

# **User Generated Short Video SDK**

## **Integration (No UI)**

### **Product Documentation**



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

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

## Trademark Notice



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

## Service Statement

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

# Contents

## Integration (No UI)

### SDK Integration

#### SDK Integration (Xcode)

#### SDK Integration (Android Studio)

## Capturing and Shoot

### Capturing and Shoot

#### iOS

#### Android

### Multi-Segment Shoot

#### iOS

#### Android

### Shoot Drafts

#### iOS

#### Android

### Adding Background Music

#### iOS

#### Android

### Voice Changing and Reverb

#### iOS

#### Android

## Preview, Clipping, and Splicing

### Video Editing

#### iOS

#### Android

### Video Splicing

#### iOS

#### Android

## Upload and Playback

### Signature Distribution

### Video Upload

#### iOS

#### Android

### Player SDK

#### iOS

#### Android

## Tencent Effect SDK

### SDK Features

### SDK Integration Guide

#### iOS

#### Android

### Migrating from UGSV Enterprise

## Advanced Features and Special Effects

### TikTok-like Special Effects

#### iOS

#### Android

### Stickers and Subtitles

#### iOS

#### Android

### Video Karaoke

#### iOS

#### Android

### Image Transition Special Effects

#### iOS

#### Android

### Customizing Video Data

#### iOS

#### Android

### Video Porn Detection

### Custom Themes

#### iOS

#### Android



# Integration (No UI)

## SDK Integration

## SDK Integration (Xcode)

Last updated : 2022-11-14 18:18:58

## 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

Decompress the downloaded SDK resource package and copy the framework files whose filename starts with `TXLiteAVSDK\_` (such as `TXLiteAVSDK_UGC.framework`) in the SDK folder to the project folder and drag them to the project.

1. Integrate the libraries based on your SDK version:
  - For TXLiteAVSDK on v9.5 or earlier, add the integrated libraries
  - For TXLiteAVSDK 10.0 or later, add the system libraries
- i. Select the project target and add the following integrated libraries:
  - Accelerate.framework
  - SystemConfiguration.framework
  - libc++.tbd
  - libsqlite3.tbd

ii. Then, the project library dependency is as shown below:

▼ Link Binary With Libraries (6 items)		×
Name	Status	
 libc++.tbd	Required	⬇
 libsqlite3.tbd	Required	⬇
 libz.tbd	Required	⬇
 Accelerate.framework	Required	⬇
 TXLiteAVSDK_UGC.framework	Required	⬇
 SystemConfiguration.framework	Required	⬇
+ —		Drag to reorder frameworks

2. Select the project target, search for `bitcode` in **Build Settings** and set **Enable Bitcode** to **NO**.

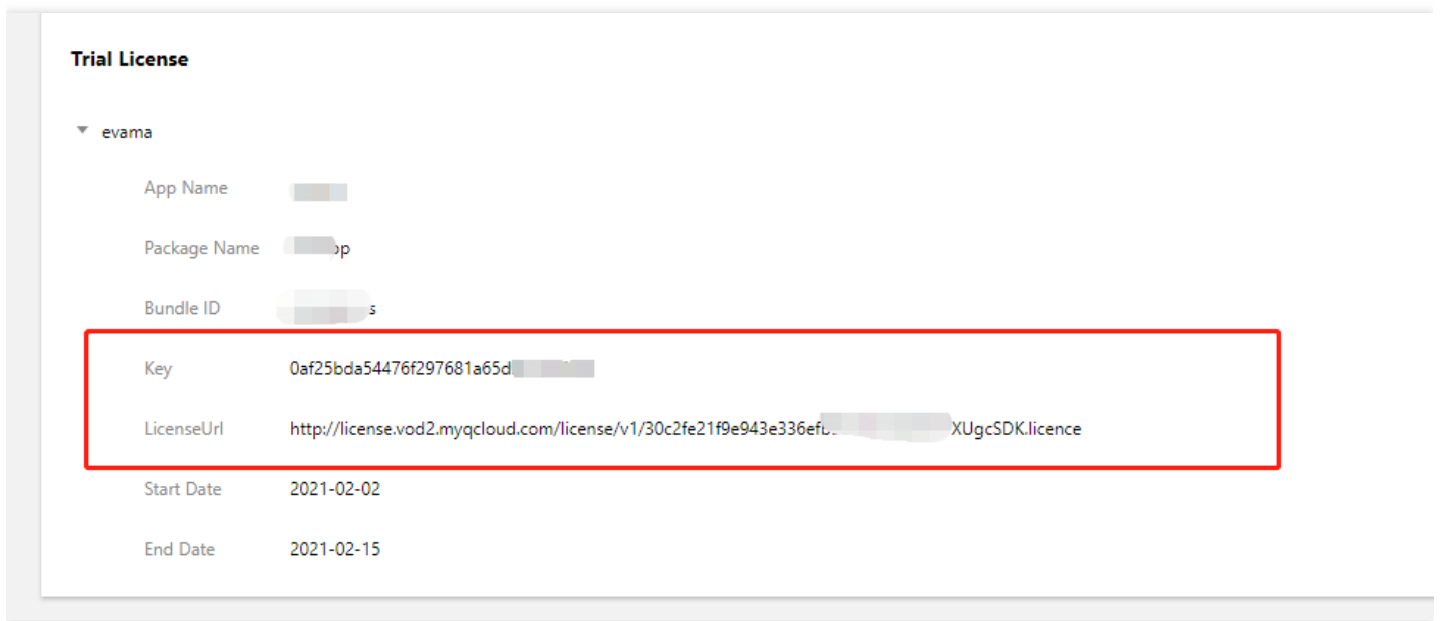
## Step 2. Configure app permissions

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

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*)application didFinishLaunchingWithOptions:(NSDictionary*)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
```

## Quick Integration of Feature Module

UGCKit is a UI component library built based on the UGSV SDK. It helps you quickly integrate different features of the SDK.

You can find UGCKit in `Demo/TXLiteAVDemo/UGC/UGCKit` of the SDK package, which you can download at [GitHub](#) or [here](#).

### UGCKit development environment requirements

- Xcode 10 or later
- iOS 9.0 or later

### Step 1. Integrate UGCKit

1. Use CocoaPods in your project and, based on your actual conditions, do either of the following:

- In the root directory of your project, run `pod init && pod install` to get the Podfile.
- Copy the **UGCKit** folder to the project root directory (the directory of the Podfile).

2. Open the Podfile and add the following:

```
pod 'UGCKit', :path => 'UGCKit/UGCKit.podspec', :subspecs => ["UGC"] #subspecs
Choose according to your SDK
```

3. To integrate basic beauty filters, copy the **BeautySettingKit** folder to the project root directory (the directory of the Podfile) and add the following to the Podfile:

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

4. To integrate the Tencent Effect SDK, copy the **xmagickit** folder to the project root directory (the directory of the Podfile) and add the following to the Podfile:

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

5. Run **pod install** and open `project name.xcworkspace`. You will find `UGCKit`, `BeautySettingKit`, and `magickit` in the `Pods/Development Pods` directory.

## Step 2. Use UGCKit

### 1. Shooting

`UGCKitRecordViewController` provides the video shooting feature. To enable the feature, just instantiate the controller and display it in the UI.

