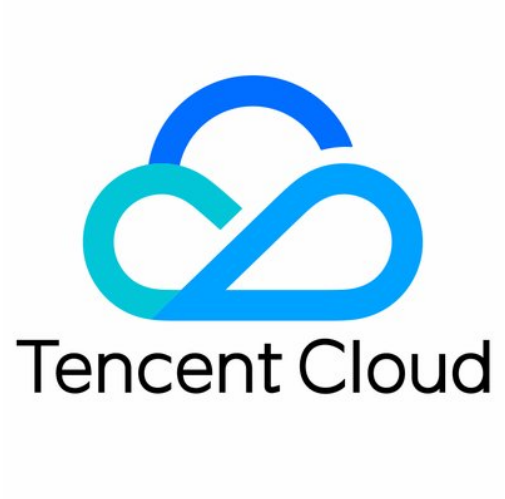
User Generated Short Video SDK

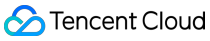
UI 統合ソリューションを含まない

製品ドキュメント



著作権声明

商標に関する声明



サービス声明

本ドキュメントは、お客様にTencent Cloudの全部また一部の製品・サービスの概要をご紹介することを目的としておりますが、一部の製品・サービス内容は変更される可能性があります。お客様がご購入されるTencent Cloud製品・サービスの種類やサービス基準などは、お客様とTencent Cloudとの間の締結された商業契約に基づきます。別段の合意がない限り、Tencent Cloudは本ドキュメントの内容に関して、明示または黙示の一切保証もしません。

カタログ:

UI 統合ソリューションを含まない SDK Integration
SDK Integration (Xcode)

最終更新日：：2025-04-01 18:10:23

Supported Platforms

The SDK is supported on iOS 8.0 or later.

Environment Requirements

- Xcode 9 or later
- iOS 12.0 or later

Directions

Step 1. Link the SDK and system libraries

How to integrate a short video SDK into your app

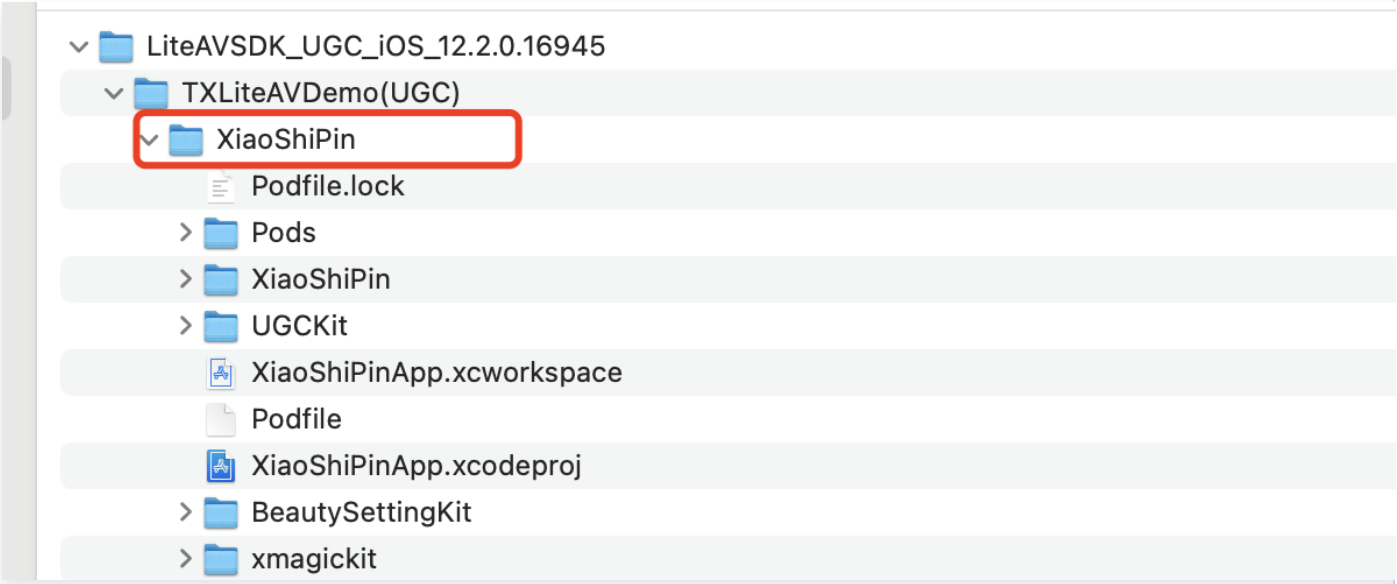
1. **Select the project target and add the following system libraries:**
   - Accelerate.framework
   - SystemConfiguration.framework
   - libc++.tbd
   - libsqlite3.tbd
   - MetalKit.framework
   - VideoToolbox.framework
   - ReplayKit.framework
   - GLKit.framework
   - OpenAL.framework
   - CoreServices.framework
2. **Unzip the downloaded SDK package and copy it to your project directory. Select the project's Target, navigate to *Build Phases*, then add the dynamic libraries (located in the SDK directory) under *Link Binary With Libraries*:**
   - TXFFmpeg.xcframework
   - TXSoundTouch.xcframework

- TXLiteAVSDK_UGC.xcframework

3. Add the following frameworks in *Embed Frameworks* and select *Code Sign On Copy*.
   - TXFFmpeg.xcframework
   - TXSoundTouch.xcframework
4. Select the project's Target, search for bitcode in *Build Settings*, and set *Enable Bitcode to NO*.

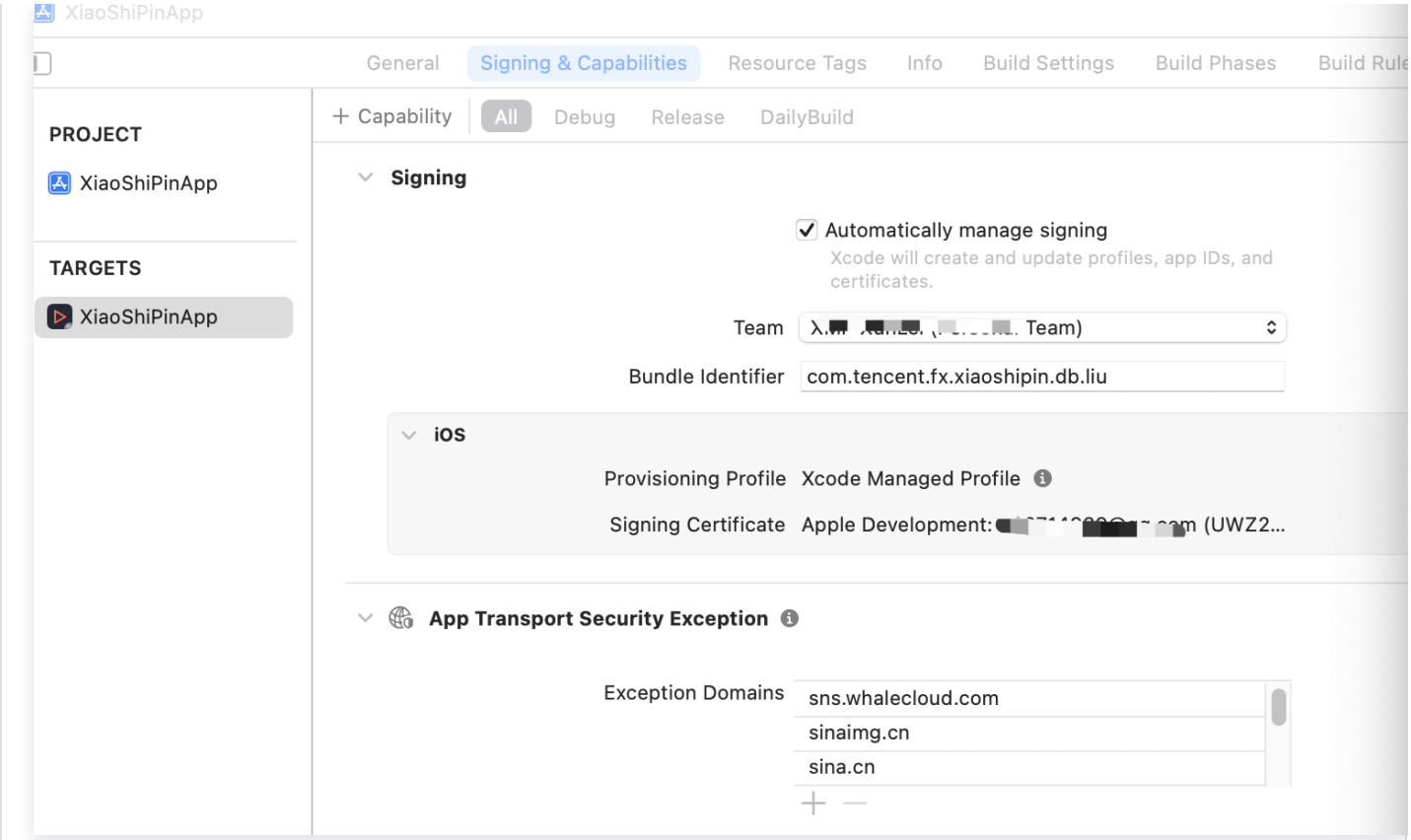
How to run the demo

1. Navigate to the TXLiteAVDemo(UGC)/XiaoShiPin directory in Terminal.



2. Run pod install.

3. Configure your iOS developer signing in Xcode.



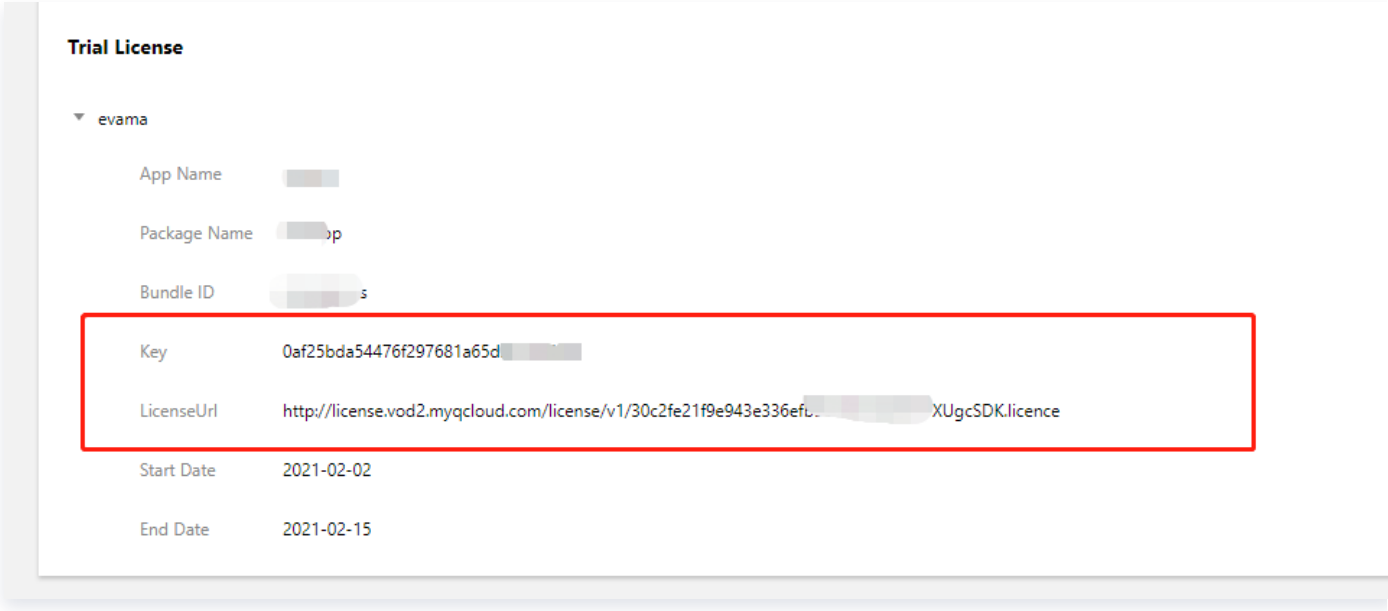
Step 2. Configure app permissions

The app needs access to the photo album, which can be configured in `Info.plist`. Right-click `Info.plist`, select **Open as** > **Source Code**, and copy and modify the code below.

```
<key>NSAppleMusicUsageDescription</key>
<string>Video Cloud Toolkit needs to access your media library to obtain
music files. It cannot add music if you deny it access.</string>
<key>NSCameraUsageDescription</key>
<string>Video Cloud Toolkit needs to access your camera to be able to
shoot video.</string>
<key>NSMicrophoneUsageDescription</key>
<string>Video Cloud Toolkit needs to access your mic to be able to shoot
videos with audio.</string>
<key>NSPhotoLibraryAddUsageDescription</key>
<string>Video Cloud Toolkit needs to access your photo album to save
edited video files.</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>Video Cloud Toolkit needs to access your photo album to edit
your video files.</string>
```

Step 3. Configure the license and get basic information

1. Follow the steps in License Application to apply for a license, and copy the key and license URL in the console.

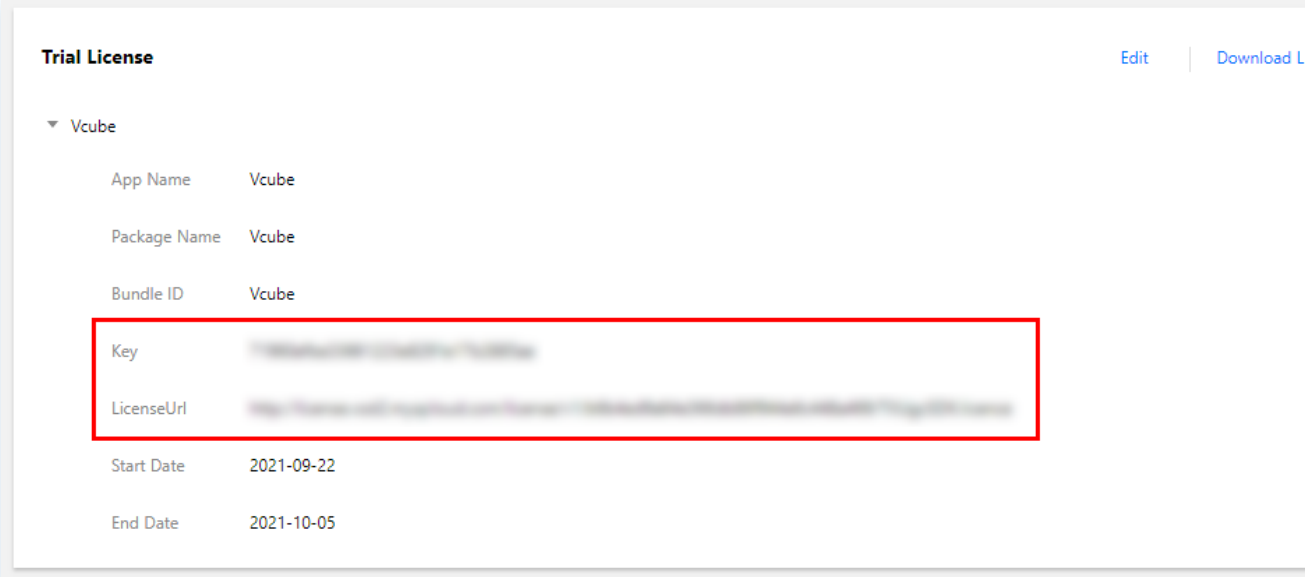


2. Before you integrate UGSV features into your application, we recommend you set `- [AppDelegate application:didFinishLaunchingWithOptions:]` as follows:

```
@import TXLiteAVSDK_UGC;
@implementation AppDelegate
- (BOOL)application:
(UIApplication*)applicationdidFinishLaunchingWithOptions:
(NSDictinoary*)options {
    NSString * const licenceURL = @"<License URL obtained>";
    NSString * const licenceKey = @"<The key obtained>";
    [TXUGCBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
@end
```

> **Note:**

If you use a **license for the SDK on v4.7** and have upgraded the SDK to v4.9, you can click **Switch to New License** in the console to generate a new license key and URL. A new license can be used only for the SDK on v4.9 or later and should be configured as described above.



Step 4. Configure logs

You can enable/disable console log printing and set the log level in `TXLiveBase`. Below are the APIs used.

- **setConsoleEnabled**

  Sets whether to print the SDK output in the Xcode console.

- **setLogLevel**

  Sets whether to allow the SDK to print local logs. By default, the SDK writes logs to the **Documents/logs** folder of the current app.

  We recommend that you enable local log printing. You may need to provide log files if you run into a problem and need technical support.

- **Viewing log files**

- To reduce the storage space taken up by log files, the UGSV SDK encrypts local logs and limits their number. You need a log decompression tool to view the content of log files.

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

Step 5. Build and run the project

If the above steps are performed correctly, you will be able to successfully compile the `HelloSDK` project. Run the app in the debug mode, and the following SDK version information will be printed in the Xcode console:

```
2017-09-26 16:16:15.767 HelloSDK[17929:7488566] SDK Version = 5.2.5541
```

Module Description

Read the documents below to learn more about different modules of the SDK.

- Video Shooting
- Video Editing
- Video Splicing
- Video Uploading
- Video Playback

SDK Integration (Android Studio)

最終更新日：：2025-04-01 18:10:24

Android Project Configuration

System requirements

We recommend you run the SDK on Android 5.0 (API level 21) or later.

Development environment

Below are the environment requirements for SDK development. You don't need to meet the same requirements for application development, but make sure that your application is compatible with the SDK.

- Android NDK: android-ndk-r12b
- Android SDK Tools: android-sdk_25.0.2
- minSdkVersion: 21
- targetSdkVersion: 26
- Android Studio (recommended)

Step 1. Integrate the SDK

Gradle

1. **Project Configuration**
   1.1 Specify dependencies in the build.gradle file located in your project's app directory
   - If using version 3.x of com.android.tools.build:gradle, add the following code:

```
dependencies {
implementation 'com.tencent.liteav:LiteAVSDK_UGC:latest.release'
}
```

   - If using version 2.x of com.android.tools.build:gradle, add the following code:

```
dependencies {
 compile 'com.tencent.liteav:LiteAVSDK_UGC:latest.release'
}
```

   1.2 In `defaultConfig`, specify the CPU architecture to be used by your application.

```
defaultConfig {
ndk {
    abiFilters "armeabi-v7a", "arm64-v8a"
}
}
```

> **Note:**
> Currently, the SDK supports armeabi-v7a and arm64-v8a.

2. Click **Sync Now**, wait for the SDK to download automatically, then build the project.

AAR

1. **Download the latest SDK.**

   Download Link: SDK Download. Copy the .aar files from the downloaded SDK directory to the specified local repository directory (the one added in Step 2.2 of the project configuration).

2. **Project Configuration**

   2.1. In the build.gradle file under your app module, add the dependency (replace LiteAVSDK_UGC_12.2.0.15065 with your actual SDK version):

   - If using version 3.x of com.android.tools.build:gradle, add the following code:

```
  dependencies {
     implementation(name: 'LiteAVSDK_UGC_12.2.0.15065', ext:
'aar')
  }
```

   - If using version 2.x of com.android.tools.build:gradle, add the following code:

```
    dependencies {
        compile(name: 'LiteAVSDK_UGC_12.2.0.15065', ext: 'aar')
    }
```

2.2. In the project-level build.gradle, add the flatDir configuration to specify the local repository directory (where the SDK files are copied):

```
allprojects {
 repositories {
        jcenter()
        flatDir {
                dirs 'libs'
        }
 }
}
```

2.3. In the build.gradle file under your app module, add the following under defaultConfig to specify NDK-compatible architectures:

```
  defaultConfig {
      ndk {
          abiFilters "armeabi-v7a", "arm64-v8a"
      }
  }
```

3. Click **Sync Now** to synchronize the project, then build it.

JAR + SO

1. **Download the latest SDK.**
   1.1 Download Link: SDK Download. Unzip the ZIP file in the SDK directory.

1.2 **Copy Files**

If you have not specified the JNI library loading path for your project, we recommend you copy the .jar package and the .so library for the corresponding architecture obtained in the previous step to the **Demo\app\src\main\jniLibs** directory (Android Studio's default path for loading JNI libraries) and put the .jar package in the `libs` folder.

2. **Project Configuration**

2.1 In the build.gradle file under your app module, add the following to reference the JAR and SO libraries:

- If using version 3.x of com.android.tools.build:gradle, add the following code:

```
dependencies {
 compile fileTree(dir: 'libs', include: ['*.jar'])
 // Import the JAR file of the UGSV SDK
 compile fileTree(dir: 'src/main/jniLibs', includes:
['*.jar'])
 ...
}
```

- If using version 2.x of com.android.tools.build:gradle, add the following code:

```
dependencies {
 compile fileTree(dir: 'libs', include: ['*.jar'])
 // Import the JAR file of the UGSV SDK
```

```
 compile fileTree(dir: 'src/main/jniLibs', includes:
['*.jar'])
 ...
}
```

2.2 In the same build.gradle file, under defaultConfig, specify NDK-compatible architectures:

```
defaultConfig {
    ndk {
            abiFilters "armeabi-v7a", "arm64-v8a"
    }
}
```

3. Click **Sync Now** to synchronize the project, then build it.

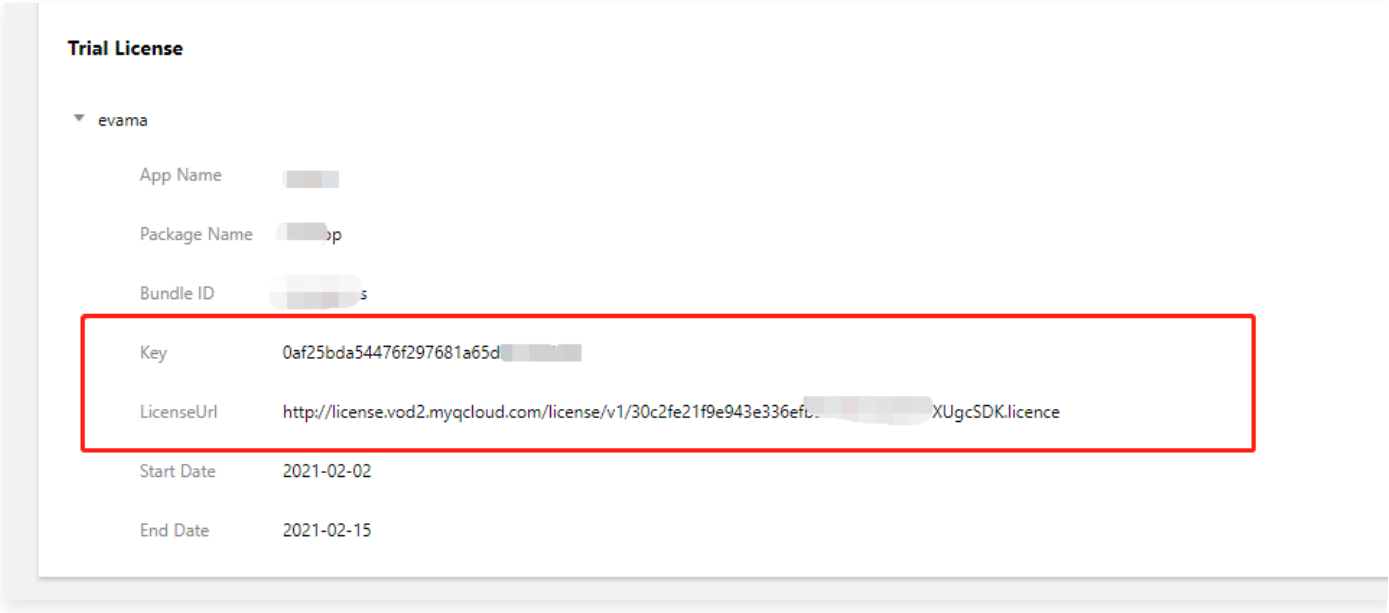
Step 2. Configure app permissions

Configure application permissions in `AndroidManifest.xml`. Audio/Video applications generally need the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.Camera"/>
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Step 3. Configure the license

1. After successfully obtaining a license, copy the license key and URL in the VOD console as shown below:



2. Before you use UGSV features in your application, we recommend that you complete the following configuration in `- Application onCreate()` :

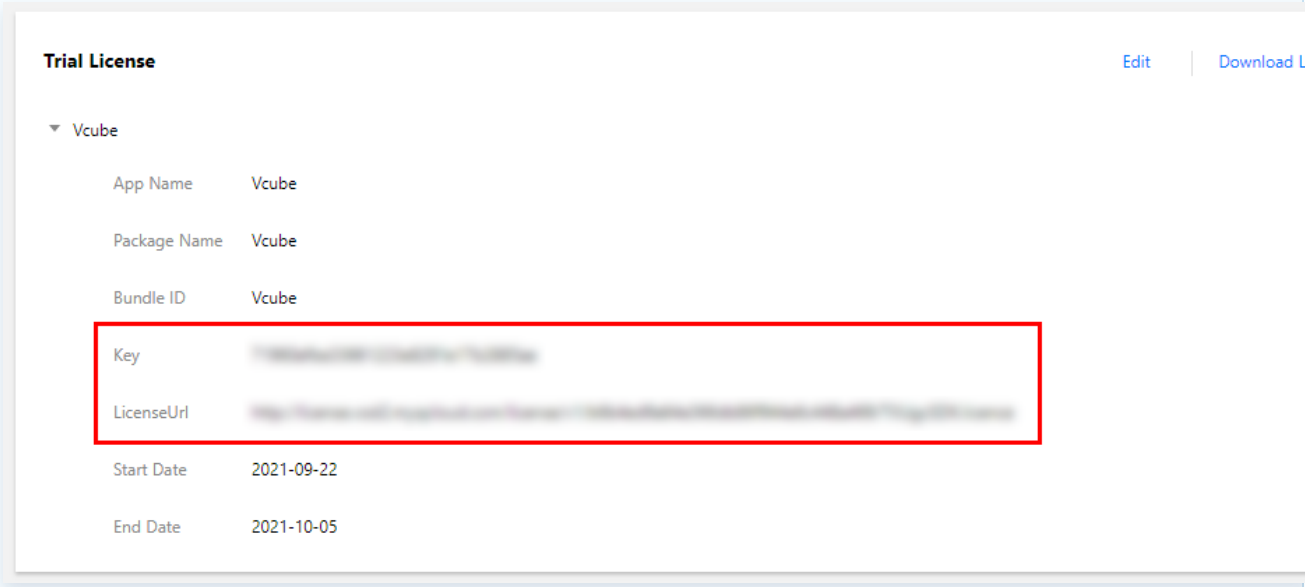
```
public class DemoApplication extends Application {
 String ugcLicenceUrl = ""; // Enter the license URL obtained from
the console.
 String ugcKey = "";        // Enter the license key obtained from the
console.

 @Override
 public void onCreate() {
     super.onCreate();
     TXUGCBase.getInstance().setLicence(instance, ugcLicenceUrl,
ugcKey);
 }
}
```

3. The latest SDK version includes short video license verification. If verification fails, you can use the following API to retrieve specific details from the license:

```
TXUGCBase.getInstance().getLicenceInfo(Context context);
```

Note:
If you use a license for the SDK on v4.7 and have upgraded the SDK to v4.9, you can click **Switch to New License** in the console to generate a new license key and URL. A new license can be used only for the SDK on v4.9 or later and should be configured as described above.



Step 4. Print logs

You can enable/disable console log printing and set the log level in `TXLiveBase`. See the sample code below.

- **setConsoleEnabled**

  Sets whether to print the SDK logs in the Android Studio console.

- **setLogLevel**

  It is used to set whether the SDK can print local logs. The SDK will write logs into the **Android/data/application package name/files/log/tencent/liteav** folder on the SD card by default. If you need technical support from Tencent Cloud, we recommend you enable this feature and provide the log file after reproducing the problem.

```
TXLiveBase.setConsoleEnabled(true);
TXLiveBase.setLogLevel(TXLiveConstants.LOG_LEVEL_DEBUG);
```

Troubleshooting

After importing the UGSV SDK, when you build and run your project, if the following error occurs:

```
Caused by: android.view.InflateException:
Binary XML file #14:Error inflating class
com.tencent.rtmp.ui.TXCloudVideoView
```

Follow the steps below to troubleshoot the problem:

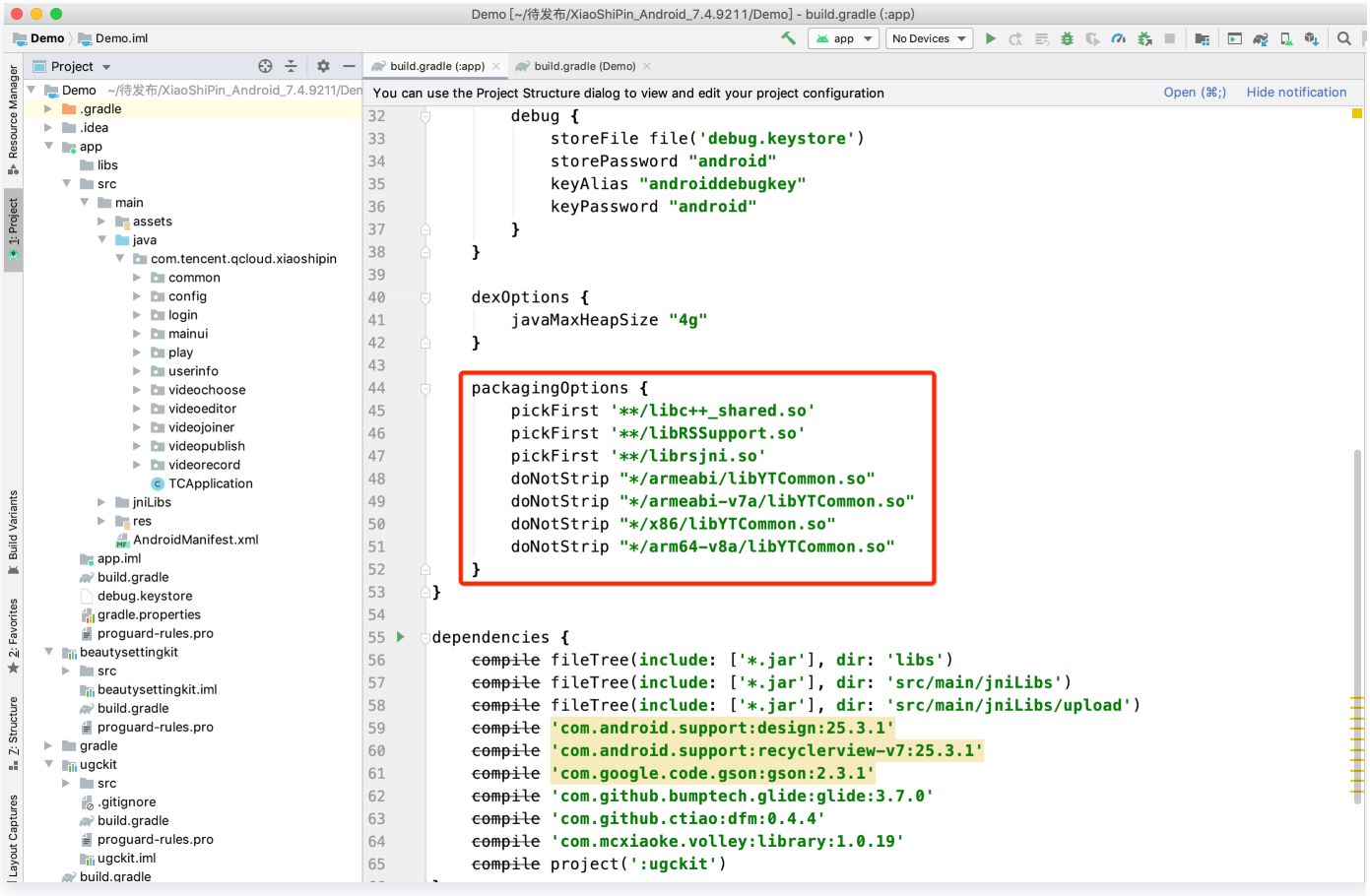
1. Check whether you have copied the JAR and SO files to the `jniLibs` directory.
2. If you use the full edition integrated with the .aar file, check whether the .so libraries for the x64 architecture are filtered out in `defaultConfig` in `build.gradle` in the project directory.

```
defaultConfig {
 ...
 ndk {
     abiFilters "armeabi-v7a", "arm64-v8a"
 }
}
```

3. Check if the package name of the SDK has been added to the "do not obfuscate" list.

```
-keep class com.tencent.** { *;}
```

4. Configure packaging options for your application.



Detailed description

See the documents below for a detailed description of different UGSV modules.

- Video Shooting
- Video Editing
- Video Splicing
- Video Upload
- Video Playback

Capturing and Shoot
Capturing and Shoot
iOS

最終更新日：：2025-04-01 17:14:08

You can implement video shooting features including speed change, beautification, filters, audio effects, and background music.

Overview of Relevant Classes

API File	Description
`TXUGCRecord.h`	Shooting videos
`TXUGCRecordListener.h`	Shooting callbacks
`TXUGCRecordEventDef.h`	Shooting events
`TXUGCRecordTypeDef.h`	Definitions of basic parameters
`TXUGCPartsManager.h`	Video segment management class, which is used for multi-segment shooting and segment deletion

Basic Workflow

1. Configure shooting parameters.
2. Enable preview.
3. Set shooting effects.
4. End shooting.

Basic code example for starting preview/recording(The following snippet demonstrates the core logic; implement the full code as needed):

```
- (void)viewDidLoad {
    // Create a view to display camera preview
    _videoRecordView = [[UIView alloc] initWithFrame:self.view.bounds];
    [self.view addSubview:_videoRecordView];
```

```
    // 1. Configure recording parameters
    TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
    param.videoQuality = VIDEO_QUALITY_MEDIUM;

    // 2. Start preview, set parameters and specify the view for preview
    [[TXUGCRecord shareInstance] startCameraSimple:param
preview:_videoRecordView];

    // 3. Set recording effects (example: add beauty filter)
    TXBeautyManager *manager = [[TXUGCRecord shareInstance]
getBeautyManager];
    [manager setBeautyStyle:TXBeautyStyleSmooth];
    [manager setBeautyLevel:5];

    // Set the delegate for video recording to receive progress and
completion notifications
    [TXUGCRecord shareInstance].recordDelegate = self;

    // Start recording
    [[TXUGCRecord shareInstance] startRecord];
    。。。。。。

    // Stop recording
    [[TXUGCRecord shareInstance] stopRecord];
}

// Recording completion callback
-(void) onRecordComplete:(TXUGCRecordResult*)result
{
    if (result.retCode == UGC_RECORD_RESULT_OK) {
        // Recording succeeded. Video file is at result.videoPath
    } else {
        // Error handling. For error codes, refer to TXUGCRecordResultCode
in TXUGCRecordTypeDef.h
    }
}
@end
```

Preview

`TXUGCRecord` in `TXUGCRecord.h` is used to implement video shooting. The first step of shooting videos is using `startCameraSimplePreview` to enable preview. Because mic and camera permissions are required for preview, you need to configure pop-up windows to request the permissions.

1. Start Preview

```
[TXUGCRecord shareInstance].recordDelegate = self; // Set recording
callback (refer to TXUGCRecordListener for callback methods)

// Configure camera and start preview
TXUGCSimpleConfig *param = [[TXUGCSimpleConfig alloc] init];
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH;  // 720p
param.frontCamera = YES;      // Use front camera
param.minDuration = 5;        // Minimum recording duration: 5s
param.maxDuration = 60;       // Maximum recording duration: 60s
param.touchFocus = NO;        // NO: autofocus; YES: manual focus

// Display camera preview in self.previewView
[[TXUGCRecord shareInstance] startCameraSimple:param
preview:self.previewView];

// Stop camera preview
[[TXUGCRecord shareInstance] stopCameraPreview];
```

2. Modify Preview Parameters

To modify preview parameters after the camera is turned on, refer to the code below:

```
// Switch video recording resolution to 540p (540x960)
[[TXUGCRecord shareInstance] setVideoResolution:
VIDEO_RESOLUTION_540_960];

// Switch video bitrate to 6500 Kbps
[[TXUGCRecord shareInstance] setVideoBitrate: 6500];

// Set zoom level to 3.
// Value 1: farthest view (normal lens); Value 5: closest view (zoomed-
in lens)
```

```
[[TXUGCRecord shareInstance] setZoom: 3];

// Switch camera: YES for front camera, NO for rear camera
[[TXUGCRecord shareInstance] switchCamera: NO];

// Toggle flash: YES to turn on, NO to turn off
[[TXUGCRecord shareInstance] toggleTorch: YES];

// Set custom video processing callback delegate
[TXUGCRecord shareInstance].videoProcessDelegate = delegate;
```

Photo Capture

After starting the camera preview, you can use the photo capture functionality.

```
// Take a photo. Before calling this API, you must start the recording
preview
// by calling either `startCameraSimplePreview` or
`startCameraCustomPreview`
[[TXUGCRecord shareInstance] snapshot:^(UIImage *image) {
    // Process the captured image here
 }];
```

Shooting Control

The shoot can be started, paused, and resumed as follows:

```
// Start recording.
// This method does not specify the recording file path; the file path
will be returned in the recording completion callback.
[[TXUGCRecord shareInstance] startRecord];

// Start recording with a specified output video file path and cover
image path
[[TXUGCRecord shareInstance] startRecord:videoFilePath
coverPath:coverPath];

// Start recording with a specified output video file path, video
segments folder path, and cover image path
```

```
[[TXUGCRecord shareInstance] startRecord:videoFilePath
videoPartsFolder:videoPartFolder coverPath:coverPath];

// Pause recording
[[TXUGCRecord shareInstance] pauseRecord];

// Resume recording
[[TXUGCRecord shareInstance] resumeRecord];

// Stop recording
[[TXUGCRecord shareInstance] stopRecord];
```

The shooting progress and result callbacks are implemented by `TXUGCRecordListener` (defined in `TXUGCRecordListener.h`).

- `onRecordProgress` is the shooting progress callback. The `millisecond` parameter indicates the recorded duration in milliseconds.

```
@optional
 (void)onRecordProgress:(NSInteger)milliSecond;
```

- `onRecordComplete` is the shooting result callback. The `retCode` and `descMsg` fields in `TXRecordResult` indicate the error code and error message respectively. `videoPath` indicates the path of the video, and `coverImage` is the first frame of the video, which is used as the thumbnail.

```
@optional
 (void)onRecordComplete:(TXUGCRecordResult*)result;
```

- `onRecordEvent` is the callback reserved for the shooting event and is not used currently.

```
@optional
 (void)onRecordEvent:(NSDictionary*)evt;
```

Shooting Settings

1. Video Image

```
// Set orientation for preview
```

```
// The valid values for `rotation` are 0, 90, 180, and 270, which
indicate the clockwise rotation angle.
// You must set the rotation before you call `startRecord` for the
setting to take effect.
[[TXUGCRecord shareInstance] setRenderRotation:rotation];

// Set the aspect ratio
// VIDEO_ASPECT_RATIO_9_16: 9:16
// VIDEO_ASPECT_RATIO_3_4: 3:4
// VIDEO_ASPECT_RATIO_1_1: 1:1
// You must set the rotation before you call `startRecord` for the
setting to take effect.
[[TXUGCRecord shareInstance] setAspectRatio:VIDEO_ASPECT_RATIO_9_16];
```

2. Speed

```
// Set the shooting speed
//    VIDEO_RECORD_SPEED_SLOWEST: Very slow
//    VIDEO_RECORD_SPEED_SLOW: Slow
//    VIDEO_RECORD_SPEED_NOMAL: Original
//    VIDEO_RECORD_SPEED_FAST: Fast
//    VIDEO_RECORD_SPEED_FASTEST: Very fast
[[TXUGCRecord shareInstance] setRecordSpeed:VIDEO_RECORD_SPEED_NOMAL];
```

3. Audio

```
// Set the mic volume. This is used to control the volume of the mic
when background music is mixed.
// Volume. The normal volume is 1. We recommend 0-2, but you can set it
to a larger value if you want louder music.
[[TXUGCRecord shareInstance] setMicVolume:volume];

// Mute/Unmute. The `isMute` parameter specifies whether to mute audio.
Audio is unmuted by default.
[[TXUGCRecord shareInstance] setMute:isMute];
```

Effects

You can add various effects to your video during shooting.

1. Watermarks

```
// Add a global watermark
// normalizationFrame: The normalized position of the watermark in
relation to the video. The SDK calculates the watermark height based on
the aspect ratio.
// Suppose the video dimensions are 540 x 960, and `frame` is set to
`(0.1, 0.1, 0.1, 0)`.
// The actual coordinates of the watermark would be:
// (540*0.1, 960*0.1, 540*0.1, 540*0.1*waterMarkImage.size.height /
waterMarkImage.size.width)
[[TXUGCRecord shareInstance] setWaterMark:waterMarkImage
normalizationFrame:frame)
```

2. Filters

```
// Set the filter style
// Set the color filter: Romantic, refreshing, elegant, pink, retro, and
more
// filterImage: The color lookup table, which must be in PNG format.
// The color lookup table used in the demo is in
`FilterResource.bundle`.
[[TXUGCRecord shareInstance] setFilter:filterImage];

 // Set the strength of filters. Value range: 0-1. Default: 0.5. The
greater the value, the stronger the filter.
[[TXUGCRecord shareInstance] setSpecialRatio:ratio];

// Set a filter combination
// mLeftBitmap: The left filter
// leftIntensity: The strength of the left filter
// mRightBitmap: The right filter
// rightIntensity: The strength of the right filter
// leftRatio: The ratio of the width of the left picture to the video
width
// You can use this API to implement "swipe to change filter".
[[TXUGCRecord shareInstance] setFilter:leftFilterImgage
leftIntensity:leftIntensity rightFilter:rightFilterImgage
rightIntensity:rightIntensity leftRatio:leftRatio];
```

3. Beauty

```
// Access Beauty Configuration Interface
TXBeautyManager *manager = [[TXUGCRecord shareInstance]
getBeautyManager];
// Set beauty style:   (TXBeautyStyleSmooth: Smooth;
TXBeautyStyleNature: Natural; TXBeautyStylePitu: Pitu)
[manager setBeautyStyle:TXBeautyStyleSmooth];
// Set beauty level (0-9)
[manager setBeautyLevel:5];
// Set whitening level (0-9)
[manager setWhitenessLevel:5];
// Set rosy level (0-9)
[manager setRuddyLevel:5];
```

Advanced Features

- Multi-segment shooting
- Drafts
- Adding background music
- Voice changing and reverb
- Customizing video data

Android

最終更新日：：2025-04-01 17:14:08

Video shoot includes features such as adjustable-speed shoot, beauty filters, filters, sound effects, and background music configuration.

Overview of Relevant Classes

Class	Feature
TXUGCRecord	Video shoot implementation
TXUGCPartsManager	Video segment management class, which is used to shoot multiple video segments and delete existing segments
ITXVideoRecordListener	Shoot callback
TXRecordCommon	Basic parameter definition, including video shoot callback and release callback APIs

Basic Workflow

1. Configure the shoot parameters.
2. Start video image preview.
3. Set the shoot effects.
4. Complete shoot.

Basic code example for starting preview/recording(The following snippet demonstrates the core logic; implement the full code as needed):

```
TXUGCRecord mTXUGCRecord = TXUGCRecord.getInstance(context);

// Create a TXCloudVideoView for camera preview
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view);

// 1. Configure recording parameters (using recommended
TXUGCSimpleConfig)
TXRecordCommon.TXUGCSimpleConfig param = new
TXRecordCommon.TXUGCSimpleConfig();
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM;
```

```
// 2. Start camera preview
mTXUGCRecord.startCameraSimplePreview(param, mVideoView);

// 3. Set recording effects (example: add beauty filter)
mTXUGCRecord.getBeautyManager().setBeautyStyle(TXBeautyManager.TXBeautyS
tyleSmooth);
mTXUGCRecord.getBeautyManager().setBeautyLevel(5);

// 4. Set recording event callback listener
mTXUGCRecord.setVideoRecordListener(new ITXVideoRecordListener() {
    @Override
    public void onRecordEvent(int eventId, Bundle bundle) {}

    @Override
    public void onRecordProgress(long millisecond) {}

    @Override
    public void onRecordComplete(TXRecordResult result) {}
});

// Start recording
int result = mTXUGCRecord.startRecord();

// Stop recording
mTXUGCRecord.stopRecord();
```

Previewing Video Image

`TXUGCRecord` (in `TXUGCRecord.java`) is used for short video shoot. The preview feature needs to be implemented first, where the `startCameraSimplePreview` function is used to start preview. As camera and mic need to be enabled before the preview can be started, prompt windows for permission application may pop up at this point.

1. Start Preview

```
// Get the singleton instance of the recorder
TXUGCRecord mTXUGCRecord =
TXUGCRecord.getInstance(this.getApplicationContext());
```

```
// Configure basic recording parameters
TXRecordCommon.TXUGCSimpleConfig param = new
TXRecordCommon.TXUGCSimpleConfig();
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH;  // 720p
param.isFront = true;            // Whether to use the front camera
param.minDuration = 5000;       // Minimum recording duration (ms)
param.maxDuration = 60000;      // Maximum recording duration (ms)
param.touchFocus = false;        // false: auto-focus; true: manual
focus

// Video preview view
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view);

// Start preview
mTXUGCRecord.startCameraSimplePreview(param, mVideoView);

// Stop camera preview
mTXUGCRecord.stopCameraPreview();
```

2. Modify Preview Parameters

After starting the camera, you can adjust the preview parameters as follows:

```
// Switch video recording resolution to 540p (540x960)
mTXUGCRecord.setVideoResolution(TXRecordCommon.VIDEO_RESOLUTION_540_960)
;

// Switch video recording bitrate to 6500 Kbps
mTXUGCRecord.setVideoBitrate(6500);

// Get the maximum zoom level supported by the camera
mTXUGCRecord.getMaxZoom();

// Set zoom level to 3. Valid range: 0 ~ getMaxZoom()
mTXUGCRecord.setZoom(3);

// Switch camera: true for front camera, false for rear camera
mTXUGCRecord.switchCamera(false);

// Toggle flash: true to turn on, false to turn off
```

```
mTXUGCRecord.toggleTorch(false);

// When param.touchFocus is true (manual focus mode), use this API to
set focus position
mTXUGCRecord.setFocusPosition(eventX, eventY);

// Set custom video processing callback
mTXUGCRecord.setVideoProcessListener(this);
```

Photo Capture

After enabling the camera preview, you can use the photo capturing feature.

```
// Take a photo. Before calling this API, you must start the recording
preview by calling either startCameraSimplePreview or
startCameraCustomPreview
mTXUGCRecord.snapshot(new TXRecordCommon.ITXSnapshotListener() {
    @Override
    public void onSnapshot(Bitmap bmp) {
        // Save or display the captured image
    }
});
```

Shooting Control

The shoot can be started, paused, and resumed as follows:

```
// Start recording
// This method does not specify the recording file path. The path will
be returned in the recording completion callback.
mTXUGCRecord.startRecord();

// Start recording with specified output video file path and cover image
path
mTXUGCRecord.startRecord(videoFilePath, coverPath);

// Start recording with specified output video file path, video segments
folder path, and cover image path
mTXUGCRecord.startRecord(videoFilePath, videoPartFolder, coverPath);
```

```
// Pause recording
mTXUGCRecord.pauseRecord();

// Resume recording
mTXUGCRecord.resumeRecord();

// Stop recording
mTXUGCRecord.stopRecord();
```

The shoot process and result will be returned through the `TXRecordCommon.ITXVideoRecordListener` API (defined in `TXRecordCommon.java`):

- `onRecordProgress` returns the shoot progress, and the `millisecond` parameter indicates the shoot duration in milliseconds.

```
@optional
void onRecordProgress(long milliSecond);
```

- `onRecordComplete` returns the shoot result, the `retCode` and `descMsg` fields of `TXRecordResult` indicate the error code and error message, respectively, `videoPath` indicates the path of the shot short video file, and `coverImage` indicates the short video's first-frame image that is automatically captured and will be used in video release.

```
@optional
void onRecordComplete(TXRecordResult result);
```

- `onRecordEvent` is the shoot event callback, which contains the event ID and event-related parameters in the format of (key,value).

```
@optional
void onRecordEvent(final int event, final Bundle param);
```

Shooting Settings

1. Video Image

```
// Set orientation for preview
```

```
// The valid values for `rotation` are 0, 90, 180, and 270, which
indicate the clockwise rotation angle.
// You must set the rotation before you call `startRecord` for the
setting to take effect.
mTXUGCRecord.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT)
;

// Set the aspect ratio
// VIDEO_ASPECT_RATIO_9_16: 9:16
// VIDEO_ASPECT_RATIO_3_4: 3:4
// VIDEO_ASPECT_RATIO_1_1: 1:1
// You must set the rotation before you call `startRecord` for the
setting to take effect.
mTXUGCRecord.setAspectRatio(TXRecordCommon.VIDEO_ASPECT_RATIO_9_16);
```

2. Speed

```
// Set the video shoot speed
// TXRecordCommon.RECORD_SPEED_SLOWEST (ultra-slow)
// TXRecordCommon.RECORD_SPEED_SLOW (slow)
// TXRecordCommon.RECORD_SPEED_NORMAL (standard)
// TXRecordCommon.RECORD_SPEED_FAST (fast)
// TXRecordCommon.RECORD_SPEED_FASTEST (ultra-fast)
mTXUGCRecord.setRecordSpeed(TXRecordCommon.VIDEO_RECORD_SPEED_NORMAL);
```

3. Audio

```
// Set the mic volume. This is used to control the volume of the mic
when background music is mixed.
// Volume. The normal volume is 1. We recommend 0-2, but you can set it
to a larger value if you want louder music.
mTXUGCRecord.setMicVolume(volume);

// Mute/Unmute. The `isMute` parameter specifies whether to mute audio.
Audio is unmuted by default.
mTXUGCRecord.setMute(isMute);
```

Effects

You can add various effects to your video during shooting.

1. Watermark

```
// Add a global watermark
// normalizationFrame: The normalized position of the watermark in
relation to the video. The SDK calculates the watermark height based on
the aspect ratio.
// Suppose the video dimensions are 540 x 960, and `frame` is set to
`(0.1, 0.1, 0.1, 0)`.
// The actual coordinates of the watermark would be:
// (540*0.1, 960*0.1, 540*0.1, 540*0.1*waterMarkImage.size.height /
waterMarkImage.size.width)
mTXUGCRecord.setWatermark(watermarkBitmap, txRect)
```

2. Filter

```
// Set the filter style
// Set the color filter: Romantic, refreshing, elegant, pink, retro, and
more
// filterImage: The color lookup table, which must be in PNG format.
// The color lookup table used in the demo is in
`FilterResource.bundle`.
mTXUGCRecord.setFilter(filterBitmap);

 // Set the strength of filters. Value range: 0-1. Default: 0.5. The
greater the value, the stronger the filter.
mTXUGCRecord.setSpecialRatio(0.5);

// Set a filter combination
// mLeftBitmap: The left filter
// leftIntensity: The strength of the left filter
// mRightBitmap: The right filter
// rightIntensity: The strength of the right filter
// leftRatio: The ratio of the width of the left picture to the video
width
// You can use this API to implement "swipe to change filter".
```

```
mTXUGCRecord.setFilter(mLeftBitmap, leftIntensity, mRightBitmap,
rightIntensity, leftRatio);
```

3. Beauty

```
// Access Beauty Configuration Interface
TXBeautyManager mTXBeautyManager  = mTXCameraRecord.getBeautyManager();
// Set beauty style:   (TXBeautyStyleSmooth: Smooth;
TXBeautyStyleNature: Natural; TXBeautyStylePitu: Pitu)
mTXBeautyManager.setBeautyStyle(TXBeautyManager.TXBeautyStyleSmooth);
// Set beauty level (0-9)
mTXBeautyManager.setBeautyLevel(5);
// Set whitening level (0-9)
mTXBeautyManager.setWhitenessLevel(5);
// Set rosy level (0-9)
mTXBeautyManager.setRuddyLevel(5);
```

Advanced Features

Multi-segment shoot
Shoot drafts
Adding background music
Voice changing and reverb
Customizing video data

Multi-Segment Shoot
iOS

最終更新日：：2025-04-01 17:14:08

Basic Workflow for Multi-Segment Video Recording:

1. Start preview.
2. Begin recording.
3. Pause recording.
4. Resume recording.
5. Stop recording.

Each pause generates a video segment. Use TXUGCPartsManager to manage recorded segments, and finally merge them.

```
// Start custom preview
recorder = [TXUGCRecord shareInstance];
[recorder startCameraCustom:param preview:preview];

// Start recording
[recorder startRecord];

// Pausing recording generates a segment, accessible via
TXUGCPartsManager
[recorder pauseRecord];

// Get the video segment manager
TXUGCPartsManager *partsManager = recorder.partsManager;

// Delete the last recorded segment
[partsManager deleteLastPart];

// Resume recording
[recorder resumeRecord];

// Stop recording and merge all segments into a single video
[recorder stopRecord];

// Get total duration of all segments
```

```
[partsManager getDuration];

// Get paths of all segments
[partsManager getVideoPathList];

// Delete the last segment
[partsManager deleteLastPart];

// Delete a specific segment (e.g., index 1)
[partsManager deletePart:1];

// Delete all segments
[partsManager deleteAllParts];

// Insert an external video (non-recorded) into the segment list
[partsManager insertPart:videoPath atIndex:0];

// Merge all segments into a final video (also triggered automatically
on stopRecord)
[partsManager joinAllParts:videoOutputPath complete:complete];
```

Android

最終更新日：：2025-04-01 17:14:08

Basic Workflow for Multi-Segment Video Recording:

1. Start preview.
2. Begin recording.
3. Pause recording.
4. Resume recording.
5. Stop recording.

Each pause generates a video segment. Use TXUGCPartsManager to manage recorded segments, and finally merge them.

```
// Start recording
mTXUGCRecord.startRecord();

// After pausing recording, a video segment will be generated and can be
managed via TXUGCPartsManager
mTXUGCRecord.pauseRecord();

// Get the segment manager instance
TXUGCPartsManager mTXUGCPartsManager = mTXUGCRecord.getPartsManager();

// Delete the last recorded video segment
mTXUGCPartsManager.deleteLastPart();

// Resume recording
mTXUGCRecord.resumeRecord();

// Stop recording and merge all segments into a single output video
mTXUGCRecord.stopRecord();

/////////////////////////////////////////////////
// Video Segment Management APIs
/////////////////////////////////////////////////

// Get total duration of all current video segments
mTXUGCPartsManager.getDuration();
```

```
// Get paths of all video segments
mTXUGCPartsManager.getPartsPathList();

// Delete the last video segment
mTXUGCPartsManager.deleteLastPart();

// Delete a specific video segment by index
mTXUGCPartsManager.deletePart(index);

// Delete all video segments
mTXUGCPartsManager.deleteAllParts();

// Insert an external video (not from current recording) into the
segment list
mTXUGCPartsManager.insertPart(videoPath, index);
```

Shoot Drafts
iOS

最終更新日：：2024-12-23 14:53:57

How the shooting drafts feature works

Starting a shooting

1. Start shooting a video.
2. Pause/End the shooting.
3. Cache the video segment locally (draft box).

Resuming the shooting

1. Preload the locally cached video segment.
2. Continue with the shooting.
3. End the shooting.

```
//Get the object of the previous shooting
record = [TXUGCRecord shareInstance];

//Start shooting a video.
[record startRecord];

//Pause the shooting and cache the video segment
[record pauseRecord:^{
NSArray *videoPathList = record.partsManager.getVideoPathList;
//Set `videoPathList` to a local path.
}];

//Get the object of the resumed shooting.
record2 = [TXUGCRecord shareInstance];

//Preload the locally cached video segment.
[record2.partsManager insertPart:videoPath atIndex:0];

//Start the shooting
[record2 startRecord];
```

```
//End the shooting. The SDK will splice together the two video segments.
[record2 stopRecord];
```

Note:

For detailed instructions, see the `UGCKitRecordViewController` class in (Demo) Source Code for All-Feature UGSV Apps.

Android

最終更新日：：2022-05-24 16:02:37

You can implement the drafts logic as follows:

First Shooting

1. Start shooting.
2. Pause/End the first shooting.
3. Cache the video segment locally (in drafts).

Second Shooting

1. Preload the locally cached video segment.
2. Resume shooting.
3. End shooting.

```
// Get the first video shooting object
mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext());

// Start shooting
mTXCameraRecord.startRecord();

// Pause shooting
mTXCameraRecord.pauseRecord();

// Get the cached video segment and write it locally
List<String> pathList =
mTXCameraRecord.getPartsManager().getPartsPathList(); // Write
`pathList` locally

// Open the application again and get the shooting object
mTXCameraRecord2 =
TXUGCRecord.getInstance(this.getApplicationContext());

// Preload the locally cached segment
mTXCameraRecord2.getPartsManager().insertPart(videoPath, 0);

// Start shooting
mTXCameraRecord2.startRecord();
```

```
// End shooting, and the SDK will compose the cached video segment with
the currently shot one
mTXCameraRecord2.stopRecord();
```

Note:
For the specific implementation method, please see the usage of the `RecordDraftManager` class in the shooting module in the UGSV application demo source code.

Adding Background Music
iOS

最終更新日：：2025-04-01 17:14:08

Adding Background Music During Shooting

```
// Get the recorder instance
TXUGCRecord *recorder = [TXUGCRecord shareInstance];

// Set BGM file path
[recorder setBGMAsset:path];

// Set BGM using AVAsset loaded from the system media library
[recorder setBGMAsset:asset];

// Play BGM
[recorder playBGMFromTime:beginTime
                   toTime:endTime
          withBeginNotify:^(NSInteger errCode) {
              // Playback start callback. errCode 0: success, others:
failure
          } withProgressNotify:^(NSInteger progressMS, NSInteger
durationMS) {
              // progressMS: elapsed time (ms), durationMS: total
duration (ms)
          } andCompleteNotify:^(NSInteger errCode) {
              // Playback complete callback. errCode 0: success, others:
failure
          }];

// Stop BGM playback
[recorder stopBGM];

// Pause BGM playback
[recorder pauseBGM];

// Resume BGM playback
```

```
[recorder resumeBGM];

// Set microphone volume (used during BGM mixing)
// volume: 1.0 is normal volume. Recommended range: 0-2. Higher values
allowed for amplification.
[recorder setMicVolume:1.0];

// Set BGM volume (used during BGM mixing)
// volume: 1.0 is normal volume. Recommended range: 0-2. Higher values
allowed for amplification.
[recorder setBGMVolume:1.0];
```

Android

最終更新日：：2025-04-01 17:14:08

Adding Background Music During Shooting

```
// Set BGM path
mTXUGCRecord.setBGM(path);

// Play BGM from startTime to endTime
mTXUGCRecord.playBGMFromTime(startTime, endTime)

// Stop BGM playback
mTXUGCRecord.stopBGM();

// Pause BGM playback
mTXUGCRecord.pauseBGM();

// Resume BGM playback
mTXUGCRecord.resumeBGM();

// Set the volume of background music (BGM). Used when playing
background music mixing to control BGM volume level
// Volume value: 1 is normal volume, recommended range 0~2. For higher
BGM volume, larger values can be set
mTXUGCRecord.setBGMVolume(x);

// Set the start and end positions for BGM playback. Must be called
before startPlay; invalid if called while paused
mTXUGCRecord.seekBGM(startTime, endTime);
```

Voice Changing and Reverb
iOS

最終更新日：：2022-05-24 16:06:06

Voice changing and reverb for video shooting:

```
// Get the `recorder` object
recorder = [TXUGCRecord shareInstance];

// Set reverb
// TXRecordCommon.VIDOE_REVERB_TYPE_0   Disable reverb
// TXRecordCommon.VIDOE_REVERB_TYPE_1   Karaoke room
// TXRecordCommon.VIDOE_REVERB_TYPE_2   Small room
// TXRecordCommon.VIDOE_REVERB_TYPE_3   Big hall
// TXRecordCommon.VIDOE_REVERB_TYPE_4   Deep
// TXRecordCommon.VIDOE_REVERB_TYPE_5   Resonant
// TXRecordCommon.VIDOE_REVERB_TYPE_6   Metallic
// TXRecordCommon.VIDOE_REVERB_TYPE_7   Husky
[recorder setReverbType:VIDOE_REVERB_TYPE_1];

// Set voice changing
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0   Disable voice changing
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1   Naughty boy
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2   Little girl
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3   Middle-aged man
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4   Heavy metal
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6   Non-native speaker
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7   Furious animal
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8   Chubby
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9   Strong electric current
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10   Robot
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11   Ethereal voice
[record setVoiceChangerType:VIDOE_VOICECHANGER_TYPE_1];
```

Note:
Voice changing and reverb take effect only for recorded human voice but not for background music.

Android

最終更新日：：2025-04-01 17:14:08

Voice changing and reverb for video shooting:

```
// Set reverb
// TXRecordCommon.VIDOE_REVERB_TYPE_0   Disable reverb
// TXRecordCommon.VIDOE_REVERB_TYPE_1   Karaoke room
// TXRecordCommon.VIDOE_REVERB_TYPE_2   Small room
// TXRecordCommon.VIDOE_REVERB_TYPE_3   Big hall
// TXRecordCommon.VIDOE_REVERB_TYPE_4   Deep
// TXRecordCommon.VIDOE_REVERB_TYPE_5   Resonant
// TXRecordCommon.VIDOE_REVERB_TYPE_6   Metallic
// TXRecordCommon.VIDOE_REVERB_TYPE_7   Husky
mTXCameraRecord.setReverb(TXRecordCommon.VIDOE_REVERB_TYPE_1);

// Set voice changing
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_0   Disable voice changing
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_1   Naughty boy
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_2   Little girl
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_3   Middle-aged man
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_4   Heavy metal
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_6   Non-native speaker
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_7   Furious animal
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_8   Chubby
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_9   Strong electric current
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_10   Robot
// TXRecordCommon.VIDOE_VOICECHANGER_TYPE_11   Ethereal voice
mTXCameraRecord.setVoiceChangerType(TXRecordCommon.VIDOE_VOICECHANGER_TY
PE_1);
```

Note:
Voice changing and reverb take effect only for recorded human voice but not for background music.

Preview, Clipping, and Splicing
Video Editing
iOS

最終更新日：：2022-05-24 15:54:21

Feature Overview

Video editing includes features such as video clipping, time-based special effects (slow motion, reverse, and loop), special effect filters (dynamic light-wave, darkness and phantom, soul out, and cracked screen), filter styles (aesthetic, rosy, blues, etc.), music mix, animated stickers, static stickers, and bubble subtitles.

Overview of Relevant Classes

Class Name	Feature
TXVideoInfoReader.h	Gets media information
TXVideoEditer.h	Edits video

Use Instructions

The following is the basic usage process of video editing:

1. Set the video path.
2. Add effects.
3. Generate a video and output it to a specified file.
4. Listen on the generation event.

Sample

```
// Here, `Common/UGC/VideoPreview` in the demo is used as the preview
view
#import "VideoPreview.h"

@implementation EditViewController
{
    TXVideoEditer *editor;
    VideoPreview *_videoPreview;
}
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _videoPreview = [[VideoPreview alloc]
initWithFrame:self.view.bounds];
    [self.view addSubview:_videoPreview];
    // Edit preview parameters
    TXPreviewParam *param = [[TXPreviewParam alloc] init];
    param.videoView = _videoPreview.renderView;
    param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;

    // 1. Initialize the editor. If you do not need preview, you can pass
in `nil` or directly call the `init` method
    TXVideoEditer *editor = [[TXVideoEditer alloc]
initWithPreview:param];

    // Set the source video path
    NSString *path = [[NSBundle mainBundle] pathForResource:@"demo"
ofType:@"mp4"]
    [editor setVideoPath: path];

    // Configure the delegation
    editor.generateDelegate = self;      // Set the callback delegation
object of the generation event, which can be used to get the generation
progress and result

    // 2. Process the video. Watermarking is used as an example here
    [editor setWaterMark:[UIImage imageNamed:@"water_mark"]
          normalizationFrame:CGRectMake(0,0,0.1,0)];
}

// 3. Generate the video. Response to a user click is used as an example
here
- (IBAction)onGenerate:(id)sender {
    NSString *output = [NSTemporaryDirectory()
stringByAppendingPathComponent:@"temp.mp4"];
    [editor generateVideo:VIDEO_COMPRESSED_720P videoOutputPath:output];
}

// 4. Get the generation progress
-(void) onGenerateProgress:(float)progress
```

```
{
}

// Get the generation result
-(void) onGenerateComplete:(TXGenerateResult *)result
{
    if (result.retCode == 0) {
        // Generated successfully
    } else {
        // Generation failed. For the specific cause, please see
`result.descMsg`.
    }
}
@end
```

Getting Video Information

The `getVideoInfo` method of `TXVideoInfoReader` can get certain basic information of a specified video file. The relevant APIs are as detailed below:

```
// Get the video file information
+ (TXVideoInfo *)getVideoInfo:(NSString *)videoPath;

/** Get the video file information
 * @param videoAsset   Video file attributes
 * @return   Video information
 */
+ (TXVideoInfo *)getVideoInfoWithAsset:(AVAsset *)videoAsset;
```

The returned `TXVideoInfo` is defined as follows:

```
/// Video information
@interface TXVideoInfo : NSObject
/// Image of the first video frame
@property (nonatomic, strong) UIImage*              coverImage;
/// Video duration in seconds
@property (nonatomic, assign) CGFloat               duration;
/// Video size in bytes
@property (nonatomic, assign) unsigned long long    fileSize;
```

```
/// Video frame rate in fps
@property (nonatomic, assign) float                 fps;
/// Video bitrate in Kbps
@property (nonatomic, assign) int                   bitrate;
/// Audio sample rate
@property (nonatomic, assign) int                   audioSampleRate;
/// Video width
@property (nonatomic, assign) int                   width;
/// Video height
@property (nonatomic, assign) int                   height;
/// Video image rotation angle
@property (nonatomic, assign) int                   angle;
@end
```

Getting Thumbnail

The thumbnail APIs are mainly used to generate the preview thumbnails displayed on the video editing page, get the video cover, and perform other relevant operations.

1. Get the thumbnails evenly distributed along the video duration by number

`getSampleImages` of `TXVideoInfoReader` can get the specified number of thumbnails at the same time intervals:

```
/** Get the list of thumbnails of the video
 * @param count   Number of the thumbnails to be obtained (at even
sampling intervals)
 * @param maxSize   Maximum thumbnail dimensions. The dimensions of the
generated thumbnails will not exceed the specified width and height.
 * @param videoAsset   Video file attributes
 * @param sampleProcess   Sampling progress
 */
+ (void)getSampleImages:(int)count
                maxSize:(CGSize)maxSize
             videoAsset:(AVAsset *)videoAsset
               progress:(sampleProcess)sampleProcess;
```

`VideoRangeSlider` in the SDK uses `getSampleImages` to get 10 thumbnails so as to construct a progress bar consisting of video preview images.

2. Get thumbnails according to the list of points in time

```
/**
 * Get the thumbnails according to the list of points in time
 * @param asset   Video file object
 * @param times   List of points in time for getting thumbnails
 * @param maxSize   Thumbnail dimensions
 */
+ (UIImage *)getSampleImagesFromAsset:(AVAsset *)asset
                                times:(NSArray<NSNumber*> *)times
                              maxSize:(CGSize)maxSize
                             progress:(sampleProcess)sampleProcess;
```

Editing and Previewing

Video editing supports two effect preview modes: **pinpoint preview** (the video image is frozen at a specified point in time) and **range preview** (a video segment within a specified time range (A—B) is looped). To use a preview mode, you need to bind a `UIView` to the SDK in order to display the video image.

1. Bind UIView

The `initWithPreview` function of `TXVideoEditer` is used to bind a `UIView` to the SDK for video image rendering. You can set whether to use the **fit** or **fill** mode by controlling `renderMode` of `TXPreviewParam`.

```
   PREVIEW_RENDER_MODE_FILL_SCREEN - Fill mode, where the video image
will cover the entire screen with no black bars present, but the video
image may be cropped.
   PREVIEW_RENDER_MODE_FILL_EDGE - Fit mode, where the video image will
be complete but black bars will exist if the aspect ratio of the video
is different from that of the screen.
```

2. Use pinpoint preview

The `previewAtTime` function of `TXVideoEditer` is used to preview the video image at a specified point in time.

```
/** Render the video image at a specified point in time
 *  @param time   Preview frame time in seconds
 */
```

```
- (void)previewAtTime:(CGFloat)time;
```

3. Use range preview

The `startPlayFromTime` function of `TXVideoEditer` is used to loop a video segment within the time range of A—B.

```
/** Play back a video segment within a time range
 * @param startTime   Playback start time in seconds
 * @param endTime   Playback end time in seconds
 */
- (void)startPlayFromTime:(CGFloat)startTime
                   toTime:(CGFloat)endTime;
```

4. Pause and resume preview

```
/// Pause the video
- (void)pausePlay;

/// Resume the video
- (void)resumePlay;

/// Stop the video
- (void)stopPlay;
```

5. Add a beauty filter

You can add filter effects such as skin brightening, romantic, and fresh to the video. The demo provides multiple filters (with resources in `Common/Resource/Filter/FilterResource.bundle`) for your choice, and you can also set custom filters.You can set a filter as follows:

```
- (void) setFilter:(UIImage *)image;
```

Here, `image` is the filter mapping image. If `image` is set to `null`, the filter effect will be removed. Demo:

```
TXVideoEditer     *_ugcEdit;
NSString * path = [[NSBundle mainBundle]
pathForResource:@"FilterResource" ofType:@"bundle"];
```

```
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

6. Set a watermark

1. Set a global watermark

You can set a watermark image for the video and specify the image position.
You can set a watermark as follows:

```
- (void) setWaterMark:(UIImage *)waterMark  normalizationFrame:
(CGRect)normalizationFrame;
```

Here, `waterMark` indicates the watermark image, and `normalizationFrame` is a normalized `frame` value relative to the video image. The values of `x`, `y`, `width`, and `height` in `frame` all range from 0 to 1.
Demo:

```
UIImage *image = [UIImage imageNamed:@"watermark"];
[_ugcEdit setWaterMark:image normalizationFrame:CGRectMake(0, 0, 0.3 ,
0.3 * image.size.height / image.size.width)];// The watermark width is
30% of the video width, and the height is proportionally scaled
according to the width
```

2. Set a post-roll watermark

You can set a post-roll watermark for the video and specify the watermark position.
You can set a post-roll watermark as follows:

```
- (void) setTailWaterMark:(UIImage *)tailWaterMark normalizationFrame:
(CGRect)normalizationFrame
                         duration:(CGFloat)duration;
```

Here, `tailWaterMark` indicates the post-roll watermark image, and `normalizationFrame` is a normalized frame relative to the video image. The values of `x`, `y`, `width`, and `height` in `frame` all range from 0 to 1. `duration` indicates the watermark duration in seconds.
Demo: set a post-roll watermark that can be displayed for 1 second in the middle of the video image

```
UIImage *tailWaterimage = [UIImage imageNamed:@"tcloud_logo"];
float w = 0.15;
float x = (1.0 - w) / 2.0;
float width = w * videoMsg.width;
float height = width * tailWaterimage.size.height /
tailWaterimage.size.width;
float y = (videoMsg.height - height) / 2 / videoMsg.height;
[_ugcEdit setTailWaterMark:tailWaterimage
normalizationFrame:CGRectMake(x,y,w,0) duration:1];
```

Compressing and Clipping

Setting video bitrate

```
/**
 * Set the video bitrate
 * @param bitrate   Video bitrate in Kbps
 *                 If the bitrate is set, it will be selected preferably
when the SDK compresses videos. Please set an appropriate bitrate. If it
is too low, the video image will be blurry; if it is too high, the video
size will be too large
 *                 We recommend you set it to a value between 600 and
12000. If this API is not called, the SDK will automatically calculate
the bitrate based on the compression quality
 */
- (void) setVideoBitrate:(int)bitrate;
```

Clipping video

Video editing operations all follow the same principle: set the operation commands first and use `generateVideo` to run all commands in sequence, which can prevent unnecessary quality loss caused by multiple compression operations on the same video.

```
TXVideoEditer* _ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
// Set the clipping start and end time
[_ugcEdit setCutFromTime:_videoRangeSlider.leftPos
toTime:_videoRangeSlider.rightPos];
// ...
```

```
// Generate the final video file
_ugcEdit.generateDelegate = self;
[_ugcEdit generateVideo:VIDEO_COMPRESSED_540P
videoOutputPath:_videoOutputPath];
```

Specify the file compression quality and output path during output, and the output progress and result will be returned as a callback through `generateDelegate` .

Advanced Features

- TikTok-like special effects
- Setting background music
- Stickers and subtitles
- Editing image

Android

最終更新日：：2022-10-25 11:13:13

Overview

Video editing supports clipping, time effects (slow motion, reverse, loop), filters (rock light, dark dream, soul out, screen split), filter styles (artistic, rosy, blues, etc.), music mixing, animated stickers, static stickers, bubble subtitles, etc.

Classes

Class	Description
`TXVideoInfoReader`	Media information obtaining
`TXVideoEditer`	Video editing

Directions

Follow the steps below to edit your video:

1. Choose the video path
2. Import your video
3. Apply effects
4. Generate a file of the editing result
5. Listen for the callback for video generation
6. Release the resources

Getting Video Information

You can use **getVideoFileInfo** of **TXVideoInfoReader** to obtain some basic video information. Below is a request sample:

```
/**
  * Acquire video information
  * @param videoPath Video file path
  * @return
  */
public TXVideoEditConstants.TXVideoInfo getVideoFileInfo(String
videoPath);
```

`TXVideoInfo` is returned and is defined as follows:

```
public final static class TXVideoInfo {
        public Bitmap coverImage;                                    //
First video frame
        public long duration;                                            //
Video duration (ms)
        public long fileSize;
// Video file size (byte)
        public float fps;
// Video frame rate (fps)
        public int bitrate;
// Video bitrate (Kbps)
        public int width;
// Video width
        public int height;
// Video height
        public int audioSampleRate;                                  //
Audio bitrate
 }
```

Below is a complete sample:

```
//sourcePath Path of the video to edit
String sourcePath = Environment.getExternalStorageDirectory() +
File.separator + "temp.mp4";
TXVideoEditConstants.TXVideoInfo info =
TXVideoInfoReader.getInstance().getVideoFileInfo(sourcePath);
```

Getting Thumbnails

The thumbnail obtaining API is used to generate thumbnail preview during video editing or get the cover of a video.

1. Get thumbnails by splitting a video evenly

Quick generation

Below is a request sample:

```
/**
  * Get the thumbnail list
  * @param count    Number of thumbnails to get
  * @param width    Thumbnail width
  * @param height   Thumbnail height
  * @param fast       Whether to get keyframes
  * @param listener Callback API for thumbnail generation
*/
public void getThumbnail(int count, int width, int height, boolean fast,
TXThumbnailListener listener)
```

The @param fast parameter supports two modes:

- Quick generation: Pass in `true` to use this mode, under which thumbnails are generated relatively quickly, but they may not correspond exactly to video frames.
- Precise generation: Pass in `false` to use this mode, under which the thumbnails generated correspond exactly to video frames, but the generation may be slow if the resolution is high.

Below is a complete sample:

```
mTXVideoEditer.getThumbnail(TCVideoEditerWrapper.mThumbnailCount, 100,
100, false, mThumbnailListener);

private TXVideoEditer.TXThumbnailListener mThumbnailListener = new
TXVideoEditer.TXThumbnailListener() {
    @Override
        public void onThumbnail(int index, long timeMs, final Bitmap
bitmap) {
            Log.i(TAG, "onThumbnail: index = " + index + ",timeMs:" +
timeMs);
        // Insert the thumbnails into the image control
        }
    };
```

Precise generation

See Importing Video below.

2. Get thumbnails according to the time list

```
List<Long> list = new ArrayList<>();
```

```
list.add(10000L);
list.add(12000L);
list.add(13000L);
list.add(14000L);
list.add(15000L);

TXVideoEditer txVideoEditer = new
TXVideoEditer(TCVideoPreviewActivity.this);
txVideoEditer.setVideoPath(mVideoPath);
txVideoEditer.setThumbnailListener(new
TXVideoEditer.TXThumbnailListener() {
        @Override
        public void onThumbnail(int index, long timeMs, Bitmap bitmap) {
            Log.i(TAG, "bitmap:" + bitmap + ",timeMs:" + timeMs);
            saveBitmap(bitmap, timeMs);
        }
});
txVideoEditer.getThumbnailList(list, 200, 200);
```

Note:

- If a time point in the list is larger than the total video duration, the last video frame will be returned.
- Time points in the list are measured in milliseconds.

Importing Video

1. Quick import

This mode supports preview during editing. You can trim a video, play a video in slow motion, apply filters, change the filter style, mix music, and add animated/static stickers and bubble subtitles, among others. It does not support video looping or reversing.

2. Complete import

This mode supports all video editing features, including the time effects looping and reversing. Videos are pre-processed in this mode.
In this mode, you can locate any time point of a video and view the corresponding video frame. Thumbnails that correspond exactly to video frames are also generated during pre-processing.
The steps of complete import and the APIs used during the process are as follows:

1. Configure precise generation of thumbnails.

```
/**
  * Configure the thumbnails generated during pre-processing
  */
public void setThumbnail(TXVideoEditConstants.TXThumbnail thumbnail)
```

2. Configure the callback for thumbnail generation.

```
/**
  * Configure the callback for thumbnail generation during pre-
processing
  * @param listener
  */
public void setThumbnailListener(TXThumbnailListener listener)
```

> **Note:**
> We recommend you do not specify thumbnail width or height as scaling by the SDK is more efficient.

3. Configure the callback for video pre-processing.

```
/**
  * Configure the callback for video pre-processing
  * @param listener
  */
public void setVideoProcessListener(TXVideoProcessListener listener)
```

4. Pre-process videos

```
public void processVideo();
```

Below is a complete sample:

```
int thumbnailCount = 10;  //Number of thumbnails to generate
TXVideoEditConstants.TXThumbnail thumbnail = new
TXVideoEditConstants.TXThumbnail();
thumbnail.count = thumbnailCount;
thumbnail.width = 100;   // Thumbnail width
```

```
thumbnail.height = 100;  // Thumbnail height
mTXVideoEditer.setThumbnail(thumbnail);
// Configure the thumbnails generated during pre-processing
mTXVideoEditer.setThumbnailListener(mThumbnailListener); //  Configure
the callback for thumbnail generation

mTXVideoEditer.setVideoProcessListener(this);                          //
Configure the callback of video pre-processing progress
mTXVideoEditer.processVideo();
// Pre-process videos
```

Preview

You can preview a video during editing in two modes. Time-point preview shows the frame of a certain time point, while time-range preview plays a video segment between two time points on loop (A<=>B). You need to bind the SDK with a UIView to display video images.

1. Configure preview layout

```
public void initWithPreview(TXVideoEditConstants.TXPreviewParam param)
```

Two video rendering modes are supported, which are defined in the constant `TXVideoEditConstants`.

```
public final static int PREVIEW_RENDER_MODE_FILL_SCREEN = 1;    // Aspect
fill. The image is stretched to fill the entire screen, and the excess
parts are cropped.
public final static int PREVIEW_RENDER_MODE_FILL_EDGE = 2;         //
Aspect fit. The image is kept intact, and there may be black bars if the
aspect ratio does not match.
```

2. Time-point preview

You can locate any time point of a video imported in the complete import mode.

```
public void previewAtTime(long timeMs);
```

3. Time-range preview

You can use `startPlayFromTime` of `TXVideoEditer` to play a video segment between two time points (A<=>B).

```
// Play a segment of a video from `startTime` to `endTime`
public void startPlayFromTime(long startTime, long endTime);
```

4. Pause and resume preview

```
// Pause preview
public void pausePlay();


// Resume preview
public void resumePlay();


// Stop preview
public void stopPlay();
```

5. Add beauty filter

You can apply filter effects such as skin brightening, romantic, and refreshing to your video. The demo offers 16 filters. You can also customize filters.
The method to set filter is:

```
void setFilter(Bitmap bmp)
```

A bitmap is a mapping of the filter image. Setting `bmp` to null means to remove the filter effect.

```
void setSpecialRatio(float specialRatio)
```

You can use this API to set the filter strength on a scale of 0.0 to 1.0.

```
void setFilter(Bitmap leftBitmap, float leftIntensity, Bitmap
rightBitmap, float rightIntensity, float leftRatio)
```

You can use this API to apply different filters to the left and right sections of your video. `leftBitmap` represents the left filter and `leftIntensity` the strength of the left filter. `rightBitmap` represents the right filter and `rightIntensity` the strength of the right filter. `leftRatio` indicates the ratio

(0.0–1.0) of the image section the left filter is applied to. If `leftBitmap` or `rightBitmap` is set to null, the filter effect for the corresponding section will be removed.

6. Watermark

1. Add a global watermark

You can add a watermark to a specified position of a video.
The method to add a watermark is as follows:

```
public void setWaterMark(Bitmap waterMark, TXVideoEditConstants.TXRect
rect);
```

`waterMark` represents the watermark image. `rect` is the normalized frame of the watermark image in relation to the video image. The value range of `x`, `y`, `width`, and `height` is 0 to 1.
Demo:

```
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXVideoEditer.setWaterMark(mWaterMarkLogo, rect);
```

2. Add an ending watermark

You can add a watermark to the end of a video at the specified location.
The method to add an ending watermark is as follows:

```
setTailWaterMark(Bitmap tailWaterMark, TXVideoEditConstants.TXRect
txRect, int duration);
```

`tailWterMark` represents the watermark image. `txRect` is the normalized frame of the watermark image in relation to the video image, and the value range of `x`, `y`, and `width` in `txRect` is from 0 to 1. `duration` indicates for how long (s) the watermark is displayed.
Demo: add an ending watermark to the center of a video and show the watermark for 3 seconds

```
Bitmap tailWaterMarkBitmap =
BitmapFactory.decodeResource(getResources(), R.drawable.tcloud_logo);
TXVideoEditConstants.TXRect txRect = new TXVideoEditConstants.TXRect();
```

```
txRect.x = (mTXVideoInfo.width - tailWaterMarkBitmap.getWidth()) / (2f *
mTXVideoInfo.width);
txRect.y = (mTXVideoInfo.height - tailWaterMarkBitmap.getHeight()) / (2f
* mTXVideoInfo.height);
txRect.width = tailWaterMarkBitmap.getWidth() / (float)
mTXVideoInfo.width;
mTXVideoEditer.setTailWaterMark(tailWaterMarkBitmap, txRect, 3);
```

Compression and Clipping

Setting video bitrate

You can specify a custom value, preferably between 600 and 12000 (Kbps), for video bitrate. The SDK will prioritize the bitrate set during video compression. Do not set the bitrate too high or too low. The former drives up the size of video files, and the latter results in blurry videos.

```
public void setVideoBitrate(int videoBitrate);
```

Clipping video

Set the start and end time for clipping

```
/**
  * Specify the time range for video clipping
  * @param startTime Start time (ms) for clipping
  * @param endTime   End time (ms) for clipping
  */
public void setCutFromTime(long startTime, long endTime)


// ...
// Generate the video file
public void generateVideo(int videoCompressed, String videoOutputPath)
```

The constants of `videoCompressed` in `TXVideoEditConstants` are as follows:

```
VIDEO_COMPRESSED_360P - Compress to 360p (360 × 640)
VIDEO_COMPRESSED_480P - Compress to 480p (640 × 480)
VIDEO_COMPRESSED_540P - Compress to 540p (960 × 540)
```

```
VIDEO_COMPRESSED_720P - Compress to 720p (1280 × 720)
VIDEO_COMPRESSED_1080P - Compress to 1080p (1920 × 1080)
```

If the resolution of the original video is lower than the configured constant, the original resolution will be used.
If the resolution of the original video is higher than the configured constant, the video will be compressed to the configured resolution.

Releasing Resources

When you no longer use the `mTXVideoEditer` object, be sure to call **release()** to release it.

Advanced Features

- TikTok-like Special Effects
- Adding Background Music
- Stickers and Subtitles
- Image Transition Special Effects

Video Splicing
iOS

最終更新日：：2025-04-01 17:14:08

The TXVideoJoiner can merge multiple videos either by sequentially concatenating them or by combining their frames in a split-screen layout.

1. Basic Video Concatenation

To simply concatenate multiple video files sequentially, refer to the following code:

```
// Create TXVideoJoiner (nil for preview param indicates no preview
needed)
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc] initWithPreview:nil];

// Set video files to concatenate (_composeArray)
[_videoJoin setVideoPathList:_composeArray];

// Set delegate for progress/completion callbacks
_videoJoin.joinerDelegate = self;

// Start merging (540p compressed output)
[_videoJoin joinVideo:VIDEO_COMPRESSED_540P
videoOutputPath:_outFilePath];
```

2. Preview Before Concatenation

To preview the concatenated videos before merging, use the following code:

```
// Configure preview view
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;

// Create TXVideoJoiner with preview
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc]
initWithPreview:param];
_videoJoin.previewDelegate = _videoPreview;
```

```
// Set video files
[_videoJoin setVideoPathList:_composeArray];

// Start preview playback
[_videoJoin startPlay];
```

Control playback with:

- startPlay: Start preview
- pausePlay: Pause preview
- resumePlay: Resume preview

3. Split-Screen Video Merging

TXVideoJoiner also supports merging multiple videos into a split-screen layout. Example code:

```
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc] initWithPreview:nil];
[_videoJoin setVideoPathList:_composeArray];
_videoJoin.joinerDelegate = self;

// Configure split-screen layout
TXSplitScreenParams* splitScreenParams = [[TXSplitScreenParams alloc]
init];
splitScreenParams.canvasWidth = 720 * 2;  // Total output width
splitScreenParams.canvasHeight = 1280;     // Total output height

// Define positions for each video
splitScreenParams.rects = @[
    [NSValue valueWithCGRect:CGRectMake(0, 0,
splitScreenParams.canvasWidth/2, splitScreenParams.canvasHeight)],
    [NSValue valueWithCGRect:CGRectMake(splitScreenParams.canvasWidth/2,
0, splitScreenParams.canvasWidth/2, splitScreenParams.canvasHeight)]
];

splitScreenParams.durationMode = ALIGNS_TO_LONGEST;  // Duration matches
longest input
[_videoJoiner setSplitScreenList:splitScreenParams];

// Set audio mix ratios (0: muted, 1: full volume)
```

```
[_videoJoiner setVideoVolumes:@[@0, @1]];

// Start merging
[_videoJoin joinVideo:VIDEO_COMPRESSED_540P
videoOutputPath:_outFilePath];
```

When configuring split-screen layout parameters via setSplitScreenList and audio mixing ratios via setVideoVolumes, these settings apply to both the final video output (generated by splitJoinVideo) andthe live preview during editing.

Android

最終更新日：：2025-04-01 17:14:09

The TXVideoJoiner can merge multiple videos either by sequentially concatenating them or by combining their frames in a split-screen layout.

1. Basic Video Concatenation

To simply concatenate multiple video files sequentially, refer to the following code:

```
mTXVideoJoiner = new TXVideoJoiner(mContext);

// Set the video source list. A return value < 0 indicates invalid or
unsupported formats in the input files.
if (mTXVideoJoiner.setVideoPathList(videoSourceList) < 0) {
    return;
}

// Set a listener to track concatenation progress and completion.
mTXVideoJoiner.setVideoJoinerListener(new TXVideoJoinerListener() {
    @Override
    public void onJoinProgress(float progress) {}

    @Override
    public void onJoinComplete(TXJoinerResult result) {}
});

// Start concatenation. Specify the output resolution and path.
// Note: If all input videos share the same format and resolution,
concatenation is fast.
// Otherwise, re-encoding is required, which slows the process.
mTXVideoJoiner.joinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P,
mOutputPath);
```

2. Preview Before Concatenation

To preview the concatenated videos before merging, use the following code:

```
mTXVideoJoiner = new TXVideoJoiner(mContext);
```

```
// Set the video source list.
if (mTXVideoJoiner.setVideoPathList(videoSourceList) < 0) {
    return;
}

// Set a listener for preview progress and completion.
mTXVideoJoiner.setTXVideoPreviewListener(new TXVideoPreviewListener() {
    @Override
    public void onPreviewProgress(int time) {}

    @Override
    public void onPreviewFinished() {}
});

// Prepare the preview view.
TXVideoEditConstants.TXPreviewParam param = new
TXVideoEditConstants.TXPreviewParam();
param.videoView = mVideoView;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoJoiner.initWithPreview(param);

// Start preview.
mTXVideoJoiner.startPlay();
```

Control playback with:

- startPlay: Start preview
- pausePlay: Pause preview
- resumePlay: Resume preview

3. Split-Screen Video Merging

TXVideoJoiner also supports merging multiple videos into a split-screen layout. Example code:

```
mTXVideoJoiner = new TXVideoJoiner(mContext);

// Set the video source list.
if (mTXVideoJoiner.setVideoPathList(videoSourceList) < 0) {
    return;
}
```

```
// Set a listener for split-screen merging progress and completion.
mTXVideoJoiner.setVideoJoinerListener(new TXVideoJoinerListener() {
    @Override
    public void onJoinProgress(float progress) {}

    @Override
    public void onJoinComplete(TXJoinerResult result) {}
});

// Configure split-screen layout parameters.
SplitScreenParam splitScreenParam = new SplitScreenParam();
splitScreenParam.canvasWidth = 720;      // Canvas width during split-
screen joining
splitScreenParam.canvasHeight = 1280;    // Canvas height
splitScreenParam.durationControlMode =
DurationControlMode.ALIGNS_TO_LONGEST; // Duration matches the longest
video

// Define positions and sizes for each video on the canvas.
TXAbsoluteRect rect1 = new TXAbsoluteRect();
TXAbsoluteRect rect2 = new TXAbsoluteRect();
splitScreenParam.rects.add(rect1);
splitScreenParam.rects.add(rect2);
mTXVideoJoiner.setSplitScreenList(splitScreenParam);

// Set audio mixing weights for each video (e.g., video1: 100% volume,
video2: 0%).
List<Float> volumes = new LinkedList<>();
volumes.add(1.0f);
volumes.add(0.0f);
mTXVideoJoiner.setVideoVolumes(volumes);

// Start split-screen merging.
// If input videos differ in format or resolution, re-encoding is
required, which may slow the process.
mTXVideoJoiner.splitJoinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P
, mOutputPath);
```

When configuring split-screen layout parameters via setSplitScreenList and audio mixing ratios via setVideoVolumes, these settings apply to both the final video output (generated by splitJoinVideo) andthe live preview during editing.

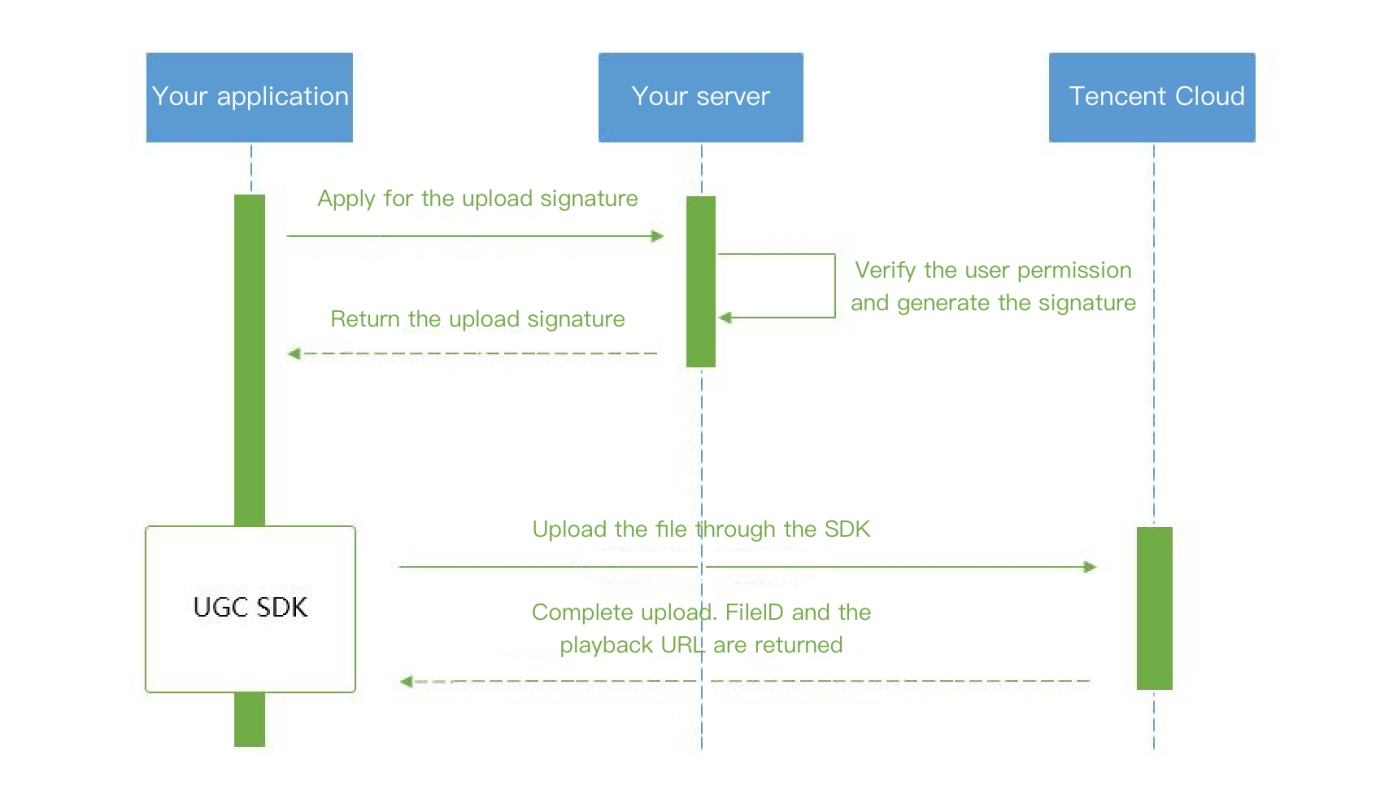
Upload and Playback Signature Distribution

最終更新日：：2020-08-28 16:53:57

Video upload from client refers to uploading local videos to the VOD platform by an end user of the application. For more information, please see Guide. This document describes how to generate a signature for upload from client.

Overview

The overall process for upload from client is as follows:



To support upload from client, you need to build two backend services: signature distribution service and event notification receipt service.

- The client first requests an upload signature from the signature distribution service.

- The signature distribution service verifies whether the client user has the upload permission. If the verification passes, a signature will be generated and distributed; otherwise, an error code will be returned, and the upload process will end.
- After receiving the signature, the client will use the upload feature integrated in the UGSV SDK to upload the video.
- After the upload is completed, the VOD backend will send an upload completion event notification to your event notification receipt service.
- If the signature distribution service specifies a video processing task flow in the signature, the VOD service will automatically process the video accordingly after the video is uploaded. Video processing in UGSV scenarios is generally AI-based porn detection.
- After the video is processed, the VOD backend will send a task flow status change event notification to your event notification receipt service.

At this point, the entire video upload and processing flow ends.

Signature Generation

For more information on the signature for upload from client, please see Signature for Upload from Client.

Signature Distribution Service Implementation Sample

```
/**
 * Calculate a signature
 */
function createFileUploadSignature({ timeStamp = 86400, procedure = '',
classId = 0, oneTimeValid = 0, sourceContext = '' }) {
    // Determine the current time and expiration time of the signature
    let current = parseInt((new Date()).getTime() / 1000)
    let expired = current + timeStamp;  // Signature validity period: 1
day
    // Enter the parameters into the parameter list
    let arg_list = {
        //required
        secretId: this.conf.SecretId,
        currentTimeStamp: current,
        expireTime: expired,
        random: Math.round(Math.random() * Math.pow(2, 32)),
        //opts
        procedure,
        classId,
        oneTimeValid,
```

```
        sourceContext
    }
    // Calculate the signature
    let orignal = querystring.stringify(arg_list);
    let orignal_buffer = new Buffer(orignal, "utf8");
    let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
    let hmac_buffer = hmac.update(orignal_buffer).digest();
    let signature = Buffer.concat([hmac_buffer,
orignal_buffer]).toString("base64");
    return signature;
}
/**
 * Respond to a signature request
 */
function getUploadSignature(req, res) {
    res.json({
        code: 0,
        message: 'ok',
        data: {
            signature: gVodHelper.createFileUploadSignature({})
        }
    });
}
```

Video Upload
iOS

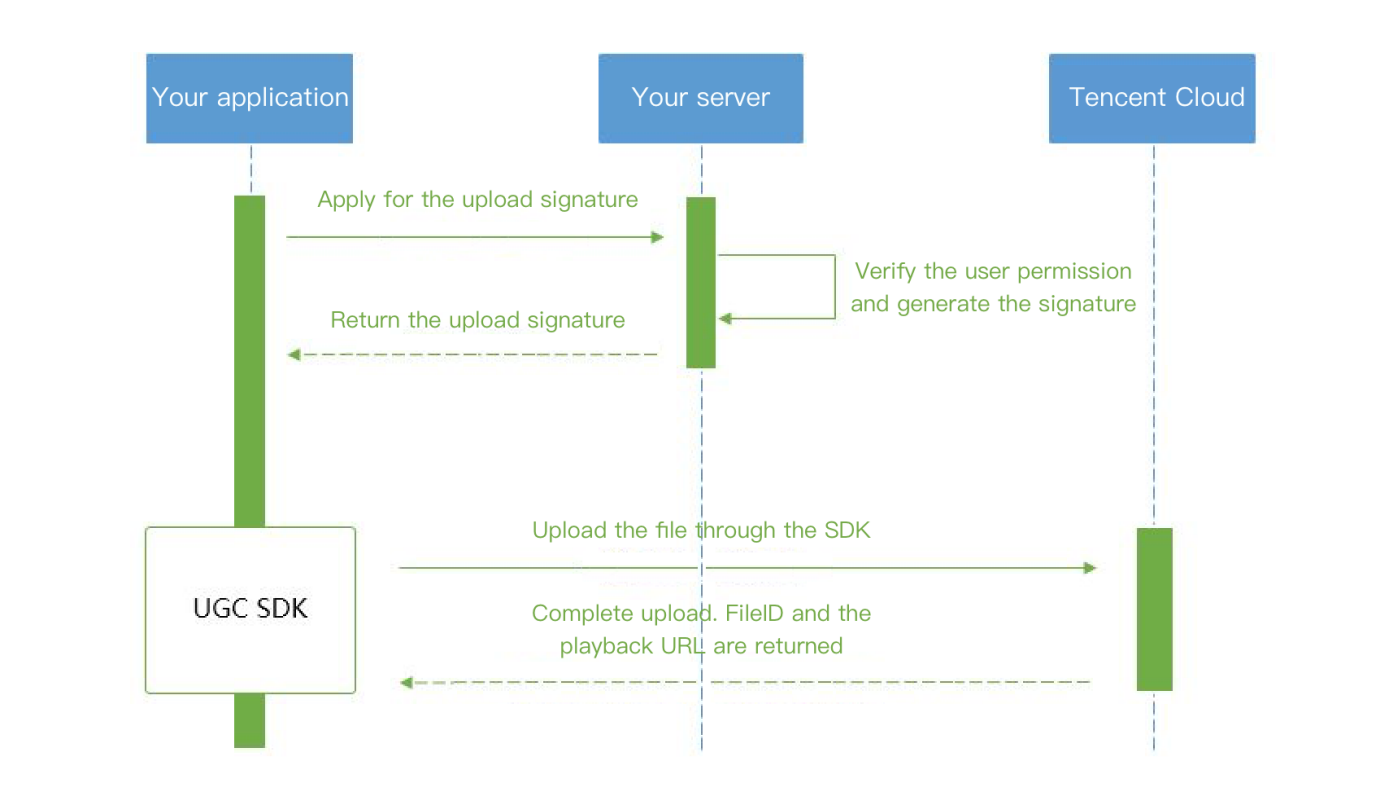
最終更新日：：2022-05-24 16:08:49

Calculating Upload Signature

Video upload from client refers to uploading local videos to the VOD platform by an end user of the application. For more information, please see Guide. This document describes how to generate a signature for upload from client.

Overview

The overall process for upload from client is as follows:



To support upload from client, you need to build two backend services: signature distribution service and event notification receipt service.

- The client first requests an upload signature from the signature distribution service.
- The signature distribution service verifies whether the client user has the upload permission. If the verification passes, a signature will be generated and delivered; otherwise, an error code will be returned, and the upload process will end.
- After receiving the signature, the client will use the upload feature integrated in the UGSV SDK to upload the video.
- After the upload is completed, the VOD backend will send an upload completion event notification to your event notification receipt service.
- If the signature distribution service specifies a video processing task flow in the signature, the VOD service will automatically process the video accordingly after the video is uploaded. Video processing in UGSV scenarios is generally AI-based porn detection.
- After the video is processed, the VOD backend will send a task flow status change event notification to your event notification receipt service.

At this point, the entire video upload and processing flow ends.

Signature generation

For more information on the signature for upload from client, please see Signature for Upload from Client.

Signature distribution service implementation sample

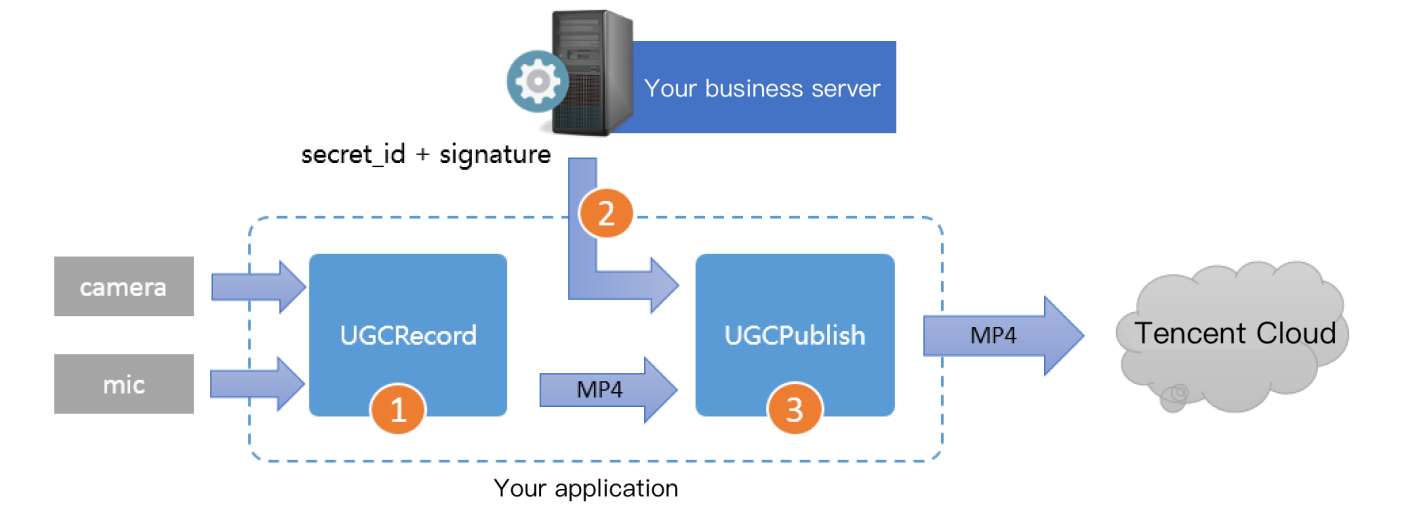
```
/**
 * Calculate a signature
 */
function createFileUploadSignature({ timeStamp = 86400, procedure = '',
classId = 0, oneTimeValid = 0, sourceContext = '' }) {
    // Determine the generation time and expiration time of the
signature
    let current = parseInt((new Date()).getTime() / 1000)
    let expired = current + timeStamp;  // Signature validity period: 1
day
    // Enter the parameters into the parameter list
    let arg_list = {
        //required
        secretId: this.conf.SecretId,
        currentTimeStamp: current,
        expireTime: expired,
        random: Math.round(Math.random() * Math.pow(2, 32)),
        //opts
        procedure,
```

```
        classId,
        oneTimeValid,
        sourceContext
    }
    // Calculate the signature
    let orignal = querystring.stringify(arg_list);
    let orignal_buffer = new Buffer(orignal, "utf8");
    let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
    let hmac_buffer = hmac.update(orignal_buffer).digest();
    let signature = Buffer.concat([hmac_buffer,
orignal_buffer]).toString("base64");
    return signature;
}
/**
 * Respond to a signature request
 */
function getUploadSignature(req, res) {
    res.json({
        code: 0,
        message: 'ok',
        data: {
            signature: gVodHelper.createFileUploadSignature({})
        }
    });
}
```

Connection Process

Publishing short video

Upload a .mp4 file to Tencent Video Cloud and get the online watch URL. Tencent Video Cloud can meet various video watch needs such as nearby scheduling, instant live streaming and playback, dynamic acceleration, and global connection, delivering a smooth watch experience.



- Step 1. Use the `TXUGCRecord` API to shoot a short video, and a .mp4 short video file will be generated after the shoot ends and be called back.
- Step 2. Your application applies for an upload signature ("credential" for the application to upload the .mp4 file to VOD) to your business server. To ensure the security, upload signatures should be distributed by your business server but not generated by the client application.
- Step 3. Use the `TXUGCPublish` API to publish the video. After the video is successfully published, the SDK will call back the watch URL to you.

Notes

- You should never write the `SecretID` or `SecretKey` for upload signature calculation into the client code of the application, as their disclosure will cause security risks. If attackers get such information by cracking the application, they can misappropriate your traffic and storage service.
- The correct practice is to generate a one-time upload signature by using the `SecretID` and `SecretKey` on your server and send the signature to the application. As the server is generally hard to be intruded, the security is guaranteed.
- When publishing a short video, please make sure that the `Signature` field is correctly passed in; otherwise, the release will fail.

Connection directions

1. Select a video

You can upload the shot or edited video as described in previous documents or upload a local video on your phone.

2. Compress the video

Use the `TXVideoEditer.generateVideo(int videoCompressed, String videoOutputPath)` API to compress the selected video. Four resolutions are supported for compression currently, and compression with customizable bitrate will be supported in the future.

3. Publish the video

Publish the generated .mp4 file to Tencent Cloud. The application needs to get the upload signature with a short validity period for file upload as instructed in Signature Distribution.
`TXUGCPublish` (in `TXUGCPublish.h`) is used to publish .mp4 files to VOD so as to meet various video watch needs such as nearby scheduling, instant live streaming and playback, dynamic acceleration, and global connection.

```
TXPublishParam * param = [[TXPublishParam alloc] init];

param.signature = _signature;                                    // Enter
the upload signature calculated in step 4

// Video file path generated by shoot, which can be obtained through the
`onRecordComplete` callback of `TXVideoRecordListener`
param.videoPath = _videoPath;
// Path of the first-frame video preview image generated by shoot. The
value is the file path specified by calling `startRecord`. You can also
specify a path and save the `UIImage` obtained in the `onRecordComplete`
callback of `TXVideoRecordListener` to the specified path. It can be set
to `nil`.
param.coverPath = _coverPath;

TXUGCPublish *_ugcPublish = [[TXUGCPublish alloc] init];
// Checkpoint restart is used for file release by default
_ugcPublish.delegate = self;                                     // Set the
`TXVideoPublishListener` callback
[_ugcPublish publishVideo:param];
```

The release process and result will be returned through the `TXVideoPublishListener` API (defined in the `TXUGCPublishListener.h` header file):

- `onPublishProgress` is used to return the file release progress, the `uploadBytes` parameter

indicates the number of uploaded bytes, and the `totalBytes` parameter indicates the total number of bytes that need to be uploaded.

```
@optional
-(void) onPublishProgress:(NSInteger)uploadBytes totalBytes:
(NSInteger)totalBytes;
```

- `onPublishComplete` is used to return the release result, the `errCode` and `descMsg` fields of `TXPublishResult` indicate the error code and error message respectively, `videoURL` indicates the VOD address of the short video, `coverURL` indicates the cloud storage address of the video cover, and `videoId` indicates the cloud storage ID the video file, with which you can call VOD's server APIs.

```
@optional
-(void) onPublishComplete:(TXPublishResult*)result;
```

- Release result.

  You can check the short video release result against the error code table.

4. Play back the video

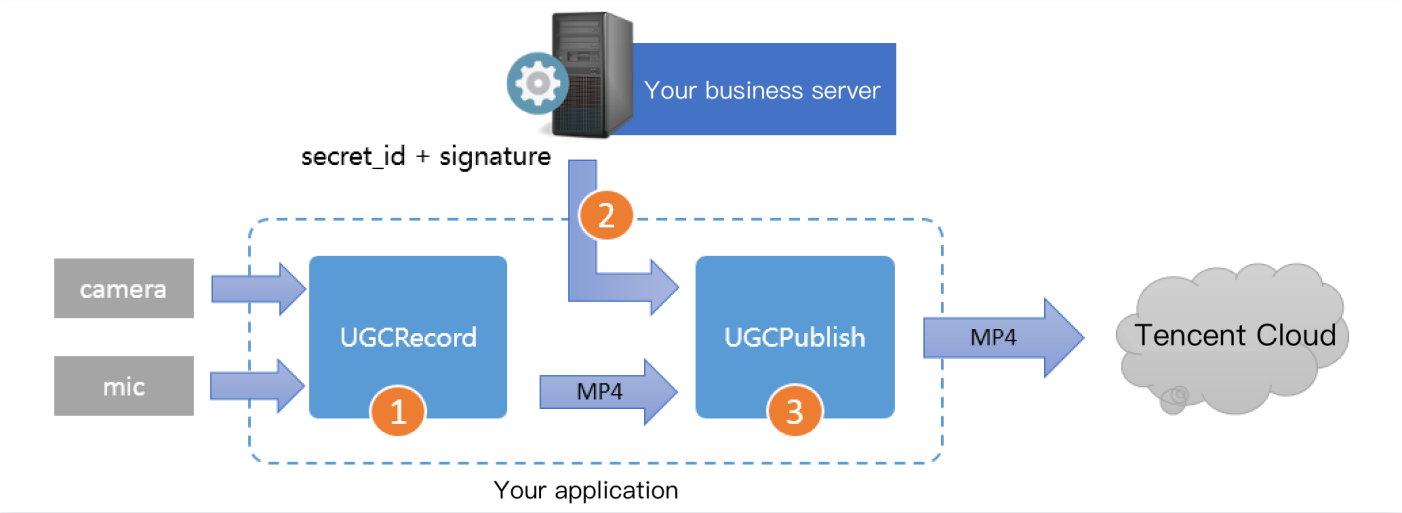
After the video is successfully uploaded in step **3**, the video `fileId`, playback URL, and cover URL will be returned. You can directly pass in the `fileId` or playback URL to the VOD player for playback.

Android

最終更新日：：2022-05-24 16:09:04

Connection Process

Short video release: upload a .mp4 file to Tencent Video Cloud and get the online watch URL. Tencent Video Cloud can meet various video watch needs such as nearby scheduling, instant live streaming and playback, dynamic acceleration, and global connection, delivering a smooth watch experience.



- Step 1. Use the `TXUGCRecord` API to shoot a short video, and a .mp4 short video file will be generated after the shoot ends and be called back.
- Step 2. Your application applies for an upload signature ("credential" for the application to upload the .mp4 file to VOD) to your business server. To ensure the security, upload signatures should be distributed by your business server but not generated by the client application.
- Step 3. Use the `TXUGCPublish` API to publish the video. After the video is successfully published, the SDK will call back the watch URL to you.

Notes

- You should never write the `SecretID` or `SecretKey` for upload signature calculation into the client code of the application, as their disclosure will cause security risks. If attackers get such information by cracking the application, they can misappropriate your traffic and storage service.
- The correct practice is to generate a one-time upload signature by using the `SecretID` and

`SecretKey` on your server and send the signature to the application.

- When publishing a short video, please make sure that the `Signature` field is correctly passed in; otherwise, the release will fail.

Connection Directions

Integrate the short video upload feature as instructed in Upload SDK for Android.

1. Select a video

Uploaded a shot, edited, or spliced video or select a local video for upload.

2. Compress the video

- Video compression can reduce the short video file size but will also reduce the video definition. You can determine whether to compress videos as needed.
- Use the `TXVideoEditer.generateVideo(int videoCompressed, String videoOutputPath)` API to compress the video. Four resolutions are supported for compression currently, and compression with customizable bitrate will be supported in the future.

3. Publish the video

Publish the generated .mp4 file to Tencent Cloud. The application needs to get the upload signature with a short validity period for file upload as instructed in Signature Distribution. `TXUGCPublish` (in `TXUGCPublish.java`) is used to publish .mp4 files to VOD so as to meet various video watch needs such as nearby scheduling, instant live streaming and playback, dynamic acceleration, and global connection.

```
mVideoPublish = new
TXUGCPublish(TCVideoPublisherActivity.this.getApplicationContext());
// Checkpoint restart is used for file release by default
TXUGCPublishTypeDef.TXPublishParam param = new
TXUGCPublishTypeDef.TXPublishParam();
param.signature = mCosSignature;                        // Enter the
upload signature calculated in step 4
// Video file path generated by shoot, which can be obtained through the
`onRecordComplete` callback of `ITXVideoRecordListener`
param.videoPath = mVideoPath;
// First-frame video preview generated by shoot, which can be obtained
through the `onRecordComplete` callback of `ITXVideoRecordListener`
param.coverPath = mCoverPath;
mVideoPublish.publishVideo(param);
```

The release process and result will be returned through the `TXRecordCommon.ITXVideoPublishListener` API (in the `TXRecordCommon.java` header file):

- `onPublishProgress` is used to return the release progress, the `uploadBytes` parameter indicates the number of uploaded bytes, and the `totalBytes` parameter indicates the total number of bytes that need to be uploaded.

```
void onPublishProgress(long uploadBytes, long totalBytes);
```

- `onPublishComplete` is used to return the release result.

```
void onPublishComplete(TXPublishResult result);
```

The fields in the `TXPublishResult` parameter and their descriptions are as detailed below:

Field	Description
errCode	Error code.
descMsg	Error message.
videoURL	VOD address of short video.
coverURL	Cloud storage address of video cover.
videoId	Cloud storage ID of video file, through which you can call VOD's server APIs.

- You can check the short video release result against the error code table.

4. Play back the video

After the video is successfully uploaded in step 3, the video `fileId`, playback URL, and cover URL will be returned. You can directly pass in the `fileId` or playback URL to the VOD player for video playback.

Player SDK
iOS

最終更新日：：2022-11-21 17:29:27

Overview

Tencent Cloud RT-Cube Player for iOS is an open-source Tencent Cloud player component. It can provide powerful playback functionality similar to Tencent Video with just a few lines of code. It has basic features such as landscape/portrait mode switching, definition selection, gestures, and small window playback, as well as special features such as video buffering, software/hardware decoding switching, and adjustable-speed playback. It supports more formats and has better compatibility and functionality than system-default players. In addition, it features instant broadcasting of the first frame and low latency and offers advanced capabilities like video thumbnail.
If the Player component cannot meet your custom requirements and you have development experience, you can integrate the RT-Cube Player SDK as instructed in Integration Guide to customize the player UI and playback features.

Prerequisites

1. Activate VOD. If you don't have an account yet, sign up first.
2. Download and install Xcode from App Store.
3. Download and install CocoaPods as instructed at the CocoaPods website.

Content Summary

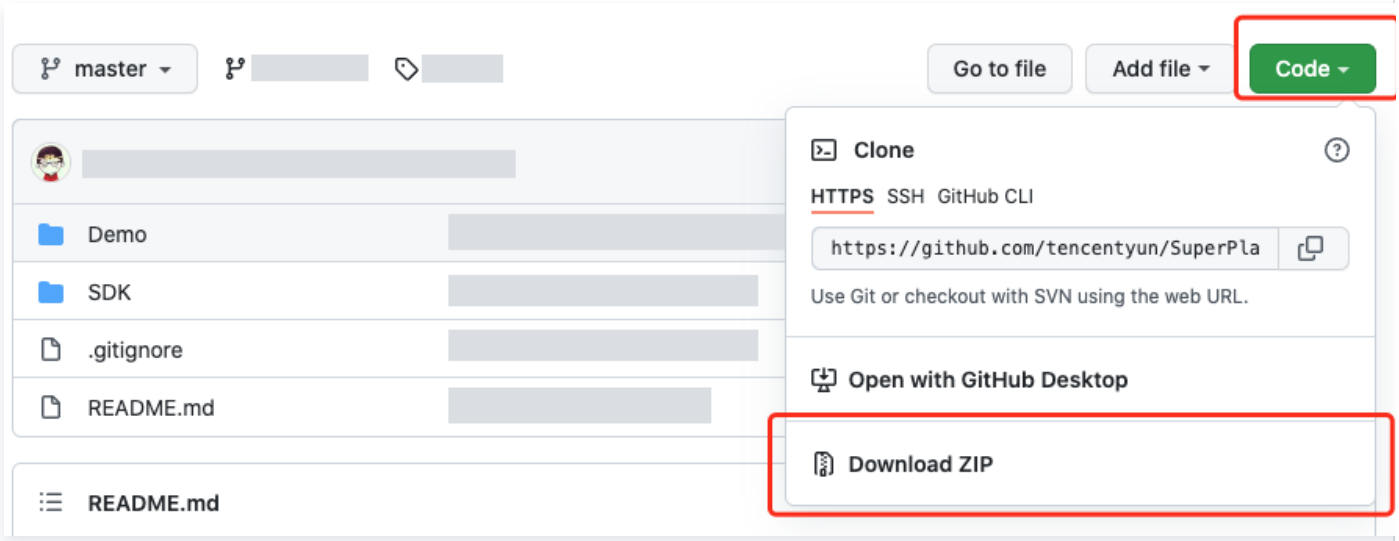
- How to integrate the RT-Cube Player component for iOS
- How to create and use a player

Directions

Step 1. Download the player code package

GitHub page: LiteAVSDK/Player_iOS
You can download a ZIP file of the Player component from the GitHub page or use the Git clone command to download the component.

Download the ZIP file
Go to the Player GitHub page and click **Code** > **Download ZIP**.



Download using a Git command

1. First, make sure that your computer has Git installed; if not, you can install it as instructed in Git Installation Tutorial.
2. Run the following command to clone the code of the Player component to your local system.

```
git clone git@github.com:tencentyun/SuperPlayer_iOS.git
```

If you see the following information, the project code has been cloned to your local system successfully.

```
Cloning to 'SuperPlayer_iOS'...
remote: Enumerating objects: 2637, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (333/333), done.
remote: Total 2637 (delta 227), reused 524 (delta 170), pack-
reused 1993
Receiving the object: 100% (2637/2637), 571.20 MiB | 3.94 MiB/s,
done.
Processing delta: 100% (1019/1019), done.
```

Below is the directory structure of the component's source code after decompression:

Filename	Description
SDK	Stores the Superplayer framework and static library.
Demo	The folder of the Player demo.
App	The entry point UI.
SuperPlayerDemo	The Player demo.
SuperPlayerKit	The Player component.

Step 2. Integrate the component

This step describes how to integrate the Player component. We recommend you integrate it through CocoaPods or manually download the SDK and then import it into your current project.

Integrate via CocoaPods

1. To install the component using CocoaPods, add the code below to the Podfile:
   Directly integrate `SuperPlayer` as a Pod:

   ```
   pod 'SuperPlayer
   ```

   To use the Player edition, add the following dependency to `podfile` :

   ```
   pod 'SuperPlayer/Player'
   ```

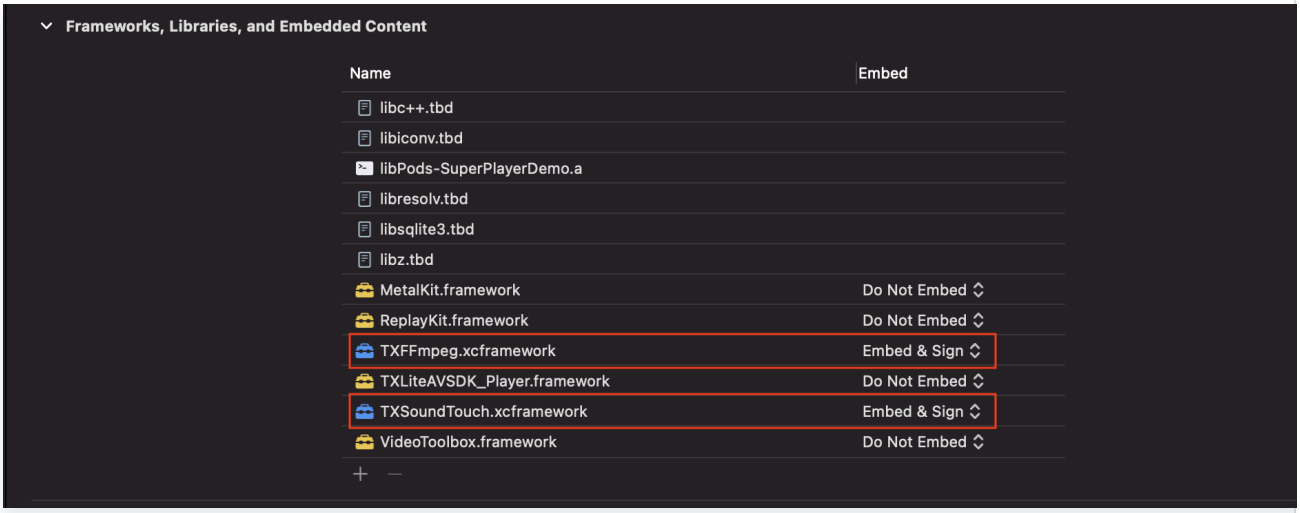
   To use the All-in-one edition, add the following dependency to `podfile` :

   ```
   pod 'SuperPlayer/Professional'
   ```

2. Run `pod install` or `pod update` .

Manually download the SDK

1. Download the SDK and demo at GitHub.
2. Import `TXLiteAVSDK_Player.framework` into your project and select **Do Not Embed**.
3. Copy `Demo/TXLiteAVDemo/SuperPlayerKit/SuperPlayer` to your project directory.
4. Third-party dependency libraries of `SuperPlayer` include `AFNetworking`, `SDWebImage`, `Masonry`, and `TXLiteAVSDK_Player`.
5. To integrate `TXLiteAVSDK_Player` manually, you need to add the required system frameworks and libraries:
   - **System frameworks**: MetalKit, ReplayKit, SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics, AVFoundation, Accelerate, MobileCoreServices, and VideoToolbox.
   - **System libraries**: libz, libresolv, libiconv, libc++, and libsqlite3.
     In addition, you need to add `TXFFmpeg.xcframework` and `TXSoundTouch.scframework` under the `TXLiteAVSDK_Player` file as dynamic libraries.



6. If you integrate `TXLiteAVSDK_Player` as a Pod, no libraries need to be added.

Step 3. Use the player features

This step describes how to create a player and use it for video playback.

1. **Create a player.**

Create a `SuperPlayerView` object to play videos (`SuperPlayerView` is the main class of the Superplayer).

```
// Import the header file
#import <SuperPlayer/SuperPlayer.h>

// Create a player
_playerView = [[SuperPlayerView alloc] init];
// Set a delegate for events
_playerView.delegate = self;
// Set the parent view. _playerView will be automatically added under
holderView.
_playerView.fatherView = self.holderView;
```

2. **License configuration.**

   If you have the required license, get the license URL and key in the RT-Cube console.
   If you don't have the required license, please contact us to get a license.
   After getting the license information, before calling relevant APIs of the SDK, initialize the license through the following API. We recommend you set the following in `- [AppDelegate application:didFinishLaunchingWithOptions:]` :

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    NSString * const licenceURL = @"<The license URL obtained>";
    NSString * const licenceKey = @"<The key obtained>";

    //TXLiveBase can be found in the "TXLiveBase.h" header file
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
```

3. **Video playback:**

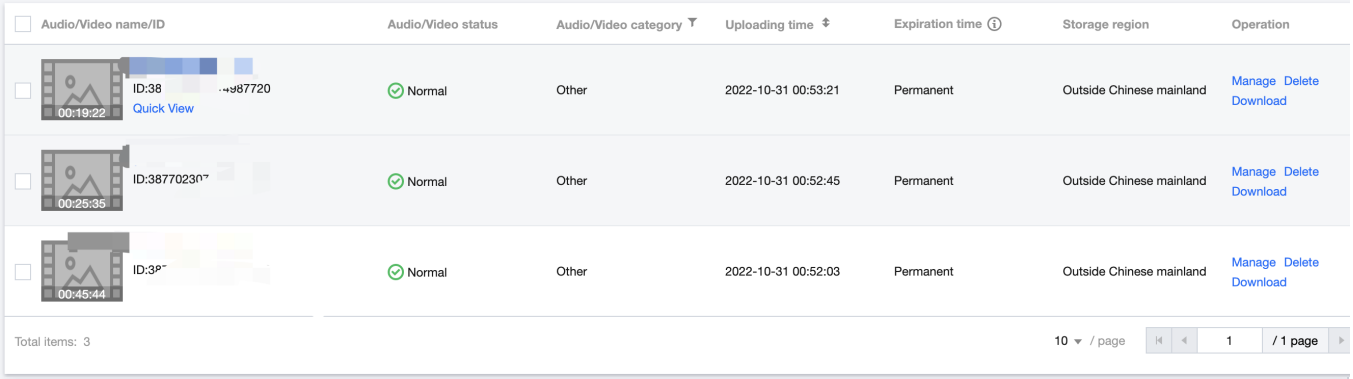
   This step describes how to play back a video. The RT-Cube Player for iOS supports playback through `FileId` in VOD or URL. We recommend you **integrate the `FileId`** because it allows you to use more VOD capabilities.

Play by VOD file ID

A video file ID is returned by the server after the video is uploaded.

1. After a video is published from a client, the server will return a file ID to the client.
2. When the video is uploaded to the server, the corresponding `FileId` will be included in the notification of upload confirmation.

   If the video you want to play is already saved with VOD, you can go to Media Assets to view its file ID.



Note:

- To play by VOD file ID, you need to use the Adaptive-HLS template (ID: 10) to transcode the video or use the player signature `psign` to specify the video to play; otherwise, the playback may fail. For more information on how to transcode a video and generate `psign`, see Play back a video with the Player component and Player Signature.
- If a "no v4 play info" exception occurs during playback through `FileId`, the above problem may exist. In this case, we recommend you make adjustments as instructed above. You can also directly get the playback link of the source video for playback through URL.
- **We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback.**

```
// If you haven't enabled hotlink protection and a "no v4 play
info" error occurs, we recommend you transcode your video using
the Adaptive-HLS template (ID: 10) or get the playback URL of the
video and play it by URL.

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
```

```
model.appId = 1400329071;// Configure AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // Configure
`FileId`
// If you enable hotlink protection, you need to enter a `psign`
(player signature) for playback. For more information on the
signature and how to generate it, see [Player Signature]
(https://intl.cloud.tencent.com/document/product/266/38099).
//model.videoId.pSign =
@"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MS
wiZmlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXA
iOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6
eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0Y
W1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-
LzU";
[_playerView playWithModelNeedLicence:model];
```

Play by URL

```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.videoURL = @"http://your_video_url.mp4";   // Enter the URL of
the video to play
[_playerView playWithModelNeedLicence:model];
```

- **Stop playback.**

  If the player is no longer needed, call `resetPlayer` to reset the player and free up memory.

  ```
  [_playerView resetPlayer];
  ```

At this point, you have learned how to create a player, use it to play videos, and stop playback.

More Features

1. Full screen playback

The Player component supports full screen playback, where it allows setting screen lock, volume and brightness control through gestures, on-screen commenting, screencapturing, and definition selection. This feature can be tried out in TCToolkit App > **Player** > **Player Component**, and you can enter the full screen playback mode by clicking the full screen icon.
You can call the API below to go full screen from the windowed playback mode:

```
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player {
  // You can customize the logic after switching to the full screen mode
here
}
```

Features of full screen playback mode

Back to windowed mode

Tap the back button to return to the windowed mode. The delegate method that will be triggered after the SDK implements the logic for exiting full screen is as follows:

```
// The back button tapping event
- (void)superPlayerBackAction:(SuperPlayerView *)player;
Triggered by tapping of the back button at the top left
// The exit full screen notification
- (void)superPlayerFullScreenChanged:(SuperPlayerView *)player;
```

Enable screen locking

Screen locking disables touch screen and allows users to enter an immersive playback mode. The SDK will handle the tapping event and no callbacks will be sent.

```
// Use the API below to enable/disable screen locking
@property(nonatomic, assign) BOOL isLockScreen;
```

On-screen comments

After the on-screen commenting feature is enabled, text comments sent by users will be displayed on the screen.
Get the `SPDefaultControlView` object and, during initialization of the player view, set an event for the on-screen comment button of `SPDefaultControlView`. The on-screen comment content and view are customized by yourself. For details, see `CFDanmakuView`, `CFDanmakuInfo`, and `CFDanmaku` in `SuperPlayerDemo`.

```
SPDefaultControlView *dv = (SPDefaultControlView
*)**self**.playerView.controlView;
[dv.danmakuBtn addTarget:**self** action:**@selector**(danmakuShow:)
forControlEvents:UIControlEventTouchUpInside];
```

CFDanmakuView: Configure the attributes of on-screen commenting during initialization.

```
// The following attributes are required--------
// On-screen time
@property(nonatomic, assign) CGFloat duration;
// On-screen time in the center, at top, and at bottom
@property(nonatomic, assign) CGFloat centerDuration;
// On-screen comment line height
@property(nonatomic, assign) CGFloat lineHeight;
// Spacing between on-screen comment lines
@property(nonatomic, assign) CGFloat lineMargin;

// Maximum number of on-screen comment lines
@property(nonatomic, assign) NSInteger maxShowLineCount;

// Maximum number of on-screen comment lines in the center, at top,
and at bottom
@property(nonatomic, assign) NSInteger maxCenterLineCount;
```

Screenshot

The Player component allows users to take and save a screenshot of a video during playback. The SDK will handle the screenshot button tapping event and no callbacks will be sent for successful or failed screenshots. Screenshots are saved to the phone album.

Change resolution
Users can change the video definition (such as SD, HD, and FHD) during playback. After the definition selection button is tapped, the SDK will implement the logic for displaying the definition selection view and handle the selection event. No callbacks will be sent.

2. Floating window playback

The Player component supports playback in a small floating window, which allows users to switch to another page of the application without interrupting the video playback. You can try out this feature in TCToolkit App > **Player** > **Player Component** by clicking **Back** in the top-left corner.

```
// Tapping the back button during playback in portrait mode will trigger
the API
[SuperPlayerWindowShared setSuperPlayer:self.playerView];
[SuperPlayerWindowShared show];
// The API triggered by tapping the floating window to return to the
main window
SuperPlayerWindowShared.backController = self;
```

3. Thumbnail

The Player component supports customizing a video thumbnail, which is displayed before the callback is received for playing back the first video frame. This feature can be tried out in TCToolkit App > **Player** > **Player Component** > **Thumbnail Customization Demo**.

- When the Player component is set to the automatic playback mode `PLAY_ACTION_AUTO_PLAY`, the video will be played back automatically, and the thumbnail will be displayed before the first video frame is loaded.
- When the Player component is set to the manual playback mode `PLAY_ACTION_MANUAL_PLAY`, the video will be played back only after the user clicks **Play**. The thumbnail will be displayed until the first video frame is loaded.

You can set the thumbnail by specifying the URL of a local or online file. For detailed directions, see the code below. If you play by VOD file ID, you can also set the thumbnail in the VOD console.

```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
model.appId = 1400329071;
```

```
model.videoId = videoId;
// Playback mode, which can be set to automatic
(`PLAY_ACTION_AUTO_PLAY`) or manual (`PLAY_ACTION_MANUAL_PLAY`)
model.action  = PLAY_ACTION_MANUAL_PLAY;
// Specify the URL of an online file to use as the thumbnail. If
`coverPictureUrl` is not set, the thumbnail configured in the VOD
console will be used.
model.customCoverImageUrl =
@"http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500005830/cc1e282086
02268011087336518/MXUW1a5I9TsA.png";
[self.playerView playWithModelNeedLicence:model];
```

4. Video playlist loop

The Player component supports looping a video playlist:

- After a video ends, the next video in the list can be played automatically or users can manually start the next video.
- After the last video in the list ends, the first video in the list will start automatically.

You can try out this feature in TCToolkit App > **Player** > **Player Component** > **Video List Loop Demo**.

```
// Step 1. Create a `NSMutableArray` for the loop data
NSMutableArray *modelArray = [NSMutableArray array];
SuperPlayerModel *model = [SuperPlayerModel new];
SuperPlayerVideoId *videoId = [SuperPlayerVideoId new];
videoId.fileId = @"8602268011437356984";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];

model = [SuperPlayerModel new];
videoId = [SuperPlayerVideoId new];
videoId.fileId = @"4564972819219071679";
model.appId = 1252463788;
model.videoId = videoId;
[modelArray addObject:model];

// Step 2. Call the loop API of `SuperPlayerView`
[self.playerView playWithModelListNeedLicence:modelArray
isLoopPlayList:YES startIndex:0];
```

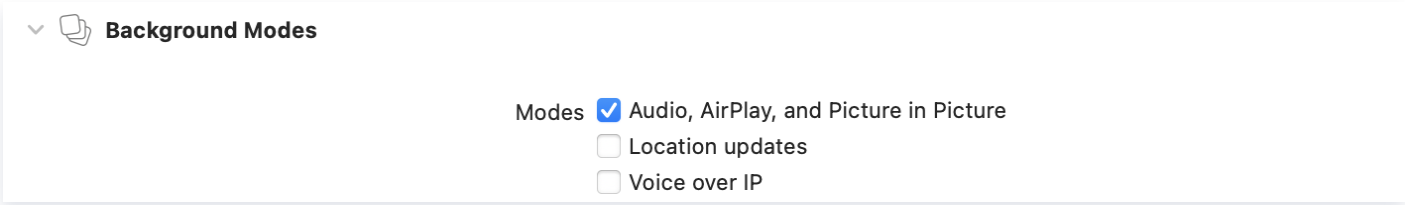
```
(void)playWithModelListNeedLicence:(NSArray *)playModelList
isLoopPlayList:(BOOL)isLoop startIndex:(NSInteger)index;
```

API parameters:

Parameter	Type	Description
playModelList	NSArray *	Loop data list
isLoop	Boolean	Whether to loop the playlist
index	NSInteger	Index of the video from which to start the playback

5. Picture-in-Picture (PiP) feature

The Picture-in-Picture (PiP) feature has been launched on iOS 9 but can currently be used only on iPads. To use PiP on an iPhone, you need to update the iOS version to iOS 14.
The Player component supports both in-app PiP and system-wide PiP. To use the feature, you need to enable background modes: In Xcode, choose your target, click **Signing & Capabilities** > **+Capability** > **Background Modes**, and select **Audio, AirPlay, and Picture in Picture**.



Code sample for using PiP capabilities:

```
// Enter the PiP mode
 if (![TXVodPlayer isSupportPictureInPicture]) {
    return;
 }
 [_vodPlayer enterPictureInPicture];

// Exit the PiP mode
[_vodPlayer exitPictureInPicture];
```

6. Video preview

The Player component supports video preview, which is useful if you want to allow non-subscribers to watch the beginning of a video. We offer parameters for you to set the video preview duration, pop-up message, and preview end screen. You can find a demo for this feature in the TCToolkit app: **Player** > **Player Component** > **Preview Feature Demo**.

```
// Step 1. Create a preview model
TXVipWatchModel *model = [[TXVipWatchModel alloc] init];
model.tipTtitle = @"You can preview 15 seconds of the video. Become a subscriber to watch the full video.";
model.canWatchTime = 15;
// Step 2. Set the preview model
self.playerView.vipWatchModel = model;
// Step 3. Call the method below to display the preview
[self.playerView showVipTipView];
```

`TXVipWatchModel` class parameter description:

Parameter	Type	Description
tipTtitle	NSString	Pop-up message
canWatchTime	float	Preview duration in seconds

7. Dynamic watermark

The Player component allows you to add a randomly moving text watermark to protect your content against piracy. Watermarks are visible in both the full screen mode and windowed mode. The text, font size, and color of a watermark are customizable. You can find a demo for this feature in the TCToolkit app: **Player** > **Player Component** > **Dynamic Watermark Demo**.

```
// Step 1. Create a video source information model
SuperPlayerModel *  playermodel   = [SuperPlayerModel new];
// Add other information of the video source
// Step 2. Create a dynamic watermark model
DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
// Step 3. Set the data of the dynamic watermark
model.dynamicWatermarkTip = @"shipinyun";
model.textFont = 30;
model.textColor = [UIColor colorWithRed:255.0/255.0 green:255.0/255.0 blue:255.0/255.0 alpha:0.8];
playermodel.dynamicWaterModel = model;
```

```
// Step 4. Call the method below to display the dynamic watermark
[self.playerView playWithModelNeedLicence:playermodel];
```

Parameters for `DynamicWaterModel`:

Parameter	Type	Description
dynamicWatermarkTip	NSString	Watermark text
textFont	CGFloat	Font size
textColor	UIColor	Text color

Demo

To try out more features, you can directly run the demo project or scan the QR code to download the TCToolkit App demo.

Running a demo project

1. In the `Demo` directory, run the `pod update` command to generate the `TXLiteAVDemo.xcworkspace` file again.
2. Double-click the file to open it, modify the certificate, and run the project on a real device.
3. After the demo is run successfully, go to **Player** > **Player Component** to try out the player features.

TCToolkit app

You can try out more features of the Player component in **TCToolkit App** > **Player**.

Android

最終更新日：：2022-11-14 18:22:09

Overview

The Tencent Cloud RT-Cube Player for Android is an open-source player component of Tencent Cloud. It integrates quality monitoring, video encryption, Top Speed Codec, definition selection, and small window playback and is suitable for all VOD and live playback scenarios. It encapsulates complete features and provides upper-layer UIs to help you quickly create a playback program comparable to mainstream video applications.
If the Player component cannot meet your custom requirements and you have development experience, you can integrate the RT-Cube Player SDK as instructed in Integration Guide to customize the player UI and playback features.

Prerequisites

1. To try out all features of the player, we recommend you activate VOD. If you don't have an account yet, sign up for one first. If you don't use the VOD service, you can skip this step; however, you will only be able to use basic player features after integration.
2. Download and install Android Studio. If you have already done so, skip this step.

Content Summary

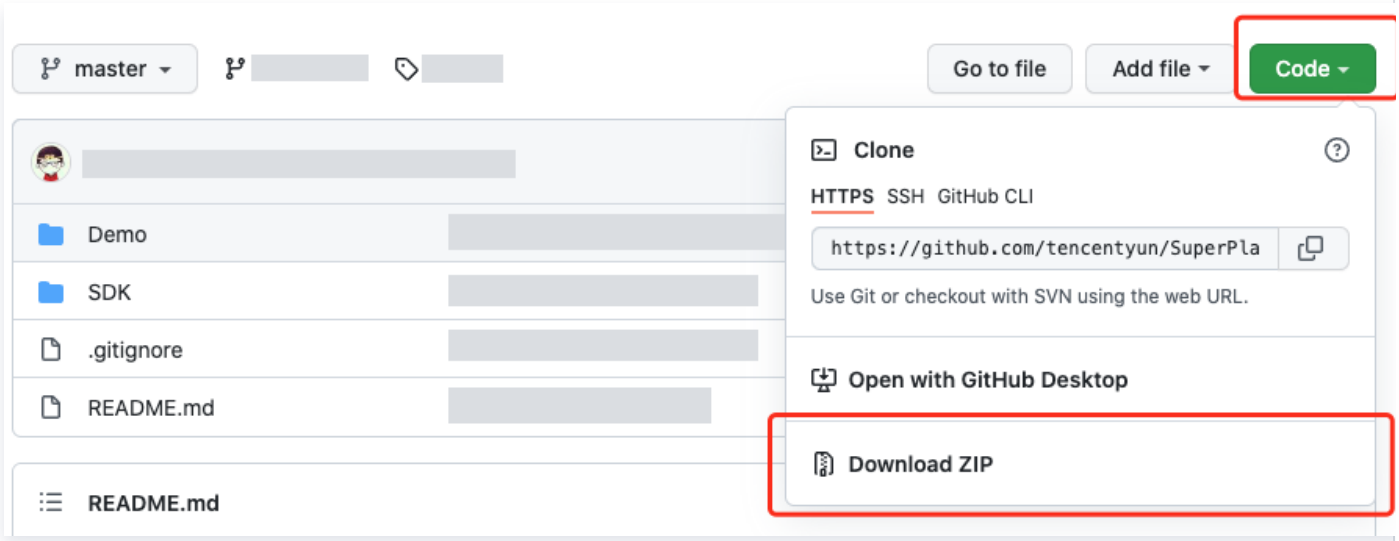
1. How to integrate the Player component for Android
2. How to create and use the player

Directions

Step 1. Download the player code package

GitHub page: LiteAVSDK/Player_Android
You can download the Player for Android by **downloading the Player component ZIP package** or **running the Git clone command**.

Download the ZIP file
Go to the Player GitHub page and click **Code > Download ZIP**.



Download using a Git command

1. First, make sure that your computer has Git installed; if not, you can install it as instructed in Git Installation Tutorial.
2. Run the following command to clone the code of the Player component to your local system.

```
git clone git@github.com:tencentyun/SuperPlayer_Android.git
```

If you see the following information, the project code has been cloned to your local system successfully.

```
Cloning to 'SuperPlayer_Android'...
remote: Enumerating objects: 2637, done.
remote: Counting objects: 100% (644/644), done.
remote: Compressing objects: 100% (333/333), done.
remote: Total 2637 (delta 227), reused 524 (delta 170), pack-
reused 1993
Receiving the object: 100% (2637/2637), 571.20 MiB | 3.94 MiB/s,
done.
Processing delta: 100% (1019/1019), done.
```

After the project is downloaded, the directory generated after decompression of the source code is as follows:

Filename	Description
LiteAVDemo(Player)	The Player demo project, which can be run directly after being imported into Android Studio.
app	The entry of the main UI
superplayerkit	The Player component (`SuperPlayerView`), which provides common features such as playback, pause, and gesture control.
superplayerdemo	The Player component demo code
common	Tool module
SDK	Player SDK, including `LiteAVSDK_Player_x.x.x.aar` (SDK provided in AAR format) and `LiteAVSDK_Player_x.x.x.zip` (SDKs provided in lib and JAR formats)
Player Documentation (Android).pdf	The Player component user guide

Step 2. Integrate the component

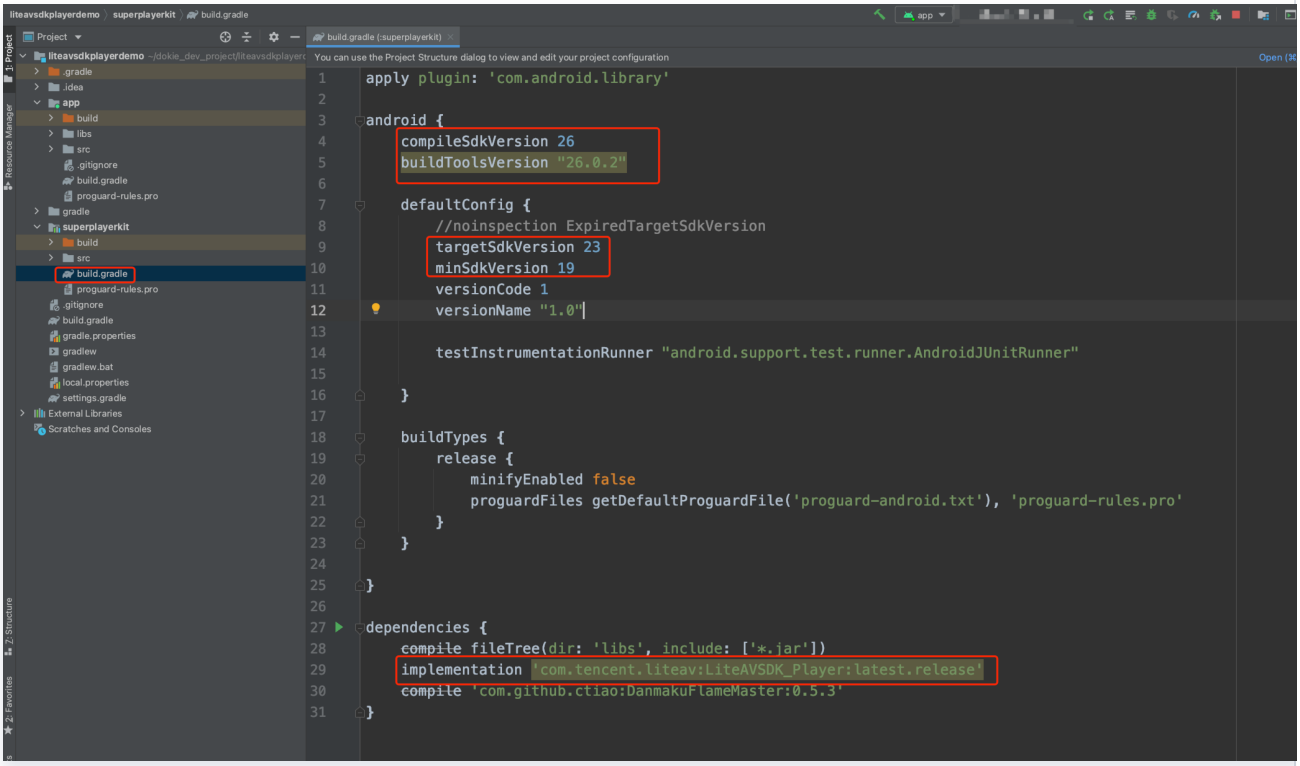
This step describes how to integrate the player. You can integrate the project by using Gradle for automatic loading, manually downloading the AAR and importing it into your current project, or importing the JAR and SO libraries.

Automatic loading in Gradle (AAR)

1. Download the SDK + demo package for Android here.
2. Copy the `Demo/superplayerkit` module to your project and then configure as follows:
   - Import `superplayerkit` into `setting.gradle` in your project directory.

```
include ':superplayerkit'
```

- Open the `build.gradle` file of the `superplayerkit` project and modify the constant values of `compileSdkVersion`, `buildToolsVersion`, `minSdkVersion`, `targetSdkVersion`, and `rootProject.ext.liteavSdk`.



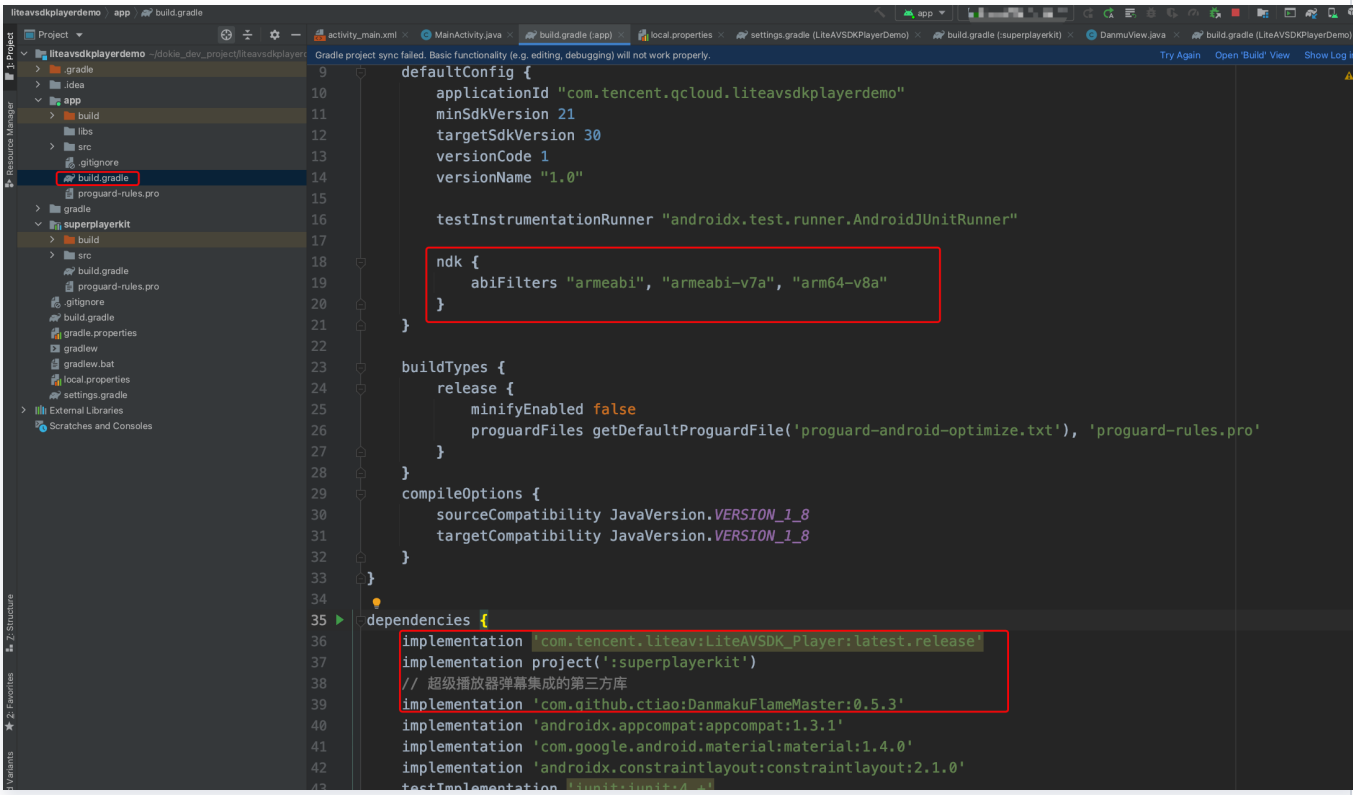

```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  // To integrate an older version, change `latest.release` to
the corresponding version number, such as `8.5.290009`
  implementation
'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

Import the `common` module into your project as instructed above and configure it.

3. Configure the `mavenCentral` repository in Gradle, and LiteAVSDK will be automatically downloaded and updated. Open `app/build.gradle` and configure as follows:



3.1 Add the `LiteAVSDK_Player` dependencies to `dependencies`.

```
dependencies {
    implementation
'com.tencent.liteav:LiteAVSDK_Player:latest.release'
    implementation project(':superplayerkit')
    // Third-party library for integration of the on-screen
commenting feature of the Player component
    implementation
'com.github.ctiao:DanmakuFlameMaster:0.5.3'
}
```

If you need to integrate an older version of the LiteAVSDK_Player SDK, view it in MavenCentral and then integrate it as instructed below:

```
dependencies {
      // Integrate the LiteAVSDK_Player SDK v8.5.10033
      implementation
'com.tencent.liteav:LiteAVSDK_Player:8.5.10033'
}
```

3.2 In the `defaultConfig` of `app/build.gradle`, specify the CPU architecture to be used by the application (currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a, which you can configure as needed).

```
ndk {
      abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

If you haven't used the download cache feature (APIs in TXVodDownloadManager) of the SDK v9.4 or earlier and don't need to play back the downloaded files in the SDK v9.5 or later, you don't need to use the SO file of the feature, which helps reduce the size of the installation package. For example, if you have downloaded a cached file by using the `setDownloadPath` and `startDownloadUrl` functions of the `TXVodDownloadManager` class in the SDK v9.4 or earlier, and the `getPlayPath` path called back by `TXVodDownloadManager` is stored in the application for subsequent playback, you will need `libijkhlscache-master.so` to play back the file at the `getPlayPath` path; otherwise, you won't need it. You can add the following to `app/build.gradle`:

```
packagingOptions{
    exclude "lib/armeabi/libijkhlscache-master.so"
    exclude "lib/armeabi-v7a/libijkhlscache-master.so"
    exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

3.3 Add the `mavenCentral` repository to the `build.gradle` in your project directory.

```
repositories {
  mavenCentral()
}
```

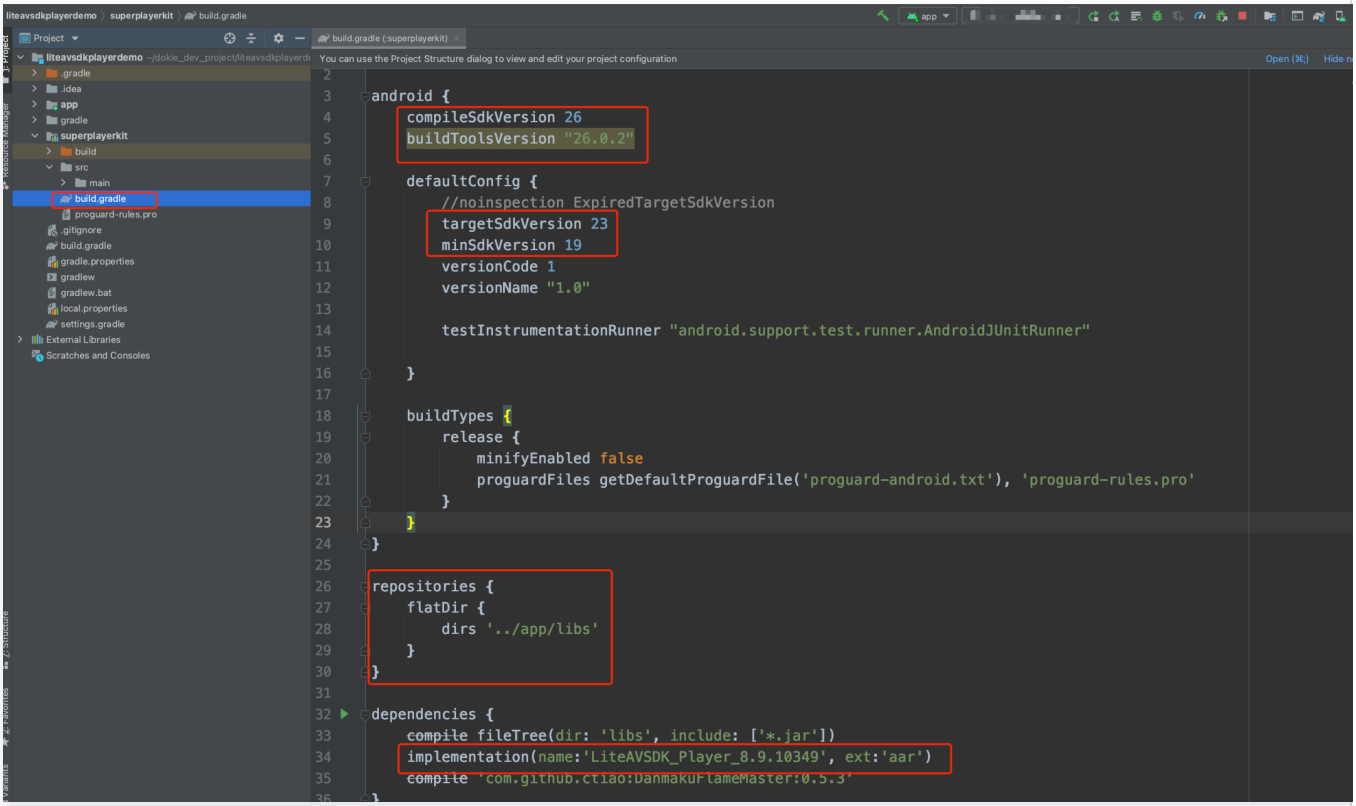
4. Click **Sync Now** to sync the SDK. If `mavenCentral` can be connected to, the SDK will be automatically downloaded and integrated into the project very soon.

Manual download in Gradle (AAR)

1. Download the SDK + demo package for Android here.
2. Import `SDK/LiteAVSDK_Player_XXX.aar` (`XXX` is the version number) into the `libs` folder under `app` and copy the `Demo/superplayerkit` module to the project.
3. Import `superplayerkit` into `setting.gradle` in your project directory.

```
include ':superplayerkit'
```

4. Open the `build.gradle` file of the `superplayerkit` project and modify the constant values of `compileSdkVersion`, `buildToolsVersion`, `minSdkVersion`, `targetSdkVersion`, and `rootProject.ext.liteavSdk`.



```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  implementation(name:'LiteAVSDK_Player_8.9.10349', ext:'aar')
}
```

Import the `common` module into your project as instructed above and configure it.

- Configure `repositories`

```
repositories {
flatDir {
        dirs '../app/libs'
}
}
```

5. Add dependencies to `app/build.gradle`:

```
compile(name:'LiteAVSDK_Player_8.9.10349', ext:'aar')
implementation project(':superplayerkit')
// Third-party library for integration of the on-screen
commenting feature of the Player component
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
```

6. Add the following to the project's `build.gradle`:

```
allprojects {
  repositories {
          flatDir {
                  dirs 'libs'
          }
  }
}
```

7. In the `defaultConfig` of `app/build.gradle`, specify the CPU architecture to be used by the application (currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a).

```
ndk {
  abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

If you haven't used the download cache feature (APIs in TXVodDownloadManager) of the SDK v9.4 or earlier and don't need to play back the downloaded files in the SDK v9.5 or later, you don't need to use the SO file of the feature, which helps reduce the size of the installation package. For example, if you have downloaded a cached file by using the `setDownloadPath` and `startDownloadUrl` functions of the `TXVodDownloadManager` class in the SDK v9.4 or earlier, and the `getPlayPath` path called back by `TXVodDownloadManager` is stored in the application for subsequent playback, you will need `libijkhlscache-master.so` to play back the file at the `getPlayPath` path; otherwise, you won't need it. You can add the following to `app/build.gradle`:

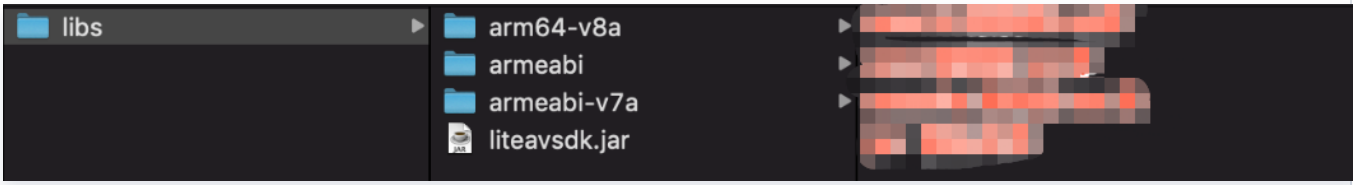
```
packagingOptions{
 exclude "lib/armeabi/libijkhlscache-master.so"
 exclude "lib/armeabi-v7a/libijkhlscache-master.so"
 exclude "lib/arm64-v8a/libijkhlscache-master.so"
}
```

8. Click **Sync Now** to sync the SDK.

SDK integration (jar + so)

If you do not want to import the AAR library, you can also integrate LiteAVSDK by importing JAR and SO libraries.

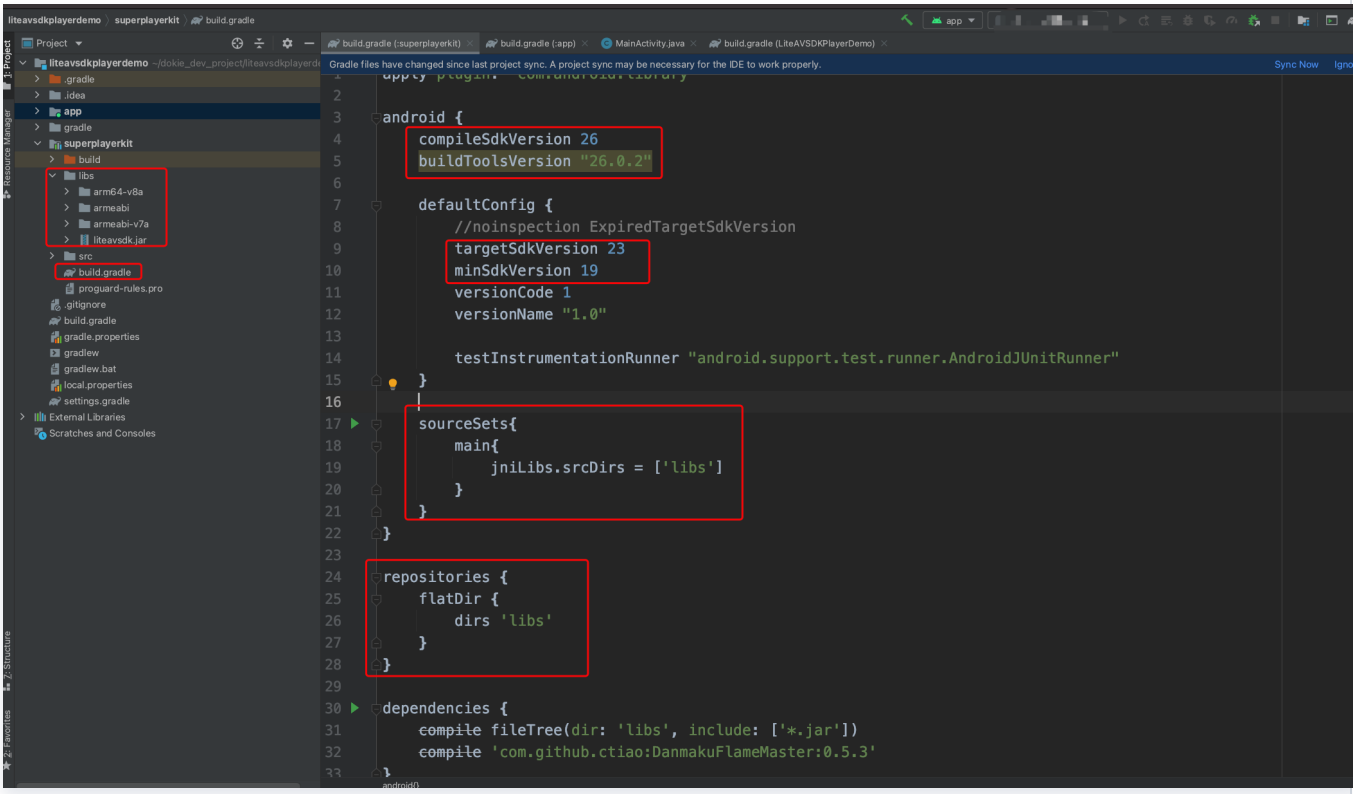
1. Download the SDK + demo package for Android here and decompress it. Find `SDK/LiteAVSDK_Player_XXX.zip` (`XXX` is the version number) in the SDK directory. After decompression, you can get the `libs` directory, which contains the JAR file and folders of SO files as listed below:



2. Copy the `Demo/superplayerkit` module to your project and import `superplayerkit` into `setting.gradle` in your project directory.

```
include ':superplayerkit'
```

3. Copy the `libs` folder obtained by decompression in step 1 to the `superplayerkit` project root directory.
4. Modify the `superplayerkit/build.gradle` file:




```
compileSdkVersion 26
```

```
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}
```

Import the `common` module into your project as instructed above and configure it.

- Configure `sourceSets` and add the SO library import code.

```
sourceSets{
    main{
            jniLibs.srcDirs = ['libs']
    }
 }
```

- Configure `repositories`, add `flatDir`, and specify the path of the local repository.

```
repositories {
flatDir {
        dirs 'libs'
}
}
```

5. In the `defaultConfig` of `app/build.gradle`, specify the CPU architecture to be used by the application (currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a).

```
ndk {
  abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

6. Click **Sync Now** to sync the SDK.

At this point, you have completed integrating the RT-Cube Player for Android.

Step 3. Configure application permissions

Configure permissions for your application in `AndroidManifest.xml`. LiteAVSDK needs the following permissions:

```
<!--network permission-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!--VOD player floating window permission -->
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"
/>
<!--storage-->
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
```

Step 4. Set obfuscation rules

In the `proguard-rules.pro` file, add the classes related to the TRTC SDK to the "do not obfuscate" list:

```
-keep class com.tencent.** { *;}
```

At this point, you have completed configuring permissions for the RT-Cube Player application for Android.

Step 5. Use the player features

This step describes how to create a player and use it for video playback.

1. **Player creation.**

   The main class of the player is `SuperPlayerView`, and videos can be played back after it is created. `FileId` or URL can be integrated for playback. Create `SuperPlayerView` in the layout file:

   ```
   <!-- Player component -->
   <com.tencent.liteav.demo.superplayer.SuperPlayerView
    android:id="@+id/superVodPlayerView"
    android:layout_width="match_parent"
    android:layout_height="200dp" />
   ```

2. **License configuration.**

   If you have the required license, get the license URL and key in the RT-Cube console.

   If you don't have the required license, contact us to get it.

After obtaining the license information, before calling relevant APIs of the SDK, initialize the license through the following API. We recommend you set the following in the `Application` class:

```
public class MApplication extends Application {

 @Override
 public void onCreate() {
     super.onCreate();
     String licenceURL = ""; // The license URL obtained
     String licenceKey = ""; // The license key obtained
     TXLiveBase.getInstance().setLicence(this, licenceURL,
licenceKey);
     TXLiveBase.setListener(new TXLiveBaseListener() {
         @Override
         public void onLicenceLoaded(int result, String reason) {
             Log.i(TAG, "onLicenceLoaded: result:" + result + ",
reason:" + reason);
         }
     });
 }
}
```

3. **Video playback.**

   This step describes how to play back a video. The RT-Cube Player for Android can be used for VOD and live playback as follows:

   - VOD playback: The Player component supports two VOD playback methods, namely, through `FileID` or URL.
   - Live playback: The Player component can use the playback through URL method for live playback. A live audio/video stream can be pulled for playback simply by passing in its URL. For more information on how to generate a Tencent Cloud live streaming URL, see Splicing Live Streaming URLs.

VOD and live playback through URL

A URL can be the playback address of a VOD file or the pull address of a live stream. A video file can be played back simply by passing in its URL.

```
SuperPlayerModel model = new SuperPlayerModel();
```

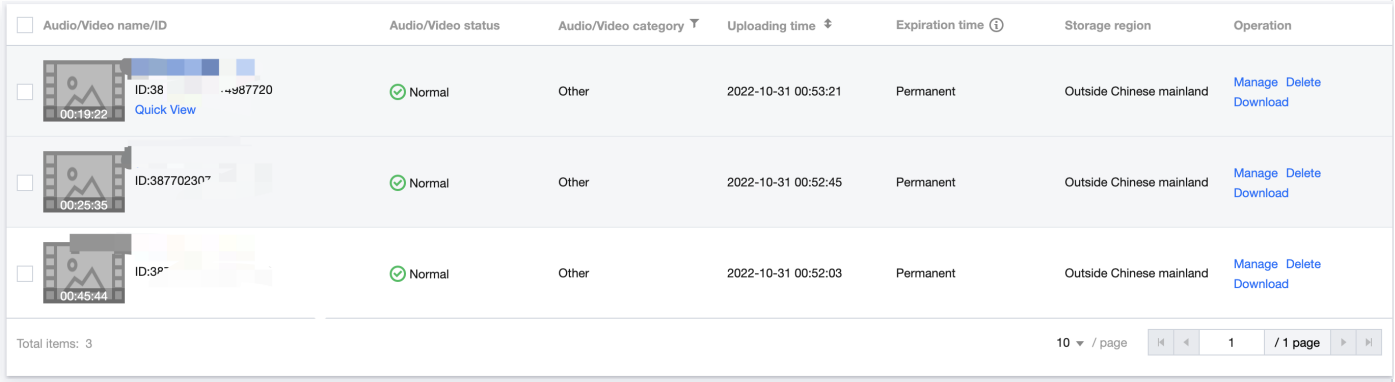
```
model.appId = 1400329073; // Configure `AppId`
model.url = "http://your_video_url.mp4";    // Configure a URL for
your video for playback
mSuperPlayerView.playWithModelNeedLicence(model);
```

VOD playback through `FileID`

A video file ID is returned by the server after the video is uploaded.

1. After a video is published from a client, the server will return a file ID to the client.
2. After a video is uploaded to the server, the notification for successful upload will contain a file ID for the video.

If the video you want to play is already saved with VOD, you can go to Media Assets to view its file ID.



Note:

- To play by VOD file ID, you need to use the Adaptive-HLS template (ID: 10) to transcode the video or use the player signature `psign` to specify the video to play; otherwise, the playback may fail. For more information on how to transcode a video and generate `psign`, see Play back a video with the Player component and Player Signature.
- If a "no v4 play info" exception occurs during playback through `FileId`, the above problem may exist. In this case, we recommend you make adjustments as instructed above. You can also directly get the playback link of the source video for playback through URL.
- **We recommend you transcode videos for playback because untranscoded videos may experience compatibility issues during playback.**

```
// If you haven't enabled hotlink protection and a "no v4 play info"
error occurs, we recommend you transcode your video using the
Adaptive-HLS template (ID: 10) or get the playback URL of the video
and play it by URL.

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// Configure AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // Configure `FileId`
// If you enable hotlink protection, you need to enter a `psign`
(player signature) for playback. For more information on the
signature and how to generate it, see [Player Signature]
(https://intl.cloud.tencent.com/document/product/266/38099).
//model.videoId.pSign =
@"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZ
mlsZUlkIjoiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsI
mV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN
2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3N
DgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
[_playerView playWithModelNeedLicence:model];
```

4. **Playback exit.**

   If the player is no longer needed, call `resetPlayer` to reset the player and free up memory.

   ```
   mSuperPlayerView.resetPlayer();
   ```

At this point, you have learned how to create a player, use it to play videos, and stop playback.

More Features

This section describes several common player features. For more features, see Demo. For features supported by the Player component, see Feature Description.

1. Full screen playback

The Player component supports full screen playback. In full screen mode, users can lock the screen, control volume and brightness with gestures, send on-screen comments, take screenshots, and switch the video

definition. You can try out this feature in TCToolkit App > **Player** > **Player Component**, and you can enter the full screen playback mode by clicking the full screen icon in the bottom-right corner.
You can call the API below to enter full screen from the windowed playback mode:

```
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.FULLSCREEN);
```

Features of full screen playback mode

Return to the window

Tap **Back** to return to the window playback mode.

```
// API triggered after tapping
mControllerCallback.onBackPressed(SuperPlayerDef.PlayerMode.FULLSCREEN);
onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

Screen lock

Screen locking disables touch screen and allows users to enter an immersive playback mode.

```
// API triggered after tapping
toggleLockState();
```

On-screen comments

After the on-screen commenting feature is enabled, text comments sent by users will be displayed on the screen.

```
// Step 1. Add an on-screen comment to the on-screen comment view
addDanmaku(String content, boolean withBorder);
// Step 2. Enable or disable on-screen commenting
toggleBarrage();
```

Screenshot

The Player component allows users to take and save a screenshot of a video during playback. Click the button in image 4 to capture the screen, and you can save the captured screenshot with the `mSuperPlayer.snapshot` API.

```
mSuperPlayer.snapshot(new TXLivePlayer.ITXSnapshotListener() {
  @Override
  public void onSnapshot(Bitmap bitmap) {
        // The captured screenshot can be saved here
  }
});
```

Change resolution

Users can change the video definition (such as SD, HD, and FHD) during playback.

```
// The API for displaying the definition selection view triggered
after the button is tapped
showQualityView();
// The callback API for tapping the definition option is as follows
mListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
    // The event of tapping the definition list view
    VideoQuality quality = mList.get(position);
    mCallback.onQualitySelect(quality);
  }
});
// Callback for the selected definition
@Override
public void onQualityChange(VideoQuality quality) {
   mFullScreenPlayer.updateVideoQuality(quality);
```

```
    mSuperPlayer.switchStream(quality);
}
```

2. Floating window playback

The Player component supports playback in a small floating window, which allows users to switch to another application without interrupting the video playback. You can try out this feature in TCToolkit App > **Player** > **Player Component** by clicking **Back** in the top-left corner.
Floating window playback relies on the following permission in `AndroidManifest`:

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"
/>
```

```
// The API triggered by switching to the floating window
mSuperPlayerView.switchPlayMode(SuperPlayerDef.PlayerMode.FLOAT);
// The API triggered by tapping the floating window to return to the
main window
mControllerCallback.onSwitchPlayMode(SuperPlayerDef.PlayerMode.WINDOW);
```

3. Thumbnail

The Player component supports customizing a video thumbnail, which is displayed before the callback is received for playing back the first video frame. You can try out this feature in TCToolkit App > **Player** > **Player Component** > **Thumbnail Customization Demo**.

- When the Player component is set to the automatic playback mode `PLAY_ACTION_AUTO_PLAY`, the video will be played back automatically, and the thumbnail will be displayed before the first video frame is loaded.
- When the Player component is set to the manual playback mode `PLAY_ACTION_MANUAL_PLAY`, the video will be played back only after the user clicks **Play**. The thumbnail will be displayed until the first video frame is loaded.

You can set the thumbnail by specifying the URL of a local or online file. For detailed directions, see the code below. If you play by VOD file ID, you can also set the thumbnail in the VOD console.

```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = "Your `appid`";
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "Your `fileId`";
```

```
// Playback mode, which can be set to automatic
(`PLAY_ACTION_AUTO_PLAY`) or manual (`PLAY_ACTION_MANUAL_PLAY`)
model.playAction = PLAY_ACTION_MANUAL_PLAY;
// Specify the URL of an online file to use as the thumbnail. If
`coverPictureUrl` is not set, the thumbnail configured in the VOD
console will be used.
model.coverPictureUrl =
"http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500005830/cc1e2820860
2268011087336518/MXUW1a5I9TsA.png"
mSuperPlayerView.playWithModelNeedLicence(model);
```

4. Video playlist loop

The Player component supports looping a video playlist:

- After a video ends, the next video in the list can be played automatically or users can manually start the next video.
- After the last video in the list ends, the first video in the list will start automatically.

This feature can be tried out in TCToolkit App > **Player** > **Player Component** > **Video List Loop Demo**.

```
// Step 1. Create a loop list<SuperPlayerModel>
ArrayList<SuperPlayerModel> list = new ArrayList<>();
SuperPlayerModel model = new VideoModel();
model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071568";
list.add(model);

model = new SuperPlayerModel();
model.videoId = new SuperPlayerVideoId();
model.appid = 1252463788;
model.videoId.fileId = "4564972819219071679";
list.add(model);
// Step 2. Call the loop API
mSuperPlayerView.playWithModelListNeedLicence(list, true, 0);
```

```
public void playWithModelListNeedLicence(List<SuperPlayerModel> models,
boolean isLoopPlayList, int index);
```

API parameters:

Parameter	Type	Description
models	List	Loop data list
isLoopPlayList	boolean	Whether to loop video playback
index	int	Index of `SuperPlayerModel` from which to start the playback

5. Preview

The Player component supports the video preview feature, which allows non-member viewers to view a preview of the video. You can pass in different parameters to control the video preview duration, prompt message, and preview end screen. You can try out this feature in Tencent Cloud Toolkit App > **Player** > **Player Component** > **Preview Feature Demo**.

```
 Method 1:
 // Step 1. Create a video model
 SuperPlayerModel mode = new SuperPlayerModel();
 //... Add the video source information
 // Step 2. Create a preview information model
 VipWatchModel vipWatchModel = new VipWatchModel("You can preview %ss
and activate the VIP membership to watch the full video",15);
 mode.vipWatchMode = vipWatchModel;
 // Step 3. Call the method for playing back videos
 mSuperPlayerView.playWithModelNeedLicence(mode);

 Method 2:
 // Step 1. Create a preview information model
 VipWatchModel vipWatchModel = new VipWatchModel("You can preview %ss
and activate the VIP membership to watch the full video",15);
  // Step 2. Call the method for setting the preview feature
 mSuperPlayerView.setVipWatchModel(vipWatchModel);
```

```
public VipWatchModel(String tipStr, long canWatchTime)
```

`VipWatchModel` API parameter description:

Parameter	Type	Description
tipStr	String	Preview prompt message
canWatchTime	Long	Preview duration in seconds

6. Dynamic watermark

The Player component allows you to add a randomly moving text watermark to protect your content against piracy. Watermarks are visible in both the full screen mode and windowed mode. The text, font size, and color of a watermark are customizable. You can find a demo for this feature in the TCToolkit app: **Player** > **Player Component** > **Dynamic Watermark Demo**.

```
 Method 1:
 // Step 1. Create a video model
 SuperPlayerModel mode = new SuperPlayerModel();
 //... Add the video source information
 // Step 2. Create a watermark information model
 DynamicWaterConfig dynamicWaterConfig = new
DynamicWaterConfig("shipinyun", 30, Color.parseColor("#80FFFFFF"));
 mode.dynamicWaterConfig = dynamicWaterConfig;
 // Step 3. Call the method for playing back videos
 mSuperPlayerView.playWithModelNeedLicence(mode);

 Method 2:
 // Step 1. Create a watermark information model
 DynamicWaterConfig dynamicWaterConfig = new
DynamicWaterConfig("shipinyun", 30, Color.parseColor("#80FFFFFF"));
  // Step 2. Call the method for setting the dynamic watermark feature
 mSuperPlayerView.setDynamicWatermarkConfig(dynamicWaterConfig);
```

```
public DynamicWaterConfig(String dynamicWatermarkTip, int tipTextSize,
int tipTextColor)
```

API parameters:

Parameter	Type	Description
dynamicWatermarkTip	String	Watermark text information
tipTextSize	int	Text size

tipTextColor	int	Text color

Demo

To try out more features, you can directly run the project demo or scan the QR code to download the TCToolkit App demo.

Running a demo project

1. Select **File** > **Open** on the navigation bar of Android Studio. In the pop-up window, select the `$SuperPlayer_Android/Demo` directory of the **demo** project. After the demo project is imported successfully, click **Run app** to run the demo.
2. After running the demo successfully, go to **Player** > **Player Component** to try out the player features.

TCToolkit app

You can try out more features of the Player component in **TCToolkit App** > **Player**.

Tencent Effect SDK
SDK Features

最終更新日：：2022-06-24 14:59:35

The Tencent Effect SDK comes in 11 editions, which fall into two categories: the basic A series and advanced S series. For details about the features of different editions, see the table below.

Basic A series

Basic A editions offer common beautification features and are for customers who do not have high requirements on face editing.

Feature		Edition					
		A1 – 01	A1 – 02	A1 – 03	A1 – 04	A1 – 05	A1 – 06
Basic	Basic beauty filtersBrightening, skin smoothing, and blush	✓	✓	✓	✓	✓	✓
	Image settingsContrast, saturation, and sharpness	✓	✓	✓	✓	✓	✓
	Basic beautificationEye enlarging and face slimming (natural, attractive, and handsome)	✓	✓	✓	✓	✓	✓
	Filters10 general filters by default	✓	✓	✓	✓	✓	✓
Extended	Stickers(10 2D general stickers for free)	–	✓	✓	✓	✓	✓
	General beautification SDKs(Face narrowing/Chin reshaping/Hairline adjustment/Nose narrowing)	–	–	✓	–	–	–
	Gesture recognition(One gesture sticker for free)	–	–	–	✓	–	–
	Keying/Virtual	–	–	–	–	✓	–

	background(Three keying stickers for free)						
	Makeup(Three full-face makeup looks for free)	-	-	-	-	-	✓
SDK Download	iOS & Android	-					

Advanced S series

Advanced S editions offer enhanced beautification features (including stickers and makeup looks) and are for customers with high requirements on face editing.

Feature		Edition				
		S1 - 00	S1 - 01	S1 - 02	S1 - 03	S1 - 04
Basic	**Basic beauty filters** Brightening, skin smoothing, and blush	✓	✓	✓	✓	✓
	Image settings Contrast, saturation, and sharpness	✓	✓	✓	✓	✓
	Advanced beautification Eye enlarging, face narrowing, face slimming (natural, attractive, and handsome), chin slimming, chin reshaping, face shortening, face reshaping, hairline adjustment, eye brightening, eye distance adjustment, eye corner adjustment, nose slimming, nose wing narrowing, cheekbone slimming, nose repositioning, teeth whitening, wrinkle removal, smile line removal, eye bag removal, mouth reshaping, lip thickness adjustment, lipstick application, blush, and facial contouring	✓	✓	✓	✓	✓

	Filters (General filters by default)	✓	✓	✓	✓	✓
	Stickers (2D general stickers by default)	--	✓	✓	✓	✓
	Advanced stickers (3D general stickers by default)	-	✓	✓	✓	✓
	Makeup Full face makeup	-	✓	✓	✓	✓
Extended	**Gesture recognition** (One gesture sticker for free)	-	-	✓	-	✓
	Keying/Virtual background (Three keying stickers for free)	-	-	-	✓	✓
SDK Download	iOS & Android	-				

SDK Integration Guide
iOS

最終更新日：：2022-11-14 18:18:58

Preparations

1. Download and decompress the demo package as described in Demos, and copy the `xmagickit` folder in the `demo/XiaoShiPin/` directory in the demo project to the directory of the Podfile of your project.
2. Add the following dependencies to your Podfile and run `pod install` to import the component.

```
pod 'xmagickit', :path => 'xmagickit/xmagickit.podspec'
```

3. Replace the `Bundle ID` with the one under the obtained trial license.

Developer environment requirements

- Xcode 11 or later: Download on App Store or here.
- Recommended runtime environment:
  - Device requirements: iPhone 5 or later. iPhone 6 and older models support up to 720p for front camera.
  - System requirements: iOS 12.0 or later.

SDK API Integration

Step 1. Initiate authentication

Add the following code to `didFinishLaunchingWithOptions` of `AppDelegate` (set `LicenseURL` and `LicenseKey` according to the authorization information you get from the Tencent Cloud website):

```
[TXUGCBase setLicenceURL:LicenseURL key:LicenseKey];

[TELicenseCheck setTELicense:LicenseURLkey:LicenseKey
completion:^(NSInteger authresult, NSString * _Nonnull errorMsg) {
              if (authresult == TELicenseCheckOk) {
                   NSLog(@"Authentication successful");
               } else {
```

```
                NSLog(@"Authentication failed");
            }
        }];
```

Authentication `errorCode` description:

Error Code	Description
0	Succeeded.
-1	The input parameter is invalid; for example, the `URL` or `KEY` is empty.
-3	Download failed. Check the network settings.
-4	The Tencent Effect SDK authorization information read from the local system is empty, which may be caused by an I/O failure.
-5	The content of the read `v_cube.license` file is empty, which may be caused by an I/O failure.
-6	The JSON field in the `v_cube.license` file is incorrect. Contact Tencent Cloud for assistance.
-7	Signature verification failed. Contact Tencent Cloud for assistance.
-8	Decryption failed. Contact Tencent Cloud for assistance.
-9	The JSON field in the `TELicense` field is incorrect. Contact Tencent Cloud for assistance.
-10	The Tencent Effect SDK authorization information parsed online is empty. Contact Tencent Cloud for assistance.
-11	Failed to write the Tencent Effect SDK authorization information to the local file, which may be caused by an I/O failure.
-12	Failed to download and failed to parse local assets.
-13	Authentication failed.
Others	Contact Tencent Cloud for assistance.

Step 2. Set the path of SDK materials

```
CGSize previewSize = [self
getPreviewSizeByResolution:self.currentPreviewResolution];
NSString *beautyConfigPath =
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) lastObject];
beautyConfigPath = [beautyConfigPath
stringByAppendingPathComponent:@"beauty_config.json"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
BOOL isDir = YES;
NSDictionary * beautyConfigJson = @{};
if ([localFileManager fileExistsAtPath:beautyConfigPath
isDirectory:&isDir] && !isDir) {
    NSString *beautyConfigJsonStr = [NSString
stringWithContentsOfFile:beautyConfigPath encoding:NSUTF8StringEncoding
error:nil];
    NSError *jsonError;
    NSData *objectData = [beautyConfigJsonStr
dataUsingEncoding:NSUTF8StringEncoding];
    beautyConfigJson = [NSJSONSerialization
JSONObjectWithData:objectData

options:NSJSONReadingMutableContainers
                                          error:&jsonError];
}
NSDictionary *assetsDict = @{@"core_name":@"LightCore.bundle",
                             @"root_path":[[NSBundle mainBundle]
bundlePath],
                             @"tnn_"
                             @"beauty_config":beautyConfigJson
};
// Init beauty kit
self.beautyKit = [[XMagic alloc] initWithRenderSize:previewSize
assetsDict:assetsDict];
```

Step 3. Add the log and event listener

```
// Register log
[self.beautyKit registerSDKEventListener:self];
```

```
[self.beautyKit registerLoggerListener:self
withDefaultLevel:YT_SDK_ERROR_LEVEL];
```

Step 4. Configure beauty filter effects

```
- (int)configPropertyWithType:(NSString *_Nonnull)propertyType withName:
(NSString *_Nonnull)propertyName withData:
(NSString*_Nonnull)propertyValue withExtraInfo:(id _Nullable)extraInfo;
```

Step 5. Render videos

In the preprocessing frame callback, construct `YTProcessInput` and pass `textureId` to the SDK for rendering.

```
 [self.xMagicKit process:inputCPU withOrigin:YtLightImageOriginTopLeft
withOrientation:YtLightCameraRotation0]
```

Step 6. Pause/Resume the SDK

```
[self.beautyKit onPause];
[self.beautyKit onResume];
```

Step 7. Add the SDK beauty filter panel to the layout

```
UIEdgeInsets gSafeInset;
#if __IPHONE_11_0 && __IPHONE_OS_VERSION_MAX_ALLOWED >= __IPHONE_11_0
if(gSafeInset.bottom > 0){
}
if (@available(iOS 11.0, *)) {
    gSafeInset = [UIApplication
sharedApplication].keyWindow.safeAreaInsets;
} else
#endif
    {
        gSafeInset = UIEdgeInsetsZero;
    }

dispatch_async(dispatch_get_main_queue(), ^{
```

```
    // Beauty filter option UI
    _vBeauty = [[BeautyView alloc] init];
    [self.view addSubview:_vBeauty];
    [_vBeauty mas_makeConstraints:^(MASConstraintMaker *make) {
        make.width.mas_equalTo(self.view);
        make.centerX.mas_equalTo(self.view);
        make.height.mas_equalTo(254);
        if(gSafeInset.bottom > 0.0){  // Adapt to full-view screen
            make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(0);
        } else {

make.bottom.mas_equalTo(self.view.mas_bottom).mas_offset(-10);
        }
    }];
        _vBeauty.hidden = YES;
});
```

Android

最終更新日：：2022-11-14 18:18:58

Step 1. Replace resources

1. Download the UGSV demo which has been integrated with the Tencent Effect SDK. This demo is built based on the Tencent Effect SDK S1-04 edition.
2. Replace resources: As the SDK edition used by the demo project may be different from the SDK edition you actually use, you need to replace the different SDK files in the demo with the files in the SDK edition you actually use as follows:
   - In the `build.gradle` file of the `xmagickit` module, find the following:

     ```
     api 'com.tencent.mediacloud:TencentEffect_S1-04:latest.release'
     ```

     Replace it with the SDK edition you purchased as described in Integrating the Tencent Effect SDK (Android).
   - If your edition contains animated effects and filters, you need to download the corresponding resources on the Tencent Effect SDK download page and put them in the following directories of the `xmagickit` module respectively:
     - Animated effects: `../assets/MotionRes`.
     - Filters: `../assets/lut`.
3. Import the `xmagickit` module from the demo into your actual project.

Step 2. Open `build.gradle` in `app` and do the following:

Replace the `applicationId` with the package name under the obtained trial license.

Step 3. Integrate the SDK APIs

You can refer to the `UGCKitVideoRecord` class of the demo.

1. **Authorize:**

```
 // For details about authentication and error codes, see
https://intl.cloud.tencent.com/document/product/1143/45385#.E6.AD.A5.E
9.AA.A4.E4.B8.80.EF.BC.9A.E9.89.B4.E6.9D.83
 XMagicImpl.checkAuth(new TELicenseCheck.TELicenseCheckListener() {
         @Override
         public void onLicenseCheckFinish(int errorCode, String msg)
{
```

```
            if (errorCode == TELicenseCheck.ERROR_OK) {
                loadXmagicRes();
            } else {
                Log.e("TAG", "auth fail, please check auth url and
key" + errorCode + " " + msg);
            }
        }
    });
```

2. Initialize the material:

```
 private void loadXmagicRes() {
    if (XMagicImpl.isLoadedRes) {
        XmagicResParser.parseRes(mActivity.getApplicationContext());
        initXMagic();
        return;
    }
    new Thread(new Runnable() {
        @Override
        public void run() {

XmagicResParser.copyRes(mActivity.getApplicationContext());

XmagicResParser.parseRes(mActivity.getApplicationContext());
            XMagicImpl.isLoadedRes = true;
            new Handler(Looper.getMainLooper()).post(new Runnable()
{
                @Override
                public void run() {
                    initXMagic();
                }
            });
        }
    }).start();
 }
```

3. Bind beauty filters to UGSV:

```
 private void initBeauty() {
```

```
        TXUGCRecord instance =
TXUGCRecord.getInstance(UGCKit.getAppContext());
        instance.setVideoProcessListener(new
TXUGCRecord.VideoCustomProcessListener() {
            @Override
            public int onTextureCustomProcess(int textureId, int width,
int height) {
                if (xmagicState == XMagicImpl.XmagicState.STARTED &&
mXMagic != null) {
                    return mXMagic.process(textureId, width, height);
                }
                return textureId;
            }

            @Override
            public void onDetectFacePoints(float[] floats) {
            }

            @Override
            public void onTextureDestroyed() {
                if (Looper.getMainLooper() != Looper.myLooper()) {  //
Not the main thread
                    boolean stopped = xmagicState ==
XMagicImpl.XmagicState.STOPPED;
                    if (stopped || xmagicState ==
XMagicImpl.XmagicState.DESTROYED) {
                        if (mXMagic != null) {
                            mXMagic.onDestroy();
                        }
                    }
                    if (xmagicState == XMagicImpl.XmagicState.DESTROYED)
{

TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListen
er(null);
                    }
                }
            }
        });
    }
```

4. **Pause/Terminate the SDK:**

`onPause()` is used to pause the beauty filter effect, which can be executed in the `Activity/Fragment` lifecycle method. The `onDestroy` method needs to be called in the GL thread (the `onDestroy()` of the `XMagicImpl` object can be called in the `onTextureDestroyed` method). For more information, see the `onTextureDestroyed` method in the sample code.

```
        @Override
        public void onTextureDestroyed() {
            if (Looper.getMainLooper() != Looper.myLooper()) {  //
Not the main thread
                boolean stopped = xmagicState ==
XMagicImpl.XmagicState.STOPPED;
                if (stopped || xmagicState ==
XMagicImpl.XmagicState.DESTROYED) {
                    if (mXMagic != null) {
                        mXMagic.onDestroy();
                    }
                }
                if (xmagicState == XMagicImpl.XmagicState.DESTROYED)
{

TXUGCRecord.getInstance(UGCKit.getAppContext()).setVideoProcessListen
er(null);
                }
            }
        }
```

5. **Add the beauty filter panel to the layout:**

```
 <RelativeLayout
    android:id="@+id/panel_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:visibility="gone"/>
```

6. **Create a beauty filter object and add the beauty filter panel.**

```
private void initXMagic() {
    if (mXMagic == null) {
        mXMagic = new XMagicImpl(mActivity, getBeautyPanel());
    } else {
        mXMagic.onResume();
    }
}
```

For detailed directions, see the `UGCKitVideoRecord` class of the demo.

Migrating from UGSV Enterprise

最終更新日：：2022-08-02 15:05:19

We have deprecated UGSV Enterprise. The beauty filter module in UGSV Enterprise is now offered as an independent SDK: Tencent Effect SDK. The SDK features more natural effects and more powerful beautification features. It also offers greater flexibility in terms of integration. This document shows you how to migrate from UGSV Enterprise to Tencent Effect SDK.

Notes

1. Modify the version number of the `glide` library in the `Xmagic` module to make it the same as the actual version number.
2. Modify the earliest version number in the `Xmagic` module to make it the same as the actual version number.

Directions

Step 1. Replace resources

1. Download the UGSV demo which has integrated the Tencent Effect SDK. This demo is built based on the Tencent Effect SDK S1-04 edition.
2. Replace the SDK files in the demo with the files for the SDK you actually use. Specifically, follow the steps below:
   - Replace the `.aar` file in the `libs` directory of the `Xmagic` module with the `.aar` file in `libs` of your SDK.
   - Replace all the files in `../src/main/assets` of the `Xmagic` module with those in `assets/` of your SDK. If there are files in the `MotionRes` folder of your SDK package, also copy them to the `../src/main/assets` directory.
   - Replace all the `.so` files in `../src/main/jniLibs` of the `Xmagic` module with the `.so` files in `jniLibs` of your SDK package (you need to decompress the ZIP files in the `jinLibs` folder to get the `.so` files for arm64-v8a and armeabi-v7a).
3. Import the `Xmagic` module in the demo into your project.

Step 2. Upgrade the SDK edition

Upgrade the SDK from UGSV Enterprise to UGSV Professional.

- **Before replacement**:
  ```
  implementation 'com.tencent.liteav:LiteAVSDK_Enterprise:latest.release'
  ```
- **After replacement**:
  ```
  implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release'
  ```

Step 3. Configure the license

1. Call the following APIs in `oncreate` of `application` in your project:

```
XMagicImpl.init(this);
XMagicImpl.checkAuth(null);
```

2. In the `XMagicImpl` class, replace the values of **license URL and key** to the ones you obtain from Tencent Cloud.

Step 4. Implement the code

The following example shows you how to implement the short video shooting view (`TCVideoRecordActivity.java`).

1. Add the following variables to the `TCVideoRecordActivity.java` class:

```
private XMagicImpl mXMagic;
private int isPause = 0;// 0: not paused; 1: paused; 2: pausing; 3:
to be terminated
```

2. Add the following code after `onCreate` in the `TCVideoRecordActivity.java` class:

```
TXUGCRecord instance = TXUGCRecord.getInstance(this);
instance.setVideoProcessListener(new
TXUGCRecord.VideoCustomProcessListener() {
  @Override
  public int onTextureCustomProcess(int textureId, int width, int
height) {
          if (isPause == 0 && mXMagic != null) {
                  return mXMagic.process(textureId, width, height);
          }
          return 0;
  }

  @Override
  public void onDetectFacePoints(float[] floats) {
  }

  @Override
  public void onTextureDestroyed() {
```

```
            if (Looper.getMainLooper() != Looper.myLooper()) {  // Not
the main thread
                    if (isPause == 1) {
                            isPause = 2;
                            if (mXMagic != null) {
                                    mXMagic.onDestroy();
                            }
                            initXMagic();
                            isPause = 0;
                    } else if (isPause == 3) {
                            if (mXMagic != null) {
                                    mXMagic.onDestroy();
                            }
                    }
            }
    }
});
XMagicImpl.checkAuth((errorCode, msg) -> {
    if (errorCode == TELicenseCheck.ERROR_OK) {
            loadXmagicRes();
    } else {
            TXCLog.e("TAG", "Authentication failed. Check the
authentication URL and key" + errorCode + " " + msg);
    }
});
```

3. Add the following code to `onStop`:

```
isPause = 1;
if (mXMagic != null) {
    mXMagic.onPause();
}
```

4. Add the following code to `onDestroy`:

```
isPause = 3;
XmagicPanelDataManager.getInstance().clearData();
```

5. Add the following code at the beginning of `onActivityResult`:

```
if (mXMagic != null) {
  mXMagic.onActivityResult(requestCode, resultCode, data);
}
```

6. Add the following two methods to the end of this class:

```
private void loadXmagicRes() {
     if (XMagicImpl.isLoadedRes) {
             XmagicResParser.parseRes(getApplicationContext());
             initXMagic();
             return;
     }
     new Thread(() -> {
             XmagicResParser.setResPath(new File(getFilesDir(),
"xmagic").getAbsolutePath());
             XmagicResParser.copyRes(getApplicationContext());
             XmagicResParser.parseRes(getApplicationContext());
             XMagicImpl.isLoadedRes = true;
             new Handler(Looper.getMainLooper()).post(() -> {
                     initXMagic();
             });
     }).start();

}
/**
* Initialize the beauty filter SDK
*/
private void initXMagic() {
     if (mXMagic == null) {
             mXMagic = new XMagicImpl(this,
mUGCKitVideoRecord.getBeautyPanel());
     }else{
             mXMagic.onResume();
     }
}
```

Step 5. Modify other classes

1. Change the type of `mBeautyPanel` in the `AbsVideoRecordUI` class to `RelativeLayout` and

the response type of `getBeautyPanel()` to `RelativeLayout`. You also need to modify the corresponding XML file and comment out the code that reports errors.

2. Comment out the code that reports errors in the `UGCKitVideoRecord` class.
3. In the `ScrollFilterView` class, delete the `mBeautyPanel` variable and comment out the code that reports errors.

Step 6. Delete the `beautysettingkit` dependencies

In the `build.gradle` file of the `ugckit` module, delete the `beautysettingkit` dependencies, compile the project, and comment out the code that report errors.

Advanced Features and Special Effects
TikTok-like Special Effects
iOS

最終更新日：：2025-04-01 17:14:09

Special Effect Filter

You can add multiple special effect filters for your videos. Currently, 11 filters are supported, all of which allow you to set the start and end time for display in the video. If multiple filters are set at the same point in time, the SDK will display the last set one.
You can set a special effect as follows:

```
- (void) startEffect:(TXEffectType)type  startTime:(float)startTime;
- (void) stopEffect:(TXEffectType)type  endTime:(float)endTime;

// The special effect type (`type` parameter) is defined in the
`TXEffectType` constant:
typedef  NS_ENUM(NSInteger,TXEffectType)
{
    TXEffectType_ROCK_LIGHT,   // Dynamic light-wave
    TXEffectType_DARK_DRAEM,   // Dark dream
    TXEffectType_SOUL_OUT,     // Soul out
    TXEffectType_SCREEN_SPLIT,// Screen split
    TXEffectType_WIN_SHADOW,   // Window blinds
    TXEffectType_GHOST_SHADOW,// Ghost shadow
    TXEffectType_PHANTOM,      // Phantom
    TXEffectType_GHOST,        // Ghost
    TXEffectType_LIGHTNING,    // Lightening
    TXEffectType_MIRROR,       // Mirror
    TXEffectType_ILLUSION,     // Illusion
};

- (void) deleteLastEffect;
- (void) deleteAllEffect;
```

You can call `deleteLastEffect()` to delete the last set special effect filter.You can call `deleteAllEffect()` to delete all set special effect filters:

Demo:
Use the first special effect filter between the first and second seconds, use the second special effect filter between the third and fourth seconds, and delete the special effect filter set between the third and fourth seconds:

```
// Use the first special effect filter between the first and second
seconds
[_ugcEdit startEffect:TXEffectType_SOUL_OUT startTime:1.0];
[_ugcEdit stopEffect:TXEffectType_SOUL_OUT startTime:2.0)];
// Use the second special effect filter between the third and fourth
seconds
[_ugcEdit startEffect:TXEffectType_SPLIT_SCREEN startTime:3.0];
[_ugcEdit stopEffect:TXEffectType_SPLIT_SCREEN startTime:4.0];
// Delete the special effect filter set between the third and fourth
seconds
[_ugcEdit deleteLastEffect];
```

Slow/Fast Motions

You can change the playback speed of multiple video segments by setting slow/fast playback as follows:

```
- (void) setSpeedList:(NSArray *)speedList;

// The `TXSpeed` parameters are as follows:
@interface TXSpeed: NSObject
@property (nonatomic, assign) CGFloat               startTime;      //
Speed change start time in s
@property (nonatomic, assign) CGFloat               endTime;        //
Speed change end time in s
@property (nonatomic, assign) TXSpeedLevel          speedLevel;     //
Speed change level
@end

Currently, multiple speed change levels are supported, which are defined
in the `TXSpeedLevel` constant:
typedef NS_ENUM(NSInteger, TXSpeedLevel) {
    SPEED_LEVEL_SLOWEST,        // Ultra-slow 0.25x of the source video's
speed
    SPEED_LEVEL_SLOW,           // Slow 0.5x of the source video's speed
```

```
    SPEED_LEVEL_NOMAL,          // Normal 1x of the source video's speed
    SPEED_LEVEL_FAST,           // Fast 1.5x of the source video's speed
    SPEED_LEVEL_FASTEST,        // Ultra-fast 2x of the source video's
speed
};
```

Demo:

```
// The SDK supports speed change of multiple video segments. This demo
only shows slow playback of one video segment.
  TXSpeed *speed =[[TXSpeed alloc] init];
  speed.startTime = 1.0;
  speed.endTime = 3.0;
  speed.speedLevel = SPEED_LEVEL_SLOW;
  [_ugcEdit setSpeedList:@[speed]];
```

Reverse Playback

You can reverse a video as follows:

```
- (void) setReverse:(BOOL)isReverse;
```

Demo:

```
[_ugcEdit setReverse:YES];
```

Video Segment Loop

You can loop a video segment, but the audio will not be looped.Set the video segment for loop as follows:

```
- (void) setRepeatPlay:(NSArray *)repeatList;

// The `TXRepeat` parameters are as follows:
@interface TXRepeat: NSObject
@property (nonatomic, assign) CGFloat               startTime;      //
Loop start time in s
@property (nonatomic, assign) CGFloat               endTime;        //
Loop end time in s
```

```
@property (nonatomic, assign) int                    repeatTimes;    //
Number of repeats
@end
```

Demo:

```
TXRepeat *repeat = [[TXRepeat alloc] init];
repeat.startTime = 1.0;
repeat.endTime = 3.0;
repeat.repeatTimes = 3;  // Number of repeats
[_ugcEdit setRepeatPlay:@[repeat]];
```

Android

最終更新日：：2025-04-01 17:14:09

Special Effect Filter

You can add multiple special effect filters for your videos. Currently, 11 filters are supported, all of which allow you to set the start and end time for display in the video. If multiple filters are set at the same point in time, the SDK will display the last set one.
Set the special effect filter:

```
/**
  * Set the start time of the special effect filter
   * @param type   Special effect filter type
   * @param startTime   Start time of special effect filter in ms
   */
public void startEffect(int type, long startTime);
/**
  * Set the end time of the special effect filter
  * @param type   Special effect filter type
  * @param endTime   End time of special effect filter in ms
*/
public void stopEffect(int type, long endTime);
```

Parameter description: @param type: special effect filter type, which is defined in the `TXVideoEditConstants` constant:

```
public static final int TXEffectType_SOUL_OUT = 0;                    //
Soul out
public static final int TXEffectType_SPLIT_SCREEN = 1;                //
Screen split
public static final int TXEffectType_DARK_DRAEM = 2;                  //
Dark dream
public static final int TXEffectType_ROCK_LIGHT = 3;                  //
Dynamic light-wave
public static final int TXEffectType_WIN_SHADDOW = 4;                 //
Window blinds
public static final int TXEffectType_GHOST_SHADDOW = 5;               //
Ghost shadow
```

```
public static final int TXEffectType_PHANTOM_SHADDOW = 6;              //
Phantom
public static final int TXEffectType_GHOST = 7;                        //
Ghost
public static final int TXEffectType_LIGHTNING = 8;                    //
Lightening
public static final int TXEffectType_MIRROR = 9;                       //
Mirror
public static final int TXEffectType_ILLUSION = 10;                    //
Illusion
```

Delete the last set special effect filter:

```
public void deleteLastEffect();
```

Delete all set special effect filters:

```
public void deleteAllEffect();
```

Below is a complete sample:
Use the first special effect filter between the first and second seconds, use the second special effect filter between the third and fourth seconds, and delete the special effect filter set between the third and fourth seconds:

```
// Use the first special effect filter between the first and second
seconds
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT,
1000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT,
2000);
// Use the second special effect filter between the third and fourth
seconds
mTXVideoEditer.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREE
N, 3000);
mTXVideoEditer.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN
, 4000);
// Delete the special effect filter set between the third and fourth
seconds
mTXVideoEditer.deleteLastEffect();
```

Slow/Fast Motions

You can change the playback speed of multiple video segments by setting slow/fast playback as follows:

```
public void setSpeedList(List speedList);

// The `TXSpeed` parameters are as follows:
public final static class TXSpeed {
    public int speedLevel;                                  // Speed
change level
    public long startTime;                                  // Start
time
    public long endTime;                                    // End
time
}

// Currently, multiple speed change levels are supported, which are
defined in the `TXVideoEditConstants` constant:
public static final int SPEED_LEVEL_SLOWEST = 0;  // Ultra-slow  0.25x
of the source video's speed
public static final int SPEED_LEVEL_SLOW = 1;     // Slow        0.5x of
the source video's speed
public static final int SPEED_LEVEL_NORMAL = 2;   // Normal      1x of
the source video's speed
public static final int SPEED_LEVEL_FAST = 3;     // Fast        1.5x of
the source video's speed
public static final int SPEED_LEVEL_FASTEST = 4;  // Ultra-fast  2x of
the source video's
```

Below is a complete sample:

```
List<TXVideoEditConstants.TXSpeed> list = new ArrayList<>();
TXVideoEditConstants.TXSpeed speed1 = new
TXVideoEditConstants.TXSpeed();
speed1.startTime = 0;
speed1.endTime = 1000;
speed1.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW;
// Slow
```

```
list.add(speed1);

TXVideoEditConstants.TXSpeed speed2 = new
TXVideoEditConstants.TXSpeed();
speed2.startTime = 1000;
speed2.endTime = 2000;
speed2.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOWEST;
// Ultra-slow
list.add(speed2);

TXVideoEditConstants.TXSpeed speed3 = new
TXVideoEditConstants.TXSpeed();
speed3.startTime = 2000;
speed3.endTime = 3000;
speed3.speedLevel = TXVideoEditConstants.SPEED_LEVEL_SLOW;
// Slow
list.add(speed3);

mTXVideoEditer.setSpeedList(list);
```

Reverse Playback

You can reverse video playback. Specifically, you can call `setReverse(true)` / `setReverse(false)` to start/stop reverse playback.
Demo:

```
mTXVideoEditer.setReverse(true);
```

Video Segment Loop

You can loop a video segment, but the audio will not be looped. Currently, Android supports loop of only one video segment thrice.
You can call `setRepeatPlay(null)` to cancel the video segment loop set previously.
Set the video segment for loop as follows:

```
public void setRepeatPlay(List repeatList);

// The `TXRepeat` parameters are as follows:
public final static class TXRepeat {
```

```
    public long startTime;              // Loop start time in ms
    public long endTime;                // Loop end time in ms
    public int  repeatTimes;            // Number of repeats
}
```

Demo:

```
long currentPts = mVideoProgressController.getCurrentTimeMs();

List repeatList = new ArrayList<>();
TXVideoEditConstants.TXRepeat repeat = new
TXVideoEditConstants.TXRepeat();
repeat.startTime = currentPts;
repeat.endTime = currentPts + DEAULT_DURATION_MS;
repeat.repeatTimes = 3;  // Currently, a video segment can be repeated
only for thrice.
repeatList.add(repeat);  // Currently, only one video segment can be
looped.
mTXVideoEditer.setRepeatPlay(repeatList);
```

Note:
To ensure smooth reverse playback, it is recommended to preprocess the video upon import by calling the processVideo method.

Stickers and Subtitles
iOS

最終更新日：：2025-04-01 17:14:09

Static Sticker

```
- (void) setPasterList:(NSArray *)pasterList;

// The `TXPaster` parameters are as follows:
@interface TXPaster: NSObject
@property (nonatomic, strong) UIImage*              pasterImage;    //
Sticker image
@property (nonatomic, assign) CGRect                frame;          //
Sticker frame (please note that the frame coordinates here are relative
to the rendering view)
@property (nonatomic, assign) CGFloat               startTime;      //
Sticker start time in s
@property (nonatomic, assign) CGFloat               endTime;        //
Sticker end time in s
@end
```

Animated Sticker

```
- (void) setAnimatedPasterList:(NSArray *)animatedPasterList;

// The `TXAnimatedPaster` parameters are as follows:
@interface TXAnimatedPaster: NSObject
@property (nonatomic, strong) NSString*             animatedPasterpath;
// Animated image file path
@property (nonatomic, assign) CGRect                frame;          //
Animated image frame (please note that the frame coordinates here are
relative to the rendering view)
@property (nonatomic, assign) CGFloat               rotateAngle;    //
Animated image rotation angle. Value range: 0-360
@property (nonatomic, assign) CGFloat               startTime;      //
Animated image start time in s
```

```
@property (nonatomic, assign) CGFloat                endTime;         //
Animated image end time in s
@end
```

Demo:

```
- (void)setVideoPasters:(NSArray*)videoPasterInfos
{
    NSMutableArray* animatePasters = [NSMutableArray new];
    NSMutableArray* staticPasters = [NSMutableArray new];
    for (VideoPasterInfo* pasterInfo in videoPasterInfos) {
        if (pasterInfo.pasterInfoType == PasterInfoType_Animate) {
            TXAnimatedPaster* paster = [TXAnimatedPaster new];
            paster.startTime = pasterInfo.startTime;
            paster.endTime = pasterInfo.endTime;
            paster.frame = [pasterInfo.pasterView
pasterFrameOnView:_videoPreview];
            paster.rotateAngle = pasterInfo.pasterView.rotateAngle * 180
/ M_PI;
            paster.animatedPasterpath = pasterInfo.path;
            [animatePasters addObject:paster];
        }
        else if (pasterInfo.pasterInfoType == PasterInfoType_static){
            TXPaster *paster = [TXPaster new];
            paster.startTime = pasterInfo.startTime;
            paster.endTime = pasterInfo.endTime;
            paster.frame = [pasterInfo.pasterView
pasterFrameOnView:_videoPreview];
            paster.pasterImage = pasterInfo.pasterView.staticImage;
            [staticPasters addObject:paster];
        }
    }
    [_ugcEditer setAnimatedPasterList:animatePasters];
    [_ugcEditer setPasterList:staticPasters];
}
```

Adding Subtitles

Video subtitling is supported. You can add subtitles to each frame of a video and set the start and end time to display each subtitle. All subtitles form a subtitle list, which can be passed to the SDK, and the SDK will automatically add the subtitles to the video at the corresponding points in time.
You can set subtitles as follows:

```
- (void) setSubtitleList:(NSArray *)subtitleList;

The `TXSubtitle` parameters are as follows:
@interface TXSubtitle: NSObject
@property (nonatomic, strong) UIImage*              titleImage;     //
Subtitle image (here, you need to convert the text loading control to an
image)
@property (nonatomic, assign) CGRect                frame;          //
Subtitle frame (please note that the frame coordinates here are relative
to the rendering view)
@property (nonatomic, assign) CGFloat               startTime;      //
Subtitle start time in s
@property (nonatomic, assign) CGFloat               endTime;        //
Subtitle end time in s
@end
```

- titleImage: subtitle image. If controls like `UILabel` are used by the upper layer, please convert the control to `UIImage` first. For detailed directions, please see the sample code of the demo.
- frame: subtitle frame (please note that the frame is relative to the frame of the rendering view passed in during `initWithPreview`). For more information, please see the sample code of the demo.
- startTime: subtitle start time.
- endTime: subtitle end time.

As the subtitle UI logic is complicated, a complete method is provided at the demo layer. We recommend you directly implement subtitling as instructed in the demo, which greatly reduces your integration costs.
Demo:

```
@interface VideoTextInfo : NSObject
@property (nonatomic, strong) VideoTextFiled* textField;
@property (nonatomic, assign) CGFloat startTime; //in seconds
@property (nonatomic, assign) CGFloat endTime;
@end

videoTextInfos = @[VideoTextInfo1, VideoTextInfo2 ...];
```

```
 for (VideoTextInfo* textInfo in videoTextInfos) {
        TXSubtitle* subtitle = [TXSubtitle new];
        subtitle.titleImage = textInfo.textField.textImage;  //UILabel
(UIView) -> UIImage
        subtitle.frame = [textInfo.textField
textFrameOnView:_videoPreview]; // Calculate the coordinates relative to
the rendering view
        subtitle.startTime = textInfo.startTime;  // Subtitle start time
        subtitle.endTime = textInfo.endTime;      // Subtitle end time
        [subtitles addObject:subtitle];           // Add the subtitle
list
   }

 [_ugcEditer setSubtitleList:subtitles];          // Set the subtitle
list
```

Android

最終更新日：：2025-04-01 17:14:09

Static Sticker

You can set a static sticker as follows:

```
public void setPasterList(List pasterList);

// The `TXPaster` parameters are as follows:
public final static class TXPaster {
    public Bitmap pasterImage;                                  // Sticker
image
    public TXRect frame;                                        // Sticker
frame (please note that the frame coordinates here are relative to the
rendering view)
    public long startTime;                                      // Sticker
start time in ms
    public long endTime;                                        // Sticker
end time in ms
}
```

Animated Sticker

You can set an animated sticker as follows:

```
public void setAnimatedPasterList(List animatedPasterList);

// The `TXAnimatedPaster` parameters are as follows:
public final static class TXAnimatedPaster {
    public String animatedPasterPathFolder;                   // Address
of animated sticker image
    public TXRect frame;                                       //
Animated sticker frame (please note that the frame coordinates here are
relative to the rendering view)
    public long startTime;                                     //
Animated sticker start time in ms
```

```
    public long endTime;                                        //
Animated sticker end time in ms
    public float rotation;
}
```

Demo:

```
List animatedPasterList = new ArrayList<>();
List pasterList = new ArrayList<>();
for (int i = 0; i < mTCLayerViewGroup.getChildCount(); i++) {
    PasterOperationView view = (PasterOperationView)
mTCLayerViewGroup.getOperationView(i);
    TXVideoEditConstants.TXRect rect = new
TXVideoEditConstants.TXRect();
    rect.x = view.getImageX();
    rect.y = view.getImageY();
    rect.width = view.getImageWidth();
    TXCLog.i(TAG, "addPasterListVideo, adjustPasterRect, paster x y = "
+ rect.x + "," + rect.y);

    int childType = view.getChildType();
    if (childType ==
PasterOperationView.TYPE_CHILD_VIEW_ANIMATED_PASTER) {
        TXVideoEditConstants.TXAnimatedPaster txAnimatedPaster = new
TXVideoEditConstants.TXAnimatedPaster();

        txAnimatedPaster.animatedPasterPathFolder =
mAnimatedPasterSDcardFolder + view.getPasterName() + File.separator;
        txAnimatedPaster.startTime = view.getStartTime();
        txAnimatedPaster.endTime = view.getEndTime();
        txAnimatedPaster.frame = rect;
        txAnimatedPaster.rotation = view.getImageRotate();

        animatedPasterList.add(txAnimatedPaster);
        TXCLog.i(TAG, "addPasterListVideo, txAnimatedPaster startTimeMs,
endTime is : " + txAnimatedPaster.startTime + ", " +
txAnimatedPaster.endTime);
    } else if (childType == PasterOperationView.TYPE_CHILD_VIEW_PASTER)
{
```

```
        TXVideoEditConstants.TXPaster txPaster = new
TXVideoEditConstants.TXPaster();

        txPaster.pasterImage = view.getRotateBitmap();
        txPaster.startTime = view.getStartTime();
        txPaster.endTime = view.getEndTime();
        txPaster.frame = rect;

        pasterList.add(txPaster);
        TXCLog.i(TAG, "addPasterListVideo, txPaster startTimeMs, endTime
is : " + txPaster.startTime + ", " + txPaster.endTime);
    }
}

mTXVideoEditer.setAnimatedPasterList(animatedPasterList);  // Set an
animated sticker
mTXVideoEditer.setPasterList(pasterList);                  // Set a
static sticker
```

Adding Subtitles

Bubble video subtitling is supported. You can add subtitles to each frame of a video and set the start and end time to display each subtitle. All subtitles form a subtitle list, which can be passed to the SDK, and the SDK will automatically add the subtitles to the video at the corresponding points in time.
You can set bubble subtitles as follows:

```
public void setSubtitleList(List subtitleList);

// The `TXSubtitle` parameters are as follows:
public final static class TXSubtitle {
        public Bitmap titleImage;                                  //
Subtitle image
        public TXRect frame;                                        //
Subtitle frame
        public long startTime;                                      //
Subtitle start time in ms
        public long endTime;                                        //
Subtitle end time in ms
}
```

```
public final static class TXRect {
        public float x;
        public float y;
        public float width;
}
```

- titleImage: subtitle image. If controls like `TextView` are used by the upper layer, please convert the control to `Bitmap` first. For detailed directions, please see the sample code of the demo.
- frame: subtitle frame (please note that the frame is relative to the frame of the rendering view passed in during `initWithPreview`). For more information, please see the sample code of the demo.
- startTime: subtitle start time.
- endTime: subtitle end time.

As the subtitle UI logic is complicated, a complete method is provided at the demo layer. We recommend you directly implement subtitling as instructed in the demo, which greatly reduces your integration costs. Demo:

```
mSubtitleList.clear();
for (int i = 0; i < mWordInfoList.size(); i++) {
    TCWordOperationView view = mOperationViewGroup.getOperationView(i);
    TXVideoEditConstants.TXSubtitle subTitle = new
TXVideoEditConstants.TXSubtitle();
    subTitle.titleImage = view.getRotateBitmap();  // Get `Bitmap`
    TXVideoEditConstants.TXRect rect = new
TXVideoEditConstants.TXRect();
    rect.x = view.getImageX();        // Get the X coordinate relative
to the parent view
    rect.y = view.getImageY();        // Get the Y coordinate relative
to the parent view
    rect.width = view.getImageWidth(); // Image width
    subTitle.frame = rect;
    subTitle.startTime = mWordInfoList.get(i).getStartTime();  // Set
the start time
    subTitle.endTime = mWordInfoList.get(i).getEndTime();      // Set
the end time
    mSubtitleList.add(subTitle);
}
mTXVideoEditer.setSubtitleList(mSubtitleList); // Set the subtitle list
```

Video Karaoke
iOS

最終更新日：：2020-09-01 14:58:16

This document describes how to implement basic duet features from scratch.

Process Overview

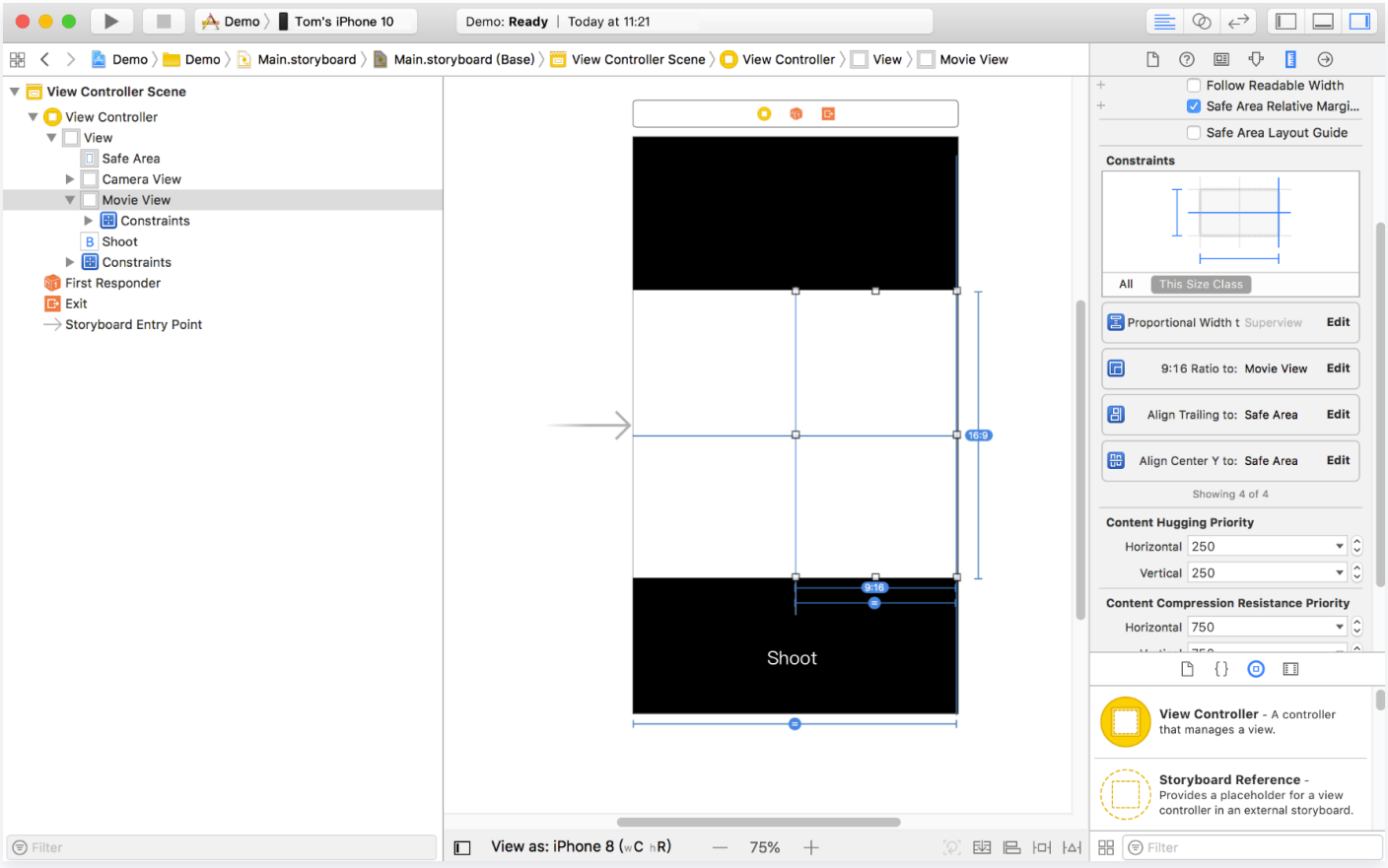
1. Place two views on the page, one for playback, and the other for shoot.
2. Place a button and progress bar for shoot and progress display, respectively.
3. Stop shoot after the video in the same duration as that of the source video has been shot.
4. Compose the shot video with the source video side by side.
5. Preview the composed video.

UI Construction

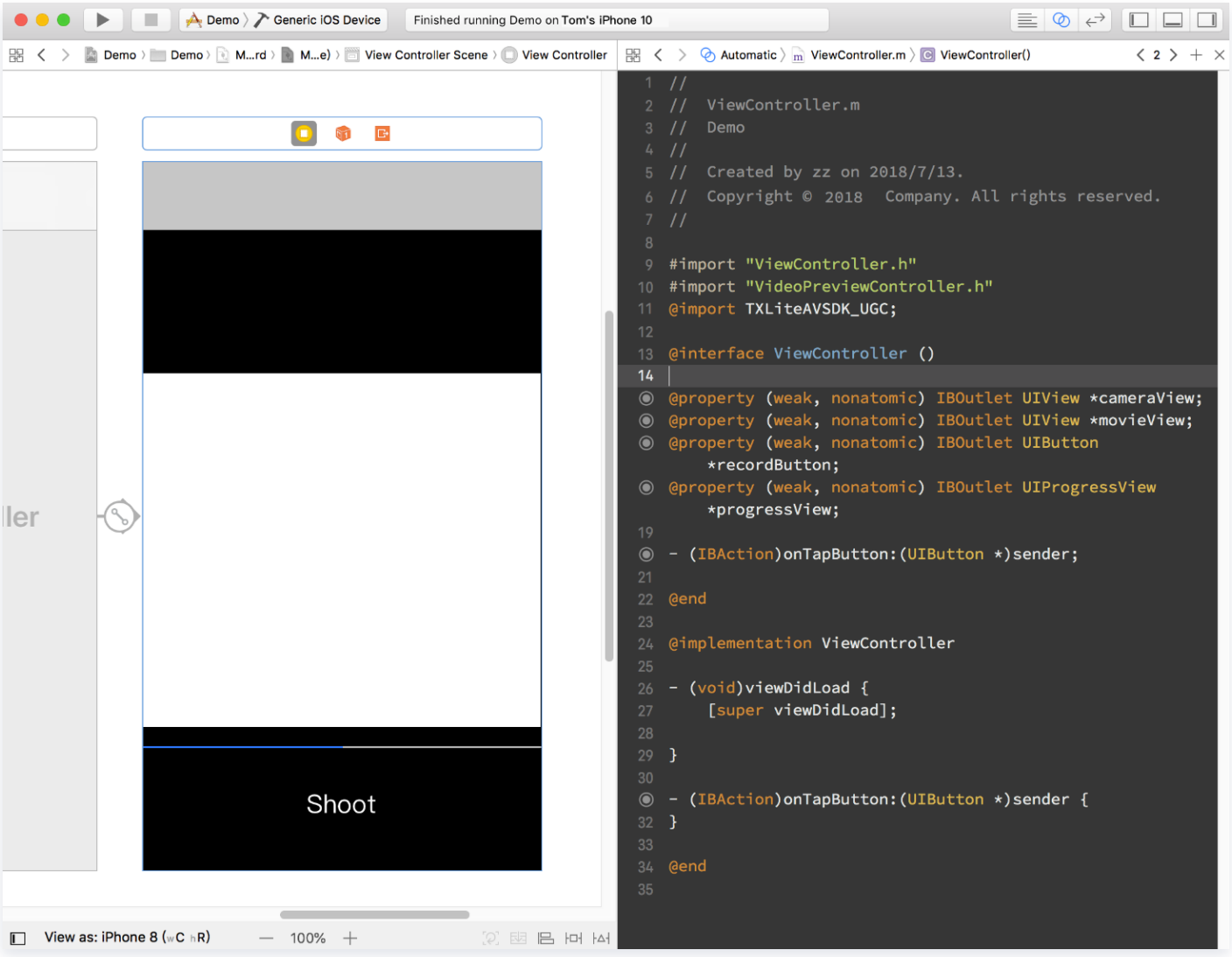
Create a project first. Open Xcode, select "File" > "New" > "Project", and name the project to create it. The project is named "Demo" in this example. To shoot a video, the camera and mic permissions are required. Add the following items to `Info`:

```
Privacy - Microphone Usage Description
Privacy - Camera Usage Description
```

You can enter desired values for the two items, such as "Shooting Video"
Configure a simple shoot page. Open `Main.storyboard`, drag two `UIView` objects into it, configure their width to 0.5 time of the `superview`, and set their aspect ratio to 16:9.



Add the progress bar, bind the page to `IBOutlet` in `ViewController.m`, and set the button `IBAction`. As the preview page needs to be redirected to after shoot, a navigation controller is required. Click the "VC" icon in yellow, select "Editor" > "Embed In" in the menu, and click "Navigation Controller" to add a layer of "Navigation Controller" onto the "ViewController". At this point, the basic UI has been constructed.



Sample Code

The duet feature mainly uses three other features: playback, shoot, and composition of the shot and source videos, which correspond to the `TXVideoEditer`, `TXUGCRecord`, and `TXVideoJoiner` SDK classes, respectively.

The SDK license needs to be configured before this feature can be used. Open `AppDelegate.m` and add the following code to it:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [TXUGCBase setLicenceURL:@"<License URL>" key:@"<License key>"];
    return YES;
}
```

Here, you need to apply for the license parameters in the UGSV Console. Once submitted, your application will be generally approved very soon, and the relevant information will be displayed on the page.

1. First, implement the declaration and initialization.

   Open `ViewContorller.m`, import the SDK, and declare the instances of the three classes above. As video playback, shoot, and composition are all async operations here, you need to listen on their events by adding the declaration for implementing the three protocols: `TXVideoJoinerListener`, `TXUGCRecordListener`, and `TXVideoPreviewListener`. After the declaration is added, the code will be as follows:

```
#import "ViewController.h"
@import TXLiteAVSDK_UGC;

@interface ViewController () <TXVideoJoinerListener,
TXUGCRecordListener, TXVideoPreviewListener>
{
    TXVideoEditer *_editor;
    TXUGCRecord   *_recorder;
    TXVideoJoiner *_joiner;

    TXVideoInfo    *_videoInfo;

    NSString       *_recordPath;
    NSString       *_resultPath;
}

@property (weak, nonatomic) IBOutlet UIView *cameraView;
@property (weak, nonatomic) IBOutlet UIView *movieView;
@property (weak, nonatomic) IBOutlet UIButton *recordButton;
@property (weak, nonatomic) IBOutlet UIProgressView *progressView;

- (IBAction)onTapButton:(UIButton *)sender;
@end
```

After preparing the member variables and API implementation declaration, initialize the member variables above in `viewDidLoad`.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Here, place a .mp4 video file or a .mov video shot on the phone
in the project
    NSString *mp4Path = [[NSBundle mainBundle] pathForResource:@"demo"
ofType:@"mp4"];
    _videoInfo = [TXVideoInfoReader getVideoInfo:mp4Path];
    TXAudioSampleRate audioSampleRate = AUDIO_SAMPLERATE_48000;
    if (_videoInfo.audioSampleRate == 8000) {
        audioSampleRate = AUDIO_SAMPLERATE_8000;
    }else if (_videoInfo.audioSampleRate == 16000){
        audioSampleRate = AUDIO_SAMPLERATE_16000;
    }else if (_videoInfo.audioSampleRate == 32000){
        audioSampleRate = AUDIO_SAMPLERATE_32000;
    }else if (_videoInfo.audioSampleRate == 44100){
        audioSampleRate = AUDIO_SAMPLERATE_44100;
    }else if (_videoInfo.audioSampleRate == 48000){
        audioSampleRate = AUDIO_SAMPLERATE_48000;
    }

    // Set the video storage path
    _recordPath = [NSTemporaryDirectory()
stringByAppendingPathComponent:@"record.mp4"];
    _resultPath = [NSTemporaryDirectory()
stringByAppendingPathComponent:@"result.mp4"];

   // Initialize the player
    TXPreviewParam *param = [[TXPreviewParam alloc] init];
    param.videoView = self.movieView;
    param.renderMode = RENDER_MODE_FILL_EDGE;
    _editor = [[TXVideoEditer alloc] initWithPreview:param];
    [_editor setVideoPath:mp4Path];
    _editor.previewDelegate = self;

    // Initialize the shoot parameters
    _recorder = [TXUGCRecord shareInstance];
```

```
    TXUGCCustomConfig *recordConfig = [[TXUGCCustomConfig alloc]
init];
    recordConfig.videoResolution = VIDEO_RESOLUTION_720_1280;
    // The frame rates of the shot video and source video must be the
same; otherwise, the audio and video may be out of sync
    // Note: the frame rate of the duet video obtained here is the
average frame rate and may be a decimal, which needs to be rounded
    recordConfig.videoFPS = (int)(_videoInfo.fps + 0.5);
    // The audio sample rates of the shot video and source video must
be the same; otherwise, the audio and video may be out of sync
    recordConfig.audioSampleRate = audioSampleRate;
    recordConfig.videoBitratePIN = 9600;
    recordConfig.maxDuration = _videoInfo.duration;
    _recorder.recordDelegate = self;

    // Enable camera preview
    [_recorder startCameraCustom:recordConfig
preview:self.cameraView];

    // Compose videos
    _joiner = [[TXVideoJoiner alloc] initWithPreview:nil];
    _joiner.joinerDelegate = self;
    [_joiner setVideoPathList:@[_recordPath, mp4Path]];
}
```

2. Next, implement the shoot feature. You only need to respond to the user click of the button to call the SDK method. For the sake of convenience, the button is reused here to display the current status, and the logic of displaying the progress is added to the progress bar.

```
- (IBAction)onTapButton:(UIButton *)sender {
    [_editor startPlayFromTime:0 toTime:_videoInfo.duration];
    if ([_recorder startRecord:_recordPath coverPath:[_recordPath
stringByAppendingString:@".png"]] != 0) {
        NSLog(@"Failed to start the camera");
    }
    [sender setTitle:@"Shooting" forState:UIControlStateNormal];
    sender.enabled = NO;
}
```

```
#pragma mark TXVideoPreviewListener
-(void) onPreviewProgress:(CGFloat)time
{
    self.progressView.progress = time / _videoInfo.duration;
}
```

3. After shoot, implement the composition. You need to specify the positions of the two videos in the output video. Here, the left and right positions are set.

```
-(void)onRecordComplete:(TXUGCRecordResult*)result;
{
    NSLog(@"Shoot is completed and composition is started");
    [self.recordButton setTitle:@"Composing..."
forState:UIControlStateNormal];

    // Get the width and height of the shot video
    TXVideoInfo *videoInfo = [TXVideoInfoReader
getVideoInfo:_recordPath];
    CGFloat width = videoInfo.width;
    CGFloat height = videoInfo.height;

    // Place the shot and source videos on the left and right,
respectively
    CGRect recordScreen = CGRectMake(0, 0, width, height);
    CGRect playScreen = CGRectMake(width, 0, width, height);
    [_joiner setSplitScreenList:@[[NSValue
valueWithCGRect:recordScreen],[NSValue valueWithCGRect:playScreen]]
canvasWidth:width * 2 canvasHeight:height];
    [_joiner splitJoinVideo:VIDEO_COMPRESSED_720P
videoOutputPath:_resultPath];
}
```

4. Implement the delegation method of the composition progress to display the progress on the progress bar.

```
-(void) onJoinProgress:(float)progress
{
    NSLog(@"Composing videos %d%%",(int)(progress * 100));
    self.progressView.progress = progress;
```

```
}
```

5. Implement the delegation method of the composition completion and switch to the preview page.

```
#pragma mark TXVideoJoinerListener
-(void) onJoinComplete:(TXJoinerResult *)result
{
    NSLog(@"Video composition completed");
    VideoPreviewController *controller = [[VideoPreviewController
alloc] initWithVideoPath:_resultPath];
    [self.navigationController pushViewController:controller
animated:YES];
}
```

At this point, the implementation is completed. The code of the video preview `VideoPreviewController` mentioned above is as follows:

- `VideoPreviewController.h`

```
#import <UIKit/UIKit.h>

@interface VideoPreviewController : UIViewController
- (instancetype)initWithVideoPath:(NSString *)path;
@end
```

- `VideoPreviewController.m:`

```
@import TXLiteAVSDK_UGC;

@interface VideoPreviewController () <TXVideoPreviewListener>
{
    TXVideoEditer *_editor;
}
@property (strong, nonatomic) NSString *videoPath;
@end

@implementation VideoPreviewController

- (instancetype)initWithVideoPath:(NSString *)path {
```

```
    if (self = [super initWithNibName:nil bundle:nil]) {
        self.videoPath = path;
    }
    return self;
}

- (void)viewDidLoad {
    [super viewDidLoad];
    TXPreviewParam *param = [[TXPreviewParam alloc] init];
    param.videoView = self.view;
    param.renderMode = RENDER_MODE_FILL_EDGE;

    _editor = [[TXVideoEditer alloc] initWithPreview:param];
    _editor.previewDelegate = self;
    [_editor setVideoPath:self.videoPath];
    [_editor startPlayFromTime:0 toTime:[TXVideoInfoReader
getVideoInfo:self.videoPath].duration];
}

-(void) onPreviewFinished
{
    [_editor startPlayFromTime:0 toTime:[TXVideoInfoReader
getVideoInfo:self.videoPath].duration];
}
@end
```

At this point, all basic duet features have been implemented. For the demo with more features, please see Source Code of Full-Featured UGSV Application Demo.

Android

最終更新日：：2020-09-01 14:57:25

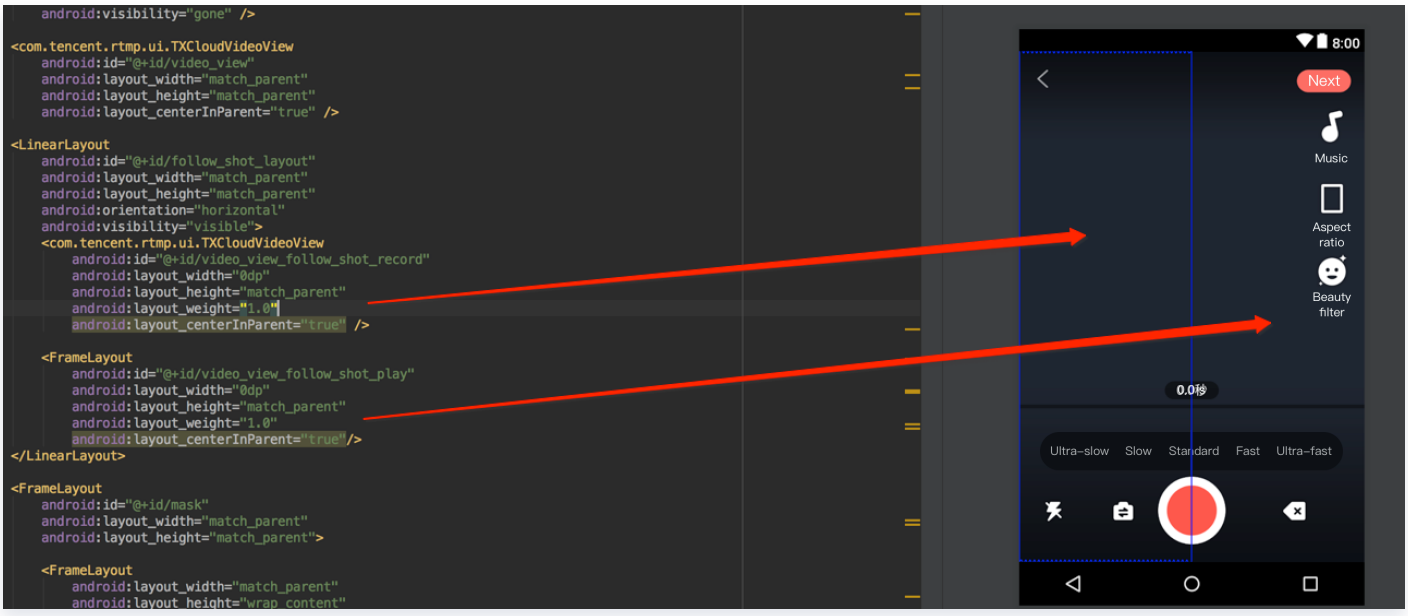
This document describes how to implement basic duet features.

Process Overview

1. Place two views on the page, one for playback, and the other for shoot.
2. Place a button and progress bar for shoot and progress display, respectively.
3. Stop shoot after the video in the same duration as that of the source video has been shot.
4. Compose the shot video with the source video side by side.
5. Preview the composed video.

UI Construction

In `activity_video_record.xml` of the shoot page `TCVideoRecordActivity`, create two views: the left one is the shoot page, and the right one is the playback page.



Sample Code

The duet feature mainly uses three other features: playback, shoot, and composition of the shot and source videos, which correspond to the `TXVideoEditer`, `TXUGCRecord`, and `TXVideoJoiner` SDK classes, respectively. You can also use `TXVodPlayer` for playback.

1. In the video list on the UGSV application homepage, select a video to enter the playback page `TCVodPlayerActivity`. Then, click "Duet" in the bottom-right corner.
   The video will be first downloaded onto the local SD card, the video information such as audio sample rate and frame rate (in fps) will be obtained, and then the shoot page will be displayed.
2. Enter the shoot page `TCVideoRecordActivity` for duet. Please pay attention to the following:
   - The maximum length of the shoot progress bar should be the length of the source video progress bar.
   - The frame rates of the shot video and source video must be the same; otherwise, the audio and video may be out of sync.
   - The audio sample rates of the shot video and source video must be the same; otherwise, the audio and video may be out of sync.
   - The rendering mode set for shoot is fit mode, where the video image can be proportionally scaled at the aspect ratio of 9:16.
   - You need to set audio mute for shoot on Android; otherwise, the source and shot videos' audios will be mixed.

```
// Shoot page
mVideoView = mVideoViewFollowShotRecord;
// Played back video
mFollowShotVideoPath =
intent.getStringExtra(TCConstants.VIDEO_EDITER_PATH);
mFollowShotVideoDuration = (int)
(intent.getFloatExtra(TCConstants.VIDEO_RECORD_DURATION, 0) * 1000);
initPlayer();
// The maximum length of the shoot progress bar should be the length
of the source video. The frame rate (`fps`) of the source video
should also be used for the shot video
mMaxDuration = (int)mFollowShotVideoDuration;
mFollowShotVideoFps =
intent.getIntExtra(TCConstants.RECORD_CONFIG_FPS, 20);
mFollowShotAudioSampleRateType =
intent.getIntExtra(TCConstants.VIDEO_RECORD_AUDIO_SAMPLE_RATE_TYPE,
TXRecordCommon.AUDIO_SAMPLERATE_48000);
// Initialize the duet API
mTXVideoJoiner = new TXVideoJoiner(this);
```

```
mTXVideoJoiner.setVideoJoinerListener(this);
```

```
// Initialize the player. Here, `TXVideoEditer` is used. You can also
use `TXVodPlayer`
mTXVideoEditer = new TXVideoEditer(this);
mTXVideoEditer.setVideoPath(mFollowShotVideoPath);
TXVideoEditConstants.TXPreviewParam param = new
TXVideoEditConstants.TXPreviewParam();
param.videoView = mVideoViewPlay;
param.renderMode =
TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoEditer.initWithPreview(param);
```

```
customConfig.videoFps = mFollowShotVideoFps;
customConfig.audioSampleRate = mFollowShotAudioSampleRateType; // The
audio sample rate of the shot video must be the same as that of the
source video
customConfig.needEdit = false;
mTXCameraRecord.setVideoRenderMode(TXRecordCommon.VIDEO_RENDER_MODE_AD
JUST_RESOLUTION); // Set the rendering mode to fit mode
mTXCameraRecord.setMute(true); // The duet audio played back by the
speaker will not be recorded, because if it is recorded by mic, the
source and shot videos' audios will be mixed
```

3. Start shoot. When the maximum shoot duration is reached, the `onRecordComplete` callback will be returned to proceed with the composition. Here, you need to specify the positions of the two videos in the result.

```
private void prepareToJoiner(){
 List<String> videoSourceList = new ArrayList<>();
 videoSourceList.add(mRecordVideoPath);
 videoSourceList.add(mFollowShotVideoPath);
 mTXVideoJoiner.setVideoPathList(videoSourceList);
 mFollowShotVideoOutputPath =
getCustomVideoOutputPath("Follow_Shot_");
 // Proportionally scale the video on the right by the width and
height of the shot video on the left
 int followVideoWidth;
```

```
 int followVideoHeight;
 if ((float) followVideoInfo.width / followVideoInfo.height >=
(float)recordVideoInfo.width / recordVideoInfo.height) {
     followVideoWidth = recordVideoInfo.width;
     followVideoHeight = (int) ((float)recordVideoInfo.width *
followVideoInfo.height / followVideoInfo.width);
 } else {
     followVideoWidth = (int) ((float)recordVideoInfo.height *
followVideoInfo.width / followVideoInfo.height);
     followVideoHeight = recordVideoInfo.height;
 }

 TXVideoEditConstants.TXAbsoluteRect rect1 = new
TXVideoEditConstants.TXAbsoluteRect();
 rect1.x = 0;                    // Top-left point position of the
first video
 rect1.y = 0;
 rect1.width = recordVideoInfo.width;   // Width and height of the
first video
 rect1.height = recordVideoInfo.height;

 TXVideoEditConstants.TXAbsoluteRect rect2 = new
TXVideoEditConstants.TXAbsoluteRect();
 rect2.x = rect1.x + rect1.width; // Top-left point position of the
second video
 rect2.y = (recordVideoInfo.height - followVideoHeight) / 2;
 rect2.width = followVideoWidth;  // Width and height of the second
video
 rect2.height = followVideoHeight;

 List<TXVideoEditConstants.TXAbsoluteRect> list = new ArrayList<>();
 list.add(rect1);
 list.add(rect2);
 mTXVideoJoiner.setSplitScreenList(list, recordVideoInfo.width +
followVideoWidth, recordVideoInfo.height); // The second and third
parameters: width and height of the video composition canvas
 mTXVideoJoiner.splitJoinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_5
40P, mFollowShotVideoOutputPath);
}
```

4. Listen on the composition callback. After `onJoinComplete`, redirect to the preview page for playback.

```
@Override
public void onJoinComplete(TXVideoEditConstants.TXJoinerResult result)
 {
 mCompleteProgressDialog.dismiss();
 if(result.retCode == TXVideoEditConstants.JOIN_RESULT_OK){
     runOnUiThread(new Runnable() {
         @Override
         public void run() {
             isReadyJoin = true;
             startEditerPreview(mFollowShotVideoOutputPath);
             if(mTXVideoEditer != null){
                 mTXVideoEditer.release();
                 mTXVideoEditer = null;
             }
         }
     });
 }else{
     runOnUiThread(new Runnable() {
         @Override
         public void run() {
             Toast.makeText(TCVideoRecordActivity.this, "Composition
failed", Toast.LENGTH_SHORT).show();
         }
     });
 }
}
```

At this point, all basic duet features have been implemented. For the complete code, please see Source Code of Full-Featured UGSV Application Demo.

Image Transition Special Effects
iOS

最終更新日：： 2025-04-01 17:14:09

The image editing feature is added starting from SDK 4.7. You can select a desired image to add effects such as transition animation, background music, and stickers.The API functions are as follows:

```
/*
 *pitureList: list of transition images, which must contain at least
three images (note: we recommend you compress the images to 720p or
lower (as shown in the demo); otherwise, the memory usage may be too
high, causing editing exceptions).
 *fps: frame rate of the video generated from the transition images in
fps. Value range: 15-30.
 * Returned values:
 *       0: set successfully
 *      -1: failed to set. Please check whether the image list exists,
the number of images is greater than or equal to 3, and the frame rate
(`fps`) is normal
 */
- (int)setPictureList:(NSArray<UIImage *> *)pitureList fps:(int)fps;

/*
 *transitionType: transition type. For more information, please see
`TXTransitionType`
 * Returned values:
 *       duration: transition video duration (note: the duration for the
same image list may vary by transition animation. You can get the
transition image duration here)
 */
- (void)setPictureTransition:(TXTransitionType)transitionType duration:
(void(^)(CGFloat))duration;


typedef NS_ENUM(NSInteger, TXTransitionType) {
    TXTransitionType_LefRightSlipping,
    TXTransitionType_UpDownSlipping,
    TXTransitionType_Enlarge,
```

```
    TXTransitionType_Narrow,
    TXTransitionType_RotationalScaling,
    TXTransitionType_FadeinFadeout,
};
```

- The `setPictureList` API is used to set the image list, which must contain at least three images. If too many images are set, the image size should be appropriate to avoid editing exceptions due to high memory usage.
- The `setPictureTransition` API is used to set the transition effect. Currently, six effects are available, and their durations may vary. You can get the transition duration through `duration` here.
- Pay attention to the API call sequence: call `setPictureList` first and then call `setPictureTransition` .
- Image editing currently does not support loop, reverse, fast/slow motions, and post-roll watermarking, but supports other video editing features. The call method is the same as that of video editing.

Android

最終更新日：： 2025-04-01 17:14:09

The image editing feature is added starting from SDK 4.9. You can select a desired image to add effects such as transition animation, background music, and stickers.The API functions are as follows:

```
/*
 * bitmapList: list of transition images, which must contain at least
three images (note: we recommend you compress the images to 720p or
lower (as shown in the demo); otherwise, the memory usage may be too
high, causing editing exceptions).
 * fps: frame rate of the video generated from the transition images in
fps. Value range: 15-30.
 * Returned values:
 *       0: set successfully
 *      -1: failed to set. Please check whether the image list exists
 */
public int setPictureList(List<Bitmap> bitmapList, int fps);

/*
 * type: transition type. For more information, please see
`TXVideoEditConstants`
 * Returned values:
 *       duration: transition video duration (note: the duration for the
same image list may vary by transition animation. You can get the
transition image duration here)
 */
public long setPictureTransition(int type)

/**
* Transition type for image to video conversion
*/
public static final int TX_TRANSITION_TYPE_LEFT_RIGHT_SLIPPING = 1;
public static final int TX_TRANSITION_TYPE_UP_DOWN_SLIPPING = 2;
public static final int TX_TRANSITION_TYPE_ROTATIONAL_SCALING = 3;
public static final int TX_TRANSITION_TYPE_ENLARGE = 4;
public static final int TX_TRANSITION_TYPE_NARROW = 5;
public static final int TX_TRANSITION_TYPE_FADEIN_FADEOUT = 6;
```

- Here, the `setPictureList` API is used to set the image list, which must contain at least three images. If too many images are set, the image size should be appropriate to avoid editing exceptions due to high memory usage.
- The `setPictureTransition` API is used to set the transition effect. Currently, six effects are available, and their durations may vary. You can get the transition duration through the returned value here.
- Pay attention to the API call sequence: call `setPictureList` first and then call `setPictureTransition`.
- Image editing currently does not support loop, reverse, and fast/slow motions, but supports other video editing features. The call method is the same as that of video editing.

Customizing Video Data
iOS

最終更新日：：2025-04-01 17:14:09

If you want to use third-party beauty libraries to add video effects during recording and editing, you can implement this in the preprocessing callbacks for both recording and editing.

Callback for Pre-Processing for Shooting

```
/**
 * Call back in the OpenGL thread, where captured images can be
processed.
 * @param texture    Texture ID
 * @param width      Texture width
 * @param height     Texture height
 * @return           Texture returned to the SDK
 * Note: The type of textures called back by the SDK is GL_TEXTURE_2D,
which must also be the texture type returned by the API. This callback
follows the calling of beauty filter APIs, and the texture format is
GL_RGBA.
 */
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width
height:(CGFloat)height;

/**
 * Call back in the OpenGL thread. You can release OpenGL resources
here.
 */
- (void)onTextureDestoryed;
```

Callback for Pre-Processing for Video Editing

```
/**
 Call back in the OpenGL thread, where captured images can be processed.
 @param textureId  Texture ID
 @param width      Texture width
```

```
 @param height     Texture height
 @param timestamp        Texture timestamp (ms)
 @return           Texture returned to the SDK
 Note: The type of textures called back by the SDK is GL_TEXTURE_2D,
which must also be the texture type returned by the API. This callback
follows the calling of beauty filter APIs, and the texture format is
GL_RGBA.
 Timestamp is the PTS of the current video frame and is measured in
milliseconds. You can customize beauty filter effects based on your
needs.
 */
- (GLuint)onPreProcessTexture:(GLuint)texture width:(CGFloat)width
height:(CGFloat)height timestamp:(UInt64)timestamp;

/**
 * Call back in the OpenGL thread. You can release OpenGL resources
here.
 */
- (void)onTextureDestoryed;
```

Android

最終更新日：：2025-04-01 17:14:09

If you want to use third-party beauty libraries to add video effects during recording and editing, you can implement this in the preprocessing callbacks for both recording and editing.

Shoot Preprocessing Callback

```
public interface VideoCustomProcessListener {
/**
  * Call back in the OpenGL thread. You can further process the captured
video image in the callback
  * @param textureId   Texture ID
  * @param width   Texture width
  * @param height   Texture height
  * @return   Texture ID returned to SDK. If no processing is required,
simply return the texture ID passed in.
  * Note: the texture type called back by the SDK is
`GLES20.GL_TEXTURE_2D`, which must also be the texture type returned to
the SDK by the API
  */
int onTextureCustomProcess(int textureId, int width, int height);


/**
  * Call back in the OpenGL thread. You can release the created OpenGL
resources in the callback
  */
void onTextureDestroyed();
}
```

Editing Preprocessing Callback

```
public interface TXVideoCustomProcessListener {
        /**
         * Call back in the OpenGL thread. You can further process the
captured video image in the callback
         *
```

```
         * @param textureId   Texture ID
         * @param width   Texture width
         * @param height   Texture height
         * @return   Texture ID returned to SDK. If no processing is
required, simply return the texture ID passed in.
         * <p>
         * Note: the texture type called back by the SDK is
`GLES20.GL_TEXTURE_2D`, which must also be the texture type returned to
the SDK by the API
         */
        int onTextureCustomProcess(int textureId, int width, int height,
long timestamp);

        /**
         * Call back in the OpenGL thread. You can release the created
OpenGL resources in the callback
         */
        void onTextureDestroyed();
    }
```

Video Porn Detection

最終更新日：：2020-09-01 14:56:48

In scenarios such as personal live streaming shoot and UGSV, the video content is unpredictable. To prevent non-compliant contents from being displayed on the VOD platform, you need to audit the uploaded videos first and transcode and distribute them after confirming that they are compliant. The Tencent Cloud UGSV solution supports AI-based porn detection in videos, which can automatically identify whether a video involves pornographic information.

Using AI-based Porn Detection

The AI-based porn detection feature can be used only after it is integrated into the video processing task flow. It depends on the video AI – content audit feature on the VOD backend, and the audit result will be sent to the application backend service as an event notification. VOD has a built-in task flow `QCVB_ProcessUGCFile` for UGSV porn detection scenarios. If you use this task flow and specify to perform AI-based porn detection, the porn detection task will be executed first, and whether to perform subsequent operations (such as transcoding, watermarking, and screencapturing) will be determined based on the porn detection result.

AI Template Overview

Template ID	Porn Processing	Sample Rate
10	End the task flow (operations such as transcoding, watermarking, and screencapturing will not be executed subsequently)	For videos whose duration is less than 500 seconds, sampling is performed once per second; for videos whose duration is greater than or equal to 500 seconds, sampling is performed once every 1% of the duration
20	Continue executing the task flow	None

AI-based Porn Detection Connection Sample

Step 1. Submit an AI-based porn detection task when generating an upload signature

```
/**
 * Generate a signature that contains an AI-based porn detection task
 */
function getUploadSignature(req, res) {
    res.json({
```

```
        code: 0,
        message: 'ok',
        data: {
            // Specify the template parameters and task flow during the
upload
            signature: gVodHelper.createFileUploadSignature({ procedure:
'QCVB_SimpleProcessFile({1,1,1,1})' })
        }
    });
}
```

Step 2. Get the AI-based porn detection result when getting the event notification

```
/**
 * Get the AI-based porn detection result of the event
 */
function getAiReviewResult(event){
    let data = event.eventContent.data;
    if(data.aiReview){
        return data.aiReview;
    }
    return {};
}
```

Custom Themes
iOS

最終更新日：：2022-11-15 11:20:02

`UGCKit` allows you to freely modify the preset text, colors, and icons.

Text

`UGCKit` offers two language packages by default: Simplified Chinese and American English. All their localized strings are stored in the default standard localized string format in `UGCKit/UGCKitResources/Localizable.strings`. You can replace the text to change the default strings or add new languages.
Below is an example of changing the Simplified Chinese animated effect names, which are in `UGCKit/UGCKitResources/zh-Hans.lproj/Localizable.strings`.

```
"UGCKit.Edit.VideoEffect.DynamicLightWave" = "Rock light";
"UGCKit.Edit.VideoEffect.DarkFantasy" = "Dark dream";
"UGCKit.Edit.VideoEffect.SoulOut" = "Soul out";
"UGCKit.Edit.VideoEffect.ScreenSplit" = "Split screen";
"UGCKit.Edit.VideoEffect.Shutter" = "Blinds";
"UGCKit.Edit.VideoEffect.GhostShadow" = "Shadow";
"UGCKit.Edit.VideoEffect.Phantom" = "Phantom";
"UGCKit.Edit.VideoEffect.Ghost" = "Ghost";
"UGCKit.Edit.VideoEffect.Lightning" = "Lightning";
"UGCKit.Edit.VideoEffect.Mirror" = "Mirror";
"UGCKit.Edit.VideoEffect.Illusion" = "Illusion";
```

To change "Illusion" to "Phantasm", you only need to modify the last line as follows:

```
"UGCKit.Edit.VideoEffect.Illusion" = "Phantasm";
```

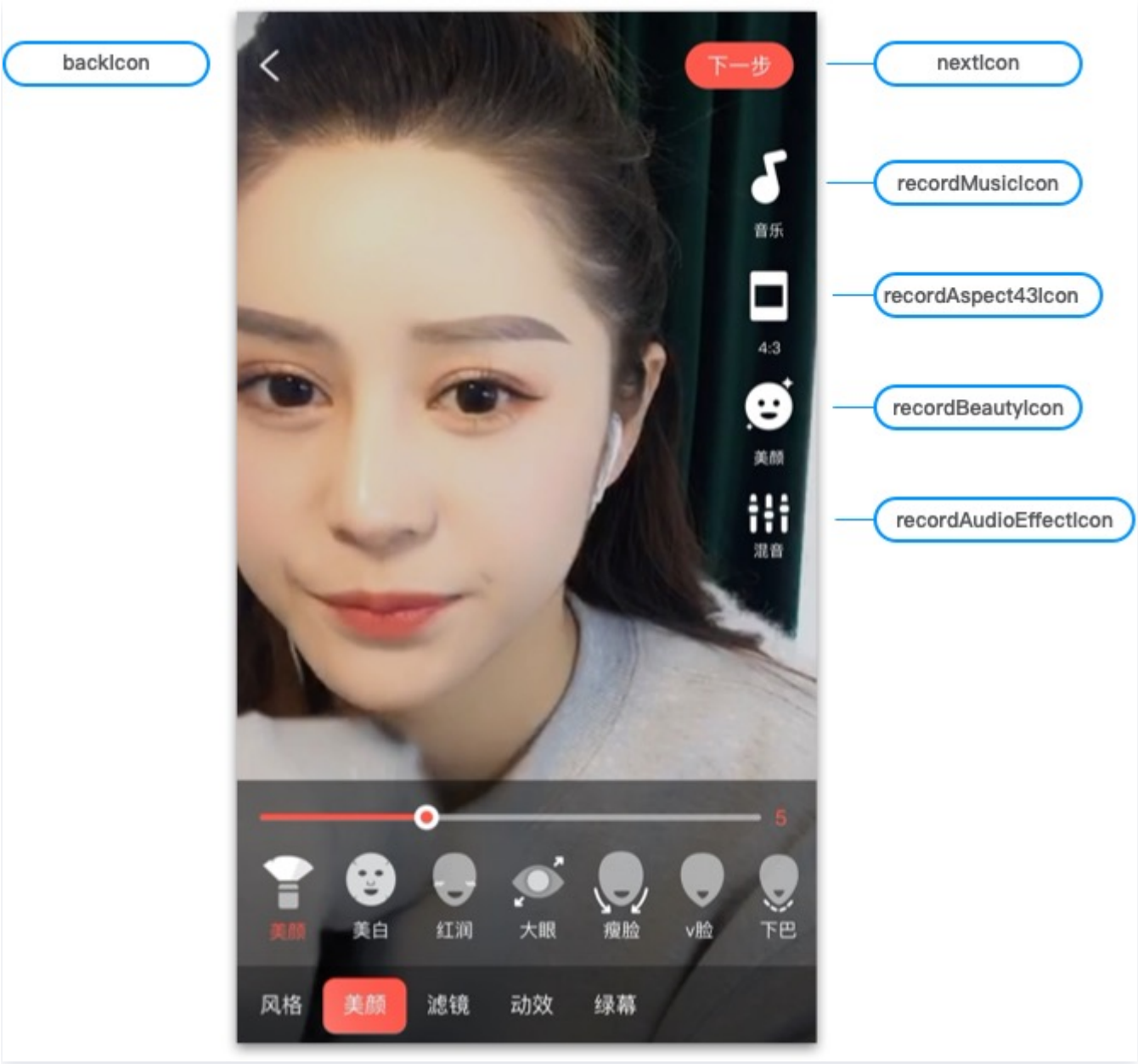
Color

The methods of getting the colors on all UIs of `UGCKit` are defined in the `UGCKitTheme` class. You can modify the attribute values to change colors. For the specific resource names, see the comments in `UGCKitTheme.h`.
Below is an example of changing the background color of the application UI:

```
UGCKitTheme *theme = [[UGCKitTheme alloc] init];
theme.backgroundColor = [UIColor whiteColor]; // Change the background
color to white
UGCKitEditViewController *editViewController = [[UKEditViewController
alloc] initWithMedia:media config:nil theme:theme]; // Use the custom
theme to create a controller
```

Icons

The icons on all UIs of `UGCKit` are in the `UGCKit.xcassets` resource file and can be replaced as needed. The specific icon filenames can be viewed in `UGCKitTheme.h`. An icon's resource name is the same as its attribute/method name, and all icons are defined in `UGCKitTheme.h`. Taking the recording UI as an example, the following names are the icons' attribute names in `UGCKitTheme` as well as the resource names in `UGCKit.xcassets`.



You can replace icons in `UGCKit` in two ways:

- Directly replace icons in `UGCKitTheme.xcassets`.
- Write code to assign values to objects in `UGCKitTheme` to change icons.

Below is the sample code for changing icons on the recording UI:

```
UGCKitTheme *theme = [[UGCKitTheme alloc] init];
theme.nextIcon = [UIImage imageName:@"myConfirmIcon"]; // Set the icon
used to complete recording
theme.recordMusicIcon = [UIImage imageName:@"myMusicIcon"]; // Set the
icon of the music feature
theme.beautyPanelWhitnessIcon = [UIImage imageNamed:@"beauty_whitness"];
// Set the icon of the brightening filter
```

```
UGCKitRecordViewController *viewController =
[[UGCKitRecordViewController alloc] initWithConfig:nil theme:theme]; //
Use the custom theme to create a controller
```

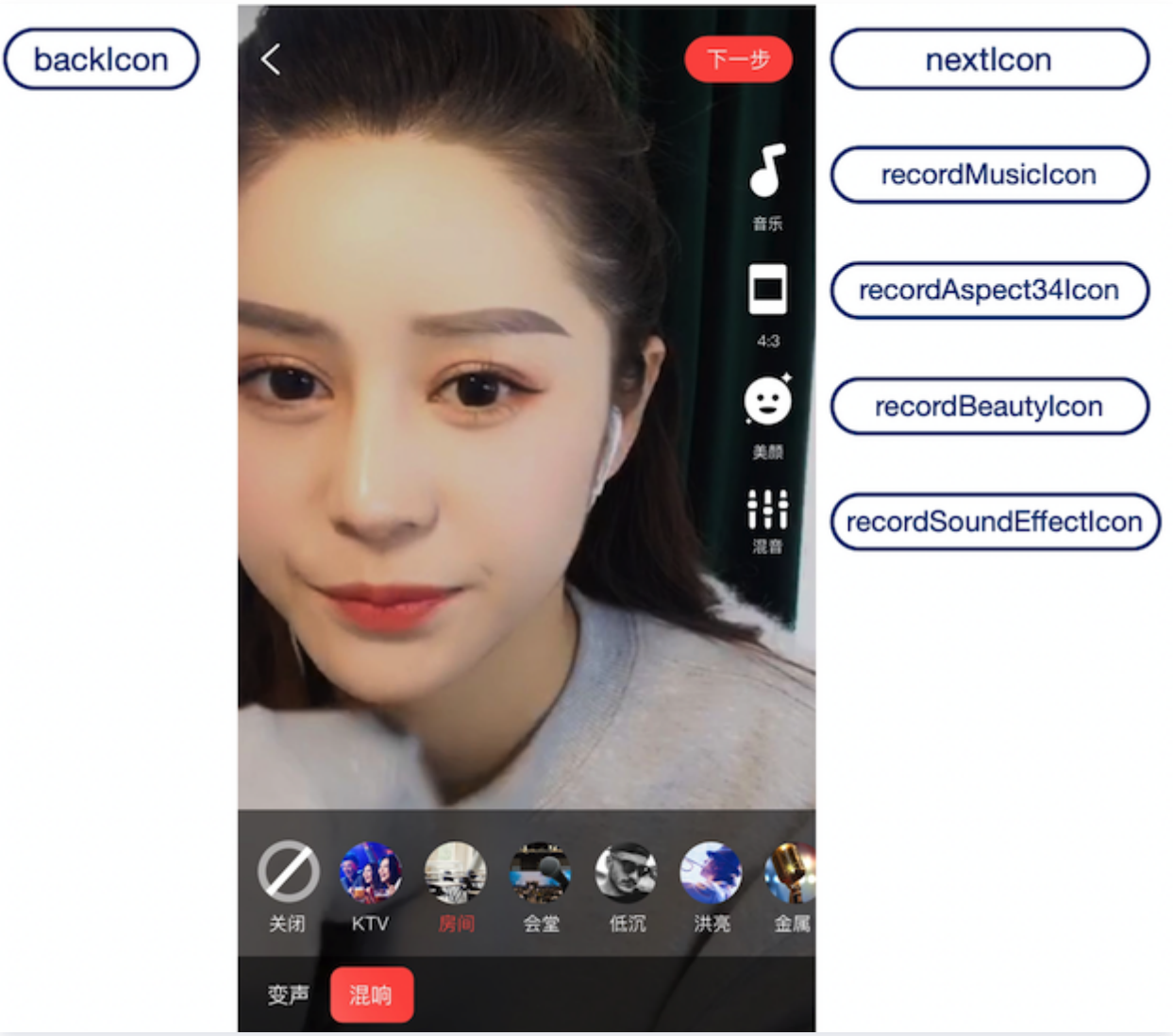
Android

最終更新日：：2022-11-14 18:25:34

`UGCKit` is a set of encapsulated interactive UIs, which is the UGSV demo app's theme by default. With simple modifications, you can customize your own theme and replace the icons, text, and colors.

Customizing the recording theme

The recording UI contains the right icon toolbar, the bottom toolbar, music panel, beauty filter panel, and sound effect panel.



1. Declare a `<style>` in `app/res/values/style.xml`, specify its parent theme as `RecordStyle`, and change the theme to the target theme. You can find all available themes in `app/ugckit/res/values.theme_style.xml`.

   Below is the sample code for replacing the music and beauty filter icons on the recording UI:

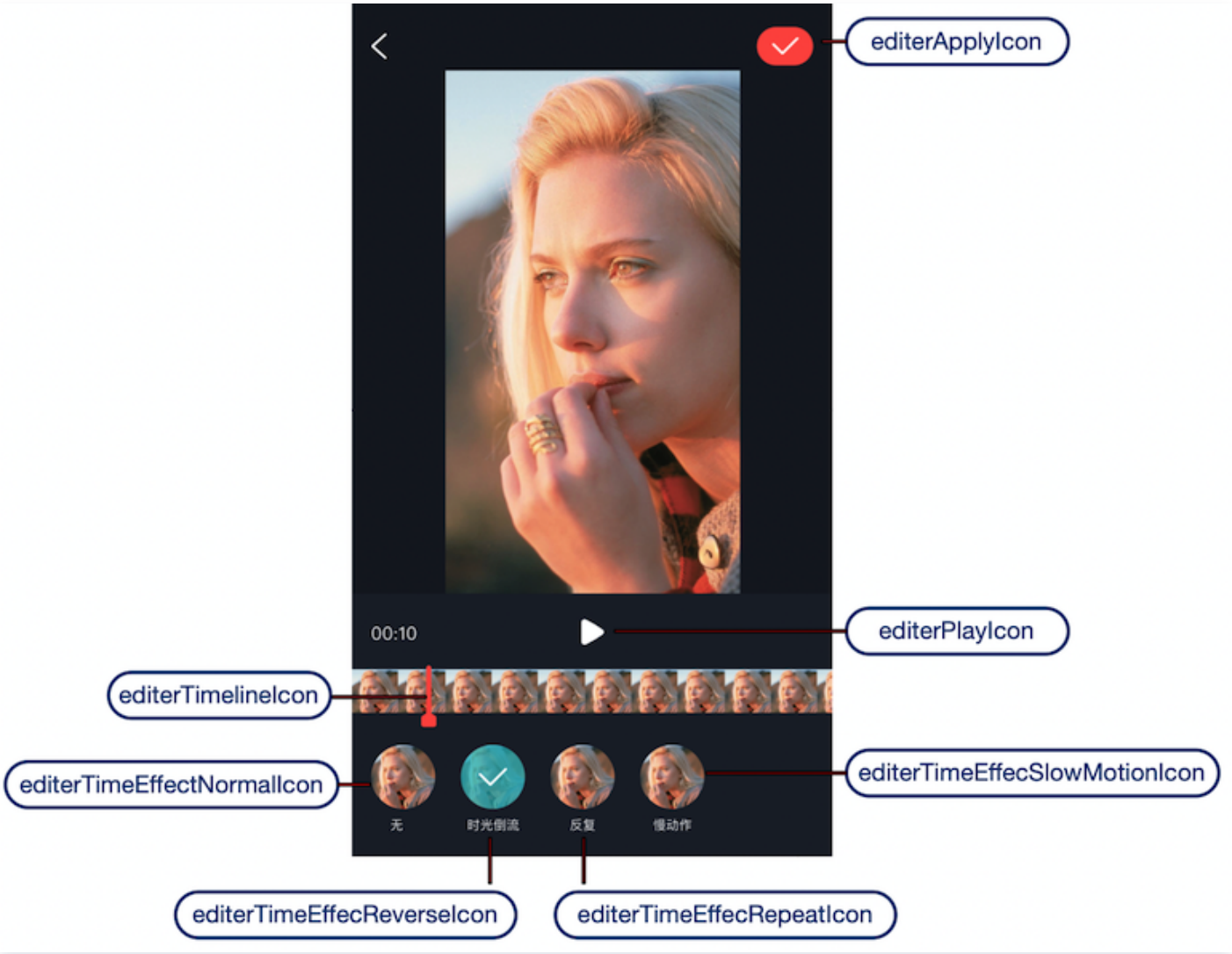
```
<style name="RecordActivityTheme" parent="RecordStyle">
 <item name="recordMusicIcon">@drawable/ic_music</item>
 <item name="recordBeautyIcon">@drawable/ic_beauty</item>
</style>
```

2. Declare the custom theme in `AndroidManifest.xml` :

```
<activity
 android:name="com.tencent.qcloud.xiaoshipin.videorecord.TCVideoRecor
dActivity"
 android:launchMode="singleTop"
 android:screenOrientation="portrait"
 android:theme="@style/RecordActivityTheme"
 android:windowSoftInputMode="adjustNothing" />
```

Customizing the editing theme

The editing UI contains the editing and clipping UI, animated effect, beauty filter, and special effect panel, and speed, filter, sticker, and bubble subtitles panels.



1. Declare a `<style>` in `app/res/values/style.xml`, specify its parent theme as `EditerStyle`, and change the theme to the target theme. You can find all available themes in `app/ugckit/res/values.theme_style.xml`.
2. Below is the sample code for replacing the playback and pause icons on the editing UI:

```
<style name="EditerActivityTheme" parent="EditerStyle">
 <item name="editorPlayIcon">@drawable/ic_play</item>
 <item name="editorPauseIcon">@drawable/ic_pause</item>
</style>
```

3. Declare the custom theme in `AndroidManifest.xml`:

```
<activity
 android:name=".videoeditor.TCVideoEffectActivity"
 android:screenOrientation="portrait"
 android:theme="@style/EditerActivityTheme" />
```