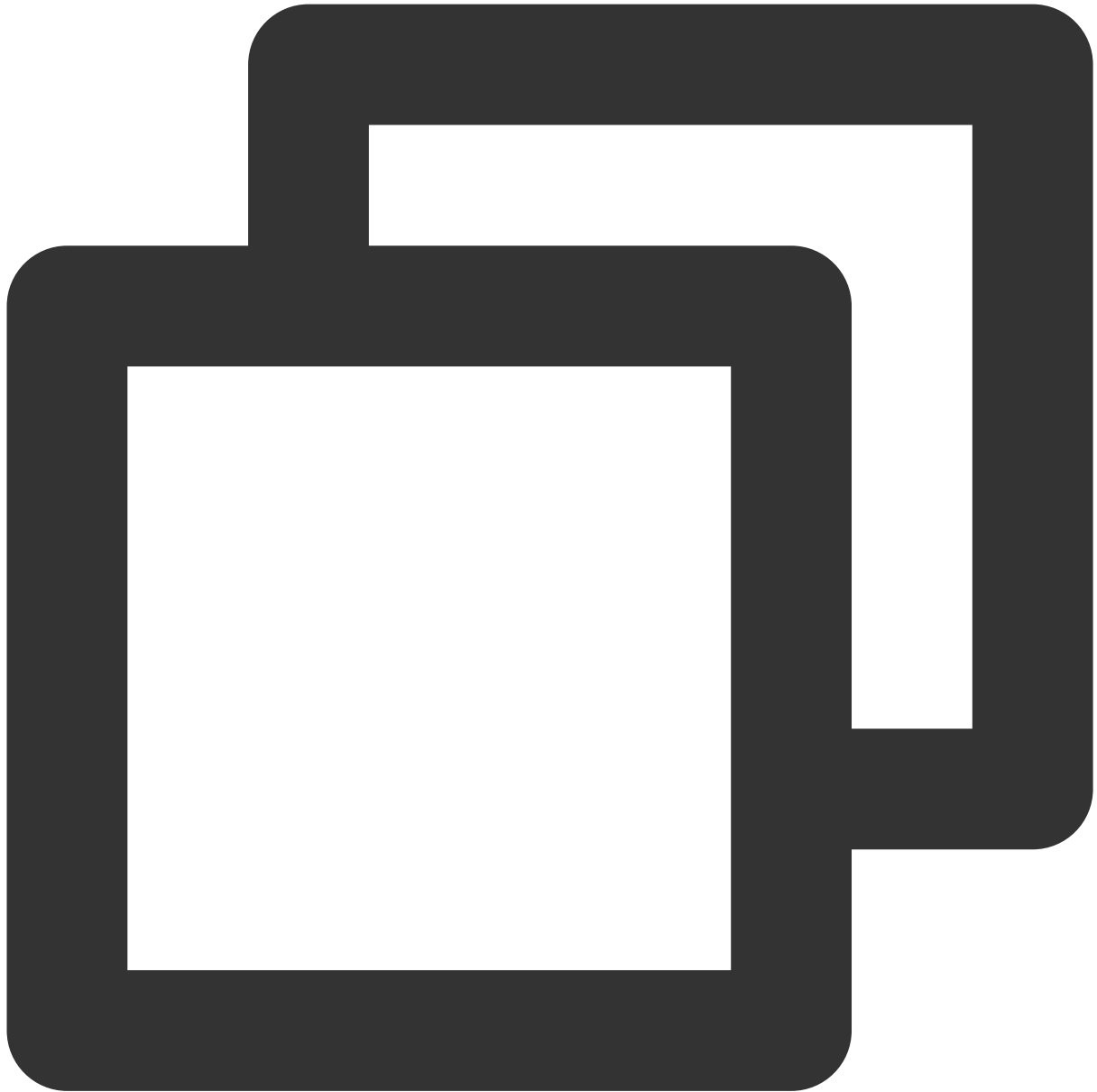
By default, `V2TXLivePusher` outputs videos in portrait resolutions. You can output landscape-mode videos to audience by modifying a parameter of the [setVideoQuality](#) API.



```
mLivePusher.setVideoQuality(mVideoResolution, isLandscape ? V2TXLiveVideoResolution
```

## Step 10. Stop publishing streams

As there can be only one `V2TXLivePusher` object running at a time, make sure that you release all the resources when stopping publishing.



```
// Stop screen sharing and release the resources
public void stopPublish() {
    mLivePusher.stopScreenCapture();
    mLivePusher.setObserver(null);
    mLivePusher.stopPush();
}
```

# Event Handling

## Listening for events

The SDK listens for publishing events and errors via the [V2TXLivePusherObserver](#) delegate. See [V2TXLiveCode](#) for a detailed list of events and error codes.

## Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common error not yet classified
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Stuttering during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.



V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the user is having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system.

# Web

Last updated : 2024-01-13 15:49:41

The TXLivePusher SDK is mainly used to publish streams for LEB (ultra-low latency streaming). It can publish audio and video the browser captures from the camera, screen, or a local media file to live streaming servers via WebRTC.

## Note :

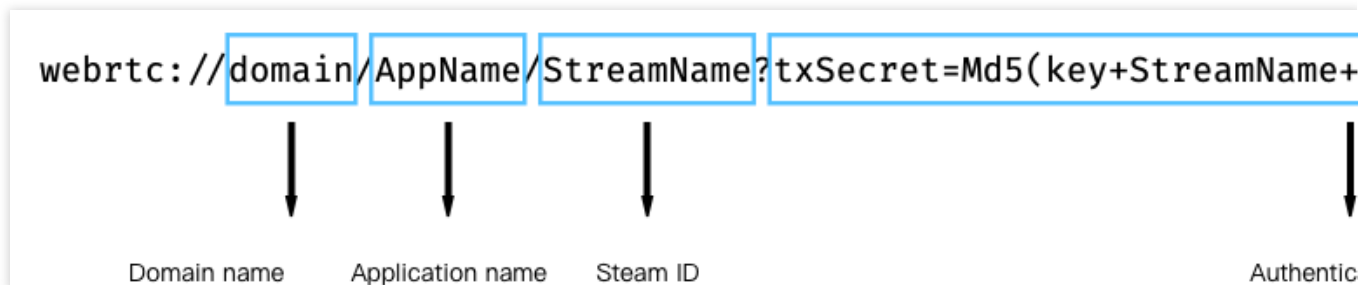
With WebRTC, each domain name can be used for the publishing of up to **concurrent 100 streams** by default. If you want to publish more streams, please [submit a ticket](#).

## Basics

Below are some basics you need to know before integrating the SDK.

### Splicing publishing URLs

To use Tencent Cloud live streaming services, you need to splice publishing URLs in the format required by Tencent Cloud, which consists of four parts.



An authentication key is not required. You can enable publishing authentication if you need hotlink protection. For details, please see [Splicing CSS URLs](#).

### Browser support

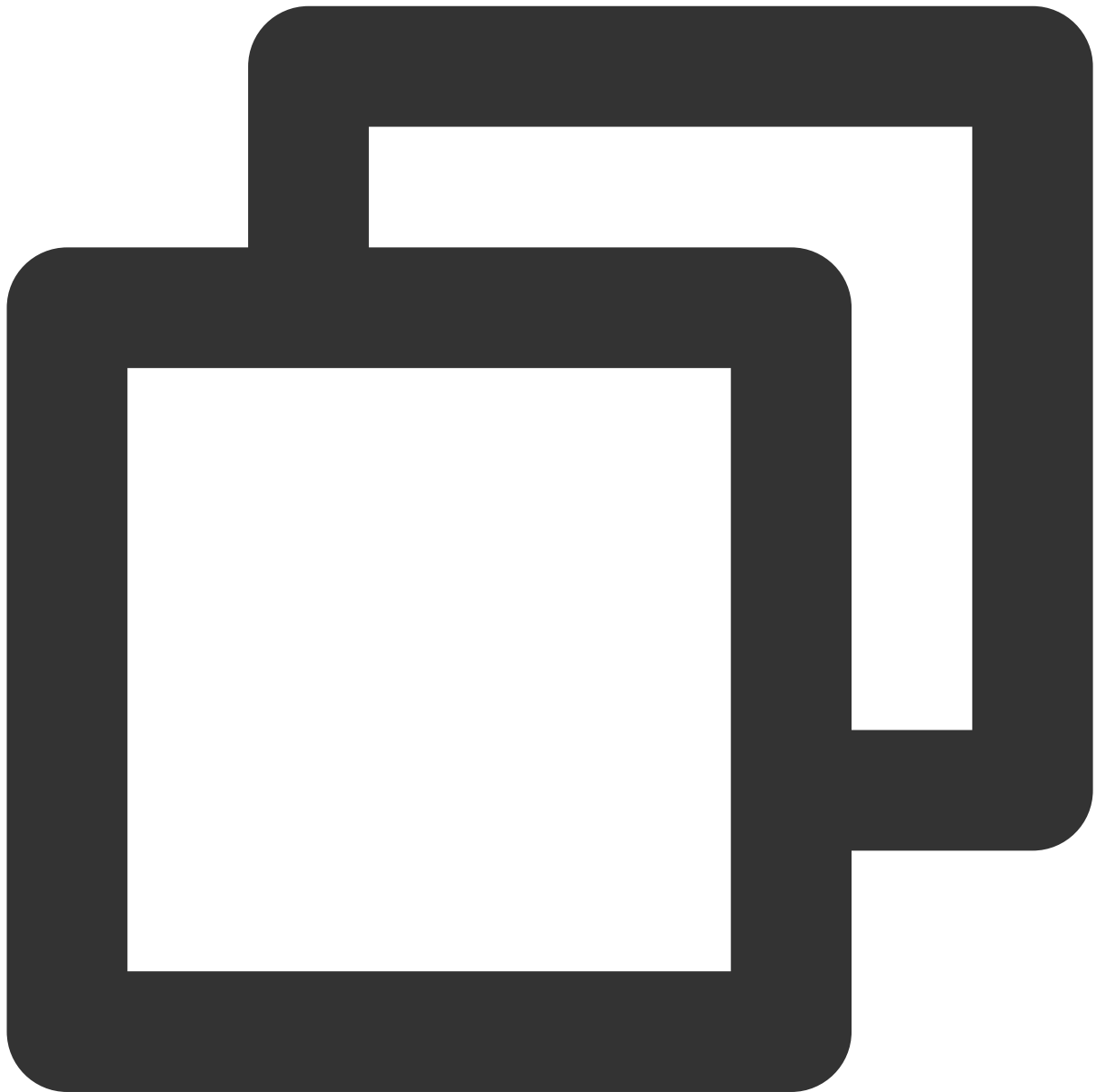
Publishing for LEB relies on WebRTC and therefore can only be used on OS and browsers that support WebRTC. The audio/video capturing feature is poorly supported on mobile browsers. For example, mobile browsers do not support screen recording, and only iOS 14.3 and above allow requesting camera access. Therefore, the publishing SDK is mainly used on desktop browsers. The latest version of Chrome, Firefox, and Safari all support publishing for LEB.

To publish streams on mobile browsers, use the [MLVB SDK](#).

## SDK Integration

### Step 1. Prepare the page

Add an initialization script to the (desktop) page from which streams are to be published.



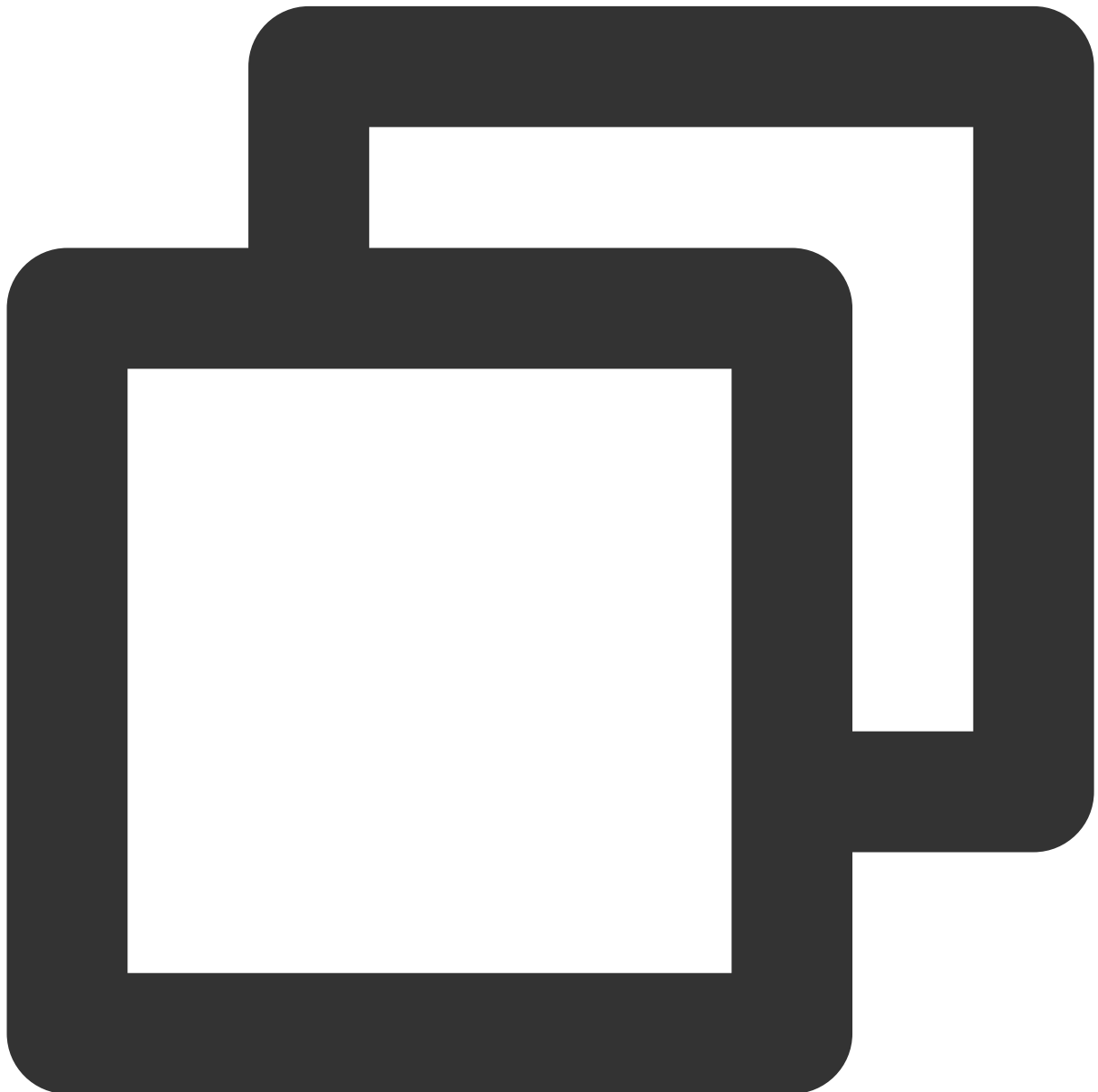
```
<script src="https://video.sdk.qcloudcdn.com/web/TXLivePusher-2.1.0.min.js" charse
```

**Note :**

The script needs to be imported into the `body` part of the HTML code. If it is imported into the `head` part, an error will be reported.

## Step 2. Add a container to the HTML page

Add a player container to the section of the page where local video is to be played. This is achieved by adding a div and giving it a name, for example, `id_local_video`. Local video will be rendered in the container. To adjust the size of the container, style the div using CSS.

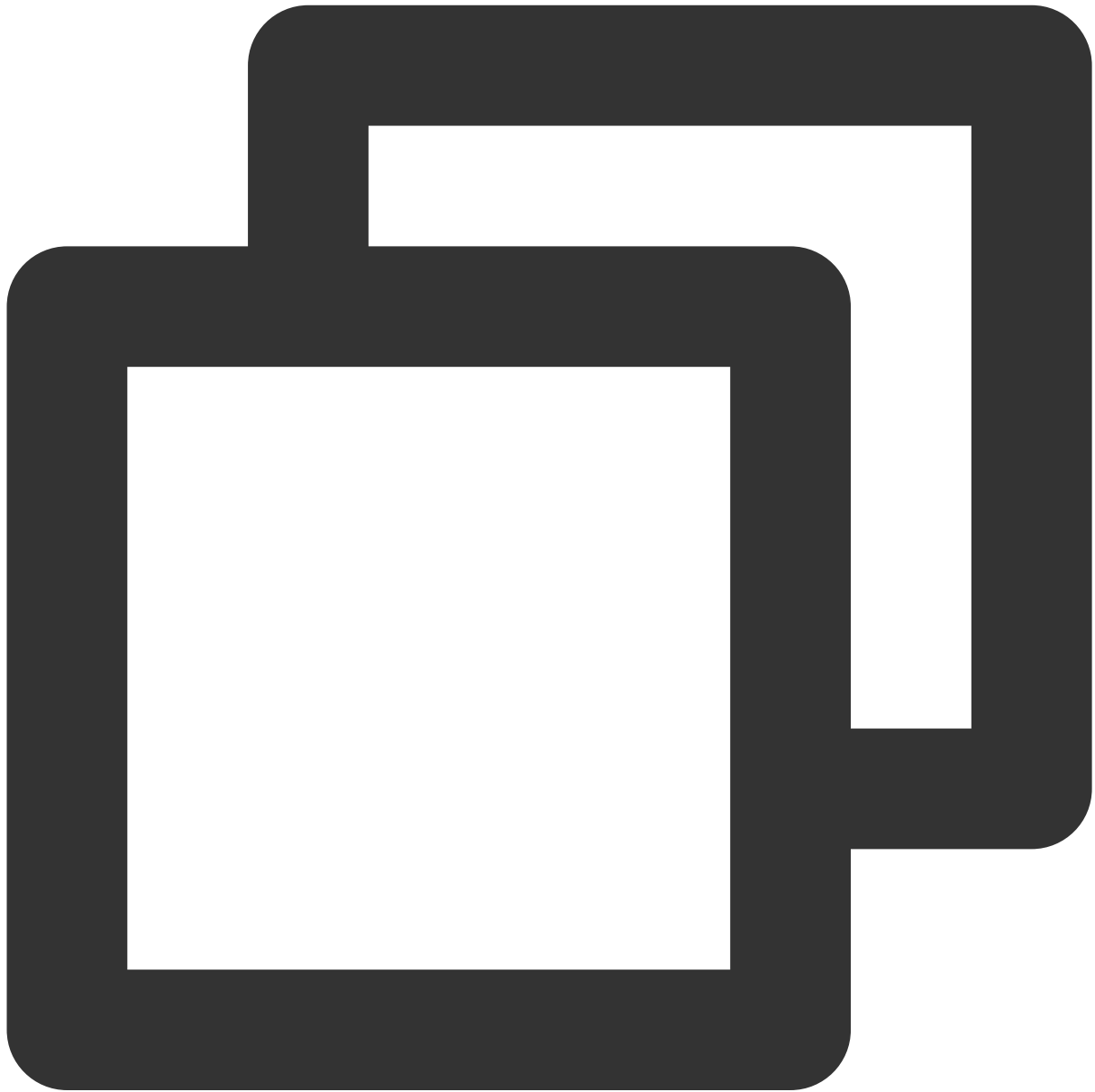


```
<div id="id_local_video" style="width:100%;height:500px;display:flex;align-items:ce
```