```
UGCKitRecordViewController *recordViewController = [[UGCKitRecordViewController
alloc] initWithConfig:nil theme:nil];
[self.navigationController pushViewController:recordViewController]
```

Shooting results are called back via the completion block, as shown below:

```
recordViewController.completion = ^(UGCKitResult *result) {
if (result.error) {
// Shooting error.
[self showAlertWithError:error];
} else {
if (result.cancelled) {
// User cancelled shooting and left the shooting view.
[self.navigationController popViewControllerAnimated:YES];
} else {
// Shooting successful. Use the result for subsequent processing.
```

```
[self processRecordedVideo:result.media];  
}  
}  
};
```

## 2. Editing

`UGCKitEditViewController` provides the slideshow making and video editing features. During instantiation, you need to pass in the media object to be edited. Below is an example that involves the editing of a shooting result:

```
- (void)processRecordedVideo:(UGCKitMedia *)media {  
    // Instantiate the editing view controller.  
    UGCKitEditViewController *editViewController = [[UKEditViewController alloc] in  
    initWithMedia:media config:nil theme:nil];  
    // Display the editing view controller.  
    [self.navigationController pushViewController:editViewController animated:YES];  
}
```

Editing results are called back via the completion block, as shown below:

```
editViewController.completion = ^(UGCKitResult *result) {  
    if (result.error) {  
        // Error.  
        [self showAlertWithError:error];  
    } else {  
        if (result.cancelled) {  
            // User cancelled shooting and left the editing view.  
            [self.navigationController popViewControllerAnimated:YES];  
        } else {  
            // Video edited and saved successfully. Use the result for subsequent processing.  
            [self processEditedVideo:result.path];  
        }  
    }  
}
```

## 3. Selecting video or image from photo album

- i. `UGCKitMediaPickerViewController` is used to select and splice media files. If multiple video files are selected, a spliced video file will be returned as follows:

```
// Configure initialization.  
UGCKitMediaPickerConfig *config = [[UGCKitMediaPickerConfig alloc] init];  
config.mediaType = UGCKitMediaTypeVideo; // Select videos.  
config.maxItemCount = 5; // Up to five can be selected.
```

```
// Instantiate the media picker view controller.
UGCKitMediaPickerViewController *mediaPickerViewController = [[UGCKitMediaPickerViewController alloc] initWithConfig:config theme:nil];
// Display the media picker view controller.
[self presentViewController:mediaPickerViewController animated:YES completion:nil];
```

ii. The selection result will be called back through the completion block. The following is a sample result:

```
mediaPickerViewController.completion = ^(UGCKitResult *result) {
    if (result.error) {
        // Error.
        [self showAlertWithError:error];
    } else {
        if (result.cancelled) {
            // User cancelled shooting and left the picker view.
            [self dismissViewControllerAnimated:YES completion:nil];
        } else {
            // Video edited and saved successfully. Use the result for subsequent processing.
            [self processEditedVideo:result.media];
        }
    }
}
```

## 4. Clipping

`UGCKitCutViewController` provides the video clipping feature. As with the editing API, you need to pass in a media project when instantiating the controller and process clipping results in `completion`.

```
UGCKitMedia *media = [UGCKitMedia mediaWithVideoPath:@"<#video path#>"];
UGCKitCutViewController *cutViewController = [[UGCKitCutViewController alloc] initWithMedia:media theme:nil];
cutViewController.completion = ^(UGCKitResult *result) {
    if (!result.cancelled && !result.error) {
        [self editVideo:result.media];
    } else {
        [self.navigationController pushViewControllerAnimated:YES];
    }
}
[self.navigationController pushViewController: cutViewController]
```

## Module Description

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

- [Video shooting](#)
- [Video editing](#)
- [Video splicing](#)
- [Video uploading](#)
- [Playback](#)



# SDK Integration (Android Studio)

Last updated : 2022-11-15 10:17:24

## 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

- AAR
- JAR + SO
- Gradle

#### 1. Create a project

##### Select the form factors and minimum SDK

Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

##### ☒ Phone and Tablet

API 18: Android 4.3 (Jelly Bean)

By targeting **API 18 and later**, your app will run on approximately **95.9%** of devices. [Help me choose](#)

☐ Include Android Instant App support

#### 2. Configure the project

- Add the code that imports the .aar package into `build.gradle` in the `App` directory of the project:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    // Import the SDK AAR file. Replace `x.y.zzzz` in `LiteAVSDK_UGC_x.y.zzzz` with the latest version number.
    compile(name: 'LiteAVSDK_UGC_10.7.1136', ext: 'aar')
    ...
}
```

- In `build.gradle` in the project directory, add `flatDir` to specify the local repository:

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

- Specify the NDK-compatible architectures in `defaultConfig` in `build.gradle` in the `App` directory of the project:

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

- Click **Sync Now** to compile the project.

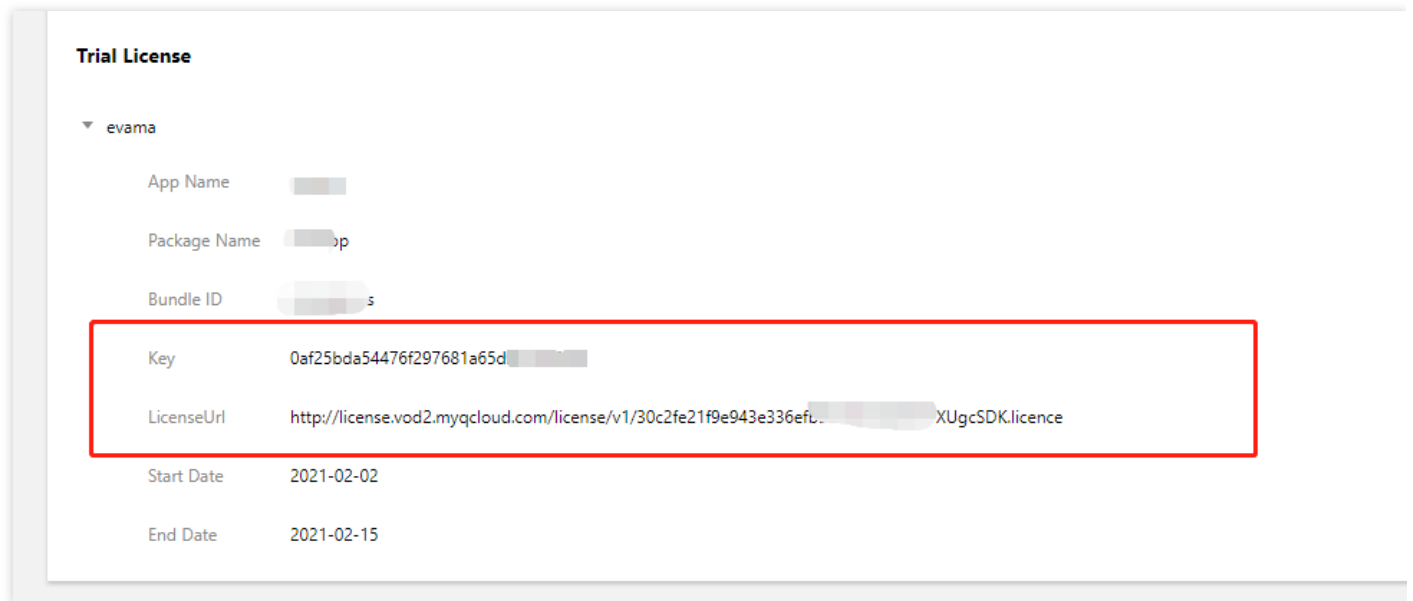
## Step 2. Configure app permissions

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:



**Trial License**

▼ evama

App Name [redacted]

Package Name [redacted]p

Bundle ID [redacted]s

Key 0af25bda54476f297681a65d [redacted]

LicenseUrl http://license.vod2.myqcloud.com/license/v1/30c2fe21f9e943e336efu...XUgcSDK.licence

Start Date 2021-02-02

End Date 2021-02-15

2. Before you use UGSV features in your application, we recommend that you complete the following configuration in

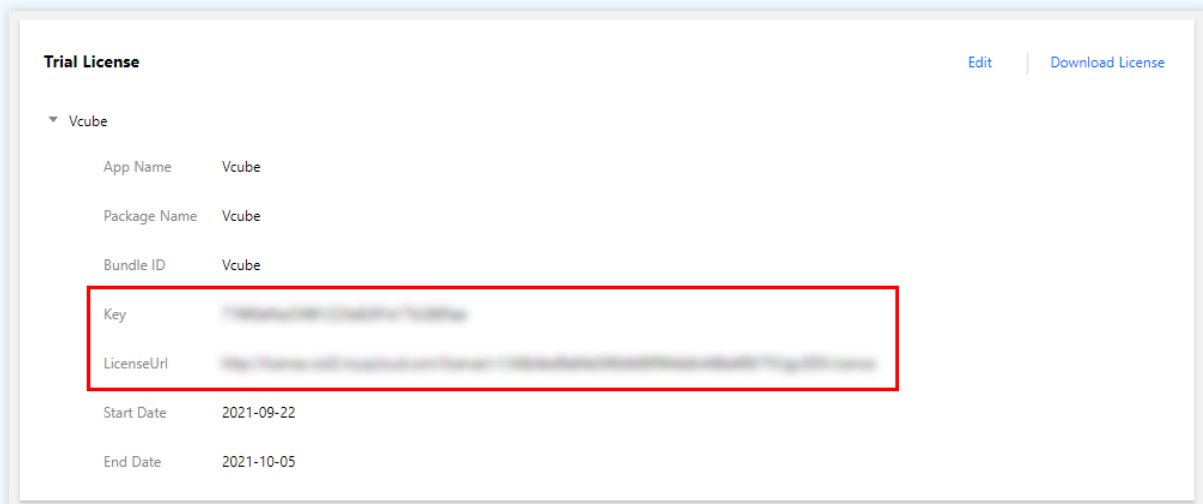
- Application onCreate() :

```
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);
    }
}
```

#### 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);
```

## Step 5. Build and run the project

Call an SDK API in your project to get the SDK version number and verify whether your project is correctly configured.

### 1. Import the SDK:

Import the SDK class in `MainActivity.java`:

```
import com.tencent.rtmp.TXLiveBase;
```

### 2. Call the API:

Call `getSDKVersion` in `onCreate` to get the version number:

```
String sdkver = TXLiveBase.getSDKVersionStr();  
Log.d("liteavsdk", "liteav sdk version is : " + sdkver);
```

### 3. Build and run the project:

If the above steps are performed correctly, you will build the project successfully and, after running it, you will see the following log information in `logcat`.

```
09-26 19:30:36.547 19577-19577/ D/liteavsdk: liteav sdk version is : 7.4.9211
```

## 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:

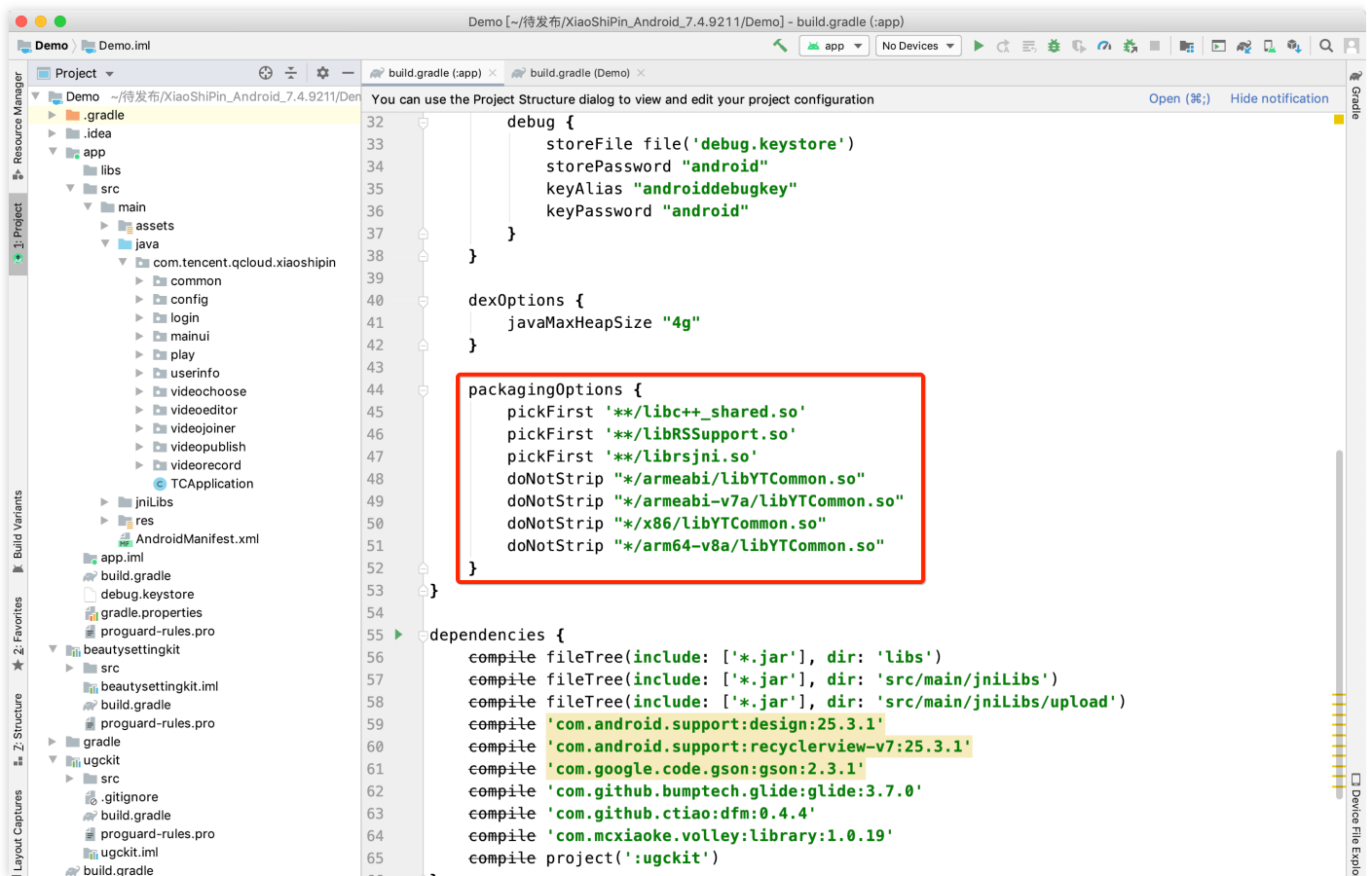
1. Check whether you have copied the JAR and SO files to the `jniLibs` directory.
2. If you use the full edition integrated with the .aar file, check whether the .so libraries for the x64 architecture are filtered out in `defaultConfig` in `build.gradle` in the project directory.

```
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.



## Integrating UGSV Modules

This section describes how to quickly integrate the UGSV SDK into your existing project to implement a complete range of short video features including shooting, editing, and composition. The code and resources mentioned in this section can be found in the SDK ZIP file as described in [SDK Download](#) and the [UGSV demo](#).

### Integrating UGCKit

#### 1. Create a project (empty activity)

- i. Create an empty Android Studio project. You can name it `ugc` and give it a custom package name. Make sure the project can be built and run successfully.
- ii. Configure `build.gradle` of the project.

```
// Top-level build file where you can add configuration options common to all
sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        # Copying starts.
        classpath 'com.android.tools.build:gradle:3.6.1'
        # Copying ends.
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        # Copying starts.
        flatDir {
            dirs 'src/main/jniLibs'
            dirs project(':ugckit').file('libs')
        }
        # Copying ends.
        jcenter() // Warning: this repository is going to shut down soon
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}

# Copying starts.
ext {
    compileSdkVersion = 29
    buildToolsVersion = "29.0.2"
    supportSdkVersion = "26.1.0"
    minSdkVersion = 21
    targetSdkVersion = 26
    versionCode = 1
    versionName = "v1.1"
    proguard = true
    rootPrj = "$projectDir/.."
    ndkAbi = 'armeabi-v7a'
    liteavSdk = "com.tencent.liteav:LiteAVSDK_UGC:latest.release"
}

# Copying ends.
```

iii. Configure `build.gradle` of your application.

```
plugins {  
    id 'com.android.application'  
}  
android {  
    # Copying starts.  
    compileSdkVersion = rootProject.ext.compileSdkVersion  
    buildToolsVersion = rootProject.ext.buildToolsVersion  
    # Copying ends.  
    defaultConfig {  
        applicationId "com.yunxiao.dev.liteavdemo"  
        # Copying starts.  
        minSdkVersion rootProject.ext.minSdkVersion  
        targetSdkVersion rootProject.ext.targetSdkVersion  
        versionCode rootProject.ext.versionCode  
        versionName rootProject.ext.versionName  
        renderscriptTargetApi = 19  
        renderscriptSupportModeEnabled = true  
        multiDexEnabled = true  
        ndk {  
            abiFilters rootProject.ext.ndkAbi  
        }  
        # Copying ends.  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
    }  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
        }  
    }  
    compileOptions {  
        sourceCompatibility JavaVersion.VERSION_1_8  
        targetCompatibility JavaVersion.VERSION_1_8  
    }  
    dependencies {  
        # Copying starts.  
        implementation fileTree(include: ['*.jar'], dir: 'libs')  
        implementation 'com.google.android.material:material:1.0.0'  
        implementation 'androidx.recyclerview:recyclerview:1.0.0'  
        implementation 'com.google.code.gson:gson:2.3.1'  
        implementation 'com.tencent.rqd:crashreport:3.4.4'
```



```
implementation 'com.tencent.rgd:nativecrashreport:3.9.2'
implementation 'com.github.castorflex.verticalviewpager:library:19.0.1'
implementation 'com.squareup.okhttp3:okhttp:3.11.0'
implementation 'de.hdodenhof:circleimageview:3.1.0'
implementation rootProject.ext.liteavSdk
implementation project(':ugckit')
implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
implementation('com.blankj:utilcode:1.25.9', {
    exclude group: 'com.google.code.gson', module: 'gson'
})
# Copying ends.
}
```

iv. Specify the Gradle version.

```
distributionUrl=https\://services.gradle.org/distributions/gradle-5.6.4-bin.zip
```

## 2. Import modules

- i. Copy the `ugckit` module to the `ugc` directory of your newly created project.
- ii. To integrate basic beauty filters, copy the `beautysettingkit` module to the `ugc` directory of the project.
- iii. To integrate the Tencent Effect SDK, copy the `xmagickit` module to the `ugc` directory of the project. For more information, see [SDK Integration Guide \(Android\)](#).
- iv. Import `ugckit` to `settings.gradle` of the project.
- v. In `UGC/settings.gradle` of the project, import the modules below:

```
include ':ugckit'
include ':beautysettingkit'
include ':xmagickit'
```

- vi. Add `ugckit` as a dependency for the app modules of your project.

```
implementation project(':ugckit')
```

### 3. Apply for a license

You need to set the license first before using `UGCKit`.

## Enabling shooting, importing, clipping, and special effects

### 1. Set the license and initialize `UGCKit`

Set the license and initialize `UGCKit` as early as possible before using UGSV features.

```
// Configure the license
TXUGCBase.getInstance().setLicence(this, ugcLicenceUrl, ugcKey);
// Initialize `UGCKit`
UGCKit.init(this);
```

### 2. Implement video shooting

i. Create an XML file for shooting and add the code below:

```
<com.tencent.qcloud.ugckit.UGCKitVideoRecord
android:id="@+id/video_record_layout"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

ii. Create an empty theme for shooting in `res/values/styles.xml` and inherit the default shooting theme of `UGCKit`.

```
<style name="RecordActivityTheme" parent="UGCKitRecordStyle"/>
```

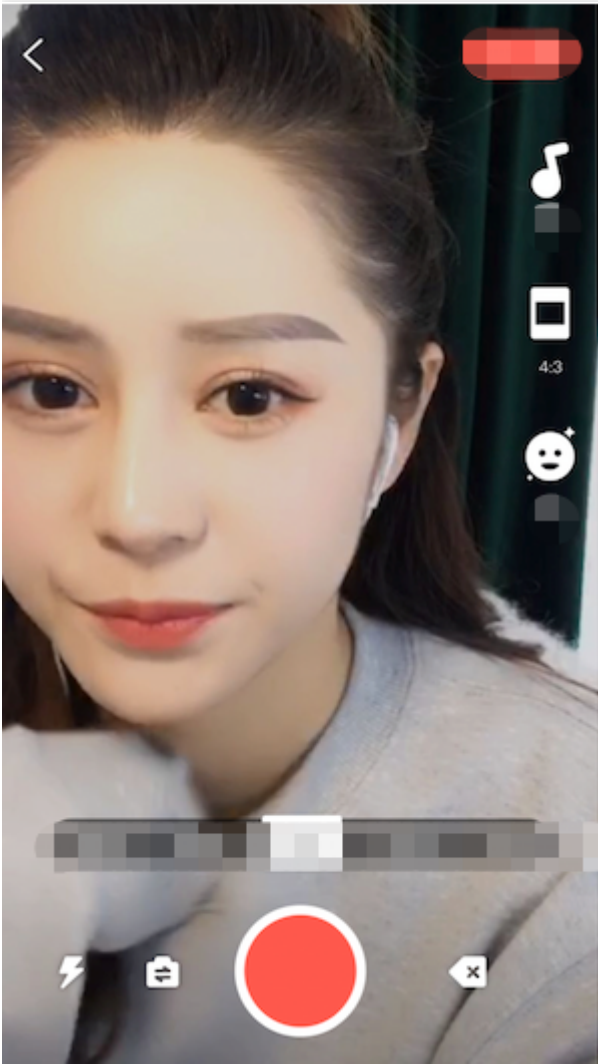
iii. Create an activity for shooting, inherit `FragmentActivity`, implement the

`ActivityCompat.OnRequestPermissionsResultCallback` API, get a `UGCKitVideoRecord` object, and set the callback.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // You must configure the theme in the code (`setTheme`) or in `AndroidManifest` (android:theme).
    setTheme(R.style.RecordActivityTheme);
    setContentView(R.layout.activity_video_record);
    // Get `UGCKitVideoRecord`
```

```
mUGCKitVideoRecord = (UGCKitVideoRecord) findViewById(R.id.video_record_layout);  
// Listen for shooting events  
mUGCKitVideoRecord.setOnRecordListener(new IVideoRecordKit.OnRecordListener()  
{  
    @Override  
    public void onRecordCanceled() {  
        // Shooting was canceled.  
    }  
    @Override  
    public void onRecordCompleted(UGCKitResult result) {  
        // Callback for ending shooting  
    }  
});  
@Override  
protected void onStart() {  
    super.onStart();  
    // Get whether the camera and audio recording permissions are granted. For details, see `Github/Demo`.  
    if (hasPermission()) {  
        // `UGCKit` takes over the shooting lifecycle. For details, see `Github/Demo`.  
        mUGCKitVideoRecord.start();  
    }  
    @Override  
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {  
        if (grantResults != null && grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
            mUGCKitVideoRecord.start();  
        }  
    }  
}
```

The UI view looks like this:



### 3. Implement video import

1. Create an XML file and add the code below:

```
<com.tencent.qcloud.ugckit.UGCKitVideoPicker
android:id="@+id/video_picker"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

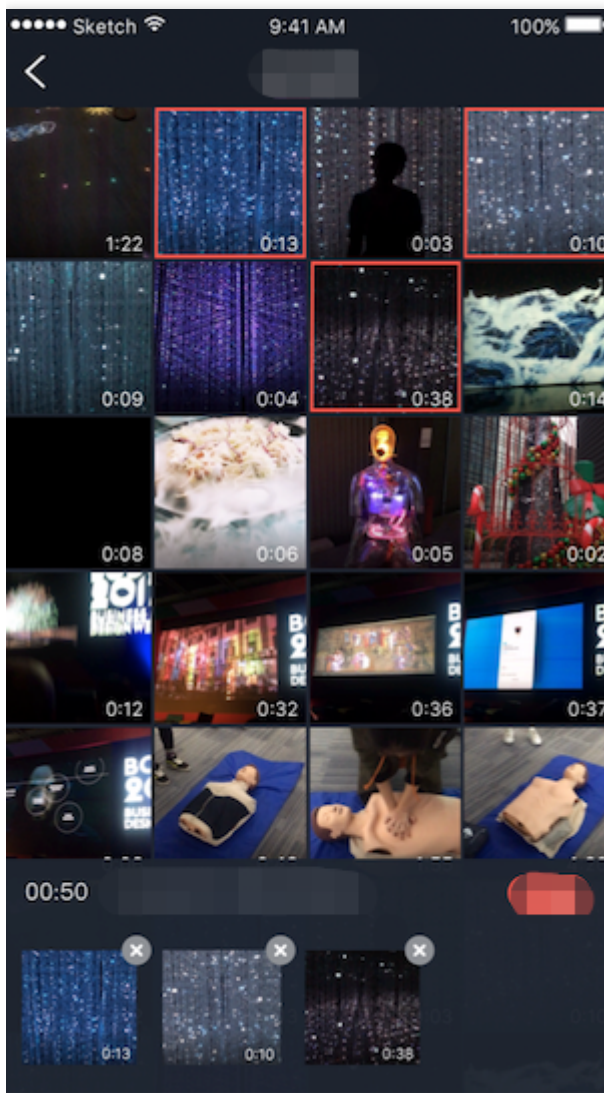
2. Create an empty theme in `res/values/styles.xml` and inherit the default video importing theme of `UGCKit`.

```
<style name="PickerActivityTheme" parent="UGCKitPickerStyle"/>
```

3. Create an activity, inherit `Activity`, get a `UGCKitVideoPicker` object, and set the callback.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // You must configure the theme in the code (`setTheme`) or in `AndroidManifest`
    (android:theme).
    setTheme(R.style.PickerActivityTheme);
    setContentView(R.layout.activity_video_picker);
    // Get `UGCKitVideoPicker`
    mUGCKitVideoPicker = (UGCKitVideoPicker) findViewById(R.id.video_picker);
    // Listen for video importing events
    mUGCKitVideoPicker.setOnPickerListener(new IPickerLayout.OnPickerListener() {
        @Override
        public void onPickedList(ArrayList<TCVideoFileInfo> list) {
            // `UGCKit` returns the paths of selected videos.
        }
    });
}
```

The UI view looks like this:



#### 4. Implement video clipping

1. Create an XML file and add the code below:

```
<com.tencent.qcloud.ugckit.UGCKitVideoCut
android:id="@+id/video_cutter"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. Create an empty theme in `res/values/styles.xml` and inherit the default video editing theme of `UGCKit`.

```
<style name="EditorActivityTheme" parent="UGCKitEditorStyle"/>
```

3. Create an activity, implement the `FragmentActivity` API, get a `UGCKitVideoCut` object, and set the callback.

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // You must configure the theme in the code (`setTheme`) or in `AndroidManifest`
    (android:theme).
    setTheme(R.style.EditorActivityTheme);
    setContentView(R.layout.activity_video_cut);
    mUGCKitVideoCut = (UGCKitVideoCut) findViewById(R.id.video_cutter);
    // Get the paths of videos imported via the previous view
    mVideoPath = getIntent().getStringExtra(UGCKitConstants.VIDEO_PATH);
    // `UGCKit` sets the video path.
    mUGCKitVideoCut.setVideoPath(mVideoPath);
    // Listen for the generation of videos
    mUGCKitVideoCut.setOnCutListener(new IVideoCutKit.OnCutListener() {

        @Override
        public void onCutterCompleted(UGCKitResult ugcKitResult) {
            // Callback for the completion of video clipping
        }

        @Override
        public void onCutterCanceled() {
            // Callback for clipping being canceled
        }
    });

    @Override
    protected void onResume() {
        super.onResume();
        // `UGCKit` takes over the lifecycle of the video clipping view. For details, see
        `Github/Demo`.
        mUGCKitVideoCut.startPlay();
    }
}
```

The UI view looks like this:



## 5. Implement video special effect editing

1. In the XML file of the editing activity, add the code below:

```
<com.tencent.qcloud.ugckit.UGCKitVideoEdit
android:id="@+id/video_edit"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

2. Create an editing activity, inherit `FragmentActivity`, get a `UGCKitVideoEdit` object, and set the callback.

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```



```
// You must configure the theme in the code (`setTheme`) or in `AndroidManifest`
(android:theme).
setTheme(R.style.EditorActivityTheme);
setContentView(R.layout.activity_video_editor);
// Set the video path (optional). You can skip this if the previous view is video
clipping and `setVideoEditFlag(true)` is called.
mVideoPath = getIntent().getStringExtra(UGCKitConstants.VIDEO_PATH);
mUGCKitVideoEdit = (UGCKitVideoEdit) findViewById(R.id.video_edit);
if (!TextUtils.isEmpty(mVideoPath)) {
mUGCKitVideoEdit.setVideoPath(mVideoPath);
}
// Initialize the player
mUGCKitVideoEdit.initPlayer();
mUGCKitVideoEdit.setOnVideoEditListener(new IVideoEditKit.OnEditListener() {
@Override
public void onEditCompleted(UGCKitResult ugckitResult) {
// Video editing completed.
}
@Override
public void onEditCanceled() {

}
});
}
@Override
protected void onResume() {
super.onResume();
// `UGCKit` takes over the lifecycle of the editing view. For details, see `Githu
b/Demo`.
mUGCKitVideoEdit.start();
}
```

The UI view looks like this:



## Detailed description

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

- [Capturing and Shooting](#)
- [Video Editing](#)
- [Video Splicing](#)
- [Video Upload](#)
- [Player SDK](#)

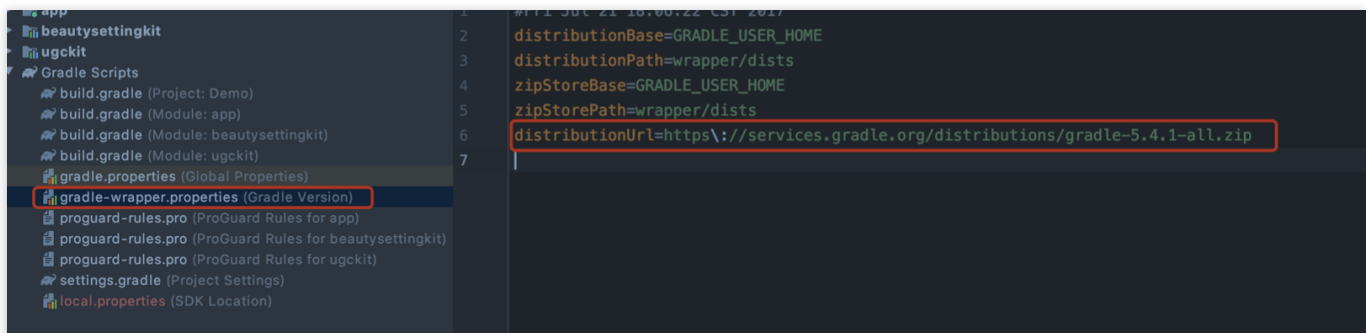
## FAQs

### Can I use AndroidX?

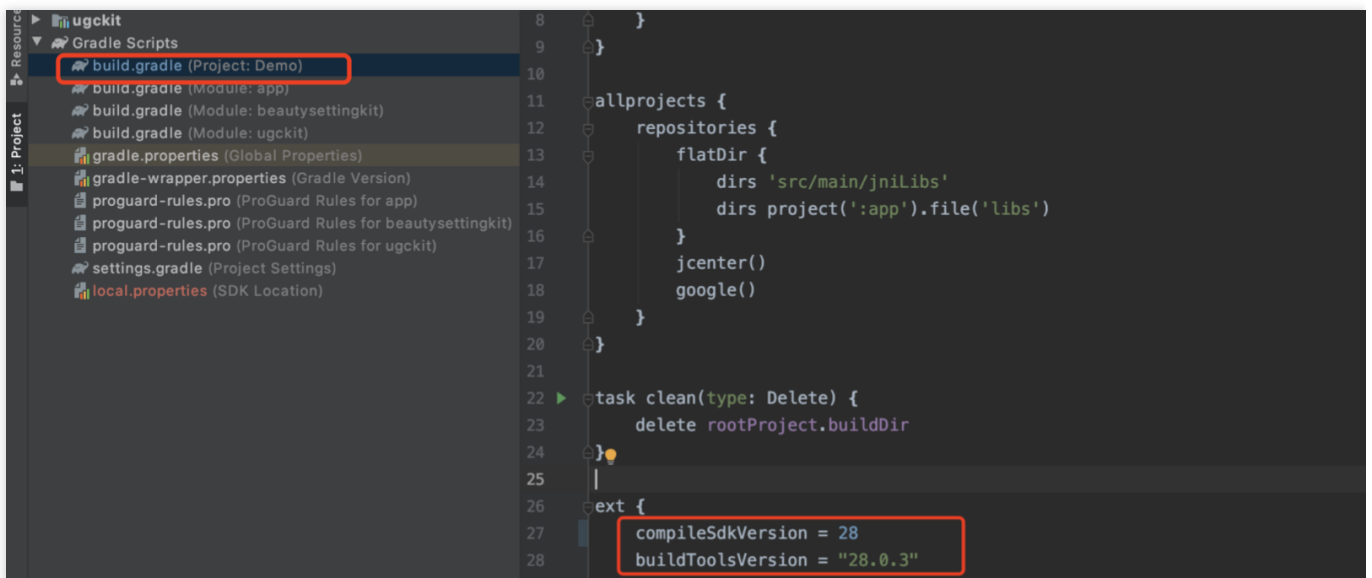
The latest version of `UGCKit` uses AndroidX. If you still use `UGCKit` based on the Android Support Library, you can update it to the latest version or switch to AndroidX as follows. Here, `UGSVSDK` is used as an example, which also uses the `UGCKit` module in its demo.

## 1. Prerequisites:

- Update Android Studio to v3.2 or later.
- Update the Android Gradle plugin to v4.6 or later.

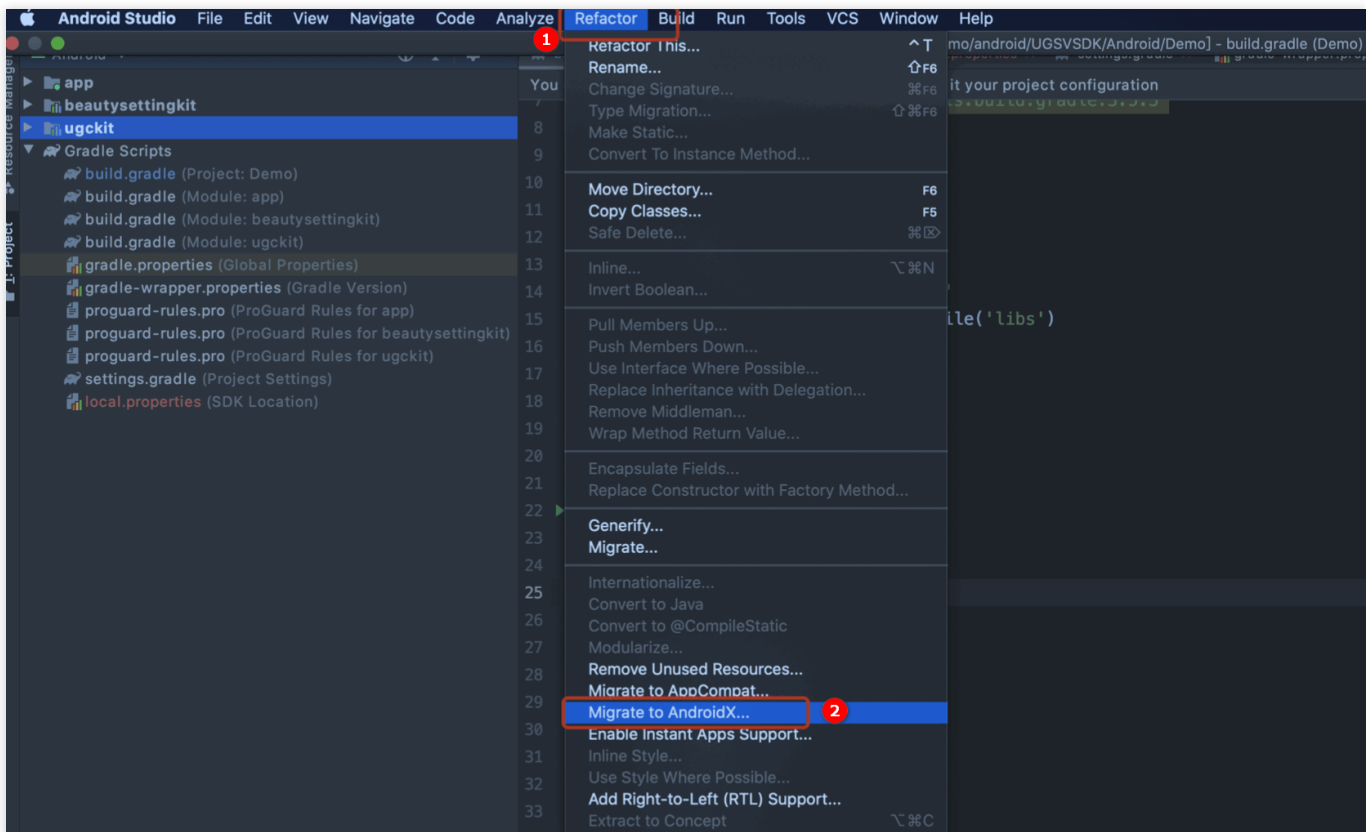


- Update `compileSdkVersion` to 28 or later.
- Update `buildToolsVersion` to 28.0.2 or later.

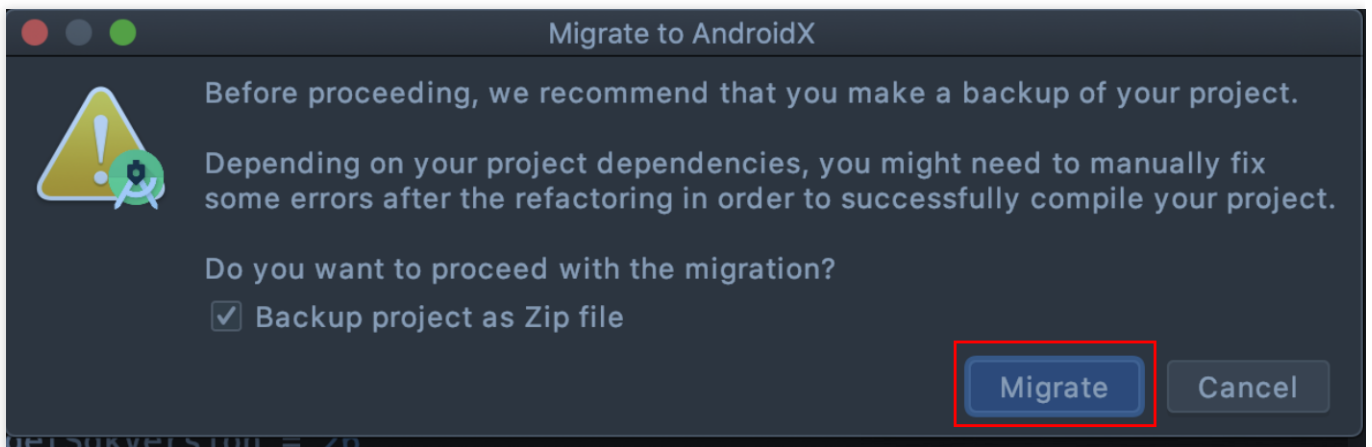


## 2. Migrate to AndroidX:

i. Import the project to Android Studio and select **Refactor > Migrate to AndroidX**.



ii. Click **Migrate** to migrate the current project to AndroidX.



What should I do if a **UGCKit** build version error occurs?

• Error message:

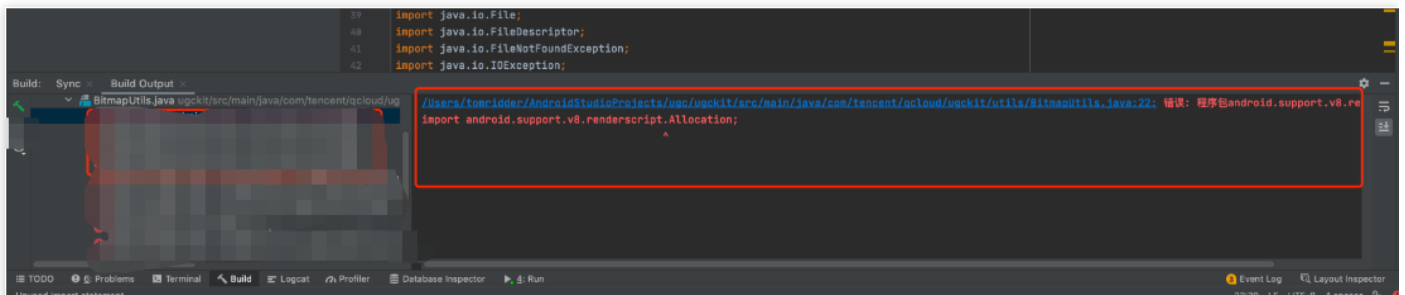
```
ERROR: Unable to find method 'org.gradle.api.tasks.compile.CompileOptions.setBootClasspath(Ljava/lang/String;)V'.
```

**Possible causes for this unexpected error include:**

- **Cause:** The problem occurs because the version of the Gradle plugin used for `UGCKit` is v2.2.3, but that of Gradle is v3.3.
- **Solution:** Check whether the versions of `Android Studio Gradle` and Gradle match. For details, see [Update the Android Gradle plugin](#).

**What should I do if the following error occurs when I build `UGCKit` ?**

- **Error message:**



- **Cause:** The problem occurs because `renderscript-v8.jar` is missing from the `ugckit` module. `renderscript-v8.jar` is responsible for image processing, blurring, and rendering.
- **Solution:** Create a `libs` folder under the `ugckit` module and add `renderscript-v8.jar`, which you can find in `\sdk\build-tools\`, to the folder.

# Capturing and Shoot

# Capturing and Shoot

# iOS

Last updated : 2022-10-25 11:27:24

## Overview

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

## Classes

The UGSV SDK offers different video shooting APIs.

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

## Video Shooting Process

The process of shooting videos is as follows:

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

## Example

```
@interface VideoRecordViewController <TXUGCRecordListener> {
    UIView *_videoRecordView;
}

@implementation VideoRecordViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    // Create a view for the camera preview
    _videoRecordView = [[UIView alloc] initWithFrame:self.view.bounds];
    [self.view addSubview:_videoRecordView];
    // 1. Configure shooting parameters
    TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];
    param.videoQuality = VIDEO_QUALITY_MEDIUM;
    // 2. Enable preview. Specify preview parameters and the view to display the preview.
    [[TXUGCRecord sharedInstance] startCameraSimple:param preview:_videoRecordView];
    // 3. Set shooting effects. In the example below, a watermark is added.
    UIImage *watermarke = [UIImage imageNamed:@"watermarke"];
    [[TXUGCRecord sharedInstance] setWaterMark:watermarke normalizationFrame:CGRectMake(0.01, 0.01, 0.1, 0)];
}

// 4. Start shooting
- (IBAction)onStartRecord:(id)sender {
    [TXUGCRecord sharedInstance].recordDelegate = self;
    int result = [[TXUGCRecord sharedInstance] startRecord];
    if(0 != result) {
        if(-3 == result) [self alert:@"Failed to start shooting." msg:@"Please check whether the camera permission is granted."];
        else if(-4 == result) [self alert:@"Failed to start shooting." msg:@"Please check whether the mic permission is granted."];
        else if(-5 == result) [self alert:@"Failed to start shooting." msg:@"License authentication failed."];
    } else {
        // Started shooting successfully
    }
}

// End shooting
- (IBAction)onStopRecord:(id)sender {
    [[TXUGCRecord sharedInstance] stopRecord];
}

// Callback for ending shooting
- (void) onRecordComplete:(TXUGCRecordResult*)result
{
    if (result.retCode == UGC_RECORD_RESULT_OK) {
        // The video was shot successfully and was saved to `result.videoPath`.
    }
}
```

```
} else {  
    // Handle errors. For the definitions of error codes, see `TXUGCRecordResultCode`  
    in `TXUGCRecordTypeDef.h`.  
}  
}  
- (void) alert: (NSString *)title msg: (NSString *)msg  
{  
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title message:msg delegate:  
        self cancelButtonTitle:@"Confirm" otherButtonTitles:nil, nil];  
    [alert show];  
}  
@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. Enable preview

```
TXUGCRecord *record = [TXUGCRecord sharedInstance];  
record.recordDelegate = self; //Configure shooting callbacks. For details, see `TXUGCRecordListener`.  
// Configure camera parameters and start preview  
TXUGCSimpleConfig * param = [[TXUGCSimpleConfig alloc] init];  
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p  
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p  
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p  
param.frontCamera = YES; // Use the front camera  
param.minDuration = 5; // Set the minimum shooting duration to 5 seconds  
param.maxDuration = 60; // Set the maximum shooting duration to 60 seconds  
param.enableBFrame = YES; // Enable B frame encoding, which produces better video  
quality at the same bitrate  
// Display the preview in `self.previewView`  
[recorder startCameraSimple:param preview:self.previewView];  
// Disable preview  
[[TXUGCRecord sharedInstance] stopCameraPreview];
```

### 2. Modify preview parameters

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



```
// Change the shooting resolution to 540p
[recorder setVideoResolution: VIDEO_RESOLUTION_540_960];
// Change the video bitrate for shooting to 6500 Kbps
[recorder setVideoBitrate: 6500];
// Set the focal length to 3. 1 indicates the widest angle of view (original), and 5 indicates the narrowest angle of view (zoomed in).
[recorder setZoom: 3];
// Switch the camera. `Yes`: Front Camera; `No`: Rear camera.
[recorder switchCamera: NO];
// Turn on/off the flashlight. `Yes`: Turn on the flashlight; `No`: Turn off the flashlight.
[recorder toggleTorch: YES];
// Set the callback for custom processing
recorder.videoProcessDelegate = delegate;
```

## Shooting control

### Start, pause, and resume shooting

```
// Start shooting a video
[recorder startRecord];
// Start shooting. You can specify the path to save the video and the path of the thumbnail image.
[recorder startRecord:videoFilePath coverPath:coverPath];
// Start shooting. You can specify the paths to save the video and video segments, as well as the path of the thumbnail image.
[recorder startRecord:videoFilePath videoPartsFolder:videoPartFolder coverPath:coverPath];
// Pause shooting
[recorder pauseRecord];
// Resume shooting
[recorder resumeRecord];
// Ending shooting
[recorder 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

```
// Shoot in landscape mode
[recorder setHomeOrientation:VIDOE_HOME_ORIENTATION_RIGHT];
// 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.
[recorder 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.
[recorder 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
[recorder setRecordSpeed:VIDEO_RECORD_SPEED_NOMAL];
```

### 3. Audio

```
// Set the mic volume. This is used to control the volume of the mic when backgro
und music is mixed.
// Volume. The normal volume is 1. We recommend 0-2, but you can set it to a larg
er value if you want louder music.
[recorder setMicVolume:volume];
// Mute/Unmute. The `isMute` parameter specifies whether to mute audio. Audio is
unmuted by default.
[recorder setMute:isMute];
```

## Picture taking

```
// Take a photo/Capture a frame from a video. This API works only if it is called
after `startCameraSimplePreview` or `startCameraCustomPreview`.
[recorder snapshot:^(UIImage *image) {
// `image` is the capturing result.
}];
```

## 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 th
e 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 / waterMarkImag
e.size.width)
[recorder 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`.
[recorder setFilter:filterImage];
// Set the strength of filters. Value range: 0-1. Default: 0.5. The greater the value, the stronger the filter.
[recorder 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".
[recorder setFilter:leftFilterImage leftIntensity:leftIntensity rightFilter:rightFilterImage rightIntensity:rightIntensity leftRatio:leftRatio];
```

## 3. Beauty filters

```
// Set the beauty filter style and strength and the strength of the skin brightening and blush effects
// beautyStyle:
// typedef NSInteger(NSInteger, TXVideoBeautyStyle) {
//     VIDEOE_BEAUTY_STYLE_SMOOTH = 0, // Smooth
//     VIDEOE_BEAUTY_STYLE_NATURE = 1, // Natural
// };
// The value range for the strength parameter is 0-9. `0` means to disable the filter. The greater the value, the stronger the filter.
[recorder setBeautyStyle:beautyStyle beautyLevel:beautyLevel whitenessLevel:whitenessLevel ruddinessLevel:ruddinessLevel];
```

# Advanced Features

[Multi-segment shooting](#)

[Drafts](#)

[Adding background music](#)

[Voice changing and reverb](#)

[Customizing video data](#)

# Android

Last updated : 2022-05-24 15:45:26

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

## Use Instructions

The following is the basic usage process of video shoot:

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

## Code Example

```
// Create `TXCloudVideoView` for camera preview
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view);
// 1. Configure the shoot parameters. The recommended configuration of `TXUGCSimpleConfig` is used here as an example
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM;
// 2. Start video image preview
mTXCameraRecord.startCameraSimplePreview(param, mVideoView);
// 3. Set the shoot effects. Watermarking is used here as an example
TXVideoEditConstants.TXRect rect = new TXVideoEditConstants.TXRect();
```

```
rect.x = 0.5f;
rect.y = 0.5f;
rect.width = 0.5f;
mTXCameraRecord.setWatermark(BitmapFactory.decodeResource(getResources(), R.drawable.watermark), rect);
// 4. Start shoot
int result = mTXCameraRecord.startRecord();
if (result != TXRecordCommon.START_RECORD_OK) {
    if (result == -4) {The video image has not been displayed yet}
    else if (result == -3) {The version is too low}
    else if (result == -5) {License verification failed}
}
else{// Started successfully}
// End shoot
mTXCameraRecord.stopRecord();
// Callback for shoot completion
public void onRecordComplete(TXRecordCommon.TXRecordResult result) {
    if (result.retCode >= 0 ) {
        // The shoot is successful, and the video file is in `result.videoPath`
    }else{
        // Process the error. For the error code definition, please see "Definition of error codes for shoot result callback" in `TXRecordCommon`.
    }
}
```

## 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

```
TXUGCRecord mTXCameraRecord = TXUGCRecord.getInstance(this.getApplicationContext());
mTXCameraRecord.setVideoRecordListