

Mobile Live Video Broadcasting IOS-based Integration Product Introduction



Copyright Notice

©2013-2018 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

IOS-based Integration

Basic Features

- Getting Started

- TXLivePusher

- TXLivePlayer

- TXVodPlayer

- Screen Recording (ReplayKit)

- Effect Feature

Advanced Feature

- SDK Internal Principles

- SDK Metric Monitoring

- QoS Traffic Control

- Coding Parameter Adjustment

- Video Data Customization

IOS-based Integration

Basic Features

Getting Started

Last updated : 2018-09-21 19:17:25

SDK Information

You can update the [LVB SDK](#) on Tencent Cloud's official website, which has the following versions:

Version	Feature
LVB simplified version	Supports push, LVB, and VOD
Independent player version	Supports LVB and VOD
Short video feature version	Supports short video and VOD
Full-featured professional version	Supports push, LVB, VOD, joint broadcasting, and short video
Commercial enterprise version	Motion effect sticker, eyes beautifying and face slimming, and green screen keying-out features are added on the basis of full-featured professional version

In professional version, for example, the decompressed SDK is composed as follows:

名称	修改日期	类型	大小
Demo	2017/8/11 21:09	文件夹	
SDK	2017/8/11 21:09	文件夹	
iOS 开发包使用指引.pdf	2017/8/11 20:15	Adobe Acrobat ...	77 KB

File Name	Description
SDK	Contains the SDK directory of framework
Demo	The simplified demo based on framework, including a simple demonstration of UI and main SDK features. Use Xcode to quickly import the demo and try it out.
iOS package user guide.pdf	Describes the basic features of SDK

Xcode Project Settings

1. Supported platform

- SDK is supported on iOS 8.0 or above.

2. Development environment

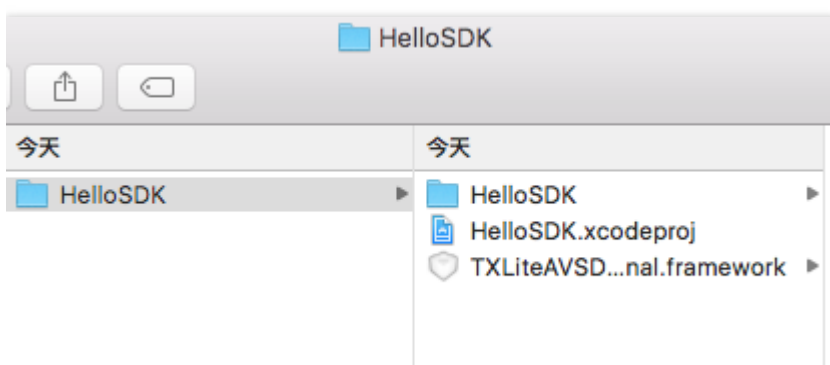
- Xcode 9 or above
- OS X 10.10 or above

3. Xcode project settings

Here, we use an iOS Application project to show you how to configure SDK in an Xcode project.

3.1 Copy SDK file

In this example, we create an iOS project named HelloSDK and copy the downloaded `TXLiteAVSDK_Professional.framework` to the project directory. The figure below shows the directory structure:



3.2 Add framework

Add `TXLiteAVSDK_Professional.framework` to the project, and also add the following dependent libraries:

- (1) `libz.tbd`
- (2) `libstdc++.tbd`
- (3) `libresolv.tbd`
- (4) `Accelerate.framework`

3.3 Add header file

Add the header file search path to **Build Settings** -> **Search Paths** -> **User Header Search Paths**. Please note that this operation is not required. If you do not add the header file search path for `TXLiteAVSDK_Professional`, "`TXLiteAVSDK_Professional/`" needs to be added before the SDK-related header file when the header file is referenced, as shown below:

```
#import "TXLiteAVSDK_Professional/TXLivePush.h"
```

4. Verification

Call SDK API in the `HelloSDK` code and obtain SDK version information to verify whether the project is correctly configured.

4.1 Reference header file

Reference the SDK header file at the beginning of `ViewController.m`:

```
#import "TXLiteAVSDK_Professional/TXLiveBase.h"
```

4.2 Add calling code

Add the following code to `viewDidLoad` method:

```
- (void)viewDidLoad {  
    [super viewDidLoad];  
    //Print SDK version information  
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);  
}
```

4.3 Compile and run the project

If all of the above steps are performed correctly, the `HelloSDK` project can be compiled successfully. Run the App in Debug mode. SDK version information is output in Xcode's Console pane.

```
2017-08-11 16:16:15.767 HelloSDK[17929:7488566] SDK Version = 3.0.1185
```

Now, the project configuration is completed.

Printing LOG

The code used to configure whether to print log from the console and set the log level in TXLiveBase is described as follows:

- **setConsoleEnabled**

Configures whether to print the output of SDK from the Xcode console.

- **setLogLevel**

Configures whether to allow SDK to print local log. By default, SDK writes log to the **Documents/logs** folder of the current App.

To get technical support, you are recommended to enable printing and provide the log file when a problem occurs. Thank you for your support.

- **View log file**

To reduce the storage size of logs, Mini LVB SDK encrypts local log files and limits the number of logs. Therefore, the log [decompression tool](#) is required to view the text content of logs.

```
[TXLiveBase setConsoleEnabled:YES];  
[TXLiveBase setLogLevel:LOGLEVEL_DEBUG];
```

- **SDK log callback:** You can print the logs to your log file by implementing the method in `TXLiveBaseDelegate` callback:

```
-(void) onLog:(NSString*)log LogLevel:(int)level WhichModule:(NSString*)module
```

TXLivePusher

Last updated : 2018-08-10 16:20:46

Basics

Push means pushing the collected and encoded audio/video data to your specified cloud video platform. Since the process involves a large amount of basic audio/video knowledge, you can only achieve desired results after lots of refinements and optimizations.

Tencent Video Cloud SDK mainly helps you push streams on smart phones. The SDK comes with easy-to-use APIs which can be driven by a single push URL.



Notes

- **Not bound to Tencent Cloud**

The SDK is not bound to Tencent Cloud. If you want to push streams to non-Tencent Cloud addresses, please set enableNearestIP in TXLivePushConfig to NO first. If you want to push streams to Tencent Cloud addresses, set enableNearestIP to Yes. Otherwise the push quality may be affected due to inaccurate ISP DNS.

- **x86 simulator debugging**

Since the SDK uses a great number of advanced features of the iOS system, we cannot ensure that all the features can function normally under the simulator in the x86 environment. Furthermore, audio and video are performance-sensitive features, the performance under the simulator will be greatly different from that on a real phone. Therefore, it is recommended to use an actual mobile phone for debugging if possible.

Preparations

- **Acquiring SDK**

[Download](#) SDK and follow the instructions in [Project Configuration](#) to add the SDK into your application development project.

- **Acquiring a test URL**

After [activating](#) the LVB service, you can use [LVB Console](#) -> [LVB Code Access](#) -> [Push Generator](#) to generate a push URL. For more information, please see [Acquiring Push/ Playback URL](#).

过期时间:	2018-01-12 23:59:59	直播码:	3891_ test	生成推流地址
推流地址:	rtmp://3891.livepush.myqcloud.com/live/3891_test?bizid=3891&txSecret=1d2f4f66920c707aa01e5d327c725555&txTime=5A58DB7F			
播放地址 (RTMP):	rtmp://3891.liveplay.myqcloud.com/live/3891_test			
播放地址 (FLV):	http://3891.liveplay.myqcloud.com/live/3891_test.flv			
播放地址 (HLS):	http://3891.liveplay.myqcloud.com/live/3891_test.m3u8			

Code Interfacing

This guide is specific to **camera LVB** and is mainly used for scenarios such as beauty show LVB and event LVB. For information on game LVB, please see relevant documents under this directory.

Step 1: Create a Pusher object

Create a **LivePush** object, which will be used later to complete the push task.

Before creating a LivePush object, you need to specify a **LivePushConfig** object to determine the configuration parameters for various LivePush push phases, such as push resolution, frames per second (FPS) and GOP (seconds between I frames).

The LivePushConfig is already equipped with some parameters we have repeatedly tuned as a result of calling alloc. If you do not wish to customize these parameters, you can simply alloc and assign them to the LivePush object. If you have experience in the related field and want to adjust the default configuration, you can read the **Advanced Guide**.

```
// Create a LivePushConfig object, which is initialized with the basic configuration by default.
TXLivePushConfig* _config = [[TXLivePushConfig alloc] init];
//In _config, you can perform certain initialization operations on push parameters (e.g. whitening, hardware acceleration, and front/rear camera). Note that _config cannot be nil.
_txLivePush = [[TXLivePush alloc] initWithConfig: _config];
```

Step 2: Rendering view

Next, we need to find a place to display the camera images. In iOS systems, a view is used as the basic interface rendering unit. Therefore, all you need to do is to prepare a view and pass it to the **startPreview** API function of the LivePush object.

- **Recommended layout!**

In fact, the SDK does not directly render the images on the view you provided. Instead, it creates a subView used for OpenGL rendering upon the view. The size of this subView will be adjusted

automatically according to that of the view you provided.



However, if you want to implement UI controls such as on-screen comment and flower presenting on the rendered screen, we recommend that you create another view at the same level to avoid screen overlay.

• How to make an animation?

You can freely make animations for a view. But note that the target attribute modified for animations is **transform**, instead of frame.

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

Step 3: Start push

After completing Step 1 and Step 2, you can use the following code to start the push:

```
NSString* rtmpUrl = @"rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
[_txLivePush startPreview:_myView]; // _myView is the view to be specified in Step 2
```

```
[_txLivePush startPush:rtmpUrl];
```

- **startPush** is used to tell the SDK to which push URL the audio/video streams are being pushed.
- The parameter of **startPreview** is the view you need to specify in Step 2; startPreview is used to associate the interface view control with the LivePush object, thus rendering the images collected by the mobile phone camera onto the screen.

Step 3+: Audio-only push

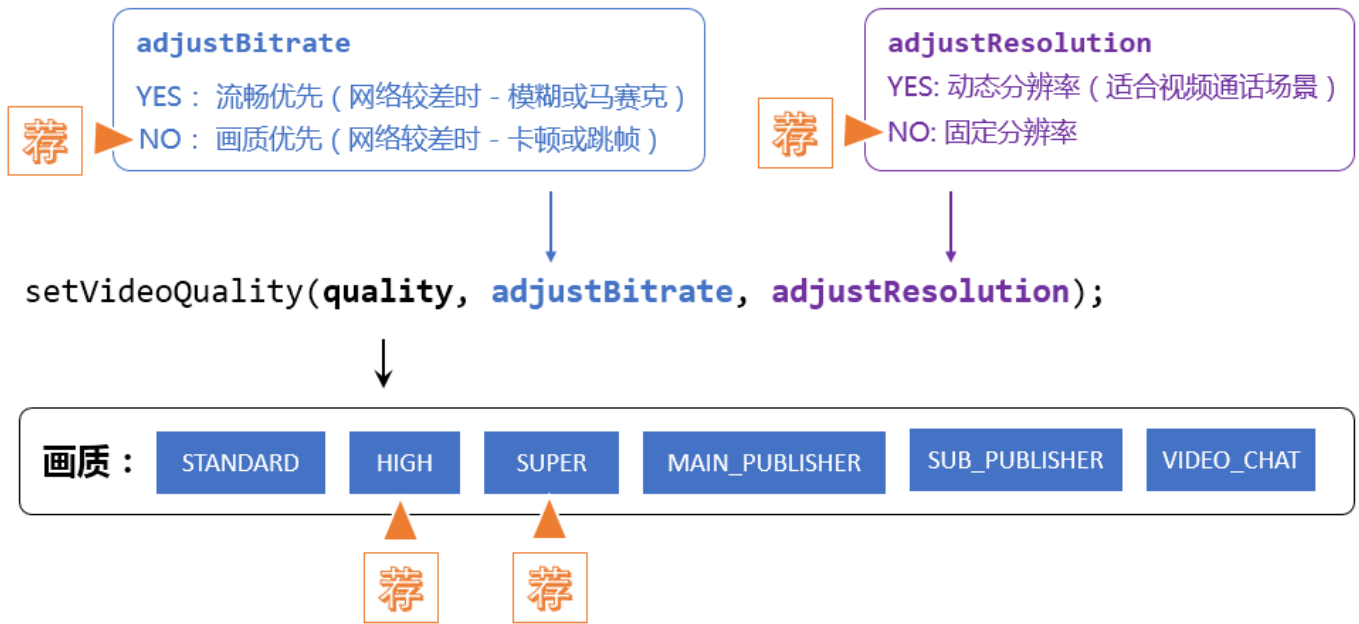
For audio-only LVB scenarios, you need to update the push configuration. Perform Step 1 and Step 2 as described above. Configure audio-only push using the following code and start the push.

```
//The API only takes effect if it is called before push starts.  
txLivePush.config.enablePureAudioPush = YES; // "YES" means enabling audio-only push. Default is "NO".  
[_txLivePublisher setConfig:_config]; // Reset config.  
  
NSString* rtmpUrl = @"rtmp://2157.livepush.myqcloud.com/live/xxxxxx";  
[_txLivePush startPush:rtmpUrl];
```

If you cannot pull streams from playback URLs in rtmp, flv and hls format after enabling audio-only push, it is because the line configuration is incorrect. Submit a ticket to us and we will help you modify the configuration.

Step 4: Configure video definition

If this is your first time to use audio-video streams, you are **not recommended** to set video parameters such as resolution and bitrate by yourself. This is because improper parameter configuration may have a negative effect on the final video quality. You can configure image definition for push using TXLivePusher::setVideoQuality API.



• Recommended parameter settings

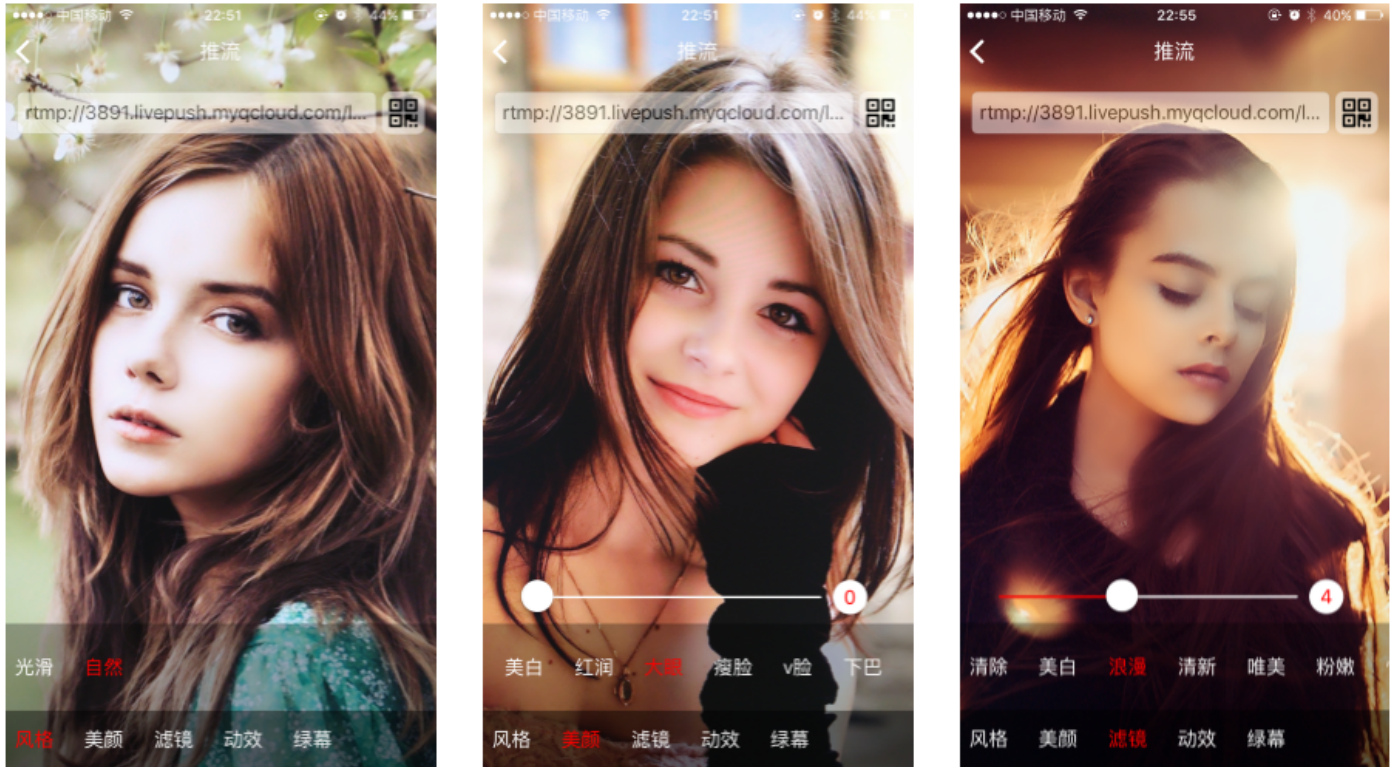
Application Scenario	quality	adjustBitrate	adjustResolution
Live show	VIDEO_QUALITY_HIGH_DEFINITION or VIDEO_QUALITY_SUPER_DEFINITION	NO	NO
Mobile game LVB	VIDEO_QUALITY_SUPER_DEFINITION	YES	YES
Joint broadcasting (primary screen)	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	YES	YES
Joint broadcasting (secondary screen)	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	NO	NO
Video chat	VIDEO_QUALITY_REALTIEM_VIDEOCHAT	YES	YES

• Internal data metrics

quality	adjustBitrate	adjustResolution	Bitrate Range	Resolution Range
---------	---------------	------------------	---------------	------------------

quality	adjustBitrate	adjustResolution	Bitrate Range	Resolution Range
STANDARD	YES	YES	300~800kbps	270x480 ~ 360x640
STANDARD	YES	NO	300~800kbps	360x640
STANDARD	NO	NO	800kbps	360x640
HIGH	YES	YES	600~1500kbps	360x640~540x960
HIGH	YES	NO	600~1500kbps	540x960
HIGH	NO	NO	1200kbps	540x960
SUPER	YES	YES	600~1800kbps	360x640~720x1280
SUPER	YES	NO	600~1800kbps	720x1280
SUPER	NO	NO	1800kbps	720x1280
MAIN_PUBLISHER	YES	YES	600~1500kbps	360x640~540x960
SUB_PUBLISHER	NO	NO	350kbps	320x480
VIDEOCHAT	YES	YES	200~800kbps	190x320~360x640

Step 5: Beauty filter



• Beauty filter

You can set beauty filter style, dermabrasion level, whitening level, and blushing level using `setBeautyStyle` API. You can obtain the best video quality using beauty filter with 540 * 960 resolution (`setVideoQuality` - `VIDEO_QUALITY_HIGH_DEFINITION`):

```
// beautyStyle : Dermabrasion style. Smooth and Natural are supported.
// beautyLevel : Dermabrasion level. The values range from 0 to 9. 0 means disabling dermabrasio
n. A higher value means a stronger effect.
// whitenessLevel : Whitening level. The values range from 0 to 9. 0 means disabling whitening. A h
igher value means a stronger effect.
// ruddinessLevel : Blushing level. The values range from 0 to 9. 0 means disabling blushing. A high
er value means a stronger effect.
(void)setBeautyStyle:(int)beautyStyle beautyLevel:(float)beautyLevel
whitenessLevel:(float)whitenessLevel ruddinessLevel:(float)ruddinessLevel;
```

• Filter

The `setFilter` API can be used to configure filter effects. A filter is actually a histogram file. Our designer group provides 8 materials which are packaged inside the Demo by default. You can use them as you like, without considering about copyright issues.

The setSpecialRatio API is used to configure the effect level of a filter (0-1). A higher value means a stronger effect. Default is 0.5.

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
if (path != nil && index != FilterType_None && _txLivePublisher != nil) {
    path = [path stringByAppendingPathComponent:lookupFileName];
    UIImage *image = [UIImage imageWithContentsOfFile:path];
    [_txLivePublisher setFilter:image];
}
```

Use PNG images if you need to customize the filters. **Do NOT use JPG images.**

Step 6: Control the camera

- **Switch between front and rear cameras**

The **front** camera is used by default (this can be changed by modifying the configuration option frontCamera in LivePushConfig). The camera is switched each time switchCamera is called. Make sure both LivePushConfig and LivePush objects have been initialized before you switch the camera.

```
//The front camera is used by default. This can be changed by modifying the configuration option frontCamera in LivePushConfig.
[_txLivePush switchCamera];
```

- **Turn the flashlight on or off**

Flashlight is only available for the rear camera. (You can find out whether the front or the rear camera is used now by checking the frontCamera member in "TXLivePush.h")

```
if(!frontCamera) {
    BOOL bEnable = YES;
    //Flashlight is on when bEnable is YES; flashlight is off when bEnable is NO
    BOOL result = [_txLivePush toggleTorch: bEnable];
    //A result of YES means the flashlight is successfully turned on, while NO means the flashlight fails to be turned on
}
```

- **Customize manual focus**

Default manual focus logic is provided in the iOS version of the SDK. Although there is no problem regarding its functionality, the logic usually fails to work when touch events of the screen are occupied.

Meanwhile, as a principle, we cannot intervene with the free interface arrangement practice.

A `setFocusPosition` function API is added to the new version of TXLivePush. You can perform manual focus based on where your finger is touching.

```
//If the customer calls this API, the focus trigger logic in the SDK will stop, avoiding repeated trigger of the focus logic  
- (void)setFocusPosition:(CGPoint)touchPoint;
```

Step 7: Set Logo watermark

Recent policies require that LVB videos must be marked with watermarks. With that in mind, we will focus on this feature that had seemed insignificant before.

Tencent Video Cloud supports two watermark setting methods. One is to set watermark in the push SDK, where the videos are marked with watermarks in the SDK before being encoded. Another is to apply watermarks in the cloud. That is, the cloud resolves videos and adds Logo watermarks to them.

We suggest that you **add watermarks with the SDK**, because there are three major problems when watermarking in the cloud:

- (1) This service increases load on the cloud machine and is not free, which will increase your cost.
- (2) It is not ideally compatible with certain situations such as resolution switching during the push process. This may cause problems like blurred screen.
- (3) It may bring about an extra 3-second video delay, which is caused by the transcode service.

The SDK requires that watermark images are in PNG format, because such images contain transparency information, which helps better solve issues like jagged screen. (Do not just change the extension of a JPG image to PNG in Windows and use it as a watermark image. Professional PNG logos need to be processed by professional art designers)

```
//Set video watermark  
_config.watermark = [UIImage imageNamed:@"watermark.png"];  
_config.watermarkPos = (CGPoint){10, 10};
```

Step 8: Local recording

You can start local recording using `startRecord` API. The recording format is MP4. You can specify the storage path for the MP4 files using `videoPath`.

- Do not change resolution and soft/hard encoding during recording. Otherwise, exceptions may exist in the generated videos.
- For cloud recording, you only need to concatenate `&record=mp4` to the end of the push URL. For more information, please see [Cloud Recording](#).

- You will be notified of the generation of a recorded file through `TXLiveRecordListener` after `stopRecord` is called.

```
-(int) startRecord:(NSString *)videoPath;  
-(int) stopRecord;
```

Step 9: Push at the backend

Usually, once the App switches to the backend, the camera's capture function will be temporarily disabled by the iOS system, which means the SDK cannot continue capturing and encoding audio/video data. This is what happens if we don't do anything:

- Phase 1 (from switching to the backend -> 10 seconds later) - CDN cannot provide video streams to viewers because it doesn't have any data, and the viewers will see a frozen display.
- Phase 2 (10 seconds -> 70 seconds) - The player at the viewer end exits because it haven't received LVB streams for a long time. Everyone leaves the room.
- Phase 3 (after 70 seconds) - The RTMP linkage of the push is disconnected by the server. The VJ needs to restart LVB to continue.

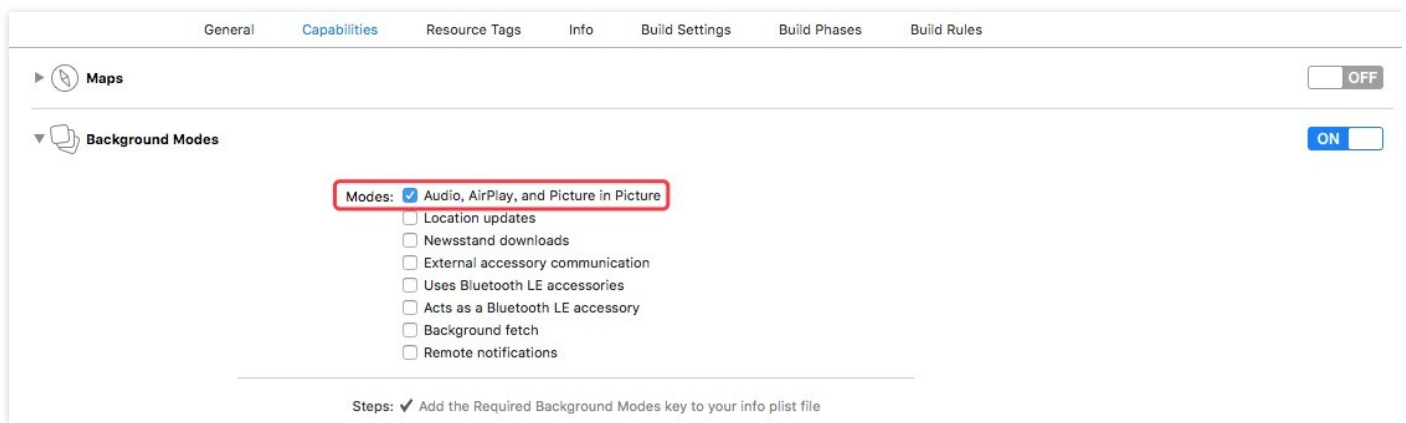
Sometimes a VJ may have to answer a phone call which will cause a pause. But even a short pause can bring unsatisfactory interaction experience as described above that leads to exit of viewers from the room. So, how can we optimize this?

We introduced a solution since **SDK 1.6.1**. Below is the result achieved at the viewer end when this

solution is used:



• 9.1) Adjust XCode configuration



• 9.2) Set pauseImg

Before push starts, you can use the `pauseImg` API of `LivePushConfig` to set a waiting picture saying like "The VJ will come back soon".

```
// 300 is the maximum duration of the image displayed at the pause of backend push (in sec).
_config.pauseTime = 300;
// 10 is the frame rate of the image displayed at the pause of backend push. The minimum is 5 and t
```


he maximum is 20.

```
_config.pauseFps = 10;
```

*// The size of the image displayed at the pause of backend push cannot exceed 1920*1920.*

```
_config.pauseImg = [UIImage imageNamed:@"pause_publish.jpg"];
```

```
[_txLivePublisher setConfig:_config];
```

• 9.3) Set temporary running of App at the backend

The App goes into sleep mode when it is switched to the backend, and consequently the SDK stops pushing streams. As a result, viewers can only see a black screen or frozen screen of the live room. The following code enables the App to run for a few minutes after it is switched to the backend, which is long enough for a VJ to answer a short phone call.

//Register before pushing message notification

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(handleEnterBackground:)  
name:UIApplicationDidEnterBackgroundNotification object:nil];
```

//Call beginBackgroundTaskWithExpirationHandler after receiving the notification

```
-(void)handleEnterBackground:(NSNotification *)notification  
{  
    [[UIApplication sharedApplication] beginBackgroundTaskWithExpirationHandler:^(  
    }];  
}
```

• 9.4) Switch to the backend

In handleEnterBackground mentioned in the last step, call the API function pausePush of TXLivePush. The SDK cannot capture camera images, but it can keep pushing streams via pauseImg you just configured.

//Switch to the backend: Add the following to the code from the last step

```
-(void)handleEnterBackground:(NSNotification *)notification  
{  
    [[UIApplication sharedApplication] beginBackgroundTaskWithExpirationHandler:^(  
    }];  
    [_txLivePush pausePush];  
}
```

• 9.5) Switch to the frontend

After the App is switched to the frontend, (in handleEnterForeground), call the API function resumePush of TXLivePush to resume capturing camera images. Note: pausePush and resumePush need to be used in pairs, because they are closely related to SDK internal state. Otherwise, many bugs will be introduced.


```
//Switch to the frontend  
- (void)handleEnterForeground:(NSNotification *)notification  
{  
    [_txLivePush resumePush];  
}
```

Step 10: Stutter alert

- What should we do if the network quality is poor at the VJ end?
- Should we lower the definition to ensure the smoothness? It will lead to blurry video images with many mosaics at the viewer end.
- Should we drop some of the video frames to maintain the image definition? It will lead to continuous stutter at the viewer end.
- Since neither of the above is satisfactory, what should we do?
- We all know that it's impossible to "eat your cake and have it."

You can capture the **PUSH_WARNING_NET_BUSY** event by using `onPlayEvent` in `TXLivePushListener`. This event indicates that the VJ's network is extremely poor and stutters occur at the viewer end.

You can prompt the VJ with a message indicating "**Poor network quality. Please move closer to your WiFi, and make sure the signal isn't blocked by any wall or obstacle**".

Step 11: Push in landscape mode

In most cases, VJs push videos in an LVB by holding the screen in a portrait orientation so that the viewers can get portrait images. However, sometimes VJs may need to hold the screen in a landscape orientation to allow the viewers to get landscape images with a wider view. In this case, push in landscape mode is required. The figures below show the difference between landscape mode and portrait mode in terms of

the images at the viewer end.



Note: The aspect ratios of images at viewer end are different between landscape mode and portrait mode. In portrait mode, the aspect ratio is 9:16, while in landscape mode, 16:9.

To implement landscape mode, you need to make two configurations:

- **Adjust image display at the viewer end**

Configure the **homeOrientation** option in LivePushConfig. It controls whether the aspect ratio of images at the viewer end is **16:9** or **6:19**. You can check whether the aspect ratio is adjusted as expected by using your player.

- **Adjust image display at the VJ end**

Next, you need to see whether the local rendering at the VJ end is normal. You can use the **setRenderRotation** API in TXLivePush to set the rotation of the images at the VJ end. This API provides four parameters (**0**, **90**, **180** and **270**) for setting the rotation angle.

Step 12: Background audio mixing

SDK 1.6.1 and later versions support background audio mixing, and VJs can choose to wear or not wear a headset. You can implement background audio mixing by using the following APIs in TXLivePush:

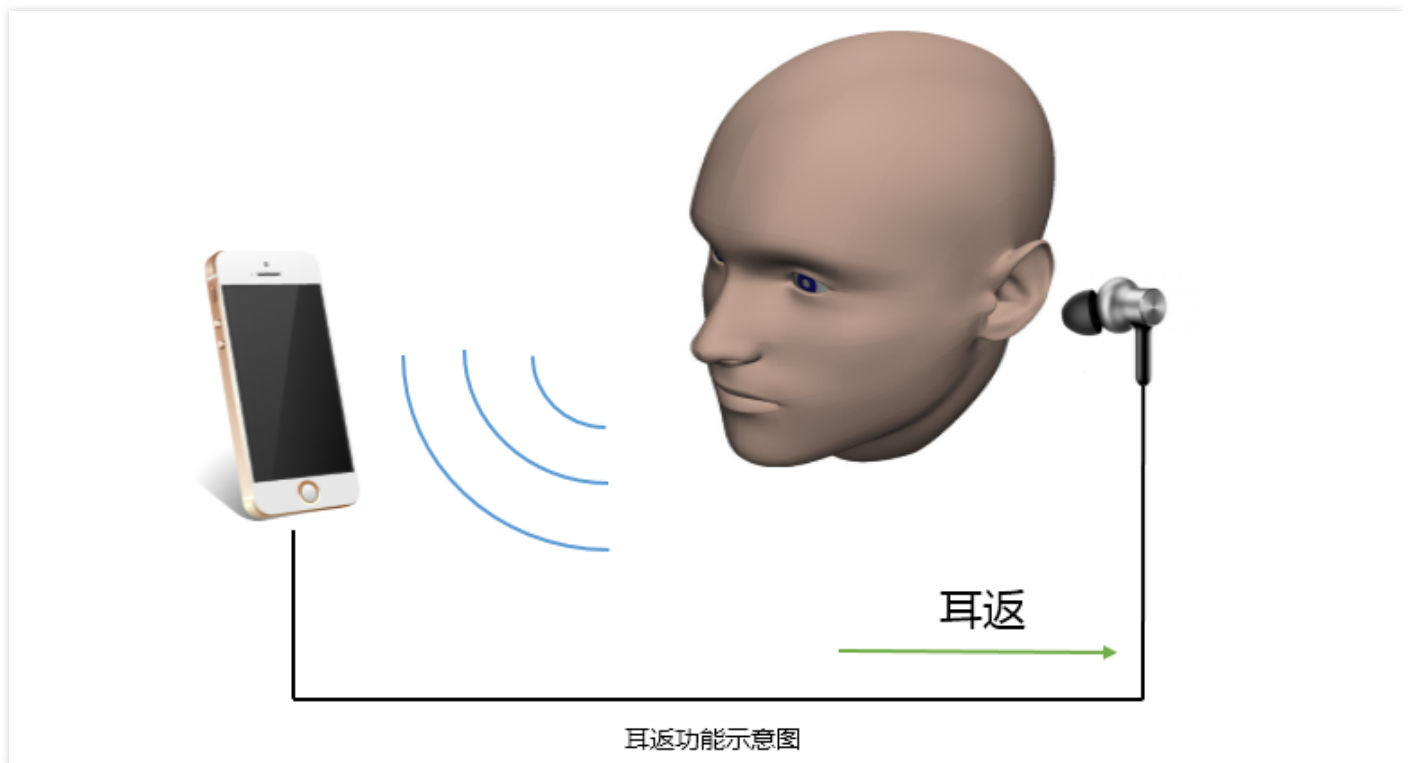
API	Description
playBGM	Passes a piece of music via path. In Mini LVB Demo , we obtain music files from the iOS local media library
stopBGM	Stops background music
pauseBGM	Pauses background music
resumeBGM	Resumes background music
setMicVolume	Sets microphone volume for audio mixing. It is recommended to add a slider in the UI to allow VJs to set volume on their own
setBGMVolume	Sets background music volume for audio mixing. It is recommended to add a slider in the UI to allow VJs to set volume on their own

Step 13: In-ear monitoring/Reverb

- In-ear monitoring**

This means when a VJ is singing with a headset on, the headset will feed back the VJ's voice in real time. This is because the VJ hears his or her own voice transmitted through bone structures in the skull (solid), while the viewers hear the voice transmitted through the air. These two voices can be very

different, thus the VJ needs to hear the voice effect at the viewer end.



In-ear monitoring can be enabled through the `enableAudioPreview` API in `TXLivePushConfig`. In joint broadcasting scenarios, it is recommended that only the primary VJ enables this feature while secondary VJs do not, because it sounds strange in real-time video/audio chats when in-ear monitoring is enabled.

- **Reverb**

This means adding certain special effects when using in-ear monitoring, such as KTV, Grand Hall, Magnetic and Metallic, to make VJs' voice more impressive to viewers. Reverb effects can be set through `setReverbType` (supported by version 1.9.2 and later), the member function of `TXLivePush`.

The following reverb effects are supported: KTV, Small Room, Grand Hall, Low-pitched, Sonorous, Metallic and Magnetic.

Step 14: End push

Ending a push is simple, but proper cleanup is required. Since only one `TXLivePush` object can run at a time, improper cleanup may adversely affect the next LVB.

```
//End a push with proper cleanup
- (void)stopRtmpPublish {
```

```
[_txLivePush stopPreview];  
[_txLivePush stopPush];  
_txLivePush.delegate = nil;  
}
```

Send messages

This feature is used to deliver certain custom messages from the pusher end to the viewer end via audio/video lines. It is applicable to the following scenarios:

- (1) Online quiz: The pusher end delivers the **questions** to the viewer end. Perfect "sound-image-question" synchronization can be achieved.
- (2) Live show: The pusher end delivers **lyrics** to the viewer end. The lyric effect can be displayed on the viewer end in real time and its image quality is not affected by video encoding.
- (3) Online education: The pusher end delivers the operations of **Laser pointer** and **Doodle pen** to the viewer end. The drawing can be performed at the viewer end in real time.

```
[_answerPusher sendMessage:[mesg dataUsingEncoding:NSUTF8StringEncoding]];
```

onPlayEvent (PLAY_EVT_GET_MESSAGE) of TXLivePlayer can be used to receive messages.

Event Handling

1. Event listening

SDK listens to push related events using the TXLivePushListener proxy. Note that the TXLivePushListener only listens to push events with prefix **PUSH_**.

2. Normal events

A notification event is prompted after each successful push. For example, receiving 1003 means that the system will start rendering the camera pictures.

Event ID	Value	Description
PUSH_EVT_CONNECT_SUCC	1001	Successfully connected to Tencent Cloud push server

Event ID	Value	Description
PUSH_EVT_PUSH_BEGIN	1002	Handshake with the server completed, everything is OK, ready to start push
PUSH_EVT_OPEN_CAMERA_SUCC	1003	The pusher has successfully started the camera (this will take 1-2 seconds on some Android phones)

3. Error notifications

The push cannot continue as the SDK detected critical problems. For example, the user disabled camera permission for the App so the camera cannot be started.

Event ID	Value	Description
PUSH_ERR_OPEN_CAMERA_FAIL	-1301	Failed to enable camera
PUSH_ERR_OPEN_MIC_FAIL	-1302	Failed to enable microphone
PUSH_ERR_VIDEO_ENCODE_FAIL	-1303	Video encoding failed
PUSH_ERR_AUDIO_ENCODE_FAIL	-1304	Audio encoding failed
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sampling rate
PUSH_ERR_NET_DISCONNECT	-1307	Network disconnected. Three failed reconnection attempts have been made. Restart the push for more retries.

4. Warning events

Some non-fatal errors occurred with SDK can be solved in most cases by throwing warning events to trigger protection or recovery logics.

- **WARNING_NET_BUSY**

VJ's network is busy. This warning can be used as a UI message for users (Step 10).

- **WARNING_SERVER_DISCONNECT**

Push request rejected by backend. This is usually caused by the miscalculated txSecret in the push URL, or because the push URL is already in use (a push URL can only be used by one pusher at a time).

Event ID	Value	Description
----------	-------	-------------

Event ID	Value	Description
PUSH_WARNING_NET_BUSY	1101	Bad network condition: data upload is blocked because upstream bandwidth is too small.
PUSH_WARNING_RECONNECT	1102	Network disconnected. Auto reconnection has been initiated (no more attempts will be made after three failed attempts).
PUSH_WARNING_HW_ACCELERATION_FAIL	1103	Failed to start hard encoding. Soft encoding is used instead.
PUSH_WARNING_DNS_FAIL	3001	RTMP - DNS resolution failed (this triggers a retry)
PUSH_WARNING_SEVER_CONN_FAIL	3002	Failed to connect to the RTMP server (this triggers a retry)
PUSH_WARNING_SHAKE_FAIL	3003	Handshake with RTMP server failed (this triggers a retry)
PUSH_WARNING_SERVER_DISCONNECT	3004	The RTMP server disconnected automatically (this triggers a retry)

TXLivePlayer

Last updated : 2018-09-26 10:20:50

Basics

This document describes the LVB playback feature of Tencent Video Cloud SDK. The following are the basics you must learn before getting started.

- **LVB and VOD**

The video source of [LVB \(LIVE\)](#) is pushed by VJ in real time. When the VJ stops broadcasting, the video image on the playback device stops. In addition, the video is broadcasted in real time, no progress bar is displayed when the player is playing the LVB URL.

The video source of [Video On-demand \(VOD\)](#) is a video file on cloud, which can be played at any time as long as it has not been deleted from the cloud. You can control the playback progress using the progress bar. The video playbacks on Tencent Video and Youku Tudou are typical VOD scenarios.

- **Supported Protocols**

Commonly used LVB protocols are as follows. It is recommended to use an LVB URL based on the FLV protocol (starting with "http" and ending with ".flv") on Apps:

直播协议	优点	缺点	播放延迟
FLV	成熟度高、高并发无压力	需集成SDK才能播放	2s - 3s
RTMP	优质线路下理论延迟最低	高并发情况下表现不佳	1s - 3s
HLS(m3u8)	手机浏览器支持度高	延迟非常高	10s - 30s

Notes

- **Is there any restriction?**

Tencent Cloud SDK **does not** impose any restrictions on the source of playback URLs, which means you can use the SDK to play videos from both Tencent Cloud and non-Tencent Cloud addresses. But the player in Tencent Video Cloud SDK only supports three LVB video address formats (FLV, RTMP and HLS (m3u8)) and three VOD address formats (MP4, HLS (m3u8) and FLV).

- **Historical factors**

In earlier versions of the SDK, TXLivePlayer works as the only Class carrying both LVB and VOD features. With the expansion of the VOD features, we have made VOD a separate set of features carried by TXVodPlayer starting from SDK 3.5. For the compilation to be successful, VOD features such as seek are still visible in TXLivePlayer.

Interfacing

Step 1: Create a player

The TXLivePlayer module in Tencent Video Cloud SDK is responsible for the LVB playback feature.

```
TXLivePlayer _txLivePlayer = [[TXLivePlayer alloc] init];
```

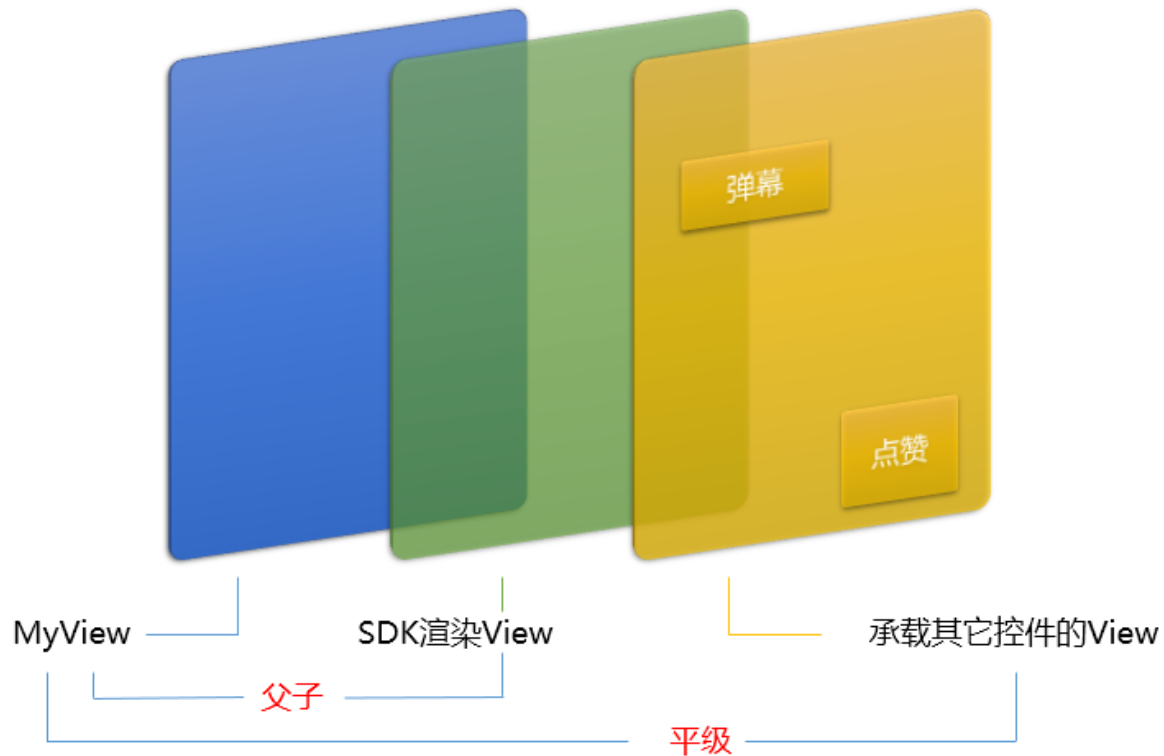
Step 2: Render a view

Next, we need to find a place to display the video images in the player. In iOS system, a view is used as the basic rendering unit. Therefore you simply need to prepare a view and configure the layout.

```
//Use setupVideoWidget to bind the view used to determine render area for the player. The first parameter "frame" has been deprecated since 1.5.2  
[_txLivePlayer setupVideoWidget:CGRectMake(0, 0, 0, 0) containView:_myView insertIndex:0];
```

Technically, the player does not directly render the video image to the view (_myView in the sample code) you provided. Instead, it creates a subView used for OpenGL rendering on top of the view.

You can adjust the size of the rendered image by simply adjusting the size and position of the view. The SDK will automatically adapt the video images to the size and position of the view.



How to make an animation?

You can freely make animations for a view. But note that the target attribute modified for animations is **transform**, instead of frame.

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

Step 3: Start playback

```
NSString* flvUrl = @"http://2157.liveplay.myqcloud.com/live/2157_xxxx.flv";
[_txLivePlayer startPlay:flvUrl type:PLAY_TYPE_LIVE_FLV];
```

Option	Enumerated Value	Description
PLAY_TYPE_LIVE_RTMP	0	The URL passed in is an RTMP-based LVB URL
PLAY_TYPE_LIVE_FLV	1	The URL passed in is an FLV-based LVB URL

Option	Enumerated Value	Description
PLAY_TYPE_LIVE_RTMP_ACC	5	Low-latency URL (only for joint broadcasting scenarios)
PLAY_TYPE_VOD_HLS	3	The URL passed in is an HLS (m3u8)-based playback URL

About HLS (m3u8)

Considering its high latency, HLS is not recommended as the playback protocol for playing LVB videos on your App (although it is suitable for playing VOD videos). Recommended playback protocols include LIVE_FLV and LIVE_RTMP.

Step 4: Adjust the view

- **view: size and position**

You can modify the size and position of the video images by adjusting the size and position of the parameter "view" of `setupVideoWidget`. The SDK will automatically adapt the video images to the size and position of the view.

- **setRenderMode: Full Screen or Self-Adaption**

Option	Description
RENDER_MODE_FILL_SCREEN	The image spread across the entire screen proportionally, with the excess parts cut out. There are no black edges in this mode, but the image may not be displayed completely because of the cut-out areas.
RENDER_MODE_FILL_EDGE	The image is scaled proportionally to adapt to the longest edge. Both the width and the height of the scaled image will not extend beyond the display area and the image is centered. In this mode, black edges maybe appear in the screen.

- **setRenderRotation: Screen rotation**

Option	Description
--------	-------------

Option	Description
RENDER_ROTATION_PORTRAIT	Normal playback (The Home button is located directly below the image)
RENDER_ROTATION_LANDSCAPE	The image rotates 270° clockwise (the Home button is directly to the left of the image)



最长边填充



完全填充



横屏模式

Step 5: Pause playback

Strictly speaking, you cannot pause LVB playback. The so-called pausing LVB playback means **freezing the image** and **turning off the sound**, but the video source keeps updating on the cloud. When you call resume, the video is resumed from the latest time, which works quite differently from VOD videos that are paused and resumed in the same way as local video files).

```
// Pause
[_txLivePlayer pause];
// Resume
[_txLivePlayer resume];
```

Step 6: End playback

To exit the current UI at the end of playback, be sure to terminate the view control using **removeVideoWidget**. Otherwise, memory leak or flickering screen will occur.

```
// Stop playback
[_txLivePlayer stopPlay];
[_txLivePlayer removeVideoWidget]; // Be sure to terminate the view control
```

Step 7: Receive messages

This feature is used to deliver certain custom messages from the pusher end to the viewer end via audio/video lines. It is applicable to the following scenarios:

- (1) Online quiz: The pusher end delivers the **questions** to the viewer end. Perfect "sound-image-question" synchronization can be achieved.
- (2) Live show: The pusher end delivers **lyrics** to the viewer end. The lyric effect can be displayed on the viewer end in real time and its image quality is not affected by video encoding.
- (3) Online education: The pusher end delivers the operations of **Laser pointer** and **Doodle pen** to the viewer end. The drawing can be performed at the viewer end in real time.

You can use this feature as follows:

- Switch the **enableMessage** toggle button in TXLivePlayConfig to **YES**.
- TXLivePlayer listens into messages by **TXLivePlayListener**, message No.: **PLAY_EVT_GET_MESSAGE (2012)**.

```
-(void) onPlayEvent:(int)EvtID withParam:(NSDictionary *)param {
    [self asyncRun:^(
        if (EvtID == PLAY_EVT_GET_MESSAGE) {
            dispatch_async(dispatch_get_main_queue(), ^{
                if ([_delegate respondsToSelector:@selector(onPlayerMessage:)]) {
                    [_delegate onPlayerMessage:param[@"EVT_GET_MSG"]];
                }
            });
        }
    )];
}
```

Step 8: Screenshot

You can capture the current image as a frame by calling **snapshot**. This feature can only capture the frames from the current live stream. To capture the entire UI, call the API of iOS system.



屏幕截图
仅截取视频画面

Step 9: Recode the captured stream

As an extension in LVB playback scenarios, Recording Captured Stream means that during the LVB, the viewer can capture and record a segment of video by clicking the "Record" button and publish the recorded content via the video delivery platform (e.g. Tencent Cloud's VOD system) so that the content can be shared through UGC message on social platforms such as the "Moment" of WeChat.



```
//The following code is used to show how to record videos in LVB scenarios
//
//Specify a TXVideoRecordListener used to synchronize the progress and result of the recording process
_txLivePlayer.recordDelegate = recordListener;
//Start the recording. It can be placed in the response function of the "Record" button. You can only
record the video source, but not the other contents such as the on-screen comments.
[_txLivePlayer startRecord: RECORD_TYPE_STREAM_SOURCE];
// ...
// ...
//End the recording. It can be placed in the response function of the "End" button
[_txLivePlayer stopRecord];
```

- The progress of recording process is indicated as a time value by the onRecordProgress of TXVideoRecordListener.
- The recorded file is in the format of MP4, and is informed by onRecordComplete of TXVideoRecordListener.
- TXUGCPublish is used to upload and publish videos. For more information on how to use TXUGCPublish, please see [Short Video - Publish Files](#).

Adjusting delay

The LVB feature of Tencent Cloud SDK, equipped with the self-developed playback engine, is not developed based on ffmpeg. Compared with open source players, it performs better in terms of LVB delay control. We provide three delay adjusting modes, suitable for Live show, game LVB, and combined scenarios.

- **Performance comparison among the three modes**

Control mode	Stutter rate	Average delay	Applicable scenarios	Principle description
Speedy mode	High (relatively smooth)	2s - 3s	Live show (online quiz)	It has the upper hand in delay control and is suitable for delay-sensitive scenarios.
Smooth mode	Lowest	>= 5s	Game LVB (Penguin e-Sports)	It is suitable for the LVB of ultra-high-bitrate games (such as battle royale games)
Auto mode	Network adaption	2s - 8s	Combined scenarios	The better the viewers' network condition, the lower the latency, and vice versa.

- **Interface codes of the three modes**

```
TXLivePlayConfig* _config = [[TXLivePlayConfig alloc] init];
//Auto mode
_config.bAutoAdjustCacheTime = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 5;
//Speedy mode
_config.bAutoAdjustCacheTime = YES;
_config.minAutoAdjustCacheTime = 1;
_config.maxAutoAdjustCacheTime = 1;
//Smooth mode
_config.bAutoAdjustCacheTime = NO;
_config.cacheTime = 5;

[_txLivePlayer setConfig:_config];

//Launch the playback after the configuration
```


For more technical information on how to deal with stutter and delay problems, please see [How to Deal with Stutter <ECI>](#)

Ultra-low latency playback

Tencent Cloud LVB player supports ultra-low delay playback with a delay of about **400ms**, which can be used in scenarios that have high requirement for delay, such as **remote prize claw** and **joint broadcasting**. Notes about this feature:

- **You don't need to activate this feature**

This feature does not need to be enabled in advance, but it requires that LVB streams reside in Tencent Cloud. Implementing low-delay linkage across cloud providers is difficult, in more than just technical terms.

- **Hotlink protection must be included in the URL**

The playback URL cannot be a normal CDN URL. It must have a hotlink protection signature. For more information on how to calculate the hotlink protection signature, please see [txTime&txSecret](#).

- **Specify the playback type as ACC**

Specify the type as **PLAY_TYPE_LIVE_RTMP_ACC** when calling the startPlay function. The SDK pulls LVB streams using the RTMP-UDP protocol.

- **This feature has restrictions on concurrent playback**

It supports **10 channels** of concurrent playback at most. Instead of being set due to limited technical capabilities, this restriction is intended to encourage you to use this feature in interaction scenarios only (for example, for VJs only in joint broadcasting and for players only in prize claw scenarios), so that you do not incur any unnecessary costs in the mere pursuit of low delay (The price of low latency lines is higher than that of CDN lines).

- **The delay performance of OBS is unsatisfactory**

If the push end is [TXLivePusher](#), set `quality` to MAIN_PUBLISHER or VIDEO_CHAT using [setVideoQuality](#). If the push end is Windows, use [Windows SDK](#). Pushing with OBS leads to unsatisfactory delay due to the excessive accumulated data at the pusher end.

Listening to SDK Events

You can bind a **TXLivePlayListener** to the TXLivePlayer object to receive notifications about the internal status of SDK through onPlayEvent (Event Notification) and onNetStatus (Quality Feedback).

1. Playback events

Event ID	Value	Description
PLAY_EVT_CONNECT_SUCC	2001	Connected to the server
PLAY_EVT_RTMP_STREAM_BEGIN	2002	Connected to the server and started to pull stream (thrown only if the playback address is RTMP)
PLAY_EVT_RCV_FIRST_I_FRAME	2003	The network has received the first renderable video packet (IDR)
PLAY_EVT_PLAY_BEGIN	2004	Video playback begins. The "loading" icon stops flashing at this point
PLAY_EVT_PLAY_LOADING	2007	Video playback is being loaded. If video playback is resumed, this will be followed by a BEGIN event
PLAY_EVT_GET_MESSAGE	2012	Used to receive messages inserted into the audio/video stream. For more information, please see Message Reception

- **Do not hide the playback view after receiving PLAY_LOADING**

The time length between PLAY_LOADING and PLAY_BEGIN can be different (sometimes 5 seconds, sometimes 5 milliseconds). Some customers consider hiding the view upon LOADING and displaying the view upon BEGIN, which will cause serious flickering (especially in LVB scenarios). It is recommended to place a translucent Loading animation on top of the video view.

2. Ending events

Event ID	Value	Description
PLAY_EVT_PLAY_END	2006	Video playback ends
PLAY_ERR_NET_DISCONNECT	-2301	Network is disconnected. Too many failed reconnection attempts. Restart the playback for more retries

- **How do I tell whether the LVB is over?**

Because of the varying implementation principles of different standards, many LVB streams usually don't throw end events (2006) and it is expected that when the VJ stops pushing stream, the SDK will soon find that data stream pull fails (WARNING_RECONNECT) and attempt to retry until the PLAY_ERR_NET_DISCONNECT event is thrown after three failed attempts.

Therefore, you need to listen to both 2006 and -2301 and use the result as the events to determine the end of LVB.

3. Warning events

You don't need to consider the following events. We listed the information of these events for synchronization purposes, according to the SDK white-box design concept

Event ID	Value	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame
PLAY_WARNING_RECONNECT	2103	Network disconnected, automatic reconnection has started (the PLAY_ERR_NET_DISCONNECT event will be thrown after three failed attempts)
PLAY_WARNING_RECV_DATA_LAG	2104	Unstable incoming packet from network: This may be caused by insufficient downstream bandwidth, or unstable outgoing stream at the VJ end
PLAY_WARNING_VIDEO_PLAY_LAG	2105	Stutter occurred during the current video playback
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to start hard-decoding; Soft-decoding is used
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	Current video frames are discontinuous and frame loss may occur
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS resolution failed (thrown only if the playback address is RTMP)
PLAY_WARNING_SEVER_CONN_FAIL	3002	Failed to connect to RTMP server (thrown only if the playback address is RTMP)

Event ID	Value	Description
PLAY_WARNING_SHAKE_FAIL	3003	RTMP server handshaking failed (thrown only if the playback address is RTMP)

Video Width and Height

What is the video resolution (in width and height)?

This question cannot be figured out if SDK only obtains one URL string. To know the width and the height of a video image in pixels, SDK needs to access the cloud server until enough information is loaded to analyze the size of the video image. Therefore, SDK can only tell the video information to your application by notification.

The **onNetStatus** notification is triggered once per second to provide real-time feedback on the current status of the pusher. Like a car dashboard, it can offer you a picture about what is happening inside the SDK, so that you can keep track of current network conditions and video information.

Evaluation parameter	Description
NET_STATUS_CPU_USAGE	Current CPU utilization (instantaneous)
NET_STATUS_VIDEO_WIDTH	Video resolution - Width
NET_STATUS_VIDEO_HEIGHT	Video resolution - Height
NET_STATUS_NET_SPEED	Current speed at which network data is received
NET_STATUS_NET_JITTER	Network jitter status. A bigger jitter means a more unstable network
NET_STATUS_VIDEO_FPS	The video frame rate of the current stream media
NET_STATUS_VIDEO_BITRATE	Video bitrate of the current stream media (in Kbps)
NET_STATUS_AUDIO_BITRATE	Audio bitrate of the current stream media (in Kbps)
NET_STATUS_CACHE_SIZE	Buffer size (jitterbuffer). A buffer length of 0 means that stutter will occur in all probability
NET_STATUS_SERVER_IP	IP of the connected server

TXVodPlayer

Last updated : 2018-08-10 16:20:59

Basics

This document describes the VOD playback feature of Tencent Video Cloud SDK. The following are the basics you must learn before getting started.

- **LVB and VOD**

The video source of [LVB \(LIVE\)](#) is pushed by VJ in real time. When the VJ stops broadcasting, the video image on the playback device stops. In addition, the video is broadcasted in real time, no progress bar is displayed when the player is playing the LVB URL.

The video source of [Video On-demand \(VOD\)](#) is a video file on cloud, which can be played at any time as long as it has not been deleted from the cloud. You can control the playback progress using the progress bar. The video playbacks on Tencent Video and Youku Tudou are typical VOD scenarios.

- **Supported Protocols**

The following are the commonly used VOD protocols. Now, HLS-based VOD URLs are most popular (starting with "http" and ending with ".m3u8"):

点播协议	优点	缺点
HLS(m3u8)	手机浏览器支持度高	大量小分片的文件组织形式，错误率和维护成本均高于单一文件
MP4	手机浏览器支持度高	格式过于复杂和娇贵，容错性很差，对播放器的要求很高
FLV	格式简单问题少，适合直播转录制场景	手机浏览器支持差，需集成SDK才能播放

Notes

Tencent Cloud SDK **does not** impose any restrictions on the source of playback URLs, which means you can use the SDK to play videos from both Tencent Cloud and non-Tencent Cloud addresses. But the player in Tencent Video Cloud SDK only supports three LVB video address formats (FLV, RTMP and HLS (m3u8)) and three VOD address formats (MP4, HLS (m3u8) and FLV).

Interfacing

Step 1: Create a player

The TXVodPlayer module in Tencent Video Cloud SDK is responsible for the VOD playback feature.

```
TXVodPlayer *_txVodPlayer = [[TXVodPlayer alloc] init];  
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

Step 2: Render a view

Next, we need to find a place to display the video images in the player. In iOS system, a view is used as the basic rendering unit. Therefore you simply need to prepare a view and configure the layout.

```
[_txVodPlayer setupVideoWidget:_myView insertIndex:0]
```

Technically, the player does not directly render the video image to the view (_myView in the sample code) you provided. Instead, it creates a subView used for OpenGL rendering on top of the view.

You can adjust the size of the rendered image by simply adjusting the size and position of the view. The SDK will automatically adapt the video images to the size and position of the view.



How to make an animation?

You can freely make animations for a view. But note that the target attribute modified for animations is **transform**, instead of frame.

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

Step 3: Start playback

TXVodPlayer supports two playback modes from which you may choose.

1. By URL

TXVodPlayer can automatically recognize the playback protocol internally. You only need to pass your playback URL to the startPlay function.

```
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startPlay:url];
```

2. By fileID

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788;
p.fileId = @"4564972819220421305";
[_txVodPlayer startPlayWithParams:p];
```

Find the file in [VOD Video Management](#). The appId and fileId are shown in the video details on the right of the page that opens.

The screenshot displays the Tencent Cloud VOD Video Management console. On the left, a sidebar contains navigation options like '点播', '服务概览', 'Web播放器管理', '视频拉取', 'Web上传', '统计分析', '全局设置', '水印管理', '归档存储', and '腾讯视频V+'. The main area is titled '云视频管理' and features a table of videos. A red box highlights a video entry with the title '点播视频上传', a video ID 'd01da476631493b87f3abc793...', and a duration of 00:03:25. A red arrow points from this video ID to the '点播视频上传' modal window on the right. The modal window shows the video details, including the video ID 'd01da476631493b87f3abc793...' and the APP ID '1252463788'.

By using fileID for playback, the player sends request to the backend for the real playback address. You will receive `PLAY_ERR_GET_PLAYINFO_FAIL` event if the network is exceptional or fileID does not exist. Otherwise, the request is successful and you will receive `PLAY_EVT_GET_PLAYINFO_SUCC`.

Step 4: Adjust the view

- **view: size and position**

You can modify the size and position of the video images by adjusting the size and position of the parameter "view" of `setupVideoWidget`. The SDK will automatically adapt the video images to the size and position of the view.

- **setRenderMode: Full Screen or Self-Adaption**

Option	Description
<code>RENDER_MODE_FILL_SCREEN</code>	The image spread across the entire screen proportionally, with the excess parts cut out. There are no black edges in this mode, but the image may not be displayed completely because of the cut-out areas.
<code>RENDER_MODE_FILL_EDGE</code>	The image is scaled proportionally to adapt to the longest edge. Both the width and the height of the scaled image will not extend beyond the display area and the image is centered. In this mode, black edges may appear in the screen.

- **setRenderRotation: Screen rotation**

Option	Description
<code>HOME_ORIENTATION_RIGHT</code>	Home button on the right
<code>HOME_ORIENTATION_DOWN</code>	Home button at the bottom
<code>HOME_ORIENTATION_LEFT</code>	Home button on the left
<code>HOME_ORIENTATION_UP</code>	Home button at the top



最长边填充



完全填充



横屏模式

Step 5: Control the playback

```
// Adjusts progress
[_txVodPlayer seek:slider.value];
// Pauses playback
[_txVodPlayer pause];
// Resumes playback
[_txVodPlayer resume];
```

Step 6: End playback

To exit the current UI at the end of playback, be sure to terminate the view control using **removeVideoWidget**. Otherwise, memory leak or flickering screen will occur.

```
// Stops playback
[_txVodPlayer stopPlay];
[_txVodPlayer removeVideoWidget]; // Be sure to terminate the view control
```

Step 7: Screenshot

You can capture the current image as a frame by calling **snapshot**. This feature can only capture the frames from the current live stream. To capture the entire UI, call the API of iOS system.



屏幕截图

仅截取视频画面

Step 8: Control the playback speed

The VOD player supports playback speed control. You can set the VOD playback speed, such as 0.5X, 1.0X, 1.2X, 2X, using the API `setRate` to speed up or slow down the playback.



变速播放

支持加速和慢播

```
//Set the speed to 1.2X
[_txVodPlayer setRate:1.2];
// ...
```

```
//Start the playback  
[_txVodPlayer startPlay:url];
```

Step 9: Local Cache [UGC version not supported]

In a short video playback scenario, the local caching of video files is a required feature. For viewers, replaying a video should not consume traffic.

- **Supported Formats**

SDK supports caching for files in HLS (m3u8) and MP4 formats.

- **When do I enable the feature?**

By default, caching feature is disabled in the SDK. It is not recommended to enable this feature for scenarios with low replay rates.

- **How do I enable the feature?**

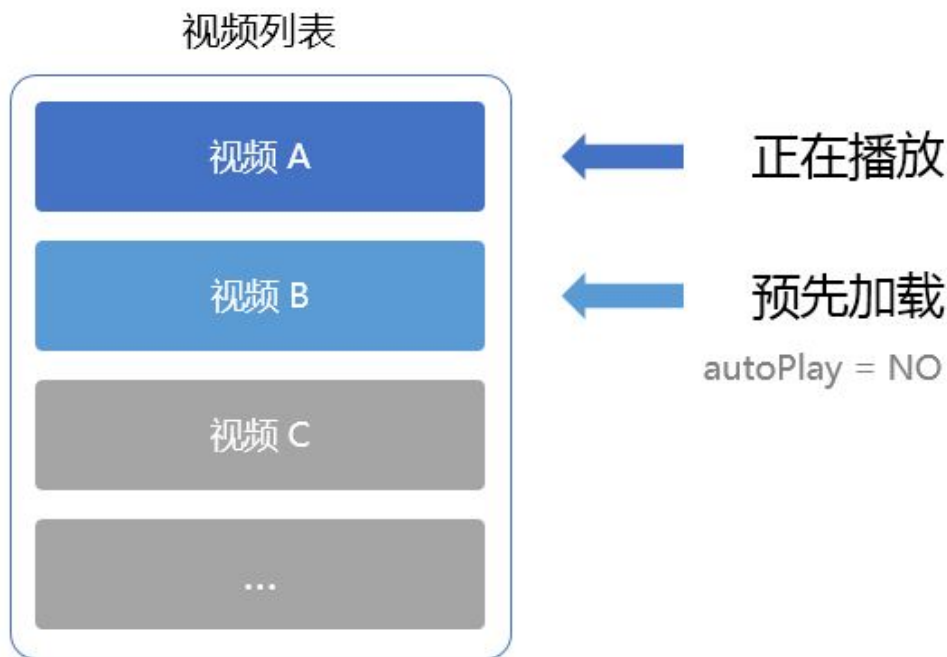
To enable this feature, you need to configure two parameters: local cache directory and the number of videos to be cached.

```
TXVodPlayConfig _config = [[TXVodPlayConfig alloc] init];  
  
// Set the cache directory  
_config.cacheFolderPath =  
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];  
  
// Set the maximum number of cached files to avoid caching too much data.  
_config.maxCacheItems = 10;  
  
[_txVodPlayer setConfig: _config];  
// ...  
//Start the playback  
[_txVodPlayer startPlay:playUrl];
```

Step 10: Preloading

In the short video playback scenario, pre-loading helps to ensure a smooth viewing experience. The URL of the next video is loaded at the background during the playback of the current video so that users can play the next video immediately when switching to it, without needing to load it from the start.

This feature allows the seamless switching in VOD, and can be enabled using the `isAutoPlay` of `TXVodPlayer` by performing the following operations:



// Play video A: If `isAutoPlay` is **set to YES**, the video **is loaded and** played immediately **when** `startPlay` **is** called.

```
NSString* url_A = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
_player_A.isAutoPlay = YES;
[_player_A startPlay:url_A];
```

// To **pre-load** video B **while** playing video A, **set** `isAutoPlay` **to** NO.

```
NSString* url_B = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
_player_B.isAutoPlay = NO;
[_player_B startPlay:url_B];
```

When video A ends and is automatically or manually switched to video B, call the "resume" function to play video B immediately.

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*)param
{
    // At the end of playback of video A, directly start the playback of video B for a seamless switching
    if (EvtID == PLAY_EVT_PLAY_END) {
        [_player_A stopPlay];
        [_player_B setupVideoWidget:mVideoContainer insertIndex:0];
        [_player_B resume];
    }
}
```

```
}  
}
```

Step 11: Pre-roll ads

autoPlay can also be used for pre-roll ads. If autoPlay is set to NO, the player loads the video immediately but does not play it immediately. Pre-roll ads can be presented during the delay between the loading and playback. When the ads end, the video is played using resume function.

Step 12: Encrypted Playback [UGC version not supported]

Video encryption is mainly used for online education and other scenarios in need of video copyright protection. To encrypt video resources, you not only need to make changes on the player, but also to encrypt and transcode the source video. The engagement of backend and terminal R&D engineers is also required. For more information, please see [Video Encryption Solution](#).

TXVodPlayer also supports encrypted playback. You can use the solution where identity verification information is carried in [URL](#). With this solution, you can call the SDK as you would for any other scenario. You can also use the solution where identity verification information is carried in [Cookie](#). With this solution, you need to configure the cookie information in the HTTP request head using the headers field of TXVodPlayConfig.

Step 13: HTTP-REF [UGC version not supported]

The headers in TXVodPlayConfig can be used to set HTTP request headers, such as the Referer field commonly used to prevent URLs from being copied (Tencent Cloud can provide a more secure hotlink protection signature solution), and the cookie field used to verify client identity information.

Step 14: Hardware acceleration

For blu-ray (1080p) quality, it is difficult to obtain smooth playback experience just by using software decoding. Therefore, if your scenario focuses on gaming LVB, it is recommended to enable hardware acceleration.

It is strongly recommended to perform **stopPlay** before the switching between software decoding and hardware decoding, and perform **startPlay** after switching. Otherwise serious blurred screen problems may occur.

```
[_txVodPlayer stopPlay];  
_txVodPlayer.enableHWAceleration = YES;  
[_txVodPlayer startPlay:_flvUrl type:_type];
```

Step 15: Multi-bitrate File [UGC version not supported]

The SDK supports hls in multiple bitrates, allowing users to switch between streams in different bitrates. After receiving the `PLAY_EVT_PLAY_BEGIN` event, you can obtain the array of supported bitrates using the following method:

```
NSArray *bitrates = [_txVodPlayer supportedBitrates]; //Obtain the array of supported bitrates
```

You can switch between different bitrates using `-[TXVodPlayer setBitrateIndex:]` at any time during the playback. Another stream of data will be pulled during the switch, which may lead to minor stuttering. The SDK has been optimized for Tencent Cloud's multi-bitrate files, which enables the switch between bitrates without any stutter.

Progress Display

VOD progress is indicated in two metrics: the **loading progress** and **playback progress**. Now, SDK uses event notification to notify the two metrics.

You can bind a **TXVodPlayerListener** listener to the `TXVodPlayer` object. Then the progress notification calls back your application via the **PLAY_EVT_PLAY_PROGRESS** event whose additional information contains the two progress metrics above.




```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*)param {
    if (EvtID == PLAY_EVT_PLAY_PROGRESS) {
        // Loading progress, in seconds, milliseconds for digits after decimal point
        float playable = [param[EVT_PLAYABLE_DURATION] floatValue];
        [_loadProgressBar setValue:playable];

        // Playback progress, in seconds, milliseconds for digits after decimal point
        float progress = [param[EVT_PLAY_PROGRESS] floatValue];
        [_seekProgressBar setValue:progress];

        // Video duration, in seconds, milliseconds for digits after decimal point
        float duration = [param[EVT_PLAY_DURATION] floatValue];
        // Used to set duration display, etc.
    }
}
```

Event Listening

Besides PROGRESS information, SDK synchronizes much other information for your applications through onPlayEvent (event notification) and onNetStatus (status feedback):

1. Playback events

Event ID	Value	Description
PLAY_EVT_PLAY_BEGIN	2004	Video playing begins. The "loading" icon stops flashing at this point.
PLAY_EVT_PLAY_PROGRESS	2005	This refers to the progress of video playback, including current playback progress, loading progress and overall duration.
PLAY_EVT_PLAY_LOADING	2007	The video is being loaded. If video playback is resumed, this will be followed by a BEGIN event.

2. Ending events

Event ID	Value	Description
PLAY_EVT_PLAY_END	2006	Video playback ended

Event ID	Value	Description
PLAY_ERR_NET_DISCONNECT	-2301	Network is disconnected. Too many failed reconnection attempts. Restart the playback for more retries
PLAY_ERR_HLS_KEY	-2305	Failed to get the HLS decoding key

3. Warning events

You can ignore the following events. They are only used to tell you the internal SDK events.

Event ID	Value	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame
PLAY_WARNING_RECONNECT	2103	Network disconnected and auto reconnection has started (the PLAY_ERR_NET_DISCONNECT event will be thrown after three failed attempts)
PLAY_WARNING_RECV_DATA_LAG	2104	Unstable inbound packet: This may be caused by insufficient downstream bandwidth, or inconsistent outbound stream from the VJ end.
PLAY_WARNING_VIDEO_PLAY_LAG	2105	Stutter occurred during the video playback
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to start hard decoding. Soft decoding is used instead.
PLAY_WARNING_VIDEO_DISCONTINUITY	2107	Discontinuous sequence of video frames. Some frames may be dropped.
PLAY_WARNING_DNS_FAIL	3001	RTMP-DNS resolution failed (thrown only if the playback address is RTMP)
PLAY_WARNING_SEVER_CONN_FAIL	3002	Failed to connect to the RTMP server (thrown only if the playback address is RTMP)
PLAY_WARNING_SHAKE_FAIL	3003	Handshake with the RTMP server failed (thrown only if the playback address is RTMP)

4. Connection events

In addition, there are several server connection events used to measure and calculate the time for server connections. Also, you don't have to be concerned:

Event ID	Value	Description
PLAY_EVT_CONNECT_SUCC	2001	Connected to the server
PLAY_EVT_RTMP_STREAM_BEGIN	2002	Connected to the server. Pull started. (thrown only if the playback address is RTMP)
PLAY_EVT_RCV_FIRST_I_FRAME	2003	The network has received the first renderable video packet (IDR)

5. Resolution events

The following events are used to obtain image change information. You don't have to be concerned about them.

Event ID	Value	Description
PLAY_EVT_CHANGE_RESOLUTION	2009	Video resolution changed
PLAY_EVT_CHANGE_ROTATION	2011	MP4 video rotation angle

Video Width and Height

What is the video resolution (in width and height)?

This question cannot be figured out if SDK only obtains one URL string. To know the width and the height of a video image in pixels, SDK needs to access the cloud server until enough information is loaded to analyze the size of the video image. Therefore, SDK can only tell the video information to your application by notification.

The **onNetStatus** notification is triggered once per second to provide real-time feedback on the current status of the pusher. Like a car dashboard, it can offer you a picture about what is happening inside the SDK, so that you can keep track of current network conditions and video information.

Evaluation parameter	Description
NET_STATUS_CPU_USAGE	Current (instant) CPU utilization
NET_STATUS_VIDEO_WIDTH	Video resolution - width
NET_STATUS_VIDEO_HEIGHT	Video resolution - height

Evaluation parameter	Description
NET_STATUS_NET_SPEED	Current speed at which network data is received
NET_STATUS_VIDEO_FPS	Video frame rate of the current stream media
NET_STATUS_VIDEO_BITRATE	Video bitrate of the current stream media (in Kbps)
NET_STATUS_AUDIO_BITRATE	Audio bitrate of the current stream media (in Kbps)
NET_STATUS_CACHE_SIZE	Buffer size (jitterbuffer). A buffer with the length of 0 means that stutter will occur.
NET_STATUS_SERVER_IP	IP of the connected server

You can call `-[TXVodPlayer width]` and `-[TXVodPlayer height]` to obtain the current width and height.

Video information

If the video is played via fileID and the request is successful, SDK will inform the upper layer of the request information. You need to resolve the information in param after receiving the `PLAY_EVT_GET_PLAYINFO_SUCC` event.

Video information	Description
EVT_PLAY_COVER_URL	Video cover URL
EVT_PLAY_URL	Video playback URL
EVT_PLAY_DURATION	Video duration

Offline Download

Offline VOD playback is a commonly needed feature. Users can download videos where there is network connection, and replay the videos where no network connection is available. The SDK supports playing local files, but this is limited to single file formats of mp4 and flv. HLS stream media files cannot be played locally, because they cannot be saved locally. To play HLS offline, you can download HLS locally using `TXVodDownloadManager`.

Step 1: Preparations

`TXVodDownloadManager` is designed as a singleton. Therefore, you cannot create multiple download objects. The following describes how it is used:

```
TXVodDownloadManager *downloader = [TXVodDownloadManager sharedInstance];  
[downloader setDownloadPath:@"<Specify a download directory> "];
```

Step 2: Start download

Start download by either url or fileID. To download via url, simply pass in the download url.

```
[downloader startDownloadUrl:@"http://1253131631.vod2.myqcloud.com/26f327f9vodgzp1253131631/f4bdf799031868222924043041/playlist.m3u8"]
```

To download via fileID, you need to pass in appID and fileID at least.

```
TXPlayerAuthParams *auth = [TXPlayerAuthParams new];  
auth.appId = 1252463788;  
auth.fileId = @"4564972819220421305";  
TXVodDownloadDataSource *dataSource = [TXVodDownloadDataSource new];  
dataSource.auth = auth;  
[downloader startDownload:dataSource];
```

See <https://cloud.tencent.com/document/product/454/12147#step-3.3A-E5.90.AF.E5.8A.A8.E6.92.AD.E6.94.BE> for how to obtain fileID

Step 3: Query task information

You need to configure callback delegate before receiving task information.

```
downloader.delegate = self;
```

Possible task callback results are as follows:

1. `-[TXVodDownloadDelegate onDownloadStart:]`

The task is started, indicating that the SDK has started downloading the video.

2. `-[TXVodDownloadDelegate onDownloadProgress:]`

Task progress. The SDK calls this API frequently during the download. You can update progress display here.

3. `-[TXVodDownloadDelegate onDownloadStop:]`

The task is stopped. After you call `stopDownload` to stop the download, this message indicates that

the task has been stopped successfully.

4. `-[TXVodDownloadDelegate onDownloadFinish:]`

The download is finished, indicating that the video is downloaded completely. The downloaded file can be played by `TXVodPlayer`.

5. `-[TXVodDownloadDelegate onDownloadError:errorMsg:]`

Download error. This API is called back when the network is disconnected during the download. The download is stopped at the same time. See `TXDownloadError` for a full list of error codes.

The callback API contains the `TXVodDownloadMediaInfo` object, because downloader can execute multiple download tasks simultaneously. You can access `url` or `dataSource` to determine the download source, and to obtain information such as download progress and file size.

Step 4: Stop download

To stop the download, call the `-[TXVodDownloadManager stopDownload:]` method, with the object returned via `-[TXVodDownloadManager startDownloadUrl:]` as the parameter.** The SDK supports resuming download from breakpoint. If the download directory is not changed, you can download the file again from the point where the download is suspended.

If you do not need to resume the download, call the `-[TXVodDownloadManager deleteDownloadFile:]` method to delete the file and free the storage.

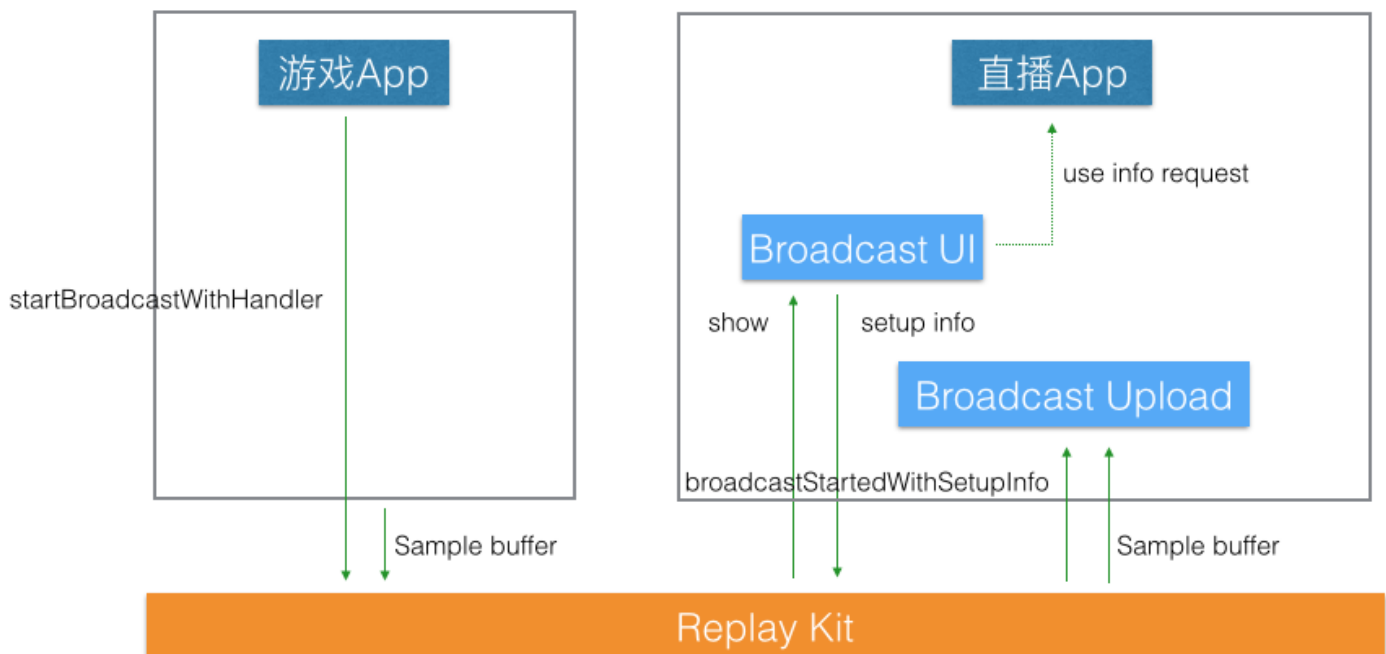
Screen Recording (ReplayKit)

Last updated : 2018-08-10 16:21:04

Overview

Screencap is a feature newly introduced in iOS 10. Based on the ReplayKit screencap video save feature on iOS 9, Apple added real-time video stream LVB feature. For official introduction, see [Go Live with ReplayKit](#).

The screencap process is divided into two sections: Game App and LVB App. When screen capping on iOS, the system does not directly run the LVB App. Instead, the LVB App provides service for the game App in the form of an extension. There are two extensions. One is used to display custom interface, which displays user information and allows the user to enter titles and so on. When the user taps on the OK button, the system switches to the other extension to send screen data. This extension cannot display UI. The structure of the entire screencap LVB process is shown below.



As an extension, Broadcast Upload has its own process. To ensure system performance, iOS systems allocate less resource to extensions, and extensions that occupy too much memory will be killed. Based on

the high quality, low latency of the original LVB feature, Tencent Cloud RTMP SDK further reduced system resource usage to ensure extension stability.

Try out the Feature

The recording process can only be completed when screencap is supported by both the game and LVB software. For the LVB software, it is recommended that you use our "Mini LVB". Download link



As iOS 10 becomes widely used, the number of games that support screencap is increasing as well. If you have no games that support screencap, you can download the game called "War of Tanks", touch the LVB option above and select "Mini LVB" to start screencap.



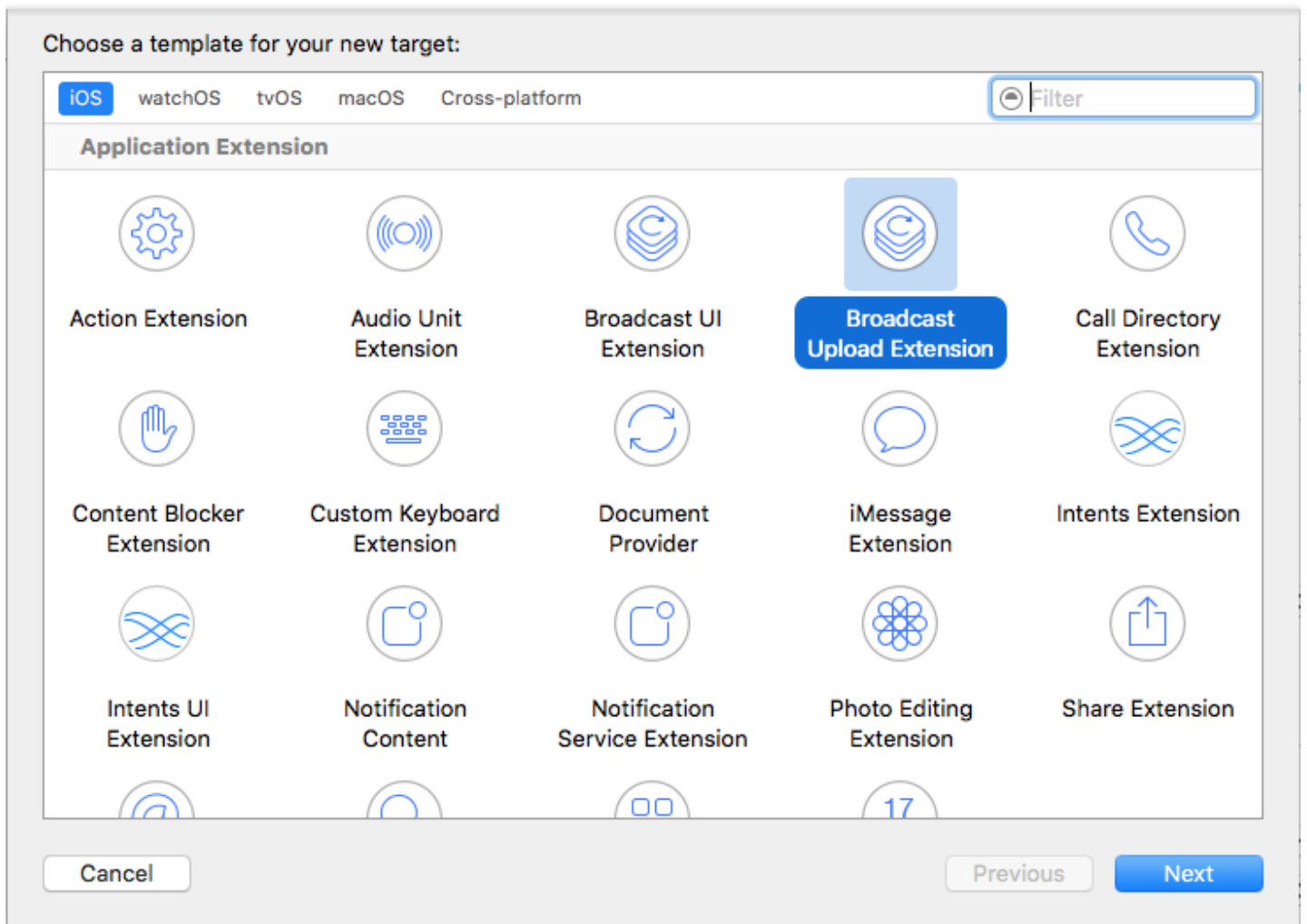
Development Environment Preparation

Xcode Preparation

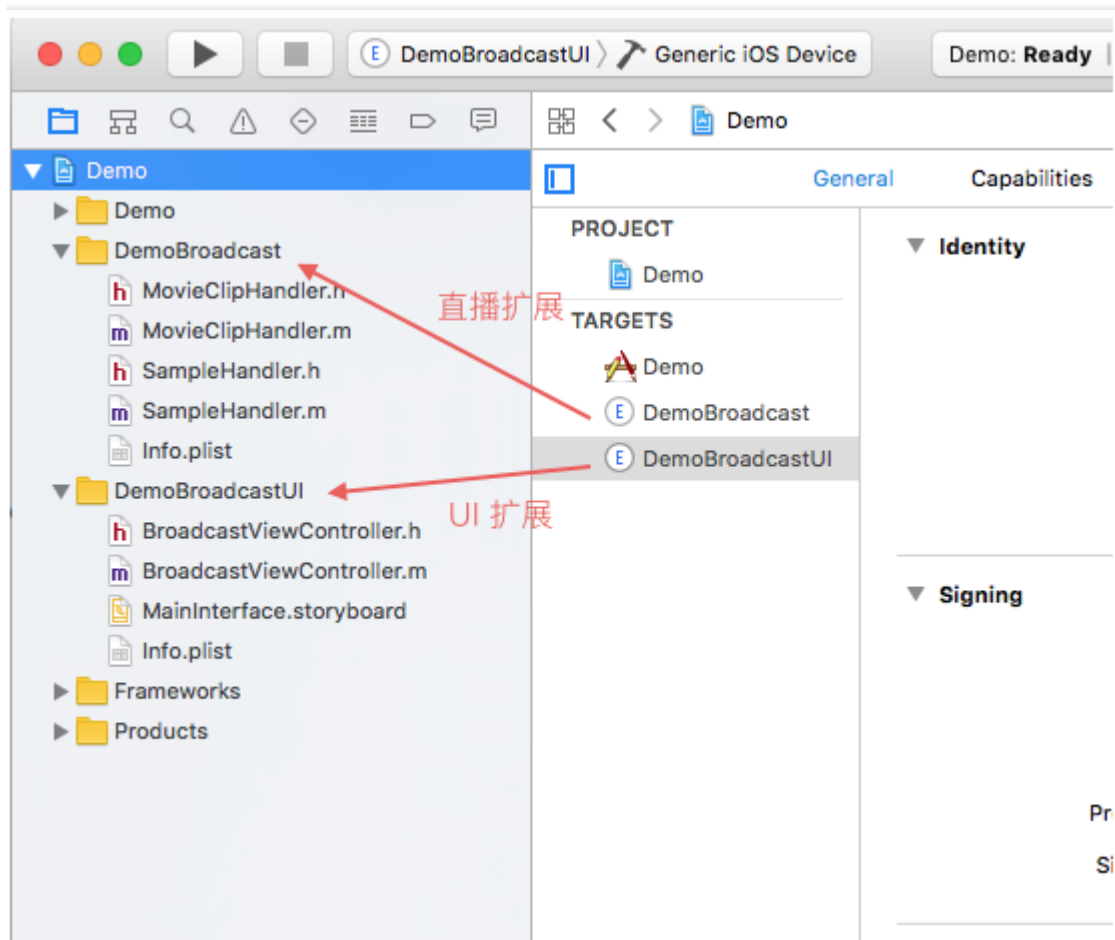
Screencap LVB is a new feature provided by iOS 10, so you'll need Xcode 8 or higher version, and the mobile phone must be upgraded to iOS 10 or above. Screencap is not supported on a simulator.

Create LVB Extension

In the current project, select "New" -> "Target...", and then "Broadcast Upload Extension", as shown in the figure



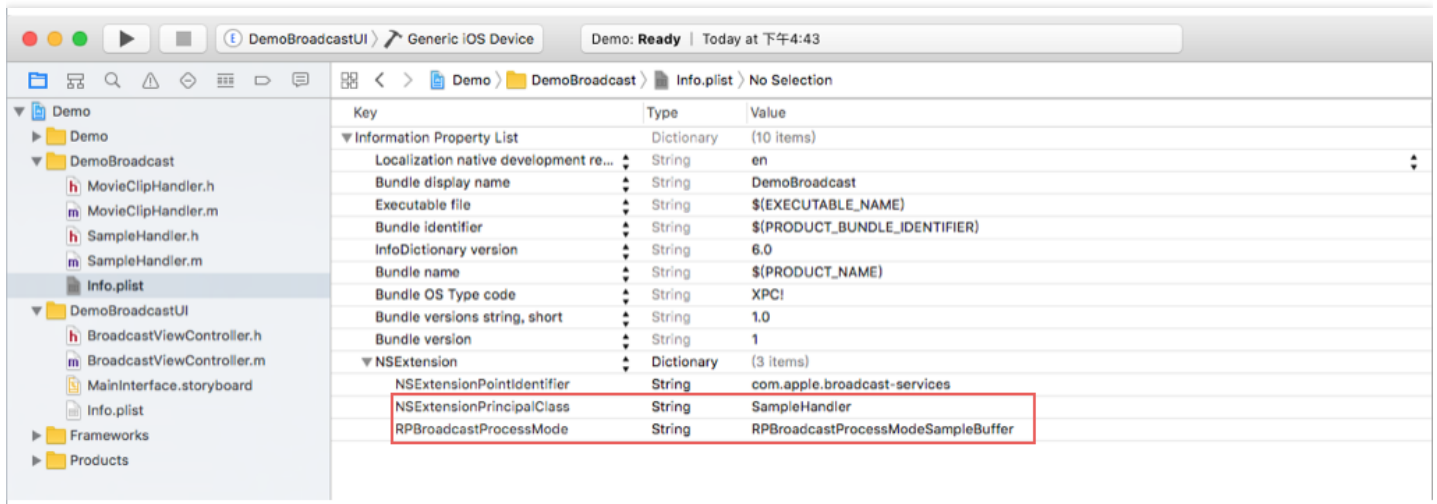
Configure Product Name. Remember to check "Include UI Extension". Click "Finish" and you will see two more directories and two more targets, LVB extension and UI extension, added to the project.



The Replay Kit of iOS 10 supports two LVB modes

1. The video and audio are encoded into a short mp4 file and handed over to the LVB extension
2. The original screen and audio data is handed over to the extension

The first mode has high latency and poor flexibility but the extension App doesn't have to be concerned with encoding issues; while the second mode allows you to customize what you send and has high configurability. Currently, the SDK only supports the second mode. Since Xcode uses the first mode by default, you need to modify the Info.plist of the LVB extension as shown in the figure



Import RTMP SDK

You need to import TXRTMPSDK.framework for the LVB extension. Importing framework to extension is the same as importing framework to main App, and dependent system libraries for the SDK are also the same. For more information, please see "[Project Configuration \(iOS\)](#)" on the official Tencent Cloud website.

Interfacing Process

Step 1: Write UI Extension

When the game App initializes LVB process, we first enter the UI extension. Here, you can customize the interface according to your product demand. If you are required to log in to the LVB software, you'd better first check the login status here, because interfaces cannot be displayed during LVB.

When the user confirms to initialize LVB, the UI extension can launch the LVB extension and attach certain custom parameters. Sample code for launching LVB extension is provided below

```
// Called when the user has finished interacting with the view controller and a broadcast stream can start
- (void)userDidFinishSetup {

// Broadcast url that will be returned to the application
NSURL *broadcastURL = [NSURL URLWithString:@"http://broadcastURL_example/stream1"];

// Service specific broadcast data example which will be supplied to the process extension during broadcast
NSString *userID = @"user1";
```

```

NSString *endpointURL = @"rtmp://2157.livepush.myqcloud.com/live/xxxxxx";
NSDictionary *setupInfo = @{ @"userID" : userID, @"endpointURL" : endpointURL };

// Set broadcast settings
RPBroadcastConfiguration *broadcastConfig = [[RPBroadcastConfiguration alloc] init];
broadcastConfig.clipDuration = 5.0; // deliver movie clips every 5 seconds

// Tell ReplayKit that the extension is finished setting up and can begin broadcasting
[self.extensionContext completeRequestWithBroadcastURL:broadcastURL
broadcastConfiguration:broadcastConfig setupInfo:setupInfo];
}

```

Step 2: Create Push Object

The project template has already provided a basic framework for the LVB extension. You simply need to add the follow code before SampleHandler.m

```

#import "SampleHandler.h"
#import "TXRTMPSDK/TXLiveSDKTypeDef.h"
#import "TXRTMPSDK/TXLivePush.h"
#import "TXRTMPSDK/TXLiveBase.h"
static TXLivePush *s_txLivePublisher;

```

s_txLivePublisher is the object we use for the push. The best location for instantiating s_txLivePublisher is in the `-[SampleHandler broadcastStartedWithSetupInfo:]` method. After starting the pusher, the UI extension will call back this function and start LVB process.

```

- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupInfo {
    if (s_txLivePublisher) {
        [s_txLivePublisher stopPush]; // End the previous push before starting the next one
    }

    TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
    config.customModeType |= CUSTOM_MODE_VIDEO_CAPTURE;
    config.autoSampleBufferSize = YES;

    config.customModeType |= CUSTOM_MODE_AUDIO_CAPTURE;
    config.audioSampleRate = 44100;
    config.audioChannels = 1;

    s_txLivePublisher = [[TXLivePush alloc] initWithConfig:config];
    NSString *pushUrl = setupInfo[@"endpointURL"]; // setupInfo is from the UI extension
    [s_txLivePublisher startPush:pushUrl];
}

```

You cannot use default configuration for the "config" of `s_txLivePublisher`. You need to customize video and audio capture configuration. For more information on how to set custom capture and how it works, please see the "[RTMP Push - Advanced Operation](#)" section in Tencent Cloud documentation.

Enable `autoSampleBufferSize` for videos. Once this option is enabled, you will not need to worry about the resolution of the push, and the SDK will set the encoder automatically based on the input resolution. If you disable this option, you will need to customize the resolution

It is recommended to enable `autoSampleBufferSize` to achieve optimal performance. This also eliminates your need to select landscape or portrait mode.

Step 3: Customize Resolution

You can specify any resolution if you don't want to use the output resolution of the screen. The SDK will adjust the video size based on the resolution you specified

```
// Specify 640*360
config.autoSampleBufferSize = NO;
config.sampleBufferSize = CGSizeMake(640, 360);
```

Step 4: Send Video

Replay Kit transfers both the audio and video to `-[SampleHandler processSampleBuffer:withType]` though callback

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sam
pleBufferType {
    switch (sampleBufferType) {
        case RPSampleBufferTypeVideo:
            // Handle audio sample buffer
            {
                [s_txLivePublisher sendVideoSampleBuffer:sampleBuffer];
            }
            return;
        }
    }
```

Video `sampleBuffer` can be sent simply by calling the `-[TXLivePush sendVideoSampleBuffer:]`.

The distribution frequency of `sampleBuffer` by the system is not fixed. If the screen remains static, it will probably take a long time before a frame of data is sent. Given this situation, the SDK implements the frame interpolation logic internally to reach the frame rate (20 fps by default) set in config.

Step 5: Send Audio

Audio is also sent to LVB extension through `-[SampleHandler processSampleBuffer:withType]`. The difference is that there are two channels of data for audio. One channel comes from inside the App, and the other comes from the microphone.

```
switch (sampleBufferType) {
case RPSampleBufferTypeAudioApp:
    // Audio from inside the App

break;
case RPSampleBufferTypeAudioMic:
    // Audio from Mic.
    {
        // Send the audio data from Mic
        [s_txLivePublisher sendAudioSampleBuffer:sampleBuffer];
    }
break;
}
```

Sending two channels of data at the same time is not supported by the SDK. You need to select which audio to use as needed.

Step 6: Pause and Resume

The current LVB can be paused in game Apps, which will prevent Samples buffer from being distributed to the LVB extension, till the user resumes the LVB. In custom capture mode, the SDK requires continuous external data source; otherwise the server will disconnect the LVB due to not receiving data for a long time.

The frame interpolation logic for video is provided in the SDK. The last frame of data will be resent when there is no video. However, no frame interpolation logic is provided for audio, video and audio will go out of sync if no data is provided for SDK. You can use the following simple code to send mute data to the SDK during a pause.

```
static dispatch_source_t s_audioTimer;

- (void)pause {
    if (s_audioTimer) {
        return;
    }

    dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
    s_audioTimer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0, queue);
}
```

```
dispatch_source_set_timer(s_audioTimer,DISPATCH_TIME_NOW,20*NSEC_PER_MSEC, 0);
dispatch_source_set_event_handler(s_audioTimer, ^{
static uint8_t _audioData[2048] = {0};
[s_txLivePublisher sendCustomPCMDData:_audioData len:sizeof(_audioData)];
});
dispatch_resume(s_audioTimer);
}

- (void)resume {
if (s_audioTimer) {
dispatch_cancel(s_audioTimer);
s_audioTimer = 0;
}
}

- (void)broadcastPaused {
// User has requested to pause the broadcast. Samples will stop being delivered.
[self pause];
}

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

Step 7: SDK Event Handling

Event Listening

You need to configure the "delegate" attribute of `TXLivePush` for SDK event listening. This delegate follows `TXLivePushListener` protocol. Underlying events will be called back through the `-(void)onPushEvent:(int)EvtID withParam:(NSDictionary*)param` API.

Due to system restrictions, LVB extension cannot trigger interface action, and thus cannot actively inform the user of a push exception. Usually, the following events will be received during screencap.

Normal Events

Event ID	Value	Description
PUSH_EVT_CONNECT_SUCC	1001	Successfully connected to Tencent Cloud push server
PUSH_EVT_PUSH_BEGIN	1002	Handshake with the server completed, everything is OK, ready to start push

Usually no action is needed for normal events.

Error Events

Event ID	Value	Description
PUSH_ERR_VIDEO_ENCODE_FAIL	-1303	Video encoding failed
PUSH_ERR_AUDIO_ENCODE_FAIL	-1304	Audio encoding failed
PUSH_ERR_UNSUPPORTED_RESOLUTION	-1305	Unsupported video resolution
PUSH_ERR_UNSUPPORTED_SAMPLERATE	-1306	Unsupported audio sampling rate
PUSH_ERR_NET_DISCONNECT	-1307	Network disconnected. Reconnection attempts have failed for three times, thus no more retries will be performed. Please restart the push manually

Video encoding failure does not affect push process directly. The SDK will handle it to ensure success of the subsequent video encoding.

Warning Events

Event ID	Value	Description
PUSH_WARNING_NET_BUSY	1101	Bad network condition: data upload is blocked because uplink bandwidth is too small
PUSH_WARNING_RECONNECT	1102	Network disconnected, automatic reconnection has started (auto reconnection will be stopped if it fails for three times)
PUSH_WARNING_HW_ACCELERATION_FAIL	1103	Failed to start hardware encoding. Software encoding is used
PUSH_WARNING_DNS_FAIL	3001	RTMP - DNS resolution failed (this will trigger retry process)
PUSH_WARNING_SEVER_CONN_FAIL	3002	Failed to connect to the RTMP server (this will trigger retry process)
PUSH_WARNING_SHAKE_FAIL	3003	RTMP server handshake failed (this will trigger retry process)

Event ID	Value	Description
PUSH_WARNING_SERVER_DISCONNECT	3004	The RTMP server actively disconnected (this will trigger retry process)

A warning event indicates that the server has encountered some internal problems, but they will not affect the push.

For the definition of all events, see the header file: **"TXLiveSDKEventDef.h"**

Step 8: End Push

To end the push, Replay Kit will call the `-[SampleHandler broadcastFinished]`. Sample code is provided below

```
- (void)broadcastFinished {  
    // User has requested to finish the broadcast.  
    if (s_txLivePublisher) {  
        [s_txLivePublisher stopPush];  
        s_txLivePublisher = nil;  
    }  
}
```

The LVB extension process may be recovered by the system after a push, make sure to perform proper cleanup work.

Effect Feature

Last updated : 2018-08-10 16:21:08

Special Effects (Eye Enlarging, Face Slimming, Dynamic Effect, Green Screen)

Feature Description

Eye enlarging, face slimming, dynamic sticker, green screen and other special effects are licensed features developed based on the face recognizing technology of YouTu Lab and the beautifying technology of Pitu. By working with YouTu Lab and Pitu, Tencent Cloud Mini LVB integrates these special effect features into the image processing process of RTMP SDK to achieve better video effects.

Integration Procedure

Application procedure:

1. Submit a ticket or contact one of our customer service representatives by calling 400-9100-100.
2. Download the [sample form](#), fill in the information, and send it to jerryqian@tencent.com and copy it to your customer service representative (important).
3. Ask your customer service representative to confirm the e-mail by replying to it. Otherwise, the e-mail may be treated as a harassing e-mail.
4. As soon as the e-mail is confirmed, we will apply for a trial License from Youtu Lab and send the License along with the package decompression password to you.

Two types of License:

- Trial License: It is **valid for one month**, and used to debug and test the dynamic effects SDK. If your App is published with a trial License, the dynamic effects will not work normally after the License expires.

- Official License: The validity period (usually one year) is subject to the contract.

Version Download

You can download the compressed SDK package (VIP version) at the bottom of the [RTMP SDK package](#). The compressed SDK package is encrypted, and you can obtain the decompression password & License in the integration procedure. After the package is decompressed successfully, you will get `Demo` and `SDK`. Resources related to special effects are placed in `SDK/Resource`.

You can distinguish the VIP version and non-VIP version by viewing the SDK's bundler id. > - The bundler id of the non-VIP version is `com.tencent.TXRTMPSDK`, and that of the VIP version `com.tencent.TXRTMPSDK.pitu`.

VIP and non-VIP SDKs can also be intuitively distinguished by size, because VIP SDK is much greater than non-VIP SDK.

Xcode Project Settings

For more information, please see [Project Configuration](#)

1. Add frameworks

The VIP edition requires to link with additional system frameworks.

1. `AssetsLibrary.framework`
2. `CoreMedia.framework`
3. `Accelerate.framework`
4. `Metal.framework`

2. Add link parameters

Under **Build Setting** -> **Other Link Flags**, add the `-ObjC` option.

3. Add Dynamic Effect Resources

Add the following files in `SDK/Resource` to the project

1. 3DFace
2. detector.bundle
3. FilterEngine.bundle
4. model
5. PE.dat
6. poseest.bundle
7. RPNSegmenter.bundle
8. ufa.bundle

Add the SegmentationShader.metal file under Demo/TXLiteAVDemo/Resource/Beauty/pitu/data/ to the project

1. SegmentationShader.metal


4. Add dynamic effect resources

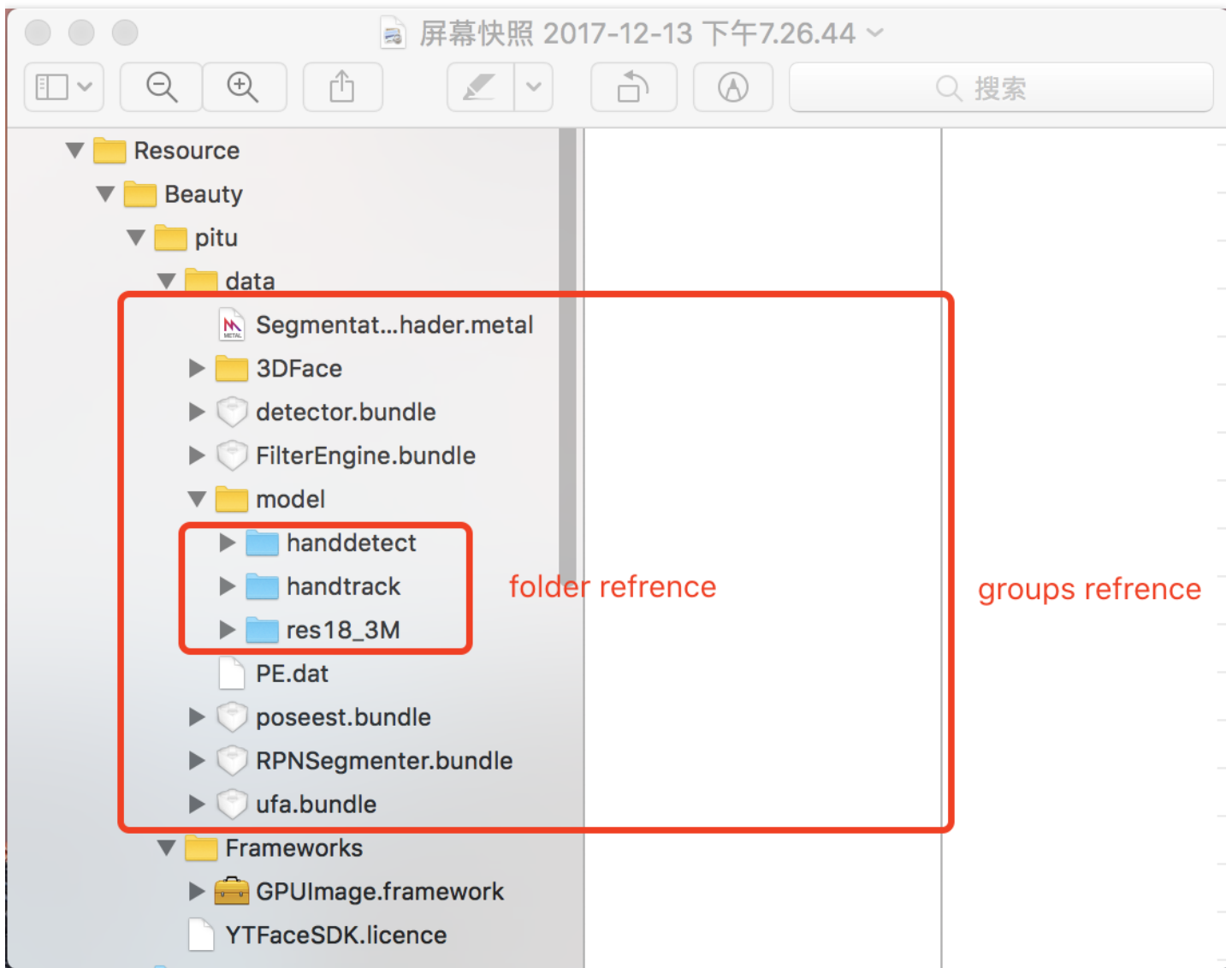
Add the resources under the Resource of the zip package to the project as groups reference. Note: handdetect, handtrack, and res18_3M are to be added as folder reference. You can directly add SegmentationShader.metal under Demo/TXLiteAVDemo/Resource/Beauty/pitu/data/. Specific operations are shown below:

Choose options for adding these files:

Destination: ☐ Copy items if needed

Added folders: ☒ Create groups
☐ Create folder references

Add to targets: ☒  TXLiteAVDemo_Enterprise



These resources must be added correctly. Otherwise, a crash may occur when the material is switched to the face-transforming type.

5. Import license files

The license of the VIP version needs to be verified to enable some of the features. You can contact one of our customer service representatives to apply for a 30-day free license for debugging.

After you obtain a license, name the license as **YTFaceSDK.licence** and place it in the project as shown in the figure above.

Each license is bound with a specific Bundle Identifier. Modifying the Bundle Identifier of the App will result in verification failure.

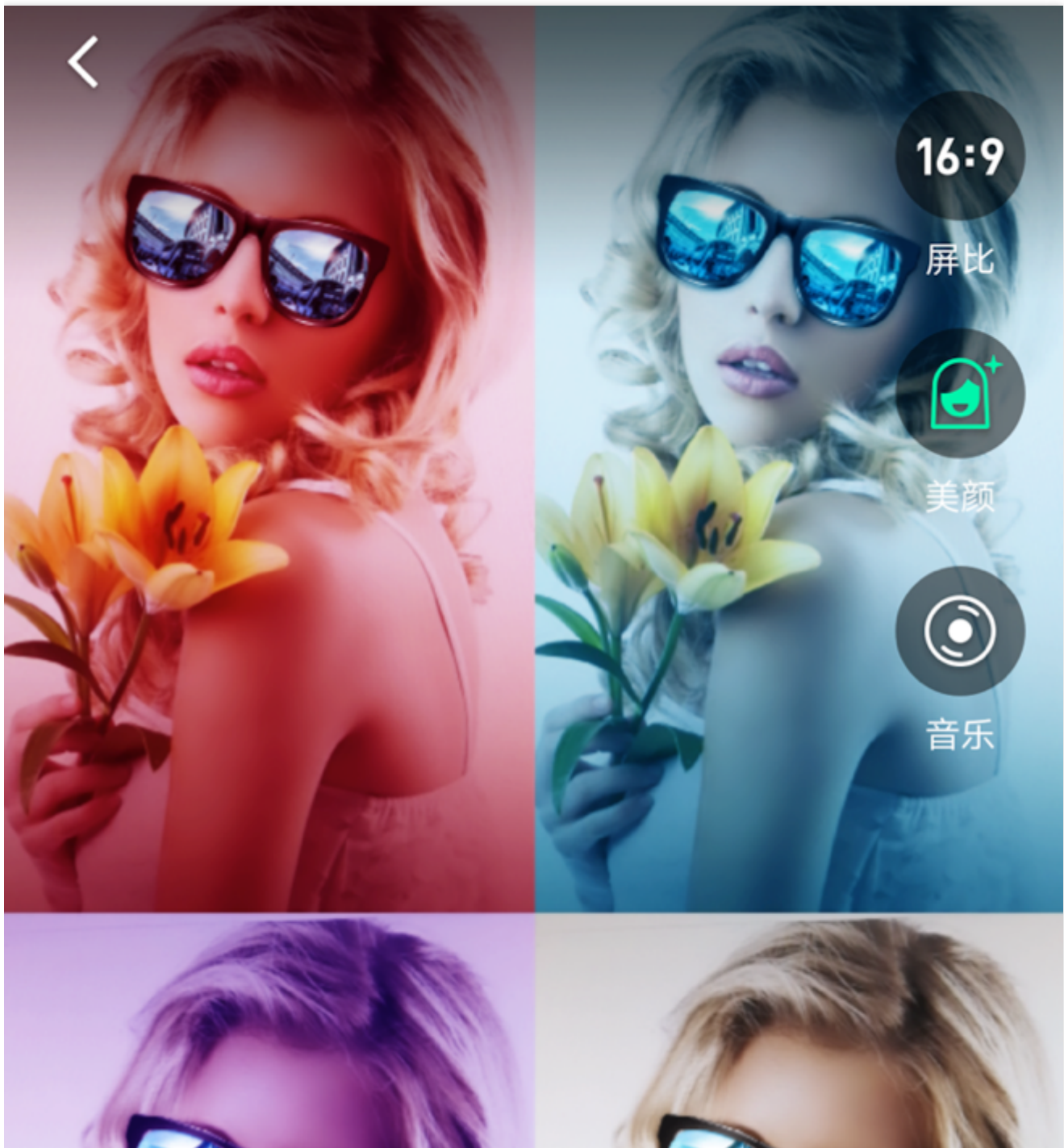
YTFaceSDK.licence cannot be renamed or modified.

You do not need to apply for licenses for iOS and Android separately. One license can be used to authorize the bundleid in iOS and the packageName in Android simultaneously.

Feature Calling

1. Dynamic sticker

Example:





A dynamic effect template is a directory, which contains a lot of resource files. The directory number and file size of each dynamic effect template vary depending on the complexity of the dynamic effect.

The sample code in Mini LVB downloads the dynamic effect resources from the backend, and then the resources are decompressed to the Resource directory. You can find the download addresses of the dynamic effect resources and thumbnails in the Mini LVB code in the following format:

```
https://st1.xiangji.qq.com/yunmaterials/{Dynamic Effect Name}.zip
```

```
https://st1.xiangji.qq.com/yunmaterials/{Motion Effect Name}.png
```

You are strongly recommended to put the dynamic effect resources on your own servers to avoid being affected by Mini LVB changes.

When the decompression is completed, you can enable the dynamic effect via the following API.

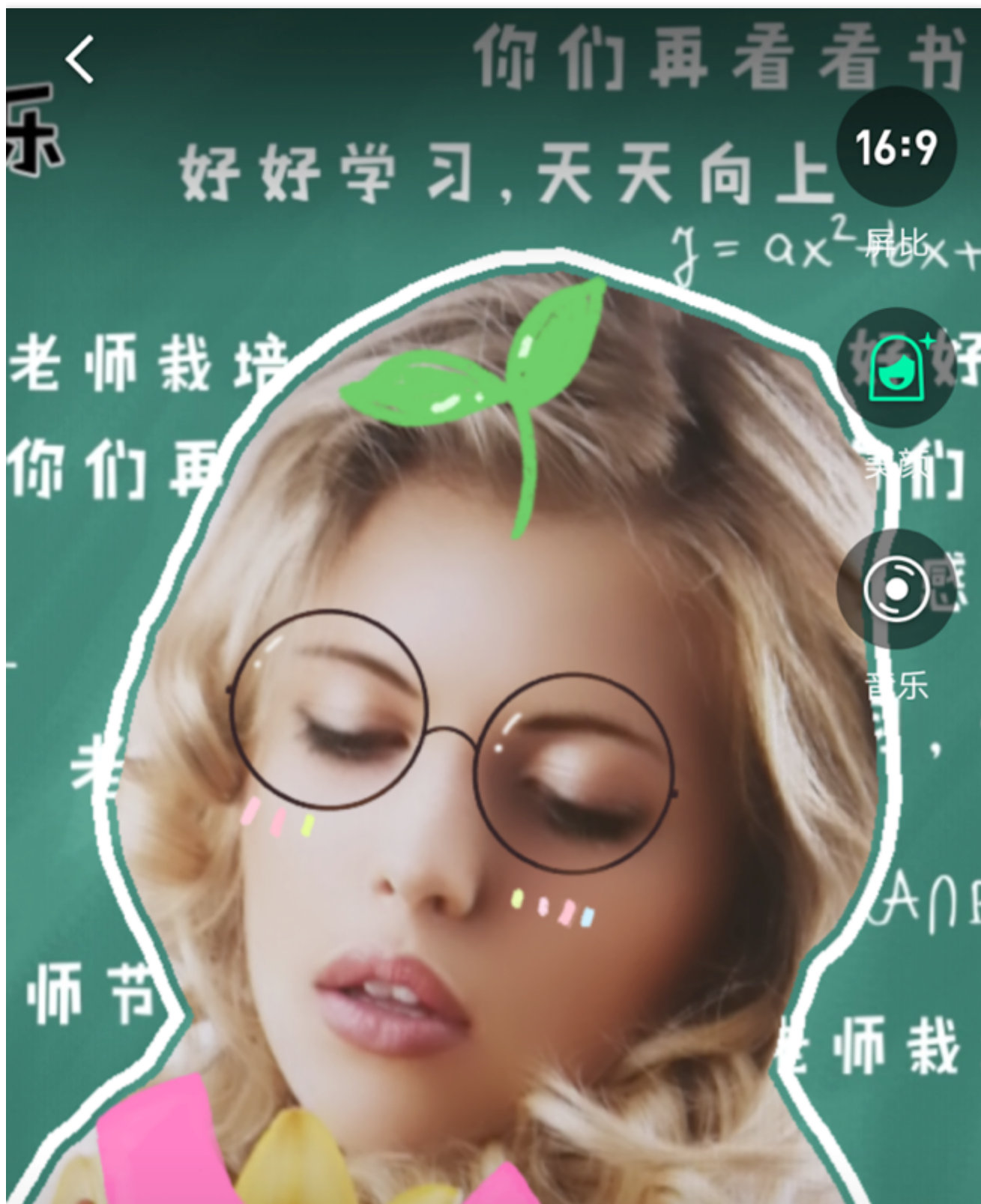
```
/**  
 * Select the dynamic effect  
 *  
 * @param tmpName: Dynamic effect name  
 * @param tmpDir: The directory in which the dynamic effect locates
```


*/

```
- (void)selectMotionTpl:(NSString *)tplName inDir:(NSString *)tplDir;
```

2. AI background keying-out

Example:





The resources for AI background keying-out need to be downloaded, and the API is the same as that of dynamic effects.

```
/**
 * Select background keying-out
 *
 * @param tmplName: Dynamic effect name
 * @param tmplDir: The directory in which the dynamic effect locates
 */
- (void)selectMotionTpl:(NSString *)tmplName inDir:(NSString *)tmplDir;
```

3. Beautifying

```
/* setEyeScaleLevel Set eye enlargement level. This parameter is valid only for value-added version.
 * Parameter:
 * eyeScaleLevel : Available value range for eye enlargement level: 0-9. 0 means disabling eye enlarge
ment. A higher value means a stronger effect.
 */
-(void) setEyeScaleLevel:(float)eyeScaleLevel;

/* setFaceScaleLevel Set face slimming level. This parameter is valid only for value-added version.
 * Parameter:
 * faceScaleLevel :Available value range for face slimming level: 0-9. 0 means disabling face slimming.
A higher value means a stronger effect.
 */
-(void) setFaceScaleLevel:(float)faceScaleLevel;
```



```

/* setFaceVLevel Set V-shaped face level. This parameter is valid only for value-added version.
* Parameter:
* faceVLevel : Available value range for V-shaped face level: 0-9. 0 means disabling V-shaped face. A
higher value means a stronger effect.
*/
- (void) setFaceVLevel:(float)faceVLevel;

/* setChinLevel Set chin stretching or contracting level. This parameter is valid only for value-added v
ersion.
* Parameter:
* chinLevel : Available value range for chin stretching or contracting level: -9-9. 0 means disabling chi
n stretching or contracting. -9 means stretching the chin to the maximum extent. 9 means contractin
g the chin to the maximum extent.
*/
- (void) setChinLevel:(float)chinLevel;

/* setFaceShortLevel Set short face level. This parameter is valid only for value-added version.
* Parameter:
* faceShortlevel : Available value range for short face level: 0-9. 0 means disabling short face. A higher
value means a stronger effect.
*/
- (void) setFaceShortLevel:(float)faceShortlevel;

/* setNoseSlimLevel Set nose narrowing level. This parameter is valid only for value-added version.
* Parameter:
* noseSlimLevel : Available value range for nose narrowing level: 0-9. 0 means disabling nose narrowi
ng. A higher value means a stronger effect.
*/
- (void) setNoseSlimLevel:(float)noseSlimLevel;

```

4. Green screen

You need to prepare an mp4 file in advance and call the following API to enable the green screen effect.

```

/**
* Set green screen file
*
* @param file: Path to the green screen mp4 is supported. nil means disabling green screen.
*/
-(void)setGreenScreenFile:(NSURL *)file;

```

Troubleshooting

1. Why does the project failed to be compiled?

1. Check whether dependent libraries `AssetsLibrary.framework`, `CoreMedia.framework`, `Accelerate.framework`, and `Metal.framework` have been added.

2. What should I do if crash occurs during project operation?

1. Check whether `-ObjC` is configured in the project.
2. Check whether Metal API Validation is set to Disabled.

3. Why don't the dynamic effects take effect?

1. Check if `YTFaceSDK.licence` is named correctly.
2. Check if the license expired (download [Query Tool](#) or contact our developers).
3. Check whether the Pitu resources are added correctly. Note that `handdetect`, `handtrack`, and `res18_3M` must be added as folder reference. The easiest way is to compare the dynamic effect files added in your project with those in our demo.
4. If you update the license, make sure to use the latest license. If you are not sure, check the validity period of the license (download [Query Tool](#) or contact our developers). If the license in the project is changed, clean the project and delete the local installation package for compilation.

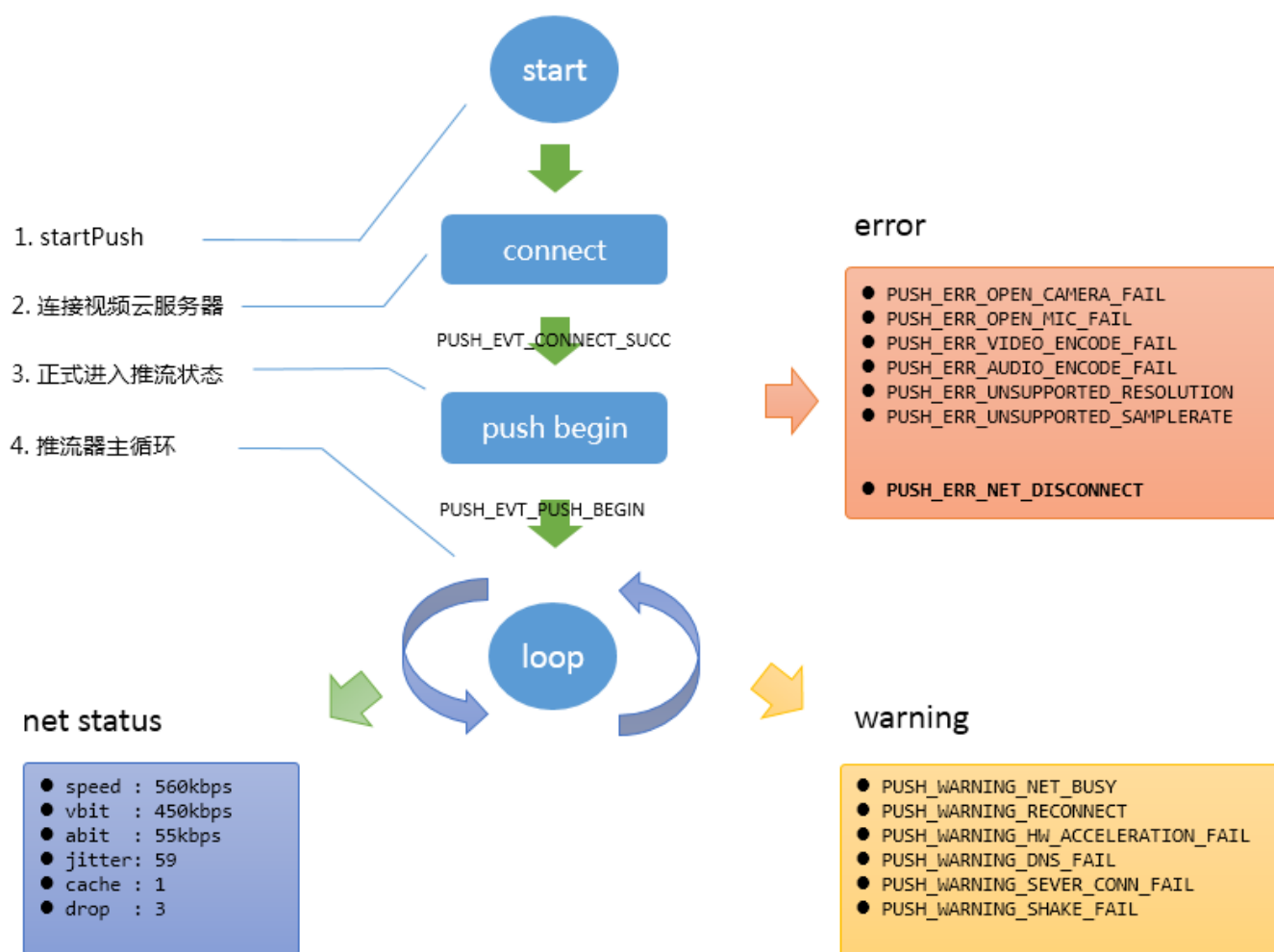
[Query Tool](#) is an xcode project, which is only supported on Mac. Other query methods will be available soon.

Advanced Feature

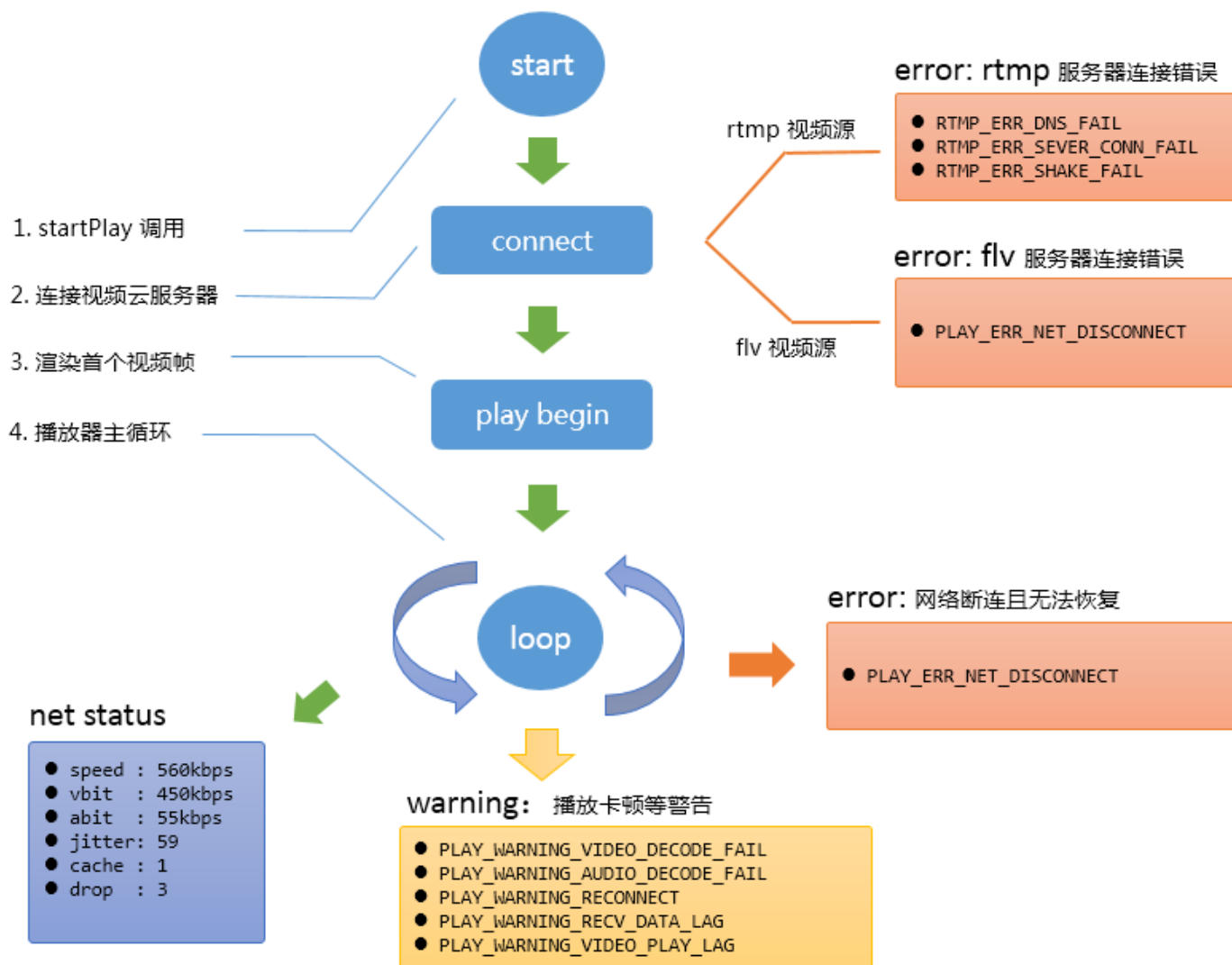
SDK Internal Principles

Last updated : 2018-07-23 15:22:47

TXLivePusher



TXLivePlayer



SDK Metric Monitoring

Last updated : 2018-07-23 15:47:21

TXLivePushListener

1. How to obtain push status?

The onNetStatus callback of TXLivePushListener synchronizes the status metrics inside SDK at an interval of 1-2 seconds. The major metrics are as follows:



TXLivePushListener - onNetStatus

推流状态	参数名称	示例指标	指标含义说明
CPU	App CPU使用率	24.5% 71.3%	App: 24.5%
	系统CPU使用率		Sys: 71.3%
RES	推流分辨率	544*960	分辨率: 544*960
SPD	网络上传速度	1219kb/s	每秒上传1219kb数据
FPS	视频帧率	14	每秒14帧画面
GOP	关键帧间隔	5s	每隔5秒编一个I帧
ARA	音频码率	64kb/s	每秒编码出64kb音频数据
VRA	视频码率	1155kb/s	每秒编码出1155kb视频数据
SVR	推流服务器IP: 端口	125.94.63.141: 443	IP地址: 125.94.63.141 端口号: 443
AUD 0	当前aec类型	1 48000, 1	当前aec类型 (0: 没有开启aec; 1: 使用系统aec; 2: 使用trae)
	音频信息		音频采样率: 48000, 声道数: 1

Push Status	Description
NET_STATUS_CPU_USAGE	CPU usage of the current process and overall CPU usage of the device
NET_STATUS_VIDEO_WIDTH	Width of the current video (in pixels)
NET_STATUS_VIDEO_HEIGHT	Height of the current video (in pixels)
NET_STATUS_NET_SPEED	Current transmission speed (in Kbps)
NET_STATUS_VIDEO_BITRATE	The output bitrate of the current video encoder, i.e., the amount of video data produced by the encoder per second (in Kbps)

Push Status	Description
NET_STATUS_AUDIO_BITRATE	The output bitrate of the current audio encoder, i.e., the amount of video data produced by the encoder per second (in Kbps)
NET_STATUS_VIDEO_FPS	Current video frame rate, i.e., the number of frames produced by the video encoder per second
NET_STATUS_CACHE_SIZE	Accumulated audio/video data size. A value ≥ 10 indicates the current upstream bandwidth is not enough to consume the audio/video data produced.
NET_STATUS_CODEC_DROP_CNT	The number of global packet drops. To avoid a vicious accumulation of delays, the SDK actively drops packets when the accumulated data exceeds the threshold. A higher number of packet drops means a more severe network problem.
NET_STATUS_SERVER_IP	The IP address of the connected push server

2. Which status metrics can be used for reference?

BITRATE vs NET_SPEED

BITRATE(= VIDEO_BITRATE + AUDIO_BITRATE) refers to the number of audio/video data bits produced by the encoder for push per second; NET_SPEED refers to the number of data bits pushed actually per second.

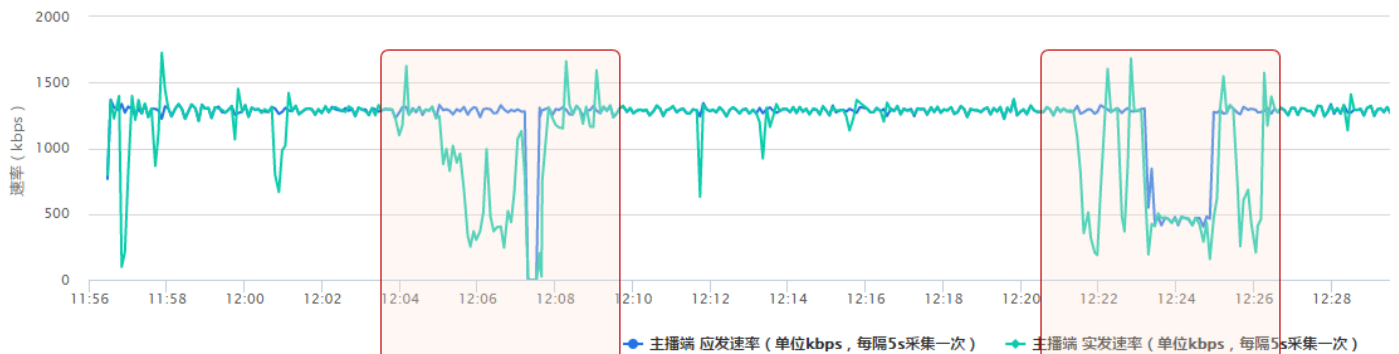
- If BITRATE == NET_SPEED in most cases, the push quality is very good;
- If BITRATE \geq NET_SPEED for a long time, audio and video data will accumulate on VJ's mobile phone and make CACHE_SIZE increase to such a point that the data is dropped by SDK to generate DROP_CNT.

CACHE_SIZE & DROP_CNT

In case of a slow upload speed at VJ end, it is likely that BITRATE \geq NET_SPEED. In this case, the audio/video data build up on VJ's phone, with the severity indicated by the CACHE_SIZE value. When the CACHE_SIZE value exceeds the threshold, SDK drops some audio/video data, thus triggering an increment of DROP_CNT.

主播端 应发速率-实发速率曲线图

图例说明：视频生成码率曲线和推流速率曲线越重合，说明主播端推送越流畅，反之主播推送不流畅，易引起播放端观看卡顿。（在图上按下鼠标左键拖动可以查看细节）



主播端 音视频数据堆积情况

图例说明：主播端不能及时发送到云端的音视频数据会被堆积起来，数据堆积超过三秒（下图红色警示线以上部分）会被主动丢弃，因此当主播的上行网络不理想时，观看端会出现



CPU_USAGE

- If **CPU utilization of system** exceeds 80%, the stability of audio/video encoding is affected, leading to random stutters in video image and sound.
- If **CPU utilization of system** reaches 100% frequently, the audio/video encoding frame rate will become insufficient, leading to serious stutters in video image and sound.

It is very common that in the practical use of an App that performs well in per-launch test, the scrolling and refreshing of interactive messages of front rooms consume a significant amount of CPU and thus lead to serious stutters in LVB video image.

SERVER_IP

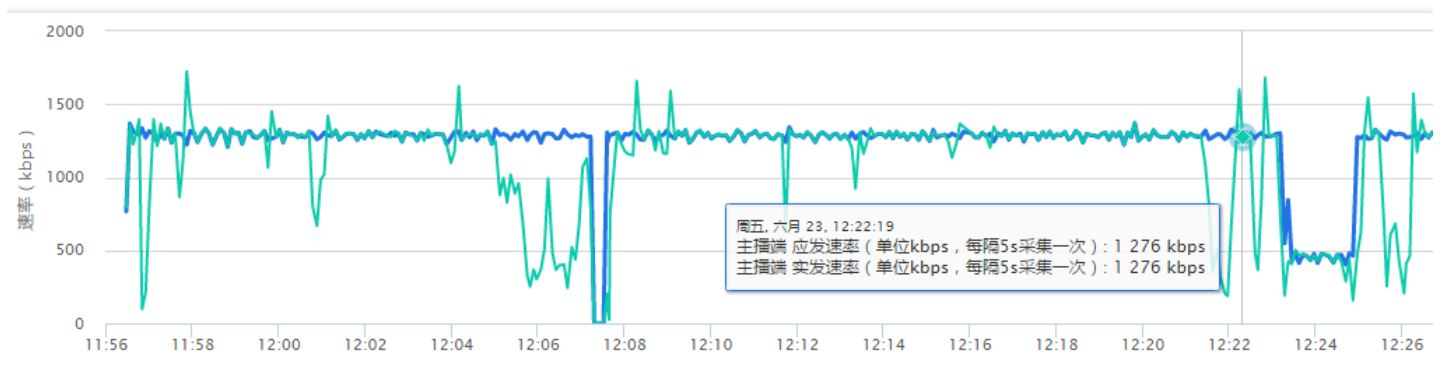
If the Ping value from VJ to the IP given by SERVER_IP is very high (for example, exceeding 500ms), the push quality will be unsatisfactory. Tencent Cloud has been providing services on an **Access to the Closest** basis. In case of the above situation, please contact us. Our OPS team will optimize the service.

3. How to comprehend Tencent Cloud push chart?

In [LVB Console - Quality Control](#), you can get a picture of the live rooms under your account and the push quality of each room:

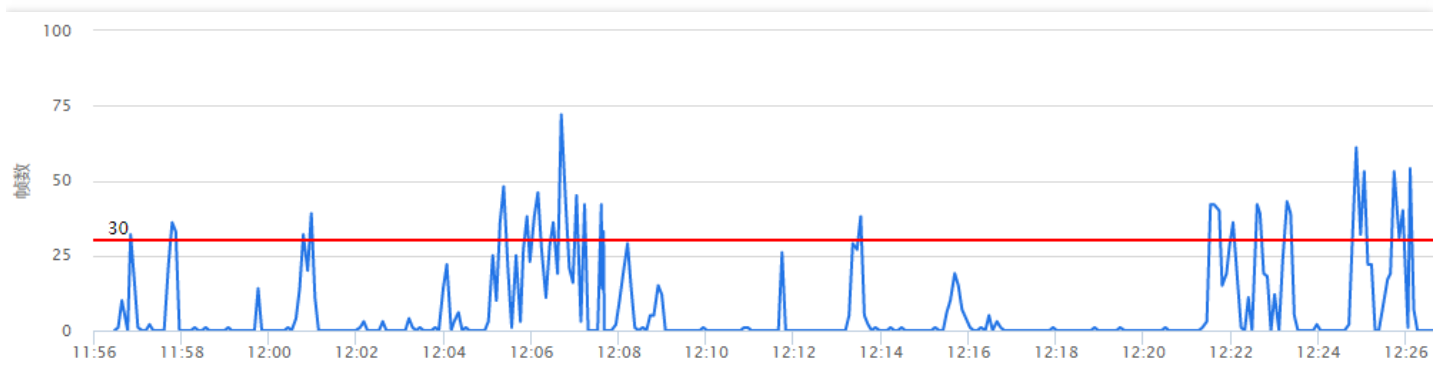
- **VJ end - expected bit rate - Actual bit rate curve**

The blue curve is the statistical curve of BITRATE, i.e. the audio and video data bits produced by the SDK. The green curve indicates the data bits actually pushed via the network. A higher fit between the two curves means a better push quality.



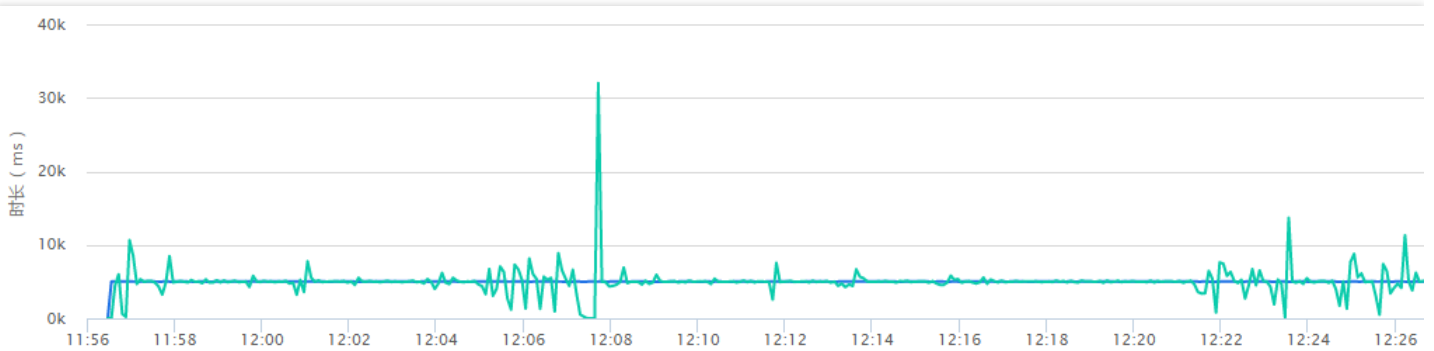
- **VJ end - Accumulation of audio/video data**

- The consistent fit between the curve and 0 scale means the entire push is very smooth and no data is accumulated.
- The points where the curve > 0 indicate accumulated data caused by network fluctuations, which may lead to slight stutters and asynchronization between video and audio at the viewer end;
- If the accumulated data exceeds the red warning level, it means that some packets has been dropped, which will inevitably result in stutters and asynchronization between video and audio at the viewer end.



• Cloud end - expected video duration - actual video duration curve

This is the statistics curve for Tencent Cloud server end, and is the only chart visible to you if you do not use Tencent Cloud SDK for push (the first two charts are invisible to you because the data is provided by SDK). A higher fit between the blue and green curves indicates a better push quality.



TXLivePlayListener

1. How to obtain the playback status data?

The onNetStatus callback of TXLivePlayListener synchronizes the status metrics inside SDK at an interval of 1-2 seconds. The major metrics are as follows:



TXLivePlayListener - onNetStatus

拉流状态	参数名称	示例指标	指标含义说明
CPU	App CPU使用率	8.3% 22.5%	App: 8.3%
	系统CPU使用率		Sys: 22.5%
RES	拉流分辨率	480*288	分辨率: 480*288
SPD	网络下载速度	321kb/s	每秒下载321kb数据
FPS	视频帧率	25	每秒25帧画面
GOP	关键帧间隔	1s	每隔1秒有一个I帧
ARA	音频码率	30kb/s	每秒需解码30kb音频数据
QUE	音频缓冲时长	1834 1880, 47, 4 -52, -47, 5.0	音频缓冲时长: 1834毫秒
	视频缓冲时长		视频缓冲时长: 1880毫秒
	视频缓冲总帧数		视频缓冲总帧数: 47帧
	视频解码器缓冲帧数		视频解码器缓冲帧数: 4帧
	音视频网络收帧时间差		音频与视频网络收帧时间差: -52毫秒
	音视频当前帧渲染时间差		音频与视频当前帧渲染时间差: -47毫秒
	平衡点		平衡点
VRA	视频码率	291kb/s	每秒需解码291kb视频数据
SVR	拉流服务器IP: 端口	183.6.224.37:1935	IP地址: 183.6.224.37 端口号: 1935
AUD 0	当前aec类型	1 48000, 2 48000, 1	当前aec类型 (0: 没有开启aec; 1: 使用系统aec; 2: 使用trae)
	原始音频信息		原始音频采样率: 48000, 原始声道数: 2
	播放音频信息		播放音频采样率: 48000, 播放声道数: 1

Playback Status

Description

Playback Status	Description
NET_STATUS_CPU_USAGE	Current (instant) CPU utilization
NET_STATUS_VIDEO_WIDTH	Video resolution - width
NET_STATUS_VIDEO_HEIGHT	Video resolution - height
NET_STATUS_NET_SPEED	Current download speed
NET_STATUS_VIDEO_FPS	Video frame rate of the current stream media
NET_STATUS_VIDEO_BITRATE	Video bitrate of the current stream media (in Kbps)
NET_STATUS_AUDIO_BITRATE	Audio bitrate of the current stream media (in Kbps)
NET_STATUS_CACHE_SIZE	Playback buffer (jitterbuffer) size. A smaller buffer size means a lower resistance against stutters.
NET_STATUS_SERVER_IP	IP of the connected server

2. Which status metrics can be used for reference?

NET_STATUS_CACHE_SIZE

This metric reflects the size of the playback buffer (jitterbuffer):

- The larger the CACHE_SIZE, the higher the delay, and the less likelihood of stutters in case of network fluctuations.
- The smaller the CACHE_SIZE, the lower the delay, and the more likelihood of stutters in case of network fluctuations.

TXLivePlayer comes with three modes for controlling the playback buffer size. For more information, please see [Basic Features - Playback] document:

播放模式	卡顿率	延迟	使用场景	原理简述
极速模式	较高	2s - 3s	美女秀场	尽可能保证较低的播放缓冲区，进而减少延迟，适合 1Mbps 左右的低码率场景。
流畅模式	较低	$\geq 5s$	游戏直播	确保随时都有较大的播放缓冲区，通过牺牲延迟来应对大码率（2M左右）下的网络波动的影响。
自动模式	自适应	2s - 8s	泛场景	缓冲区大小自动调节： 网络越好，延迟越低；网络越差，延迟越高。

QoS Traffic Control

Last updated : 2018-07-23 12:03:25

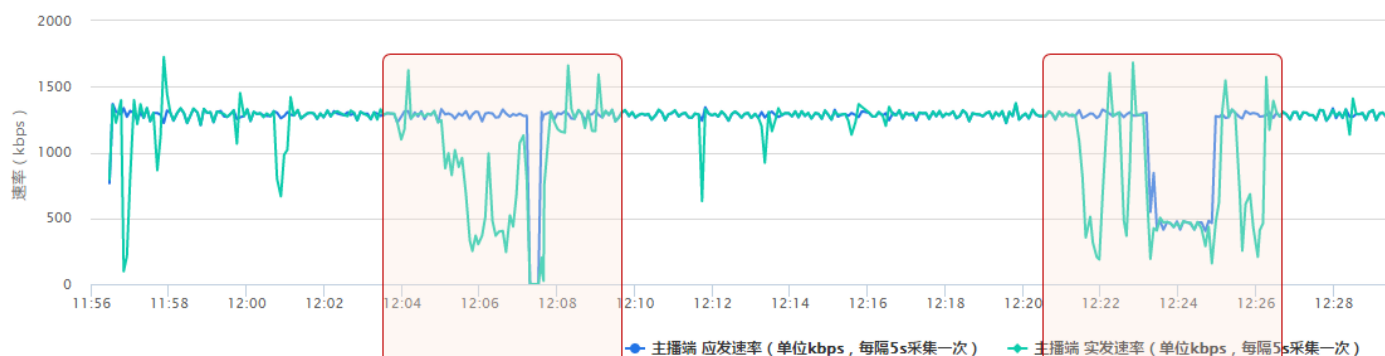
Background

RTMP push quality is crucial to the viewing experience. A poor push quality at VJ end will cause the stutter at all the viewer ends. According to the statistics, over 80% of LVB stutters among Video Cloud customers are caused by a poor RTMP push quality.

Among the push quality issues, the biggest issue is caused by unsatisfactory uplink network at VJ end. Insufficient uplink bandwidth can make audio/video data build up and then be dropped at VJ end, thus leading to the stutter or even long freezing of video images at viewer end.

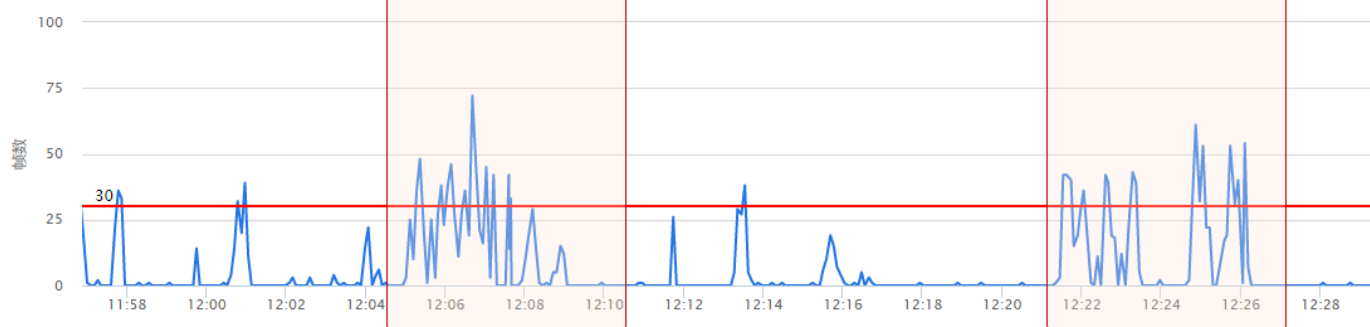
主播端 应发速率-实发速率曲线图

图例说明：视频生成码率曲线和推流速率曲线越重合，说明主播端推送越流畅，反之主播推送不流畅，易引起播放端观看卡顿。（在图上按下鼠标左键拖动可以查看细节）



主播端 音视频数据堆积情况

图例说明：主播端不能及时发送到云端的音视频数据会被堆积起来，数据堆积超过三秒（下图红色警示线以上部分）会被主动丢弃，因此当主播的上行网络不理想时，观看端会出现



Therefore, dealing with the stutter of uplink network at VJ end can effectively improve the push quality, thus delivering a better viewing experience, especially in the domestic environment where uplink bandwidth is restricted widely by ISPs.

But network condition does not hinge on our will. If a VJ uses a 4-Mbps broadband package at home, it is not impossible to change it to 8Mbps just because the VJ installs a new App. Therefore, we can choose to **actively adapt to the uplink network**.

Quick integrating

You can enable the Qos traffic control with the parameters of the setVideoQuality API of TXLivePusher. The SDK will then decide the video definition according to VJ's uplink network condition.



- **quality**

The SDK provides six basic options which are developed and configured based on our rich experience with our huge customer base. Among them, STANDARD, HIGH, and SUPER are intended for LVB mode, MAIN_PUBLISHER and SUB_PUBLISHER are intended for primary and secondary screens of joint broadcasting, and VIDEOCHAT is used for real-time audio/video.

- **adjustBitrate**

Specifies whether to enable Qos traffic control. If enabled, the SDK decides the video definition according to VJ's uplink network condition. The disadvantage is that blurry screens and many mosaics may occur in case of poor network condition on the VJ end.

- **adjustResolution**

Specifies whether to support dynamic resolution. If enabled, the SDK selects a appropriate resolution based on the current video bitrate for better definition. The disadvantage is that files generated by recording live streaming with dynamic resolutions may have compatibility issue with many players.

quality	Qos (开启)	Qos (关闭)	动态分辨率 (启用)	动态分辨率 (关闭)
STANDARD	300kbps – 800kbps	800kbps	<ul style="list-style-type: none"> ● 270*480 ● 360*640 	360*640
HIGH	600kbps – 1200kbps	1200kbps	<ul style="list-style-type: none"> ● 360*640 ● 540*960 	540*960
SUPER	600kbps – 1800kbps	1800kbps	<ul style="list-style-type: none"> ● 360*640 ● 540*960 ● 720*1280 	720*1280
VIDEOCHAT	200kbps – 800kbps	不支持关闭	<ul style="list-style-type: none"> ● 190*320 ● 270*480 ● 360*640 	不支持关闭

Fine Adjustment

If the default parameters in `setVideoQuality` are not enough, you can customize more parameters via `TXLivePushConfig`:

- **enableAutoBitrate**

Specifies whether to enable the adaptive bitrate, i.e. Qos traffic control. If enabled, the SDK decides the video definition according to VJ's uplink network condition.

- **autoAdjustStrategy**

Available only when the adaptive bitrate is enabled, otherwise the setting is invalid. The `autoAdjustStrategy` supports the following four strategies:

Strategy Name	Description
AUTO_ADJUST_BITRATE_STRATEGY_1	Constantly detects and adjusts the network speed throughout the LVB process. It is intended for scenarios such as live shows.
AUTO_ADJUST_BITRATE_RESOLUTION_STRATEGY_1	Adjusts the bitrate while adjusting the resolution accordingly to keep a balance between bitrate and resolution.
AUTO_ADJUST_BITRATE_STRATEGY_2	Quickly detects and adjusts the network speed in the first half minutes of LVB accordingly, and then minimize the adjustment. It is intended for mobile game scenarios.

Strategy Name	Description
AUTO_ADJUST_BITRATE_RESOLUTION_STRATEGY_2	Adjusts the bitrate while adjusting the resolution accordingly to keep a balance between bitrate and resolution.

- **videoBitrateMin**: Represents the minimum bitrate, which is available only when the adaptive bitrate is enabled, otherwise the setting is invalid.
- **videoBitrateMax**: Represents the maximum bitrate, which is available only when the adaptive bitrate is enabled, otherwise the setting is invalid.
- **videoBitratePIN**: Represents the original bitrate, $\text{videoBitrateMin} \leq \text{videoBitratePIN} \leq \text{videoBitrateMax}$.

Coding Parameter Adjustment

Last updated : 2018-07-23 16:04:08

Customizing Parameter

You can customize video/audio encoding parameters by setting the Config object. Now, the following setting APIs are supported:

Parameter Name	Description	Default Value
audioSampleRate	Audio sampling rate: The number of samples per second are taken from an audio signal by a recording device	44100
enableNAS	Noise suppression: When this is enabled, background noises can be filtered out (applicable when the sampling rate is below 32,000)	Off
enableHWAceleration	Video hard-coding: When this is enabled, video capture up to 720 p, 30 fps is supported.	On
videoFPS	Video frame rate: The number of frames produced by the video encoder per second. Most of the phones don't support encoding above 30 FPS, so setting FPS to 20 is recommended.	20
videoResolution	Video resolution: Four types of 16:9 resolutions are available	640 * 360
videoBitratePIN	Video bitrate: The amount of data produced by the video encoder per second (in Kbps)	800
enableAutoBitrate	Bitrate adaptation: Adjust the video bitrate automatically based on the network condition	Off
videoBitrateMax	Maximum output bitrate: This option takes effect only when bitrate adaption is enabled.	1,200
videoBitrateMin	Minimum output bitrate: This option takes effect only when bitrate adaption is enabled.	800

Parameter Name	Description	Default Value
videoEncodeGop	Keyframe interval (in second): The interval at which one I frame is output	3 seconds
homeOrientation	Set the rotation angle of video image, e.g., whether to push in landscape mode	0: home is on the right; 1: home is at the bottom; 2: home is on the left; 3: home is at the top
beautyFilterDepth	Beauty filter level: levels 1 to 9 are supported; the higher the level, the more obvious the effect. 0 means Off	Off
frontCamera	Front or rear camera by default	Front
watermark	Watermark image (UIImage object)	Tencent Cloud Logo (demo)
watermarkPos	The position of the watermark image relative to the coordinate in the upper-left corner	(0, 0)

Setting Method

You are recommended to set these parameters before enabling push, since most of them only take effect when the push is restarted. The reference codes are as follows:

```
//Declare _config and _pusher in member variables
....
//Initialize _config
_config = [[TXLivePushConfig alloc] init];

//Modify the audio sampling rate to 44100 and fixed video bitrate to 800
_config.audioSampleRate = 44100;
_config.enableAutoBitrate = NO;
_config.videoBitratePIN = 800;

//Initialize _pusher
_pusher = [[TXLivePush alloc] initWithConfig: _config];
```

Video Data Customization

Last updated : 2018-07-23 16:01:48

Customizing Push Images

Solution 1: Modify OpenGL texture

Some customers with a strong R&D capability want to customize images (e.g., adding captions) while reusing the overall process of RTMP SDK. In this case, follow the steps below.

- **Set callback for video processing**

You can customize video images by setting **videoProcessDelegate** proxy for **TXLivePush**.

@protocol TXVideoCustomProcessDelegate <NSObject>

```
/**
 * Perform a callback in the OpenGL thread, where you can conduct the secondary processing of captured images.
 * @param textureId Texture ID
 * @param width Width of texture
 * @param height Height of texture
 * @return Texture returned to SDK
 * Note: The texture type called back from the SDK is GL_TEXTURE_2D, and the one returned by the API to the SDK must also be GL_TEXTURE_2D.
 */
-(GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width height:(CGFloat)height;

/**
 * Perform a callback in the OpenGL thread, where you can release the OpenGL resources created.
 */
-(void)onTextureDestroyed;

@end
```

- **Process the video data in the callback function**

Implement onPreProcessTexture function of TXVideoCustomProcessDelegate to achieve the customized processing of video images. The texture specified by textureId is a texture of type GL_TEXTURE_2D.

To work with texture data, you need to have some basic knowledge about OpenGL. In addition, a huge calculation amount is not recommended. This is because `onPreProcessTexture` has the same call frequency as FPS, and too heavy processing is likely to cause the GPU overheating.

Solution 2: Capture data by yourself

If you only want to use SDK for encoding and push (for example, you have interfaced with SenseTime and other products), bring the audio and video capture and preprocessing (such as beauty filter, filter) under the control of your own code by following the steps below:

- **Step1. Do not call TXLivePusher's startPreview API any longer**

In this way, SDK itself does not capture video and audio data any more, but only conducts preprocessing, encoding, traffic control, data delivery and other push-related operations.

- **Step2. Set customModeType via TXLivePushConfig**

```
#define CUSTOM_MODE_AUDIO_CAPTURE 0X001 //Customer captures their own audios.
#define CUSTOM_MODE_VIDEO_CAPTURE 0X002 //Customer captures their own videos.

//If both audios and videos need to be captured by customer, the customModeType can be set to 3.
@interface TXLivePushConfig : NSObject
@property(nonatomic, assign) int customModeType;
@end
```

- **Step3. Use sendVideoSampleBuffer to populate SDK with Video data**

`SendVideoSampleBuffer` is used to populate the SDK with the captured and processed video data. RGBA and NV12 formats are supported currently.

```
TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
config.customModeType = CUSTOM_MODE_VIDEO_CAPTURE; // Customize data capturing
TXLivePush pusher = [[TXLivePush alloc] initWithConfig:config];

//You can populate the RGBA or NV12 data you captured and processed.
[pusher sendVideoSampleBuffer:sampleBuffer];
```

- **Step4. Use sendAudioSampleBuffer to populate SDK with Audio data**

`SendAudioSampleBuffer` is used to populate the SDK with the captured and processed audio data. Please use 16-bit, 48000-Hz PCM mono audio data.

This function supports two `RPSampleBufferTypes`: `RPSampleBufferTypeAudioApp` and `RPSampleBufferTypeAudioMic`. The former is used for replaykit, and the latter for both general microphone capturing and replaykit.

```
TXLivePushConfig* config = [[TXLivePushConfig alloc] init];
config.customModeType = CUSTOM_MODE_AUDIO_CAPTURE; // Customize data capturing
TXLivePush pusher = [[TXLivePush alloc] initWithConfig:config];

//You can populate the audio data you captured and processed.
[s_txLivePublisher sendAudioSampleBuffer:sampleBuffer withType:RPSampleBufferTypeAudioMic];
```

Customize Playback Data

- **Configure the `TXVideoCustomProcessDelegate` attribute of `TXLivePlayConfig`**

```
@interface TXLivePlayer : NSObject
// After configuration, each frame of Player goes through onPlayerPixelBuffer.
@property(nonatomic, weak) id <TXVideoCustomProcessDelegate> videoProcessDelegate;
```

- **Capture image data of Player with the `onPlayerPixelBuffer` callback.**

The image format of `pixelBuffer` is **NV12** if Player is in hardware decoding mode, and **i420** if Player is in software decoding mode.

If `onPlayerPixelBuffer` returns YES, the SDK stops image rendering, which can solve the OpenGL thread conflict.

```
@protocol TXVideoCustomProcessDelegate <NSObject>
@optional

/**
 * Video rendering object callback
 * @param pixelBuffer Render the image
 * @return If YES is returned, the SDK stops rendering; if NO is returned, the SDK rendering module keeps working.
 * Note: the data type of the rendered image is renderPixelFormatType set in config.
 */
-(BOOL)onPlayerPixelBuffer:(CVPixelBufferRef)pixelBuffer;
@end
```