### Step 3. Publish streams

#### 1. Generate an instance of the publishing SDK:

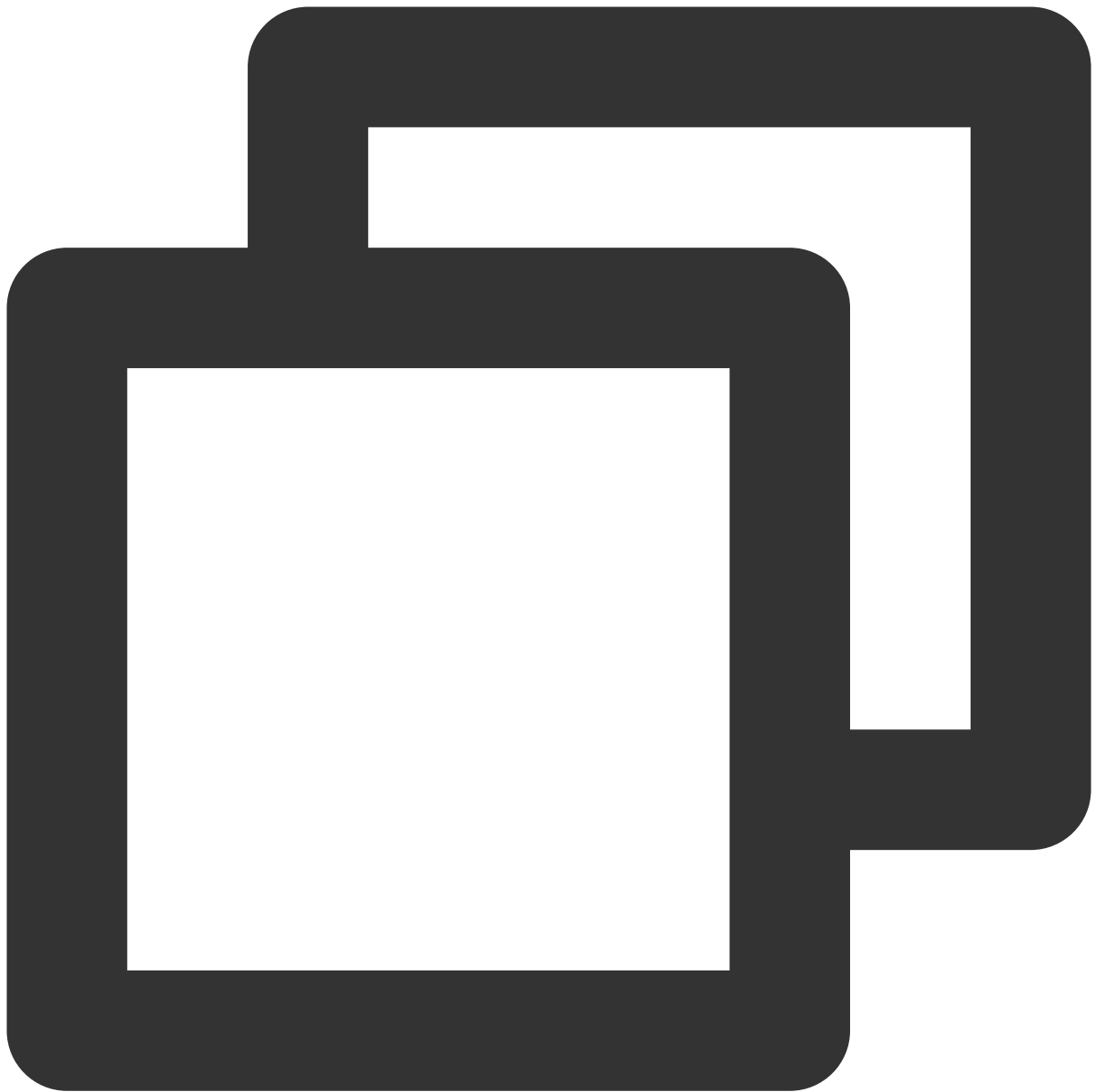
Generate an instance of the global object `TXLivePusher`. All subsequent operations will be performed via the instance.



```
var livePusher = new TXLivePusher();
```

#### 2. Specify the local video player container:

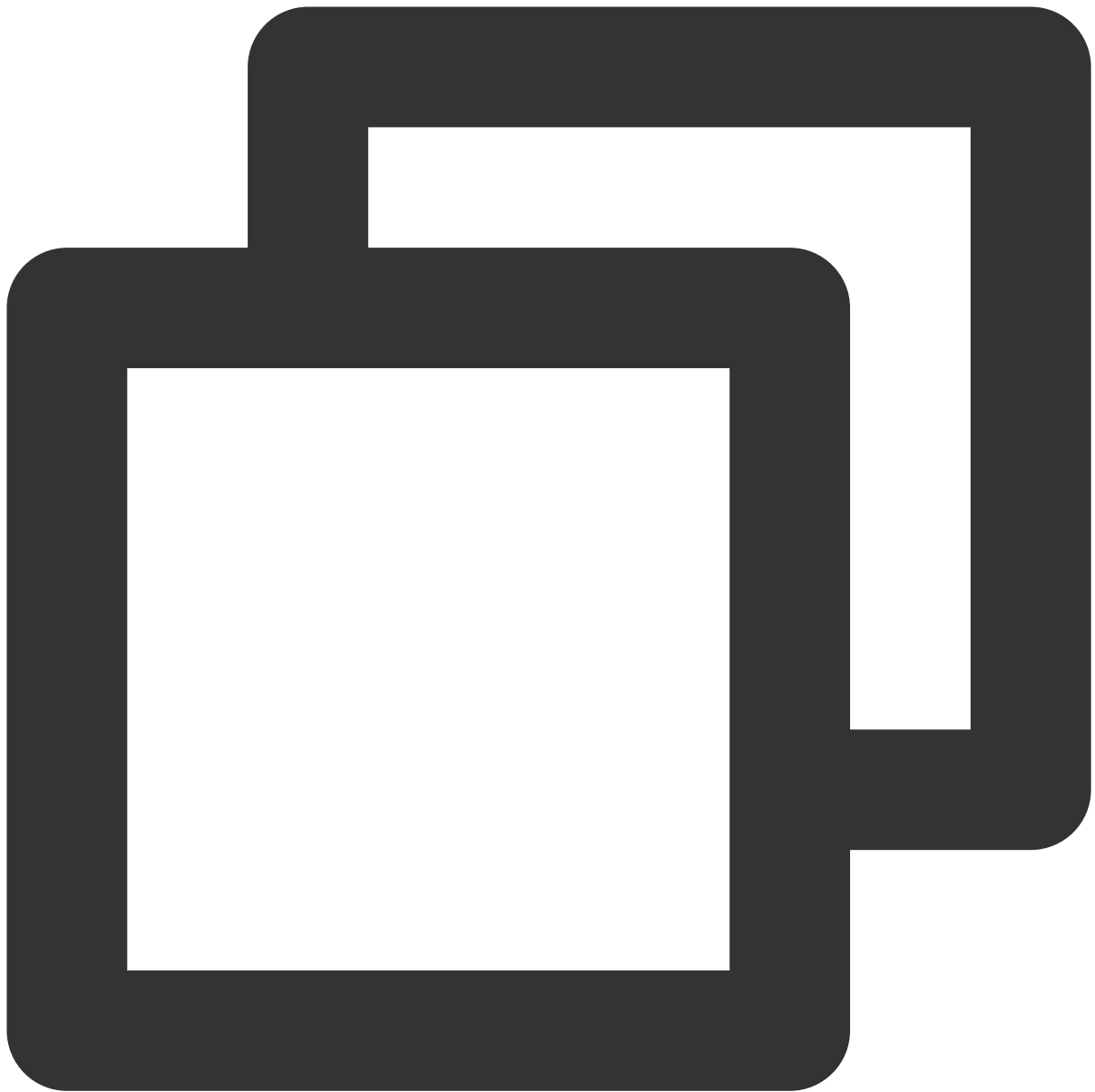
Specify the div for the local video player container, which is where audio and video captured by the browser will be rendered.



```
livePusher.setRenderView('id_local_video');
```

**Note :**

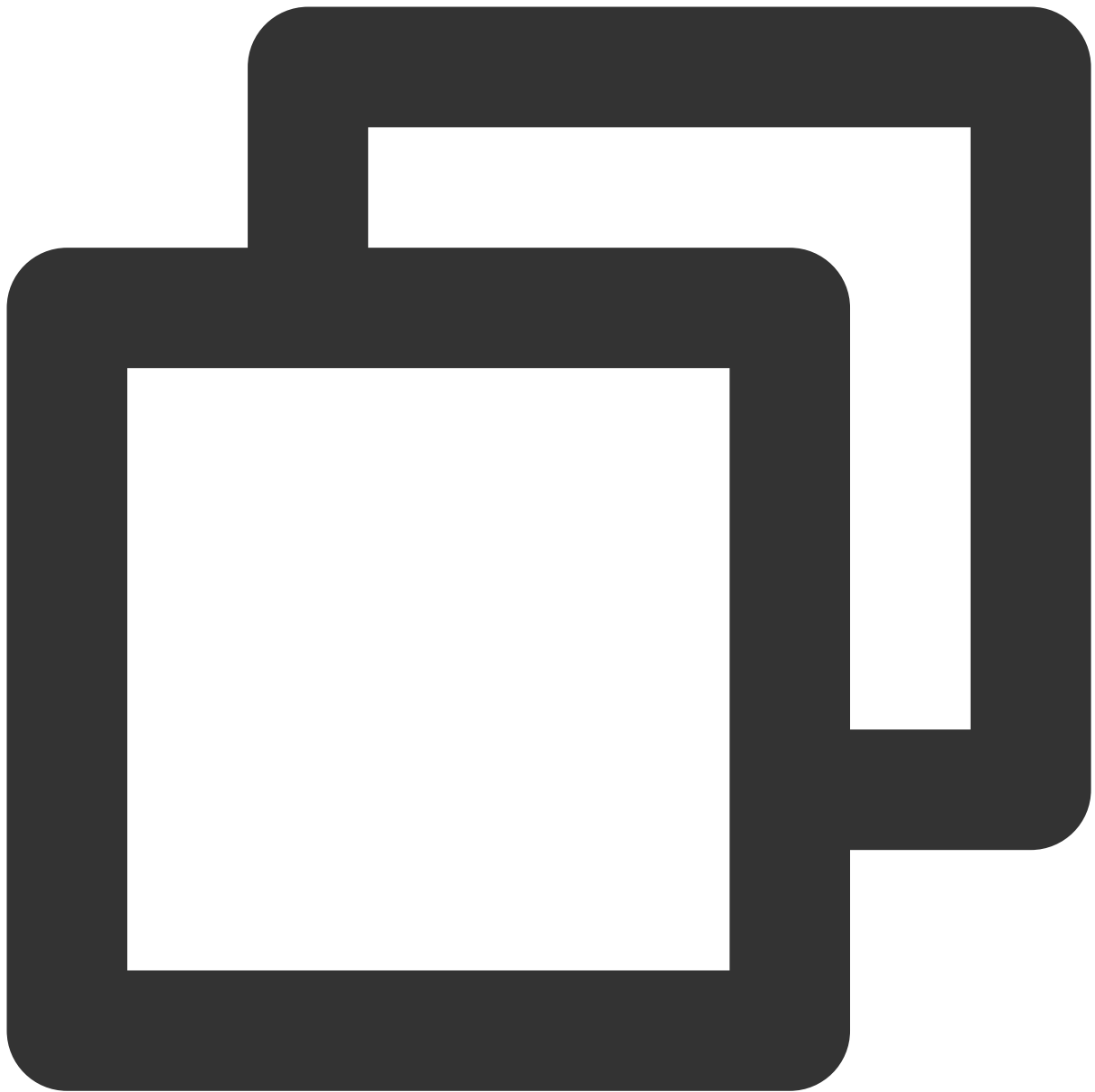
The video element generated via `setRenderView` is unmuted by default. To mute video, obtain the video element using the code below.



```
document.getElementById('id_local_video').getElementsByTagName('video')[0].muted =
```

### 3. Set audio/video quality:

Audio/video quality should be set before capturing. You can specify quality parameters if the default settings do not meet your requirements.

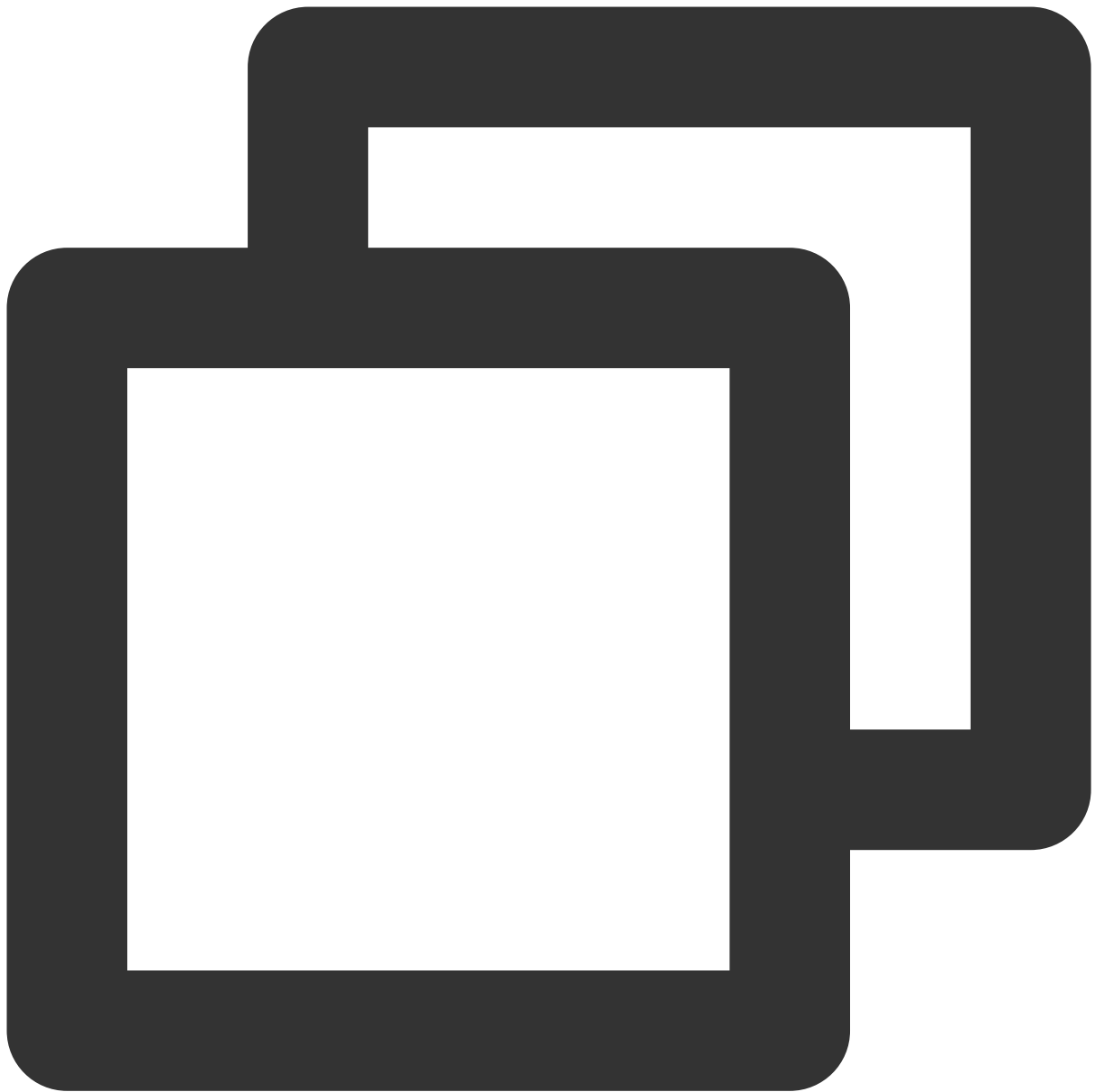


```
// Set video quality
livePusher.setVideoQuality('720p');
// Set audio quality
livePusher.setAudioQuality('standard');
// Set the frame rate
livePusher.setProperty('setVideoFPS', 25);
```

#### 4. Capture streams:

You can capture streams from the camera, mic, screen and local media files. If capturing is successful, the player container will start playing the audio/video captured.

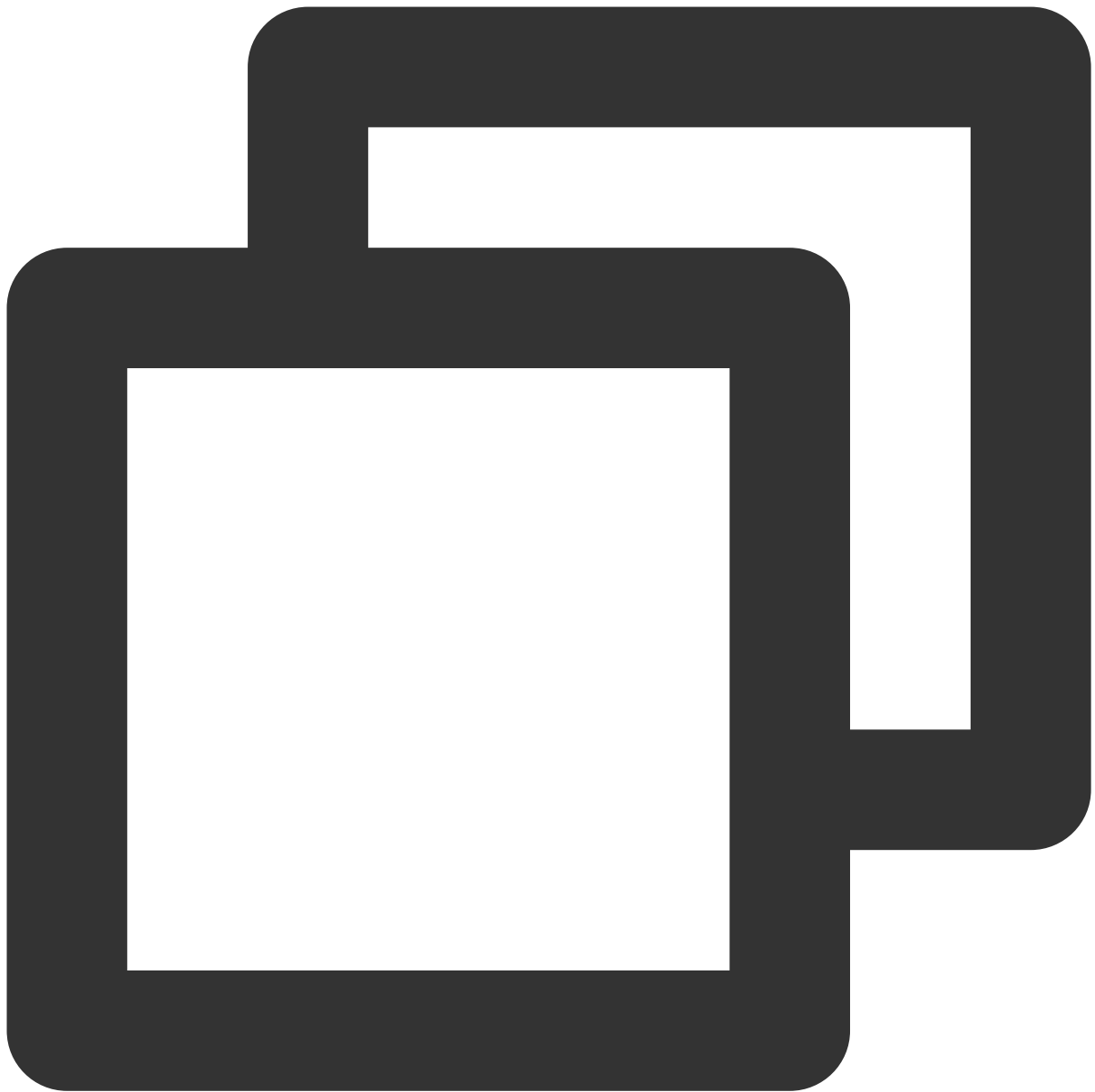




```
// Turn the camera on
livePusher.startCamera();
// Turn the mic on
livePusher.startMicrophone();
```

## 5. Publish streams:

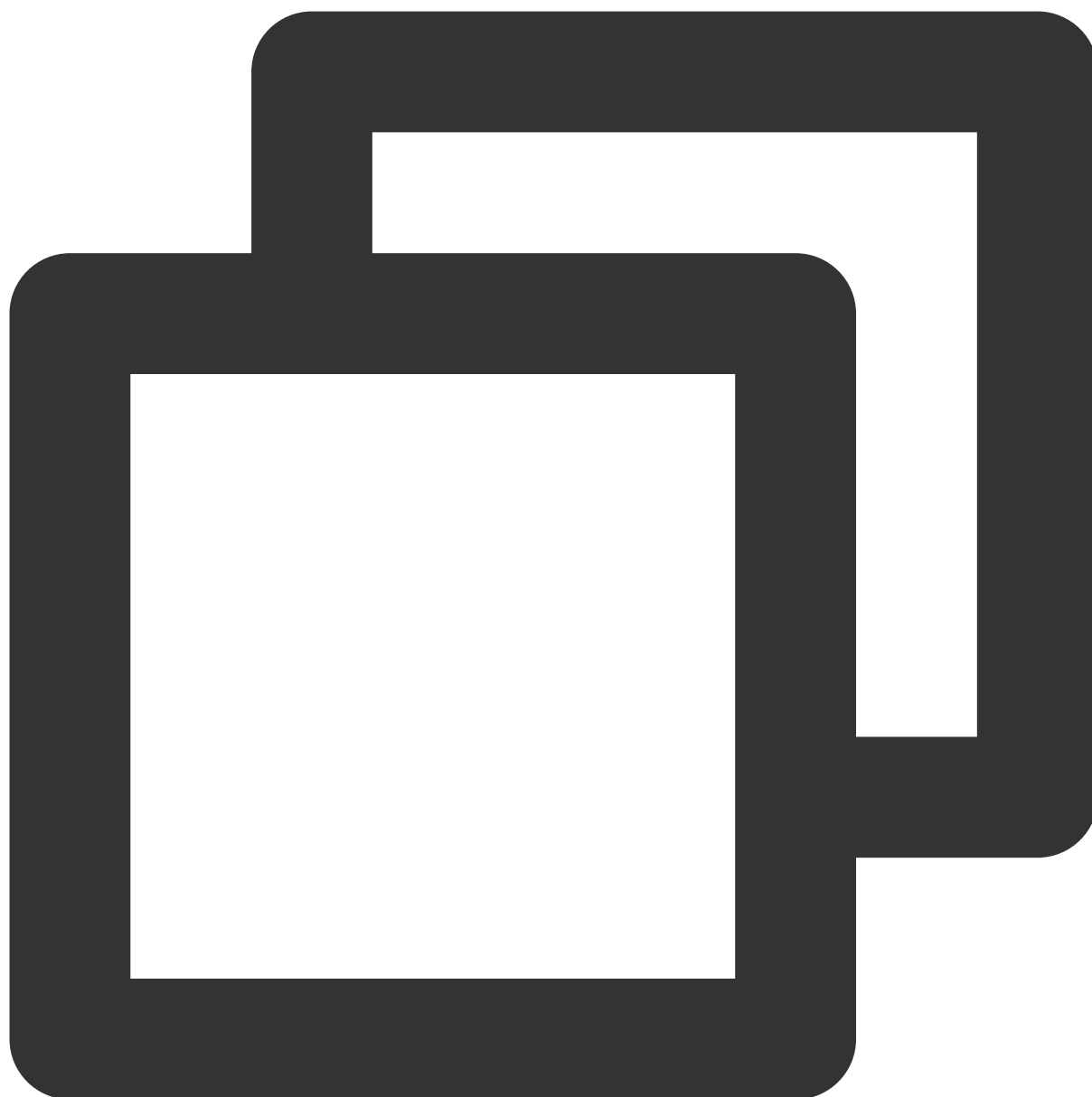
Pass in the LEB publishing URL to start publishing streams. For the format of publishing URLs, please see [Splicing CSS URLs](#). You need to replace the prefix `rtmp://` with `webrtc://`.



```
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
```

**Note :**

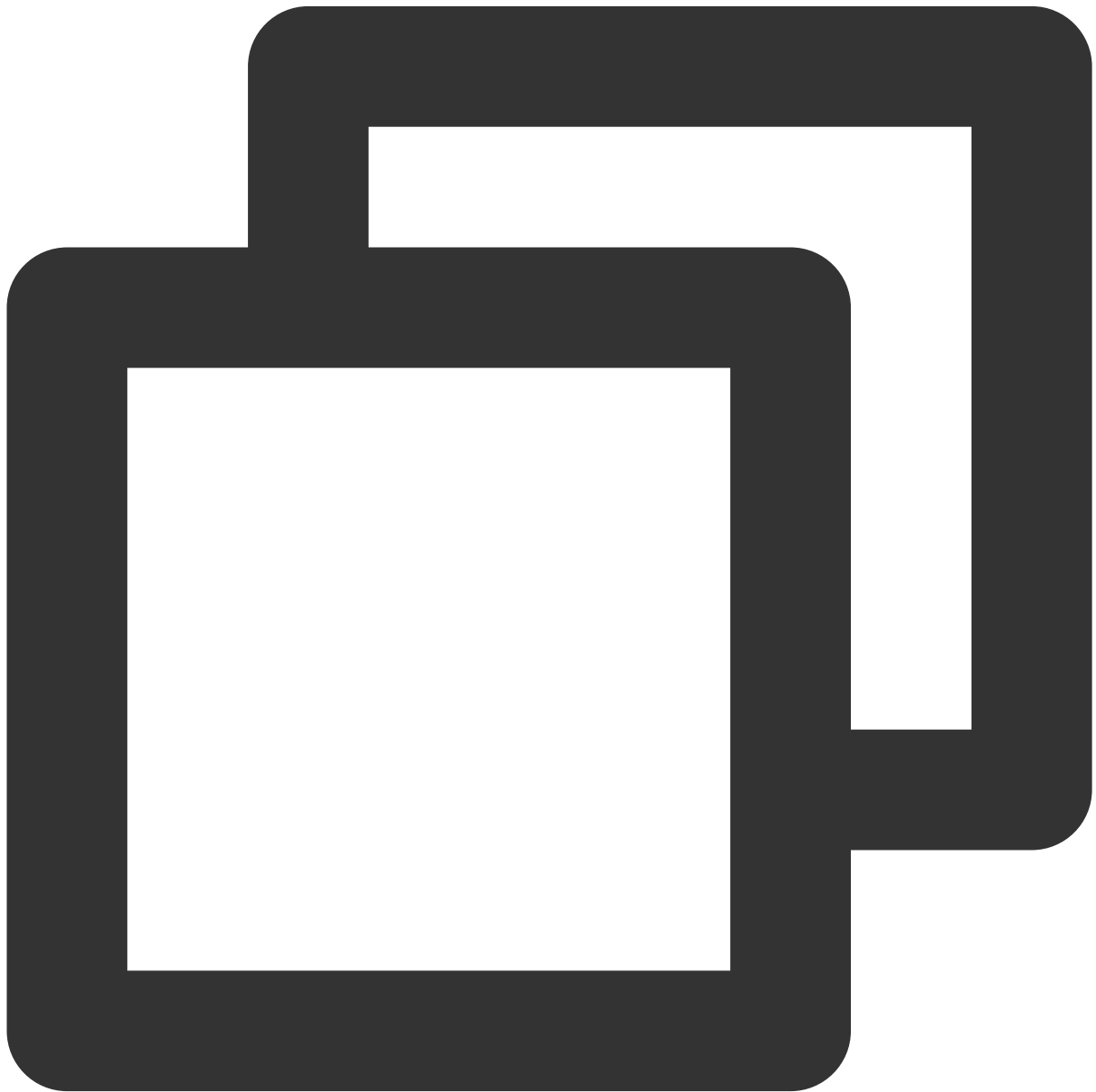
Before publishing, make sure that audio/video streams are captured successfully, or you will fail to call the publishing API. You can use the code below to publish streams automatically after audio/video is captured, that is, after the callback for capturing the first audio or video frame is received. If both audio and video are captured, publishing starts only after both the callback for capturing the first audio frame and that for the first video frame are received.



```
var hasVideo = false;
var hasAudio = false;
var isPush = false;
livePusher.setObserver({
  onCaptureFirstAudioFrame: function() {
    hasAudio = true;
    if (hasVideo && !isPush) {
      isPush = true;
      livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');
    }
  },
},
```

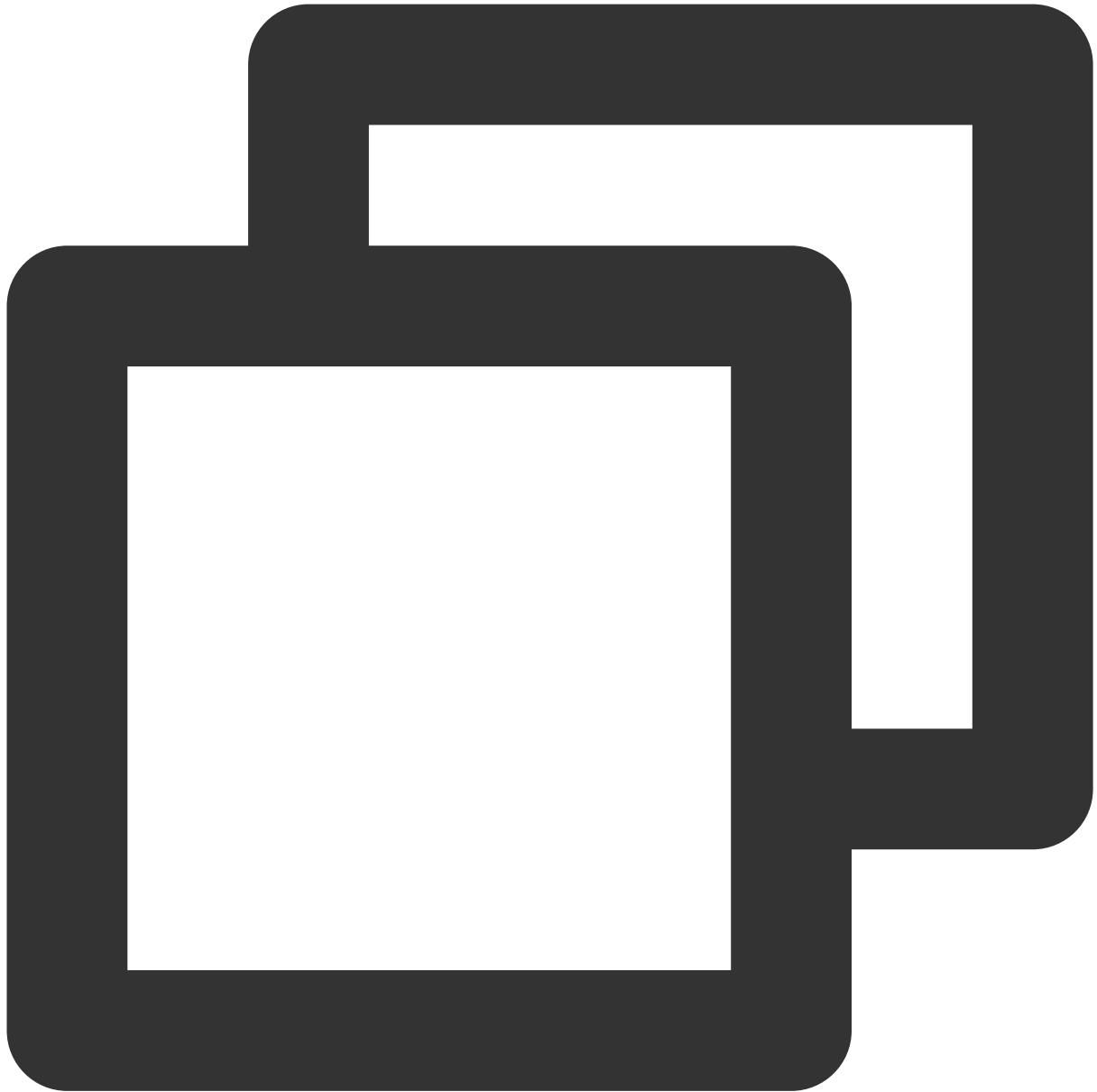
```
onCaptureFirstVideoFrame: function() {  
    hasVideo = true;  
    if (hasAudio && !isPush) {  
isPush = true;  
livePusher.startPush('webrtc://domain/AppName/StreamName?txSecret=xxx&txTime=xxx');  
    }  
}  
});
```

## 6. Stop publishing:



```
livePusher.stopPush();
```

## 7. Stop capturing audio and video:

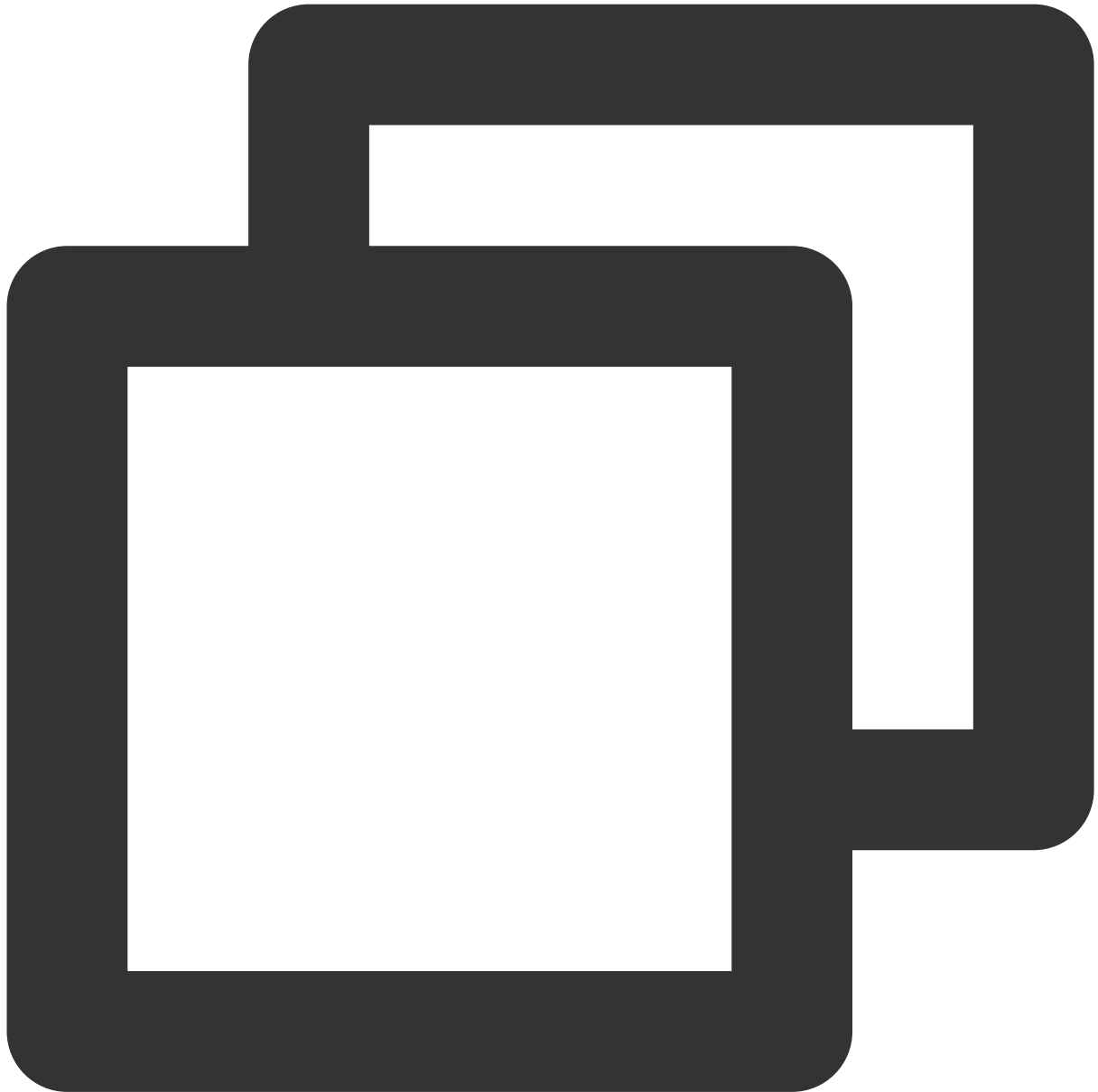


```
// Turn the camera off  
livePusher.stopCamera();  
// Turn the mic off  
livePusher.stopMicrophone();
```

## Advanced Features

## Compatibility

The SDK provides a static method to check whether a browser supports WebRTC.

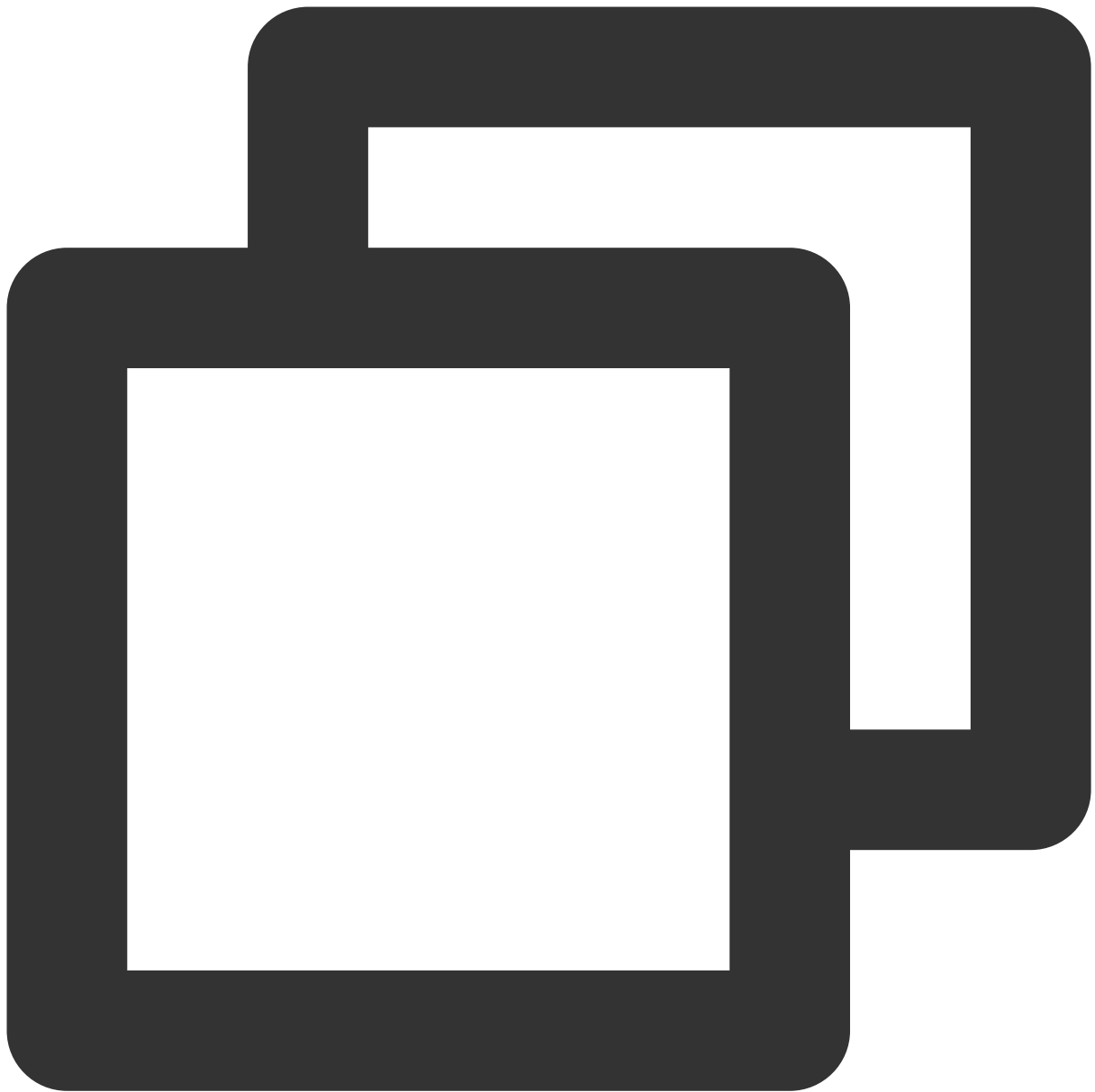


```
TXLivePusher.checkSupport().then(function(data) {  
    // Whether WebRTC is supported  
    if (data.isWebRTCSupported) {  
        console.log('WebRTC Support');  
    } else {  
        console.log('WebRTC Not Support');  
    }  
    // Whether H.264 is supported
```

```
    if (data.isH264EncodeSupported) {  
        console.log('H264 Encode Support');  
    } else {  
        console.log('H264 Encode Not Support');  
    }  
});
```

## Event callbacks

The SDK supports callback event notifications. You can set an observer to receive callbacks of the SDK's status and WebRTC-related statistics. For details, see [TXLivePusherObserver](#).



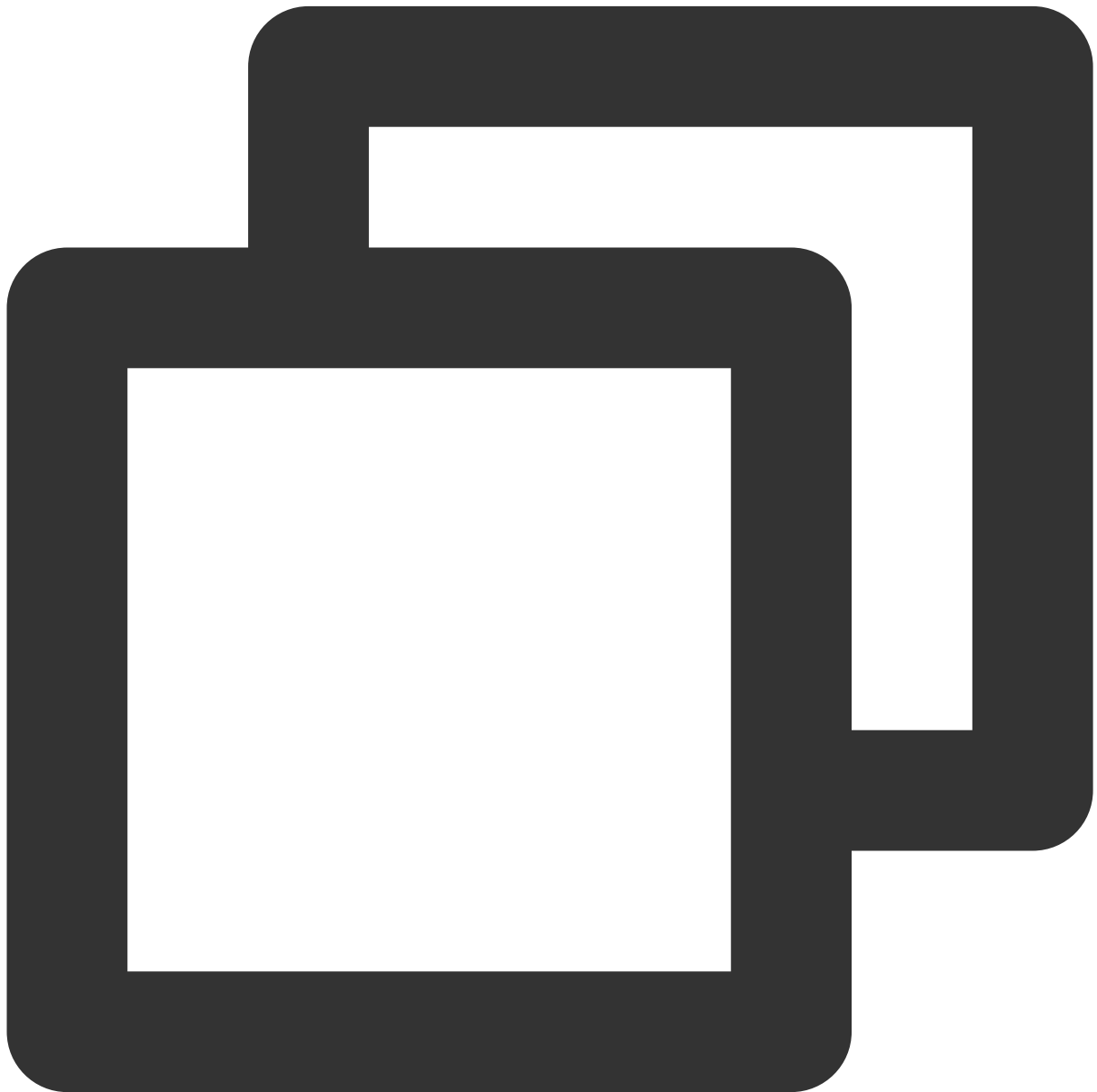
```
livePusher.setObserver({
  // Warnings for publishing
  onWarning: function(code, msg) {
    console.log(code, msg);
  },
  // Publishing status
  onPushStatusUpdate: function(status, msg) {
    console.log(status, msg);
  },
  // Publishing statistics
  onStatisticsUpdate: function(data) {
```



```
        console.log('video fps is ' + data.video.framesPerSecond);  
    }  
});
```

## Device management

You can use a device management instance to get the device list, switch devices, and perform other device-related operations.



```
var deviceManager = livePusher.getDeviceManager();  
// Get the device list  
deviceManager.getDevicesList().then(function(data) {  
    data.forEach(function(device) {  
        console.log(device.deviceId, device.deviceName);  
    });  
});  
// Switch cameras  
deviceManager.switchCamera('camera_device_id');
```

# Flutter

## Publishing from Camera

Last updated : 2024-01-13 15:49:41

### Feature Overview

Publishing from the camera refers to the process of collecting video and audio data from the mobile phone's camera and mic, encoding the data, and pushing it to cloud-based live streaming platforms. [Tencent Cloud live\\_flutter\\_plugin](#) provides the camera push capabilities via the `v2_tx_live_pusher` APIs.

### Notes

**About running projects on x86 emulators:** The SDK uses many audio and video APIs of the iOS system, most of which cannot be used on the x86 emulator built into macOS. Therefore, we recommend that you test your project on a real device.

### Sample Code

Platform	GitHub Address	Key Class
iOS	<a href="#">GitHub</a>	CameraPushViewController.m
Android	<a href="#">GitHub</a>	CameraPushMainActivity.java
Flutter	<a href="#">GitHub</a>	live_camera_push.dart

#### Note:

In addition to the above sample code, regarding frequently asked questions among developers, Tencent Cloud offers an easy-to-understand API example project, which you can use to quickly learn how to use different APIs.

iOS: [MLVB-API-Example](#)

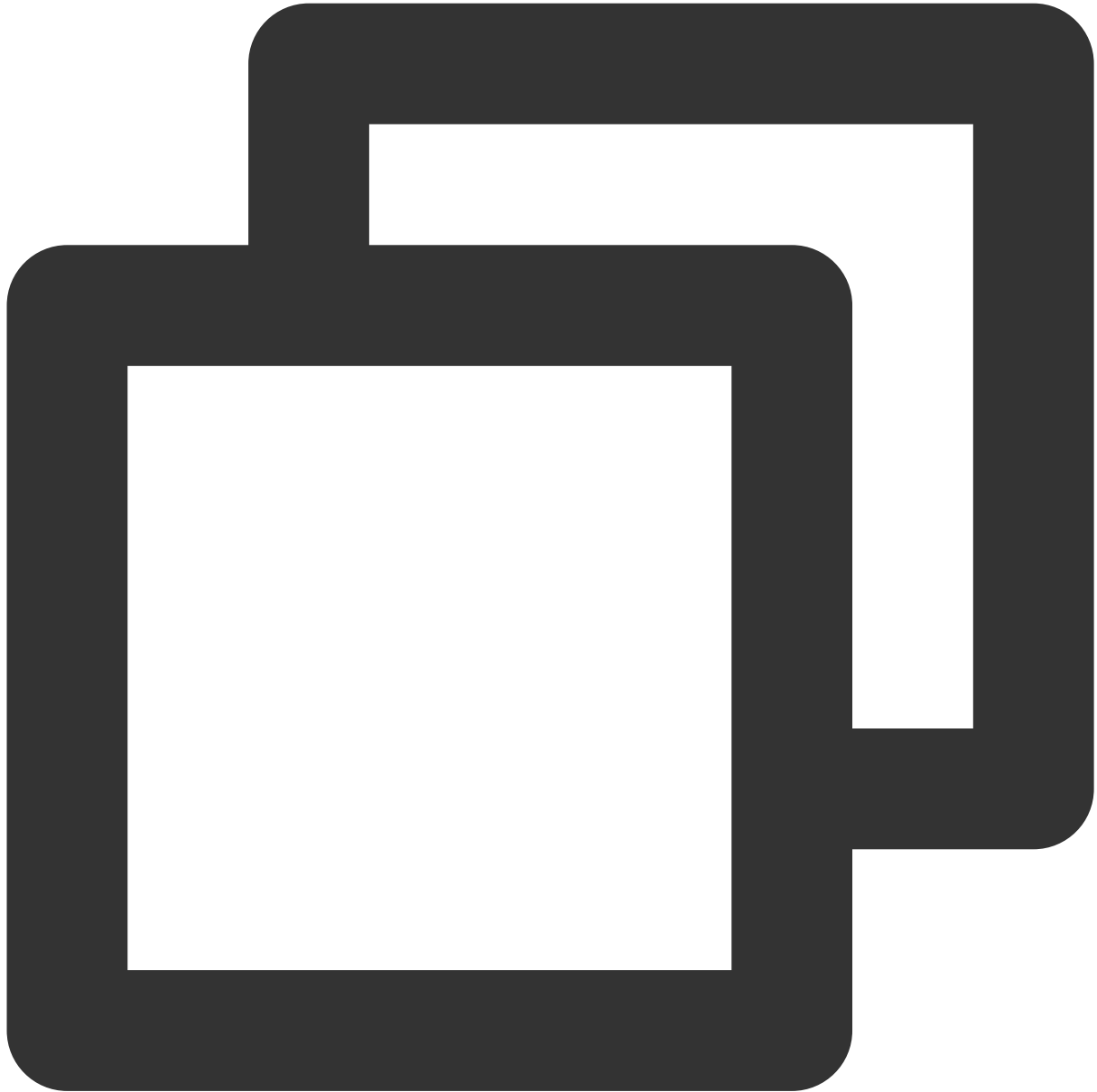
Android: [MLVB-API-Example](#)

Flutter: [Live-API-Example](#)

### Getting Started

## 1. Set dependencies

Integrate `live_flutter_plugin` into your application as instructed in [SDK Integration Guide](#).



```
dependencies:  
  live_flutter_plugin: latest version number
```

## 2. Configure a license for the SDK

1. Get the license:

If you have the required license, get the license URL and key in the [CSS console](#).

**Create Official License**[Price Overview](#)

An official license is valid for a year. Click [Create] to purchase one. Please make sure that the bundle ID and package name entered are correct as the information cannot be modified after submission.

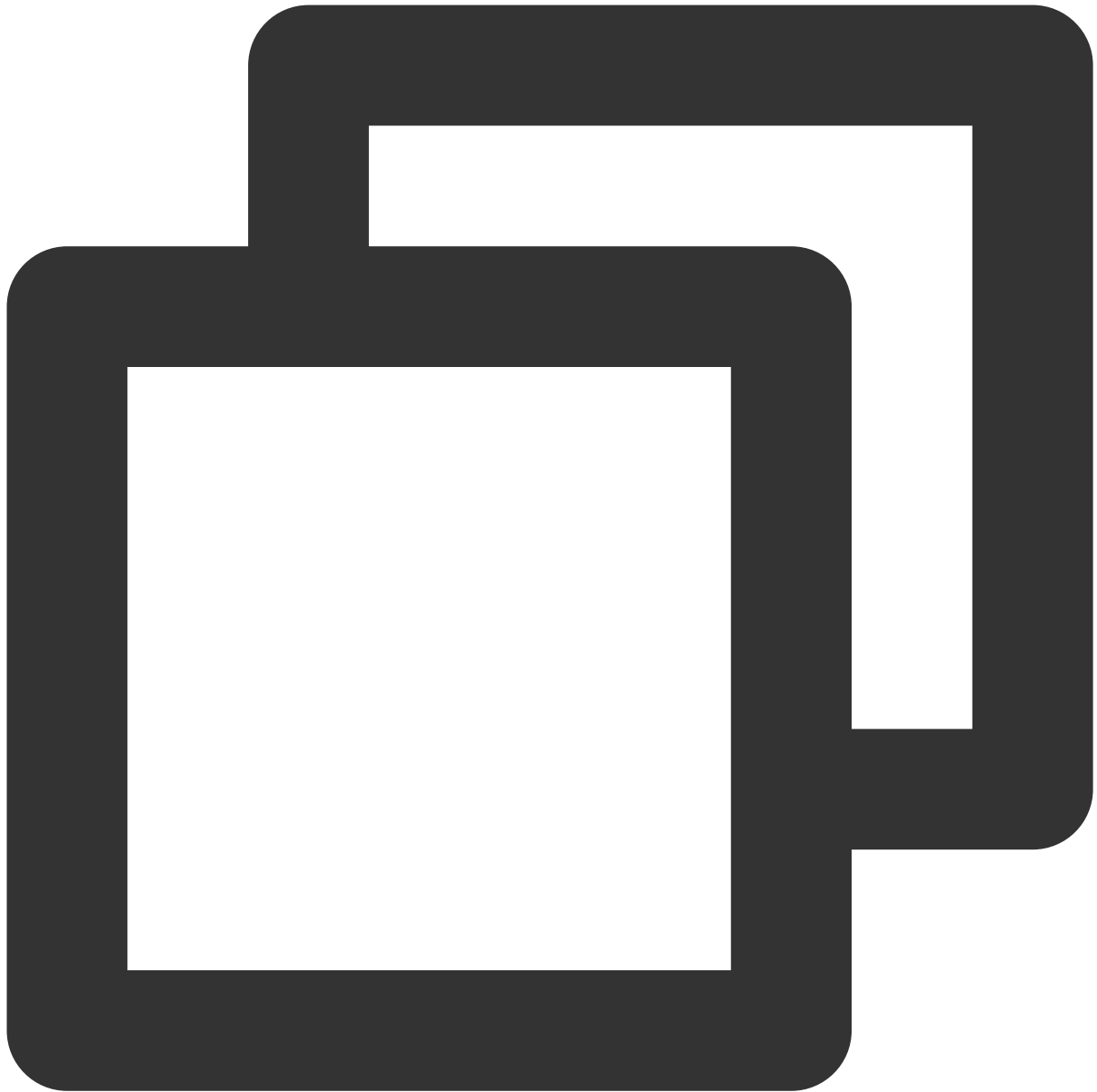
**▼ Official License** LiveApplication Name Package Name com Bundle Id con Key LicenseUrl .licence

Start Date 2021-12-03

End Date 2022-12-03

If you don't have the required license, apply for a license as instructed in [New License and Renewal](#).

2. Before your application calls features of `live_flutter_plugin`, complete the following configuration:



```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

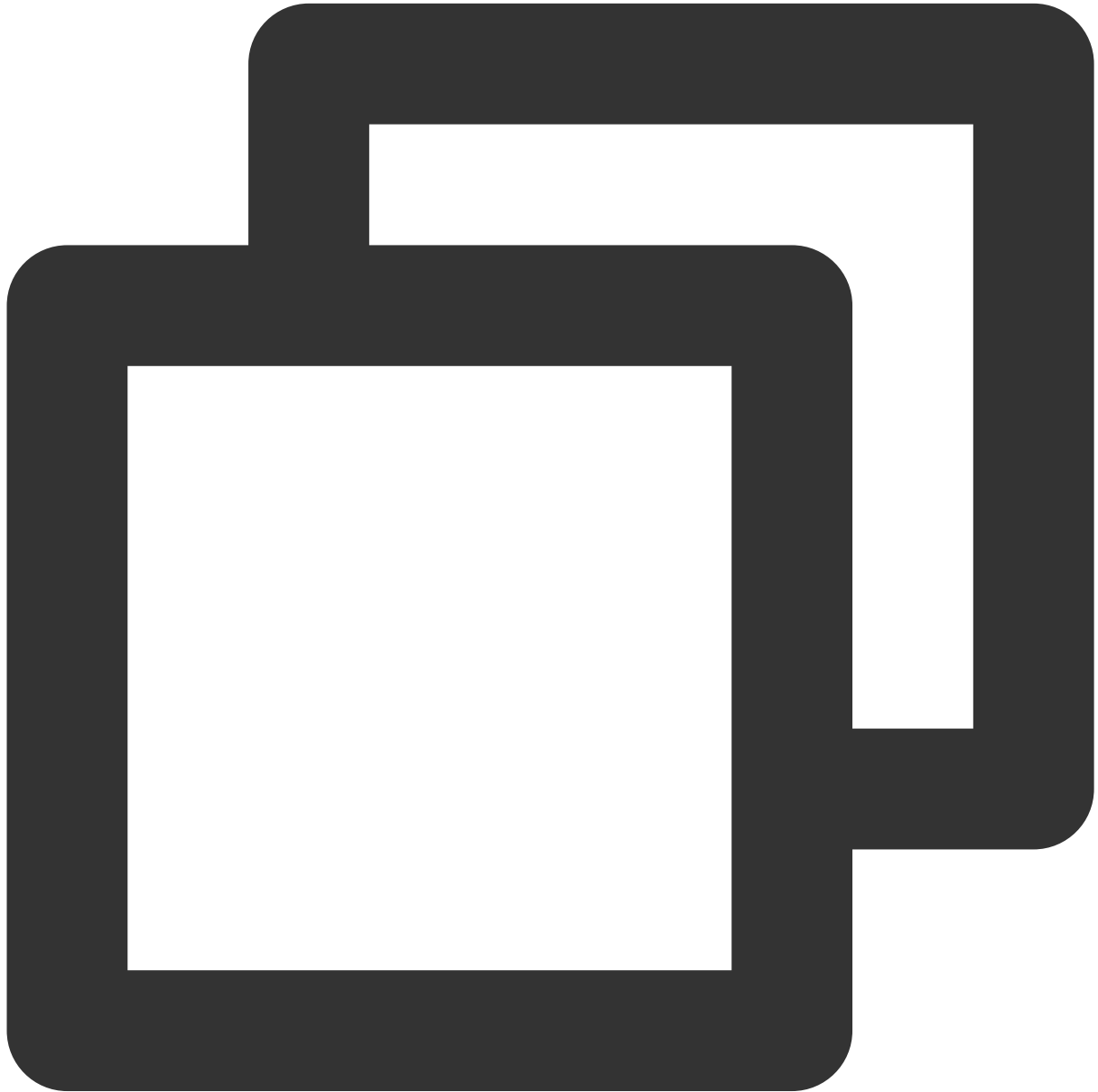
/// Tencent Cloud license management page (https://consoleintl.cloud.tencent.com/)
setupLicense() {
  // The license URL of the current application
  var LICENSEURL = "";
  // The license key of the current application
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

**Note:**

The `packageName/BundleId` configured in the license must be the same as that of the application; otherwise, stream push will fail.

**3. Initialize the `V2TXLivePusher` component**

Create a `V2TXLivePusher` object and specify `V2TXLiveMode` .

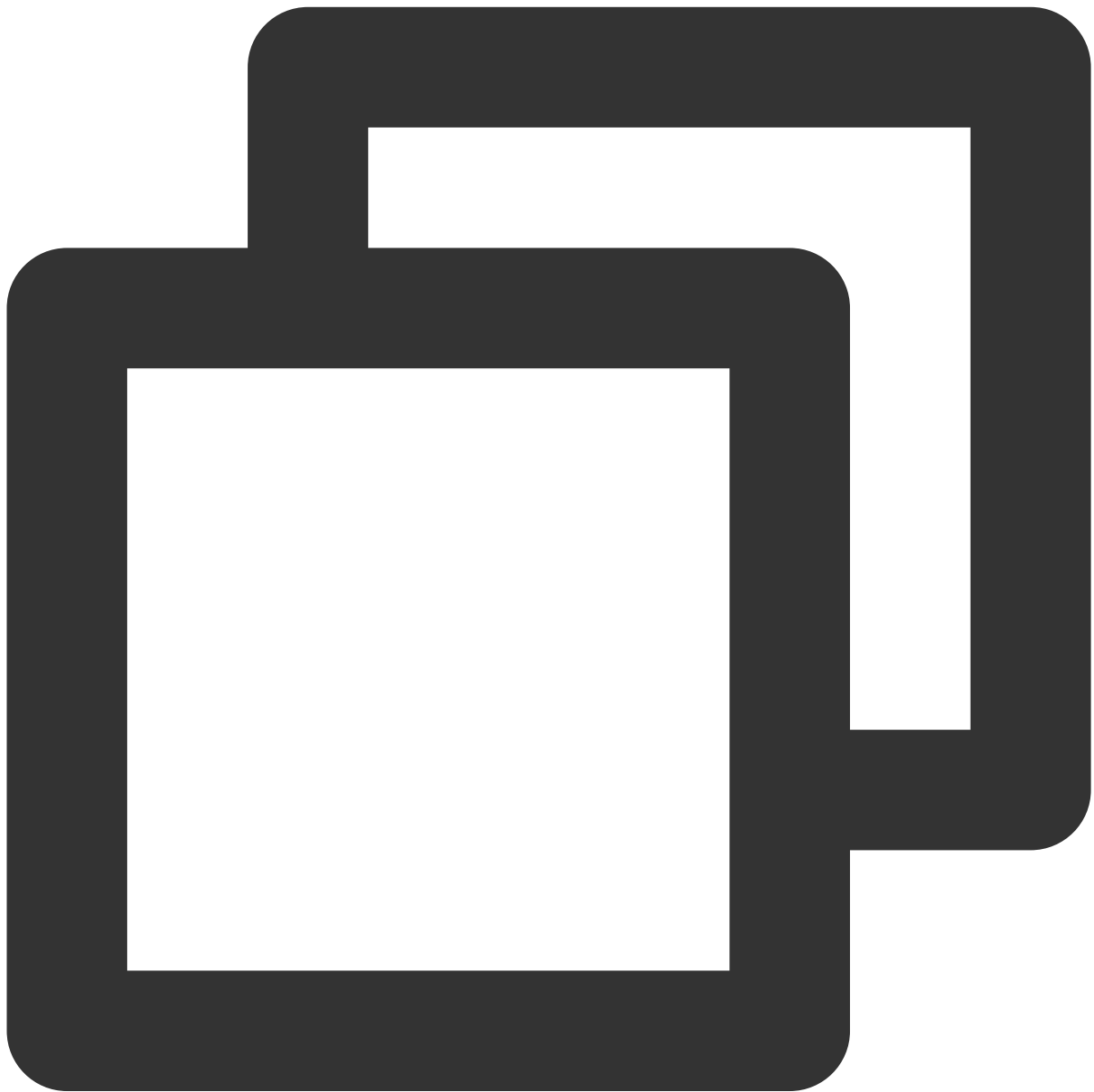


```
import 'package:live_flutter_plugin/v2_tx_live_pusher.dart';

/// Initialize `V2TXLivePusher`
```

```
initPusher() {  
  _livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTC);  
}
```

#### 4. Set the video rendering view



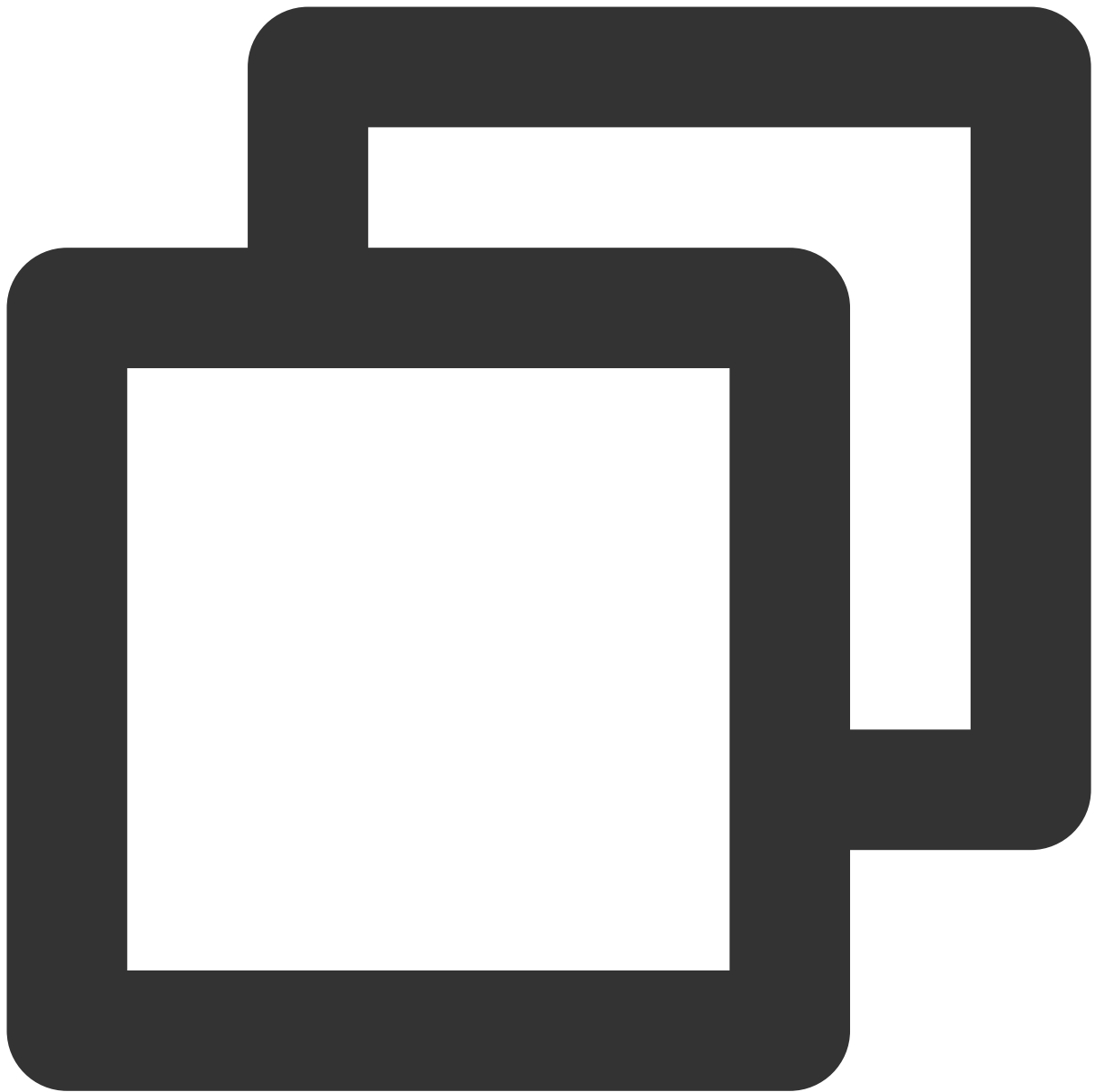
```
import 'package:live_flutter_plugin/widget/v2_tx_live_video_widget.dart';  
  
/// The video rendering view widget  
Widget renderView() {
```



```
return V2TXLiveVideoWidget(  
    onViewCreated: (viewId) async {  
        /// Set the video rendering view  
        _livePusher.setRenderViewID(_renderViewId);  
        /// Enable camera preview  
        _livePusher.startCamera(true);  
    },  
);  
}
```

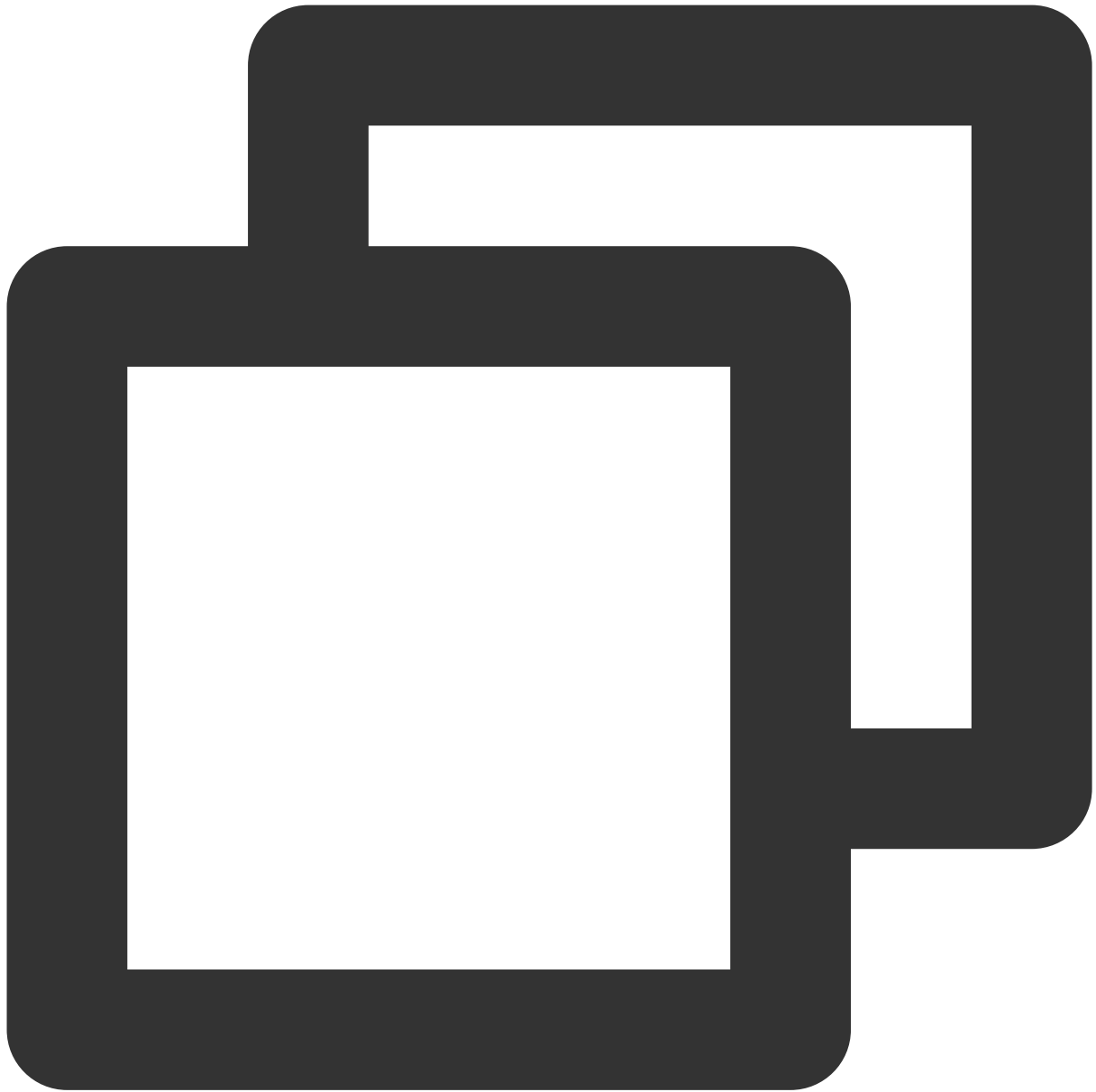
## 5. Start and stop publishing

After calling `startCamera` to enable camera preview, you can call the [startPush](#) API in `V2TXLivePusher` to start publishing. You can use [TRTC's URL](#) or an [RTMP URL](#) for publishing. The former uses UDP. It offers better streaming quality and supports co-anchoring.



```
/// Start stream push
startPush() async {
  // Generate a stream push address of RTMP/TRTC
  var url = "";
  // Start stream push
  await _livePusher.startPush(url);
  // Turn the mic on
  await _livePusher.startMicrophone();
}
```

After stream push ends, you can call the [stopPush](#) API in `V2TXLivePusher` to stop stream push.



```
/// Stop stream push
stopPush() async {
  // Turn the camera off
  await _livePusher.stopCamera();
  // Turn the mic off
  await _livePusher.stopMicrophone();
  // Stop stream push
  await _livePusher.stopPush();
}
```

**Note:**

If you have enabled camera preview, please disable it when you stop publishing streams.

**How do I get a valid stream push URL?**

Activate CSS. In the **CSS console**, go to **Auxiliary Tools** > **Address Generator** to generate a stream push URL.

For more information, see [Publishing/Playback URL](#).

The screenshot shows the 'Address Generator' interface. It includes the following fields and controls:

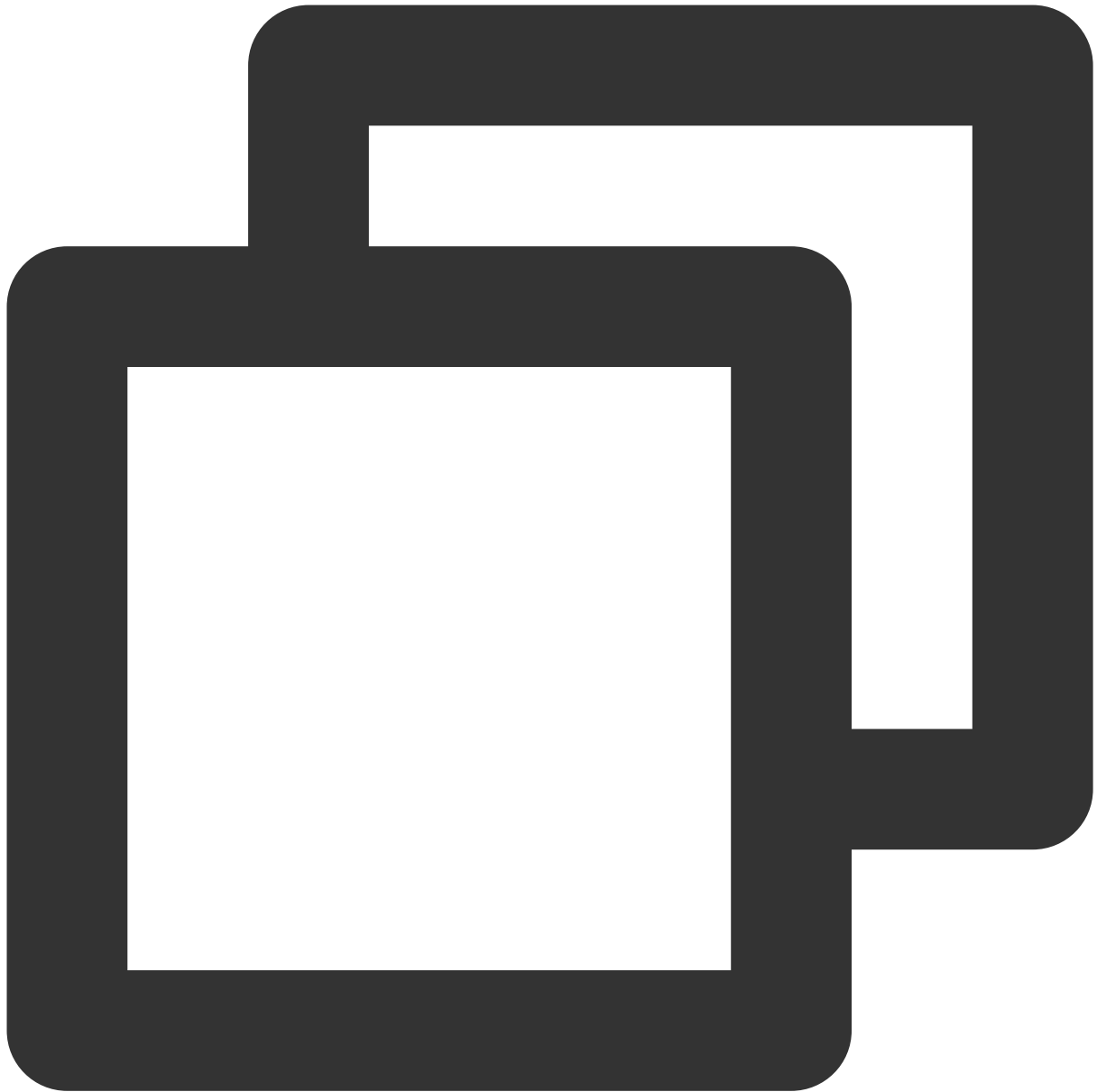
- Domain Type**: A dropdown menu set to 'Push Domain'. Below it, a note states: 'If you select push domain, a push address will be generated; and if you select playback domain, a playback address will be generated. If there is no available domain, the system will automatically select a domain for you.'
- AppName**: A text input field containing 'live'. Below it, a note states: 'Use "live" by default. Only letters, digits, and symbols are supported.'
- StreamName**: A text input field containing 'liveteststream'. Below it, a note states: 'Only support letters, digits, and symbols.'
- Expiration Time**: A dropdown menu set to 'UTC+8' and a date/time picker showing '2022-09-16 21:43:52'. Below it, a note states: 'The expiration time of playback address is the setting timestamp plus the playback authentication expiration time, and the push address expiration time.'

At the bottom, there are two buttons: 'Generate Address' (a blue button) and 'Address Resolution Sample' (a blue link).

**Why is `V2TXLIVE_ERROR_INVALID_LICENSE` returned?** If the `startPush` API returns `V2TXLIVE_ERROR_INVALID_LICENSE`, it means your license verification failed. Please check your configuration against [Step 2. Configure a license for the SDK](#).

## 6. Publish audio-only streams

If your live streaming scenarios involve audio only, you can skip [Step 4](#) or do not call `startCamera` before `startPush`.



```
/// Start stream push
startPush() async {
  // Initialize `V2TXLivePusher`
  _livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTC);
  // Generate a stream push address of RTMP/TRTC
  var url = "";
  // Start stream push
  await _livePusher.startPush(url);
  // Turn the mic on
  await _livePusher.startMicrophone();
}
```

**Note:**

If you publish audio-only streams but no streams can be pulled from an RTMP, FLV, or HLS playback URL, there is a problem with your line configuration. Please [submit a ticket](#) for help.

## 7. Set video quality

You can call the [setVideoQuality](#) API in `V2TXLivePusher` to set image definition on the viewer end. The video image watched by the host is the source video without encoding or compression and is not subject to settings. However, viewers can perceive the encoding quality of the video encoder set in `setVideoQuality`. For more information, see [Setting Video Quality](#).

## 8. Set the beauty filter style and skin brightening and rosy skin effects

You can call the [getBeautyManager](#) API in `V2TXLivePusher` to get the `TXBeautyManager` instance so as to further set the beauty filter effect.

### Beauty filter style

The SDK has three built-in skin smoothing algorithms, each of which corresponds to a beauty filter style. You can select the one most suitable for your product needs. For more information, see the [TXBeautyManager.h](#) file.

Beauty Filter Style	Description
<code>TXBeautyStyleSmooth</code>	The smooth style, which features more obvious skin smoothing effects and is suitable for live showrooms.
<code>TXBeautyStyleNature</code>	The natural style, which retains more facial details and is more natural.
<code>TXBeautyStylePitu</code>	The Pitu style, which uses the beauty filter algorithm developed by YouTu Lab. Its effect combines the smooth style and the natural style, that is, it retains more skin details than the smooth style and delivers more obvious skin smoothing effects than the natural style.

You can call the [setBeautyStyle](#) API in `TXBeautyManager` to set the beauty filter style.

Item	Configuration	Description
Beauty filter strength	Via the <code>setBeautyLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Skin brightening filter strength	Via the <code>setWhitenessLevel</code> API in <code>TXBeautyManager</code>	Value range: 0-9. `0` means the filter is disabled. The greater the value, the more obvious the effect.
Rosy skin filter	Via the <code>setRuddyLevel</code> API in	Value range: 0-9. `0` means the filter is

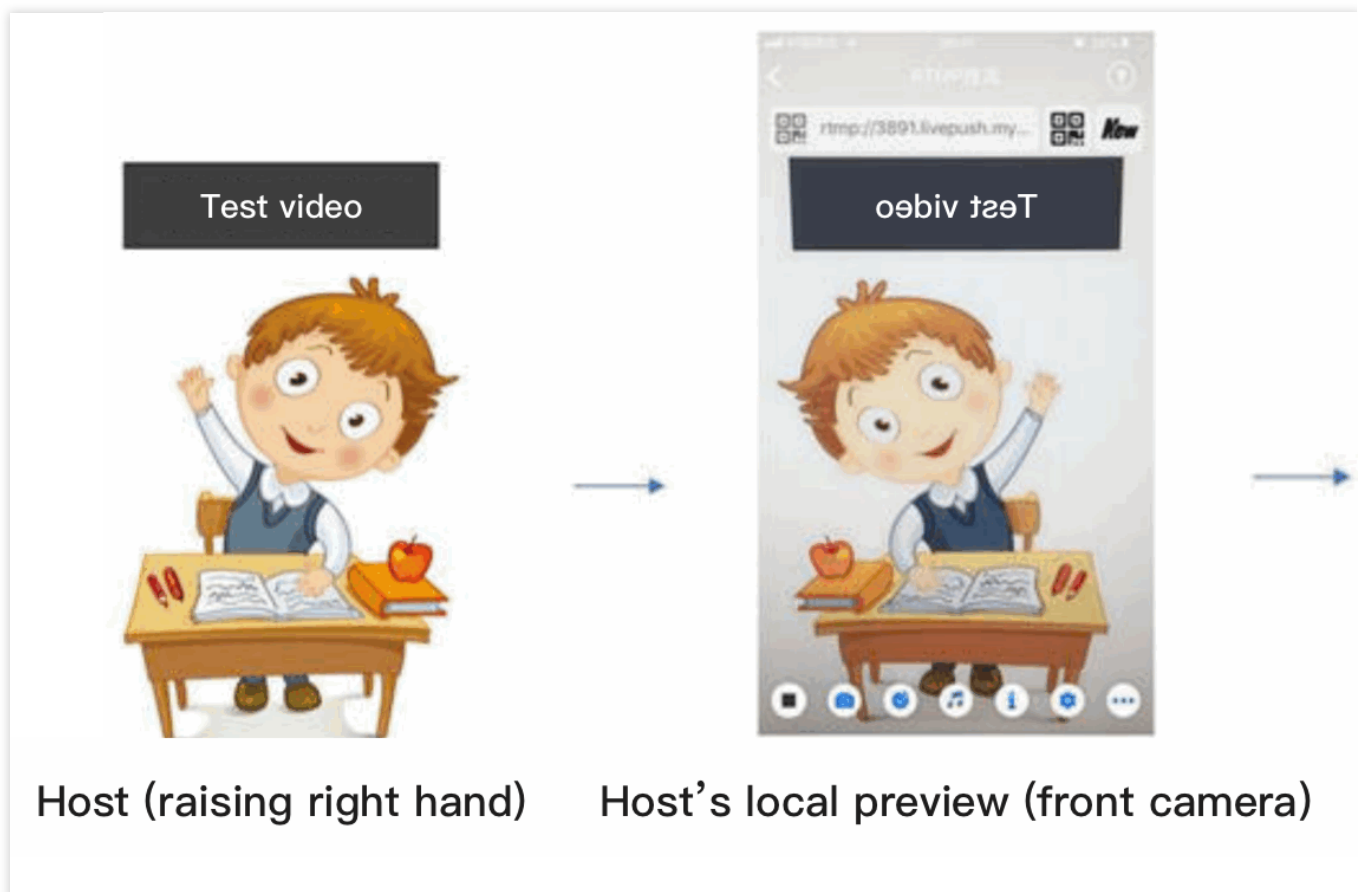
strength	`TXBeautyManager`	disabled. The greater the value, the more obvious the effect.
----------	-------------------	---

## 9. Manage devices

`V2TXLivePusher` offers a set of APIs for device control. You can use `getDeviceManager` to get the `TXDeviceManager` instance for device management. For detailed directions, see [TXDeviceManager API](#).

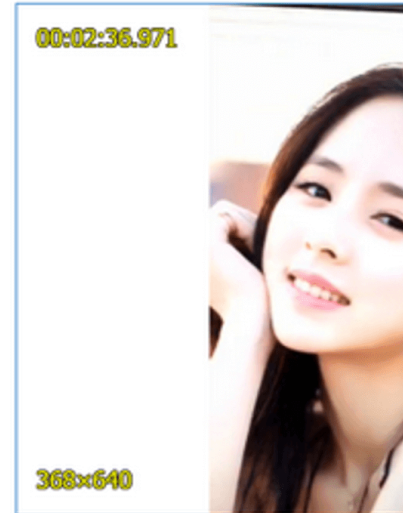
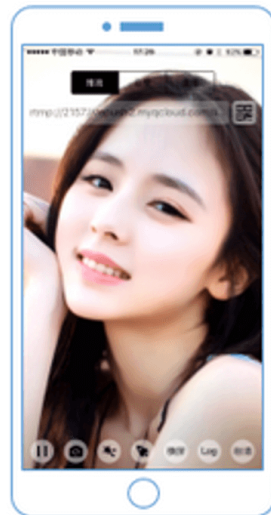
## 10. Set the mirror effect on the audience side

You can call `setRenderMirror` of `V2TXLivePusher` to change the camera mirroring mode, so as to change the mirroring effect of the video image seen by viewers. If the host uses the front camera for live streaming, the image will be reversed by the SDK by default.



## 11. Publish streams in landscape mode

In most cases, hosts stream while holding their phones vertically, and audience watch videos in portrait resolutions (e.g., 540 × 960). However, there are also cases where hosts hold phones horizontally, and ideally, audience should watch videos in landscape resolutions (960 × 540).



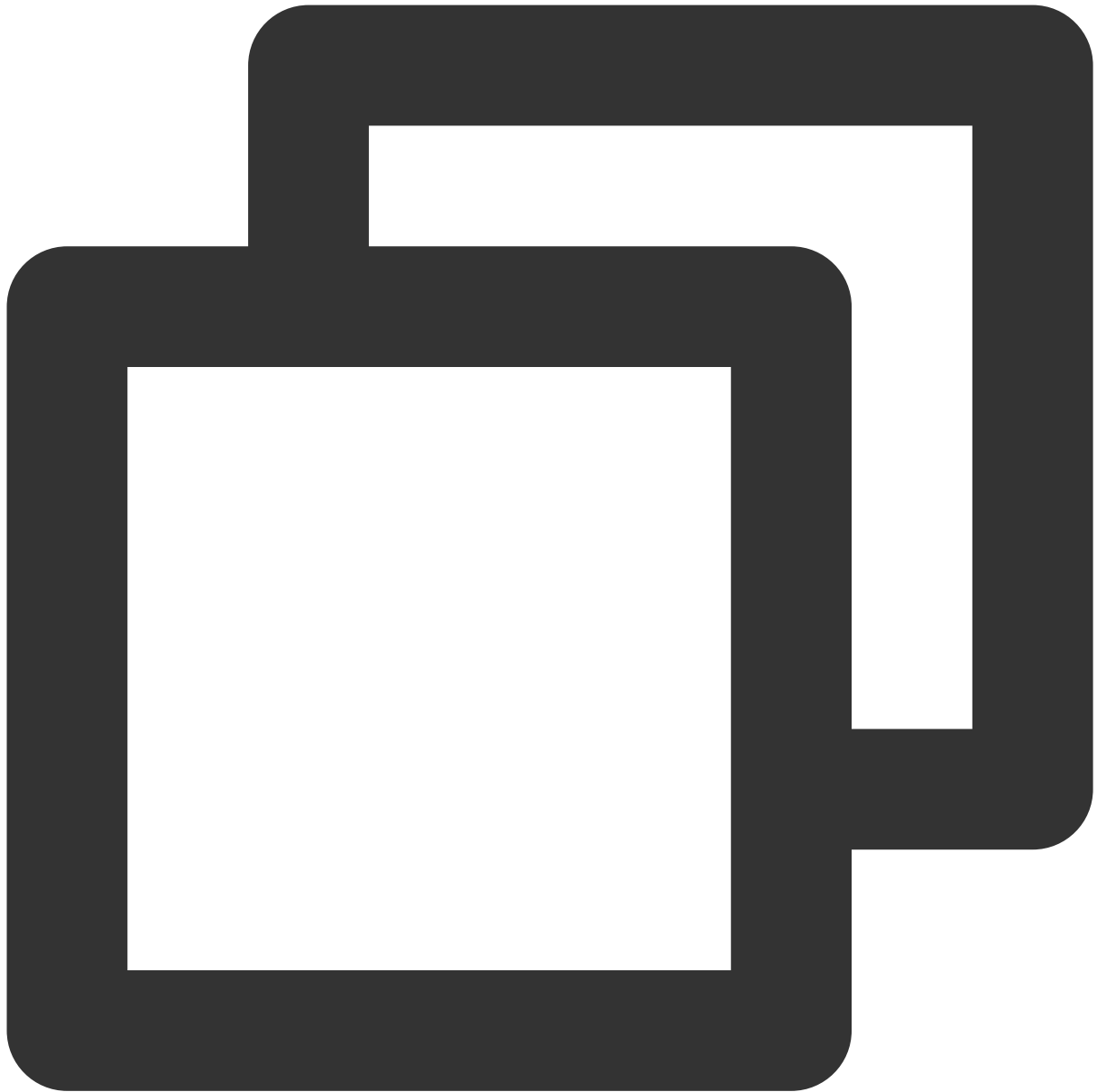
Anchor (vertical publishing streaming)



Anchor (horizontal publishing streaming)

`V2TXLivePusher` pushes video images in portrait mode by default. To push video images in landscape mode, you can modify the parameters in the `setVideoQuality` API to set the image orientation on viewers' devices.





```
// Video encoding parameters
var param = V2TXLiveVideoEncoderParam();
param.videoResolutionMode = isLandscape ? V2TXLiveVideoResolutionMode.v2TXLiveVideo
_livePusher.setVideoQuality(param);
```

## 12. Set audio effects

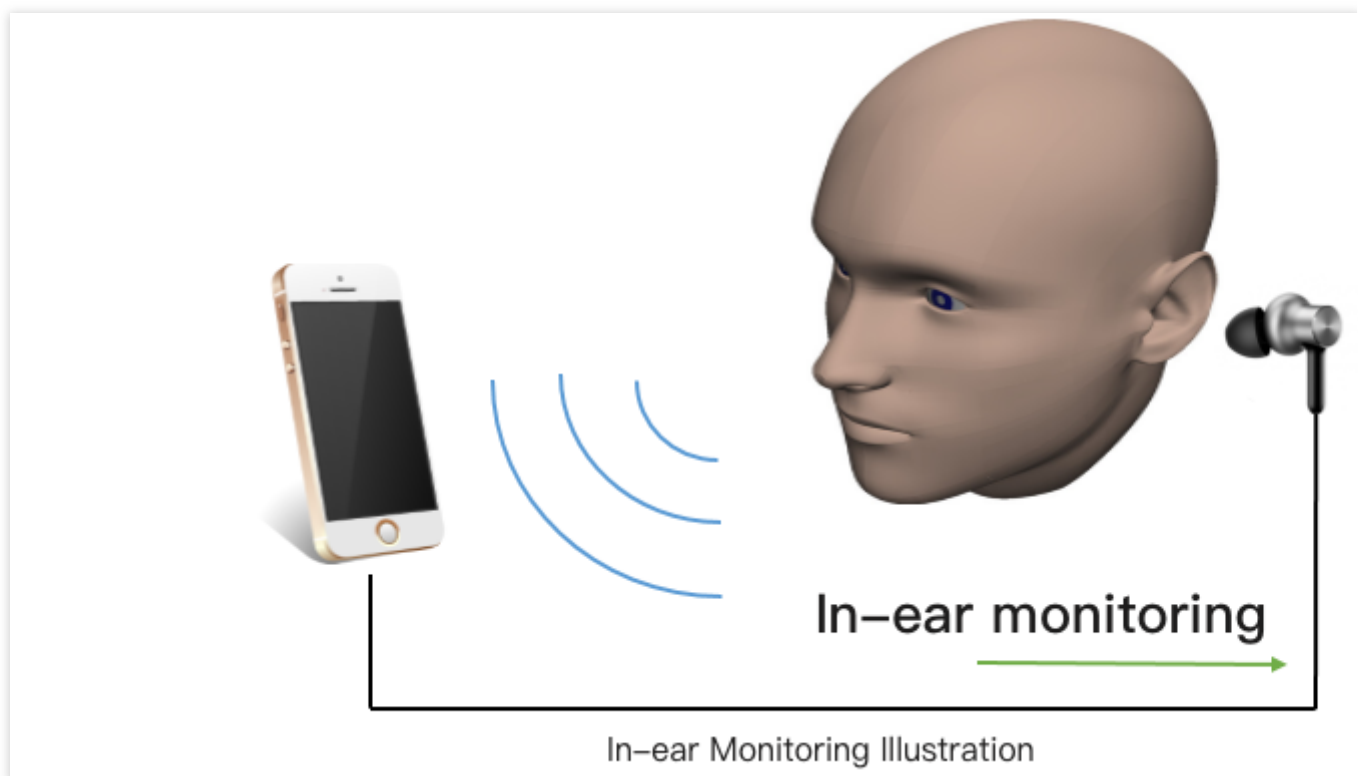
Call [getAudioEffectManager](#) in `V2TXLivePusher` to get a `TXAudioEffectManager` instance, which can be used to mix background music and set in-ear monitoring, reverb, and other audio effects. Background music mixing

means mixing into the published stream the music played by the host's phone so that the audience can also hear the music.

Call the `enableVoiceEarMonitor` API in `TXAudioEffectManager` to enable in-ear monitoring, which allows hosts to hear their vocals in earphones when they sing.

Call the `setVoiceReverbType` API in `TXAudioEffectManager` to add reverb effects such as karaoke, hall, husky, and metal. The effects are applied to the videos watched by the audience.

Call the `setVoiceChangerType` API in `TXAudioEffectManager` to add voice changing effects such as little girl and middle-aged man to enrich host-audience interaction. The effects are applied to the videos watched by the audience.

**Note:**

For detailed directions, see [TXAudioEffectManager API](#).

## Event Handling

### Listening for events

The SDK listens on push events and errors via the [V2TXLivePusherObserver](#) delegate. See [v2\\_tx\\_live\\_code library](#) for a detailed list of events and error codes.

### Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common unclassified error occurred.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Latency during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the

		user is having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system.

# Publishing from Screen

Last updated : 2024-01-13 15:49:41

## Overview

Publishing from the camera refers to the process of collecting video and audio data from the mobile phone's camera and mic, encoding the data, and pushing it to cloud-based live streaming platforms. Tencent Cloud

`live_flutter_plugin` provides the camera push capabilities via `V2TXLivePusher` APIs.

## Sample Code

Tencent Cloud offers an easy-to-understand API example project to help you quickly learn how to use different APIs.

Platform	GitHub Address
iOS	<a href="#">GitHub</a>
Android	<a href="#">GitHub</a>
Flutter	<a href="#">GitHub</a>

## Environment Requirements

### Android

Screen recording is supported in Android 5.0 and above.

Floating windows need to be enabled manually on some mobile phones and systems.

### iOS

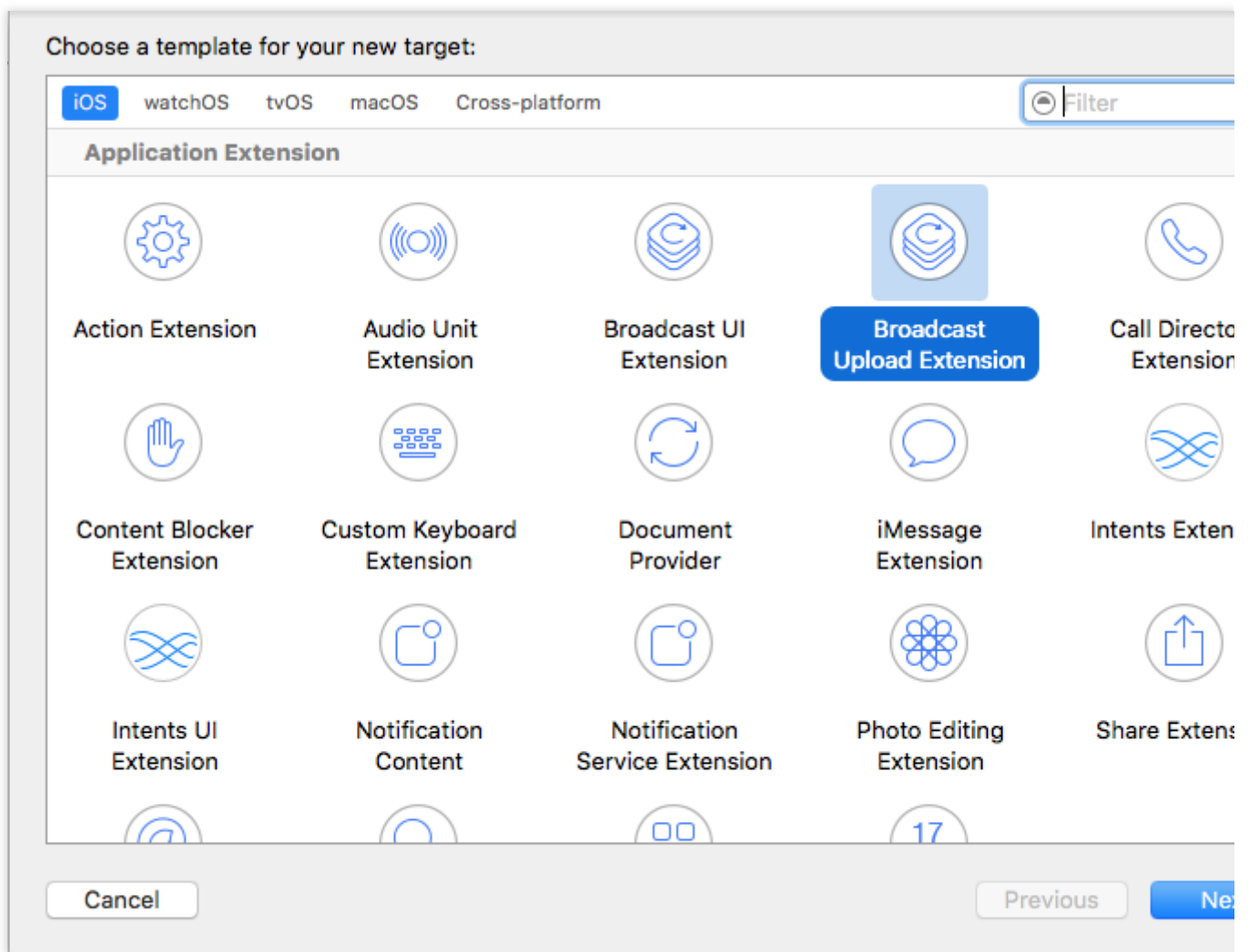
Screen recording is a new feature in iOS 10. In addition to using ReplayKit to record video from the screen, which is possible in iOS 9, with iOS 10, users can also stream live video from the screen. For details, see [Go Live with ReplayKit](#). In iOS 11, Apple made ReplayKit more usable and more universally applicable and launched [ReplayKit2](#), going from supporting ReplayKit alone to allowing the recording of the entire screen. Therefore, we recommend using ReplayKit2 in iOS 11 to enable the screen sharing feature. Screen sharing relies on extensions, which operate as independent processes. However, to ensure system smoothness, iOS allocates limited resources to extensions and may kill extensions with high memory usage. Given this, Tencent Cloud has further reduced the memory usage of LiteAVSDK while retaining its high streaming quality and low latency to ensure the stability of extensions.

**Note:**

This document describes how to use ReplayKit 2 on iOS 11 to push streams from the screen. The parts about the use of the SDK also apply to other custom stream push scenarios. For more information, see the code sample in the `Live Demo Screen` folder of the [demo](#).

**1. Create the live streaming extension**

Open your project with Xcode and select **New > Target... > Broadcast Upload Extension**, as shown below.



Enter a product name and click **Finish**. A new directory with the product name entered will appear in your project.

Under the directory, there is an automatically generated `SampleHandler` class, which is responsible for screen recording operations.

**Note:**

Xcode 9 or later is required, and your iPhone must be updated to iOS 11 or later. Screen recording is not supported on emulators.

**2. Import `TXLiteAVSDK_ReplayKitExt.framework`**

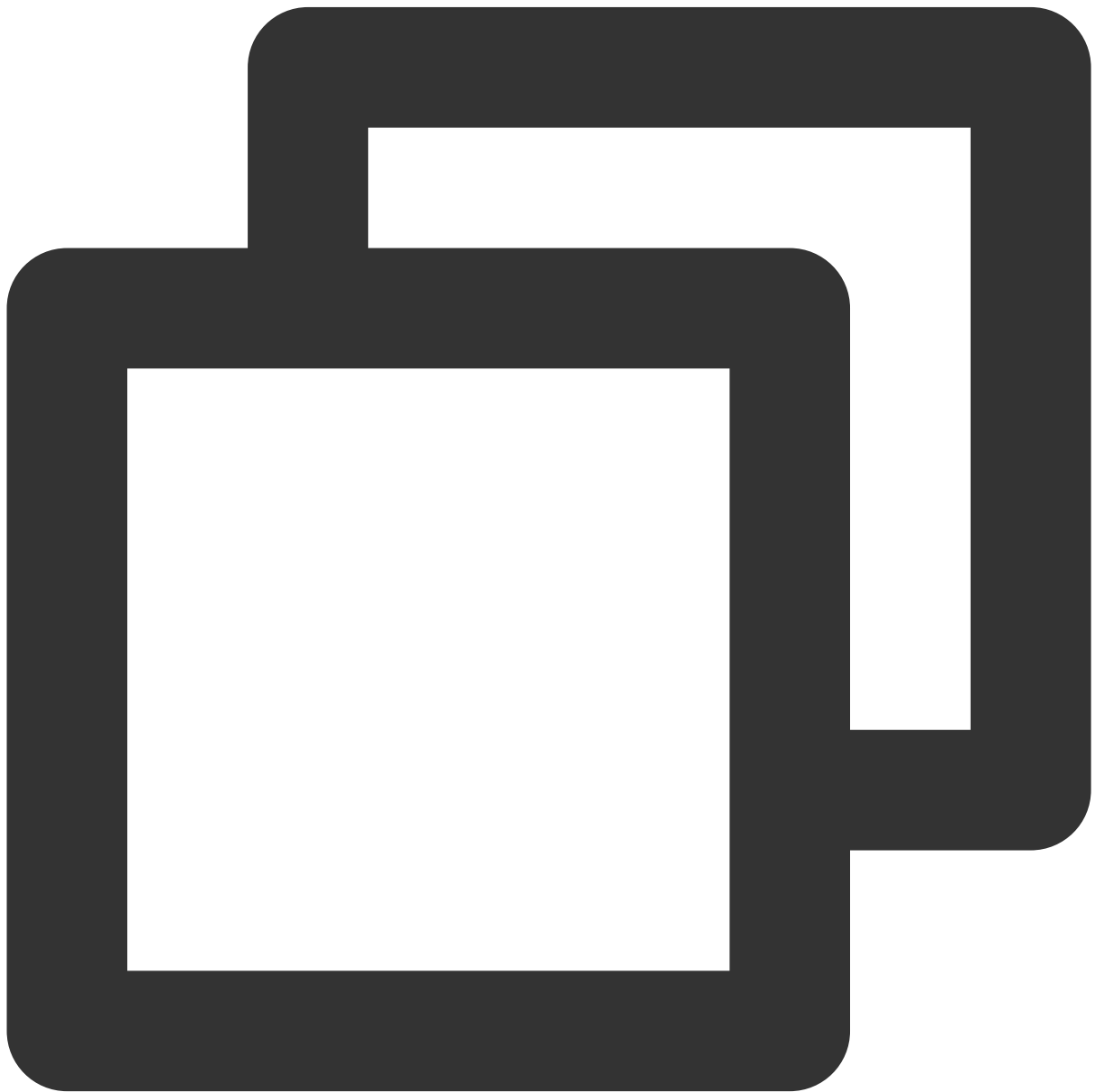
Import `TXLiteAVSDK_ReplayKitExt.framework` into the live streaming extension the same way you import a

framework into the host application. The system libraries the SDK depends on are also the same. For more information, see [iOS](#).

## Getting Started

### 1. Set dependencies

Integrate `live_flutter_plugin` into your application as instructed in [SDK Integration Guide](#).



```
dependencies:
```

```
live_flutter_plugin: latest version number
```

## 2. Configure a license for the SDK

### 1. Get the license:

If you have the required license, get the license URL and key in the [CSS console](#).

**Create Official License**[Price Overview](#)

An official license is valid for a year. Click [Create] to purchase one. Please make sure that the bundle ID and package name entered are correct as the information cannot be modified after submission.

▼ Official License

Live

Application Name

Package Name

Bundle Id

Key

LicenseUrl

Start Date

End Date

com

com

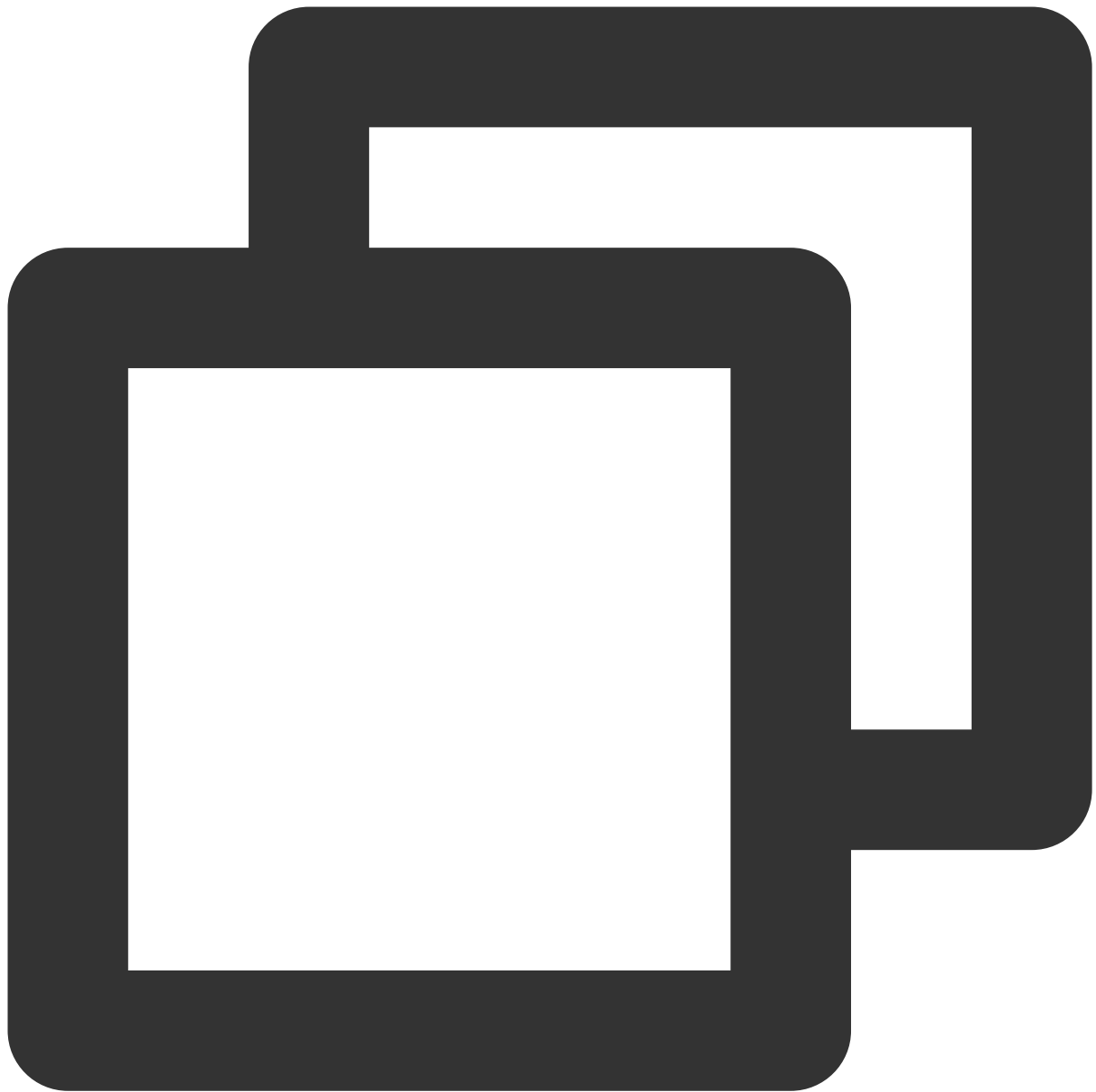
2021-12-03

2022-12-03

If you don't have the required license, apply for a license as instructed in [New License and Renewal](#).

### 2. Before your application calls features of `live_flutter_plugin`, complete the following configuration:





```
import 'package:live_flutter_plugin/v2_tx_live_premier.dart';

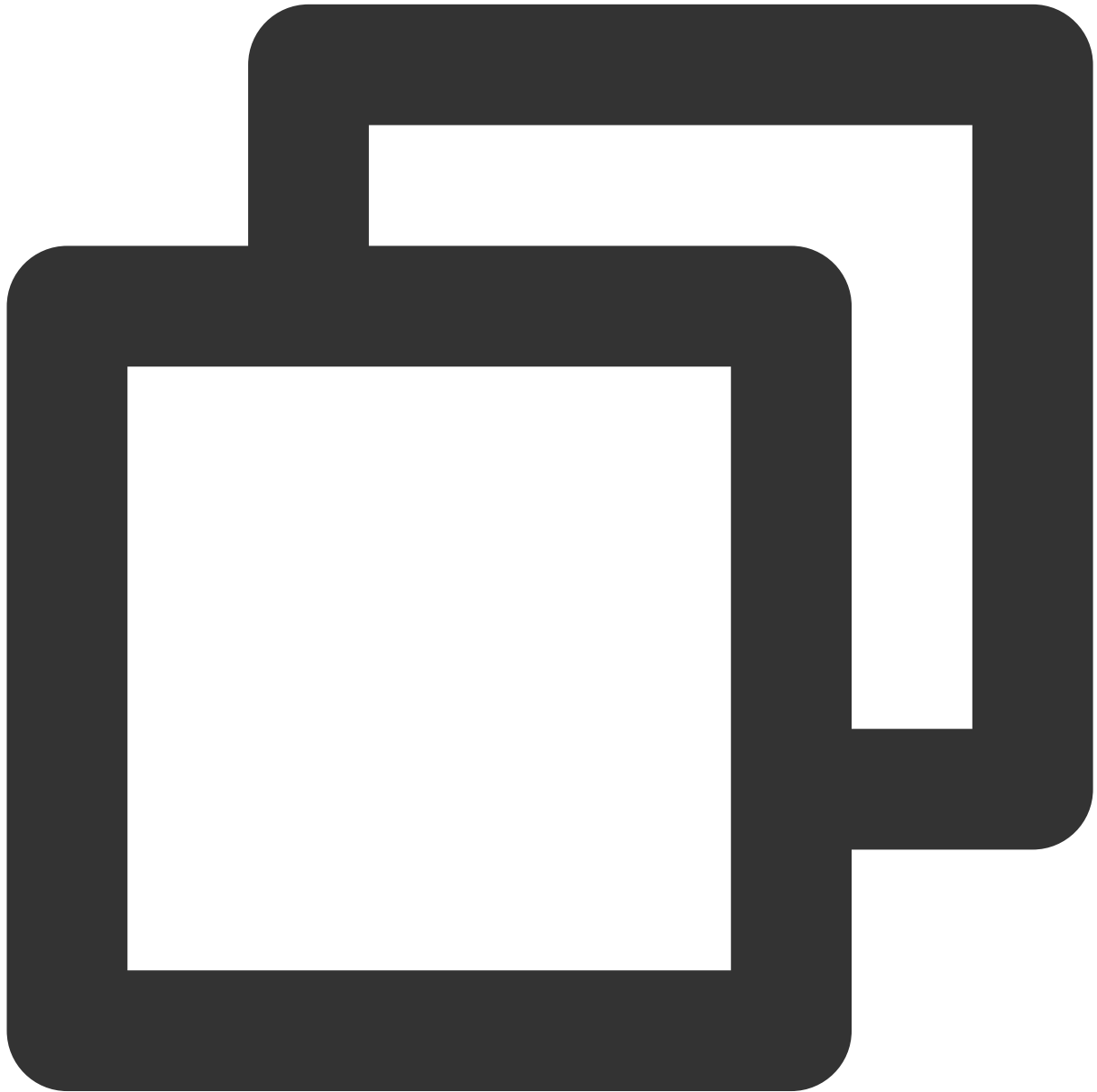
/// Tencent Cloud license management page (https://consoleintl.cloud.tencent.com/)
setupLicense() {
  // The license URL of the current application
  var LICENSEURL = "";
  // The license key of the current application
  var LICENSEURLKEY = "";
  V2TXLivePremier.setLicence(LICENSEURL, LICENSEURLKEY);
}
```

**Note:**

The `packageName/BundleId` configured in the license must be the same as that of the application; otherwise, stream push will fail.

### 3. Create a pusher object

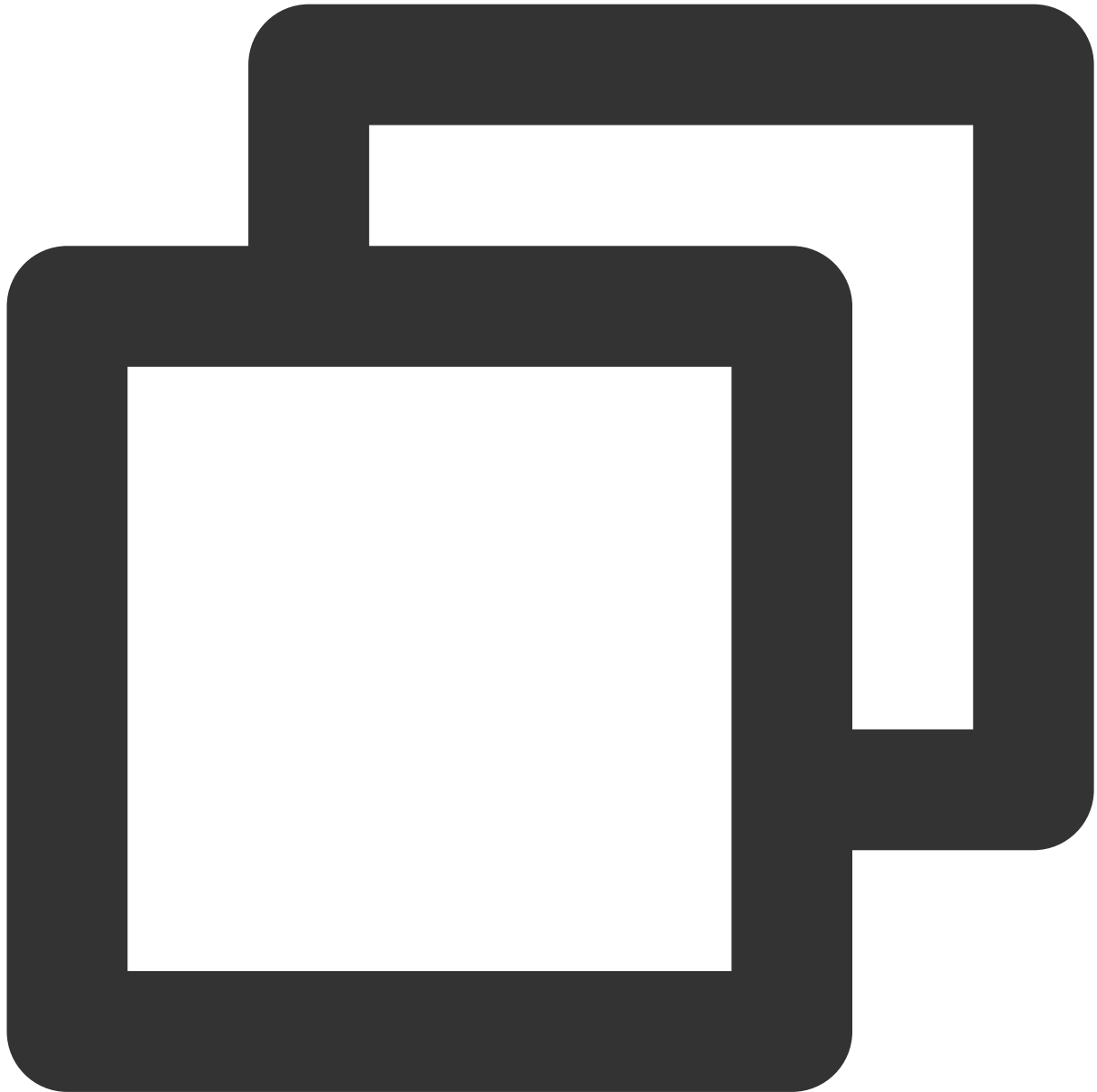
Create a **V2TXLivePusher** object, which will be responsible for publishing operations.



```
V2TXLivePusher livePusher = V2TXLivePusher(V2TXLiveMode.v2TXLiveModeRTMP);
```

#### 4. Start stream push

After completing [Step 1](#), you can use the code below to start publishing streams:



```
String rtmpUrl = "rtmp://2157.livepush.myqcloud.com/live/xxxxxx";  
livePusher.startMicrophone();  
livePusher.startScreenCapture();  
livePusher.startPush(rtmpUrl);
```

##### How can I obtain a valid publishing URL?

Activate CSS. In the **CSS console**, go to **Auxiliary Tools** > [Address Generator](#) to generate a stream push URL.

For more information, see [Publishing/Playback URL](#).

Domain Type \*

Push Domain ▼

If you select push domain, a push address will be generated; and if you select playback domain, a playback address will be generated. If there is no avai

AppName \*

live

Use "live" by default. Only letters, digits, and symbols are supported.

StreamName \*

livetteststream

Only support letters, digits, and symbols.

Expiration Time

UTC+8 ▼

2022-09-16 21:43:52

The expiration time of playback address is the setting timestamp plus the playback authentication expiration time, and the push address expiration time

Generate Address

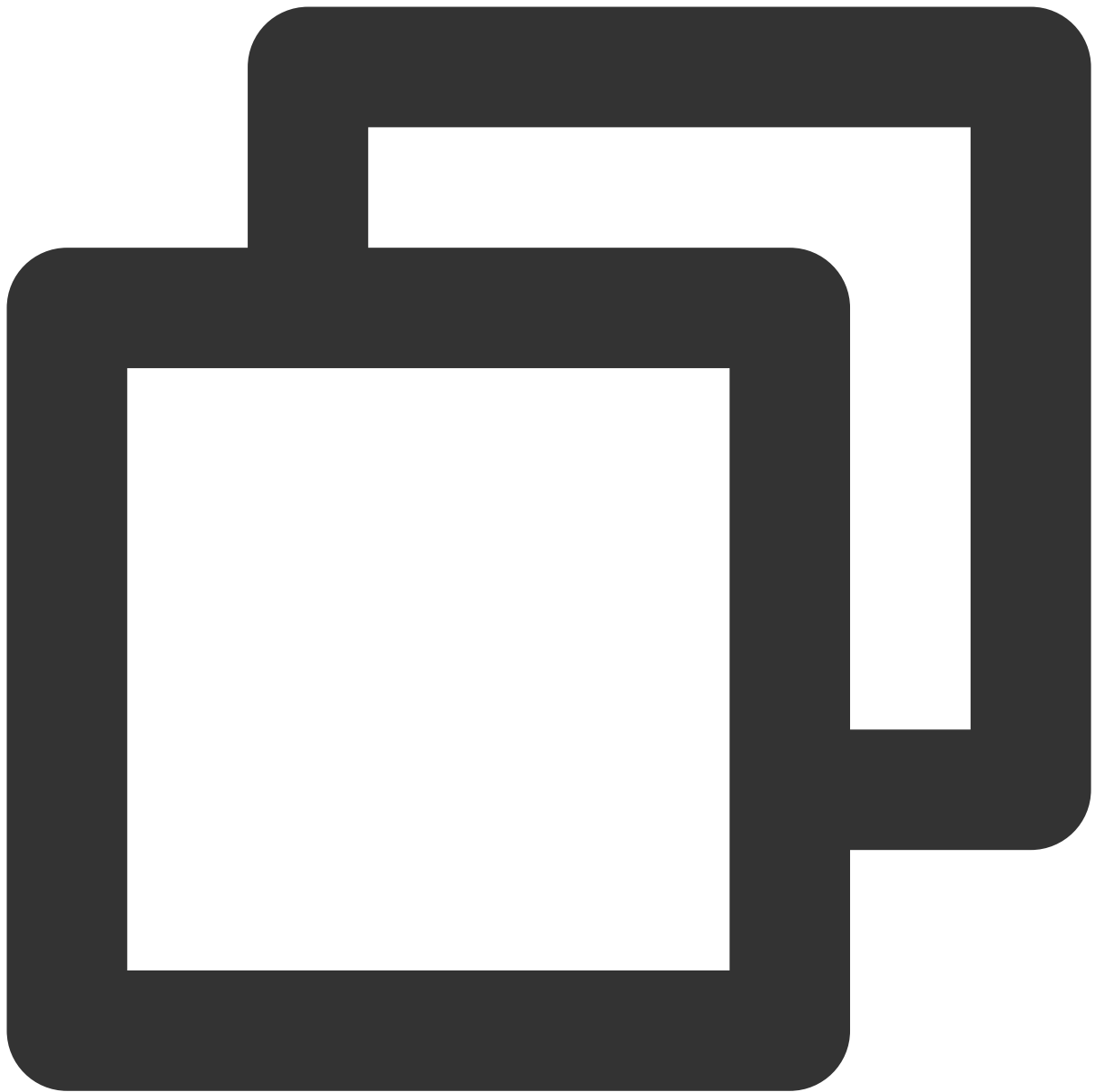
[Address Resolution Sample](#)

**Why is `V2TXLIVE_ERROR_INVALID_LICENSE` returned?**If the `startPush` API returns

`V2TXLIVE_ERROR_INVALID_LICENSE`, it means your license verification failed. Please check your configuration against [Step 2. Configure a license for the SDK](#).

## 5. Stop stream push

As there can be only one `V2TXLivePusher` object running at a time, make sure that you release all the resources when stopping publishing.



```
// Stop screen sharing and release the resources
void stopPush() {
    livePusher.stopMicrophone();
    livePusher.stopScreenCapture();
    livePusher.stopPush();
}
```

## Event Handling

## Listening for events

The SDK listens on push events and errors via the [V2TXLivePusherObserver](#) delegate. See [v2\\_tx\\_live\\_code library](#) for a detailed list of events and error codes.

## Errors

An error indicates that the SDK encountered a serious problem that made it impossible for stream publishing to continue.

Event ID	Code	Description
V2TXLIVE_ERROR_FAILED	-1	A common unclassified error occurred.
V2TXLIVE_ERROR_INVALID_PARAMETER	-2	An invalid parameter was passed in during API calling.
V2TXLIVE_ERROR_REFUSED	-3	The API call was rejected.
V2TXLIVE_ERROR_NOT_SUPPORTED	-4	The API cannot be called.
V2TXLIVE_ERROR_INVALID_LICENSE	-5	Failed to call the API due to invalid license.
V2TXLIVE_ERROR_REQUEST_TIMEOUT	-6	The server request timed out.
V2TXLIVE_ERROR_SERVER_PROCESS_FAILED	-7	The server could not handle your request.

## Warnings

A warning indicates that the SDK encountered a problem whose severity level is warning. Warning events trigger tentative protection or recovery logic and can often be resolved.

Event ID	Code	Description
V2TXLIVE_WARNING_NETWORK_BUSY	1101	Bad network connection: data upload blocked due to limited upstream bandwidth.
V2TXLIVE_WARNING_VIDEO_BLOCK	2105	Stuttering during video playback.
V2TXLIVE_WARNING_CAMERA_START_FAILED	-1301	Failed to turn the camera on.
V2TXLIVE_WARNING_CAMERA_OCCUPIED	-1316	The camera is occupied. Try a different camera.
V2TXLIVE_WARNING_CAMERA_NO_PERMISSION	-1314	No access to the camera. This usually occurs on mobile

		devices and may be because the user denied the access.
V2TXLIVE_WARNING_MICROPHONE_START_FAILED	-1302	Failed to turn the mic on.
V2TXLIVE_WARNING_MICROPHONE_OCCUPIED	-1319	The mic is occupied. This occurs when, for example, the user is having a call on the mobile device.
V2TXLIVE_WARNING_MICROPHONE_NO_PERMISSION	-1317	No access to the mic. This usually occurs on mobile devices and may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_NOT_SUPPORTED	-1309	The system does not support screen sharing.
V2TXLIVE_WARNING_SCREEN_CAPTURE_START_FAILED	-1308	Failed to start screen recording. If this occurs on a mobile device, it may be because the user denied the access.
V2TXLIVE_WARNING_SCREEN_CAPTURE_INTERRUPTED	-7001	Screen recording was stopped by the system.

## FAQs

ReplayKit2 is a new framework introduced by Apple in iOS 11, for which relatively few official documents have been released. The framework is still being improved, and problems have been found. See below for some common questions you may have when using ReplayKit2.

### 1. When does screen recording stop automatically?

Screen recording stops automatically when the screen locks or there is an incoming call. At such times, the `broadcastFinished` function in `SampleHandler` will be invoked, and you can send a notification to users about the interruption.

### 2. Why does screen recording stop sometimes during screen sharing?

The problem usually occurs after landscape/portrait mode switch if the resolution for stream publishing is set high. The broadcast upload extension is allocated a memory of only 50 MB and will be killed if its memory usage exceeds the limit. Given this, we recommend that you set the resolution to 720p or lower.

### 3. Why are images streamed from the screen of iPhone X distorted?

iPhone X has a notch at the top of the screen, so video captured from the screen is not in the aspect ratio of 16:9. If

you set the output resolution for stream publishing to 16:9, for example, to HD (960 × 540), the images published will be slightly distorted because their original aspect ratio is not 16:9. We recommend that you set the resolution according to your screen size. Besides, if you play video streamed from the screen of iPhone X in aspect fit mode, the video may have black bars, and if you play it in aspect fill mode, the video may be cropped.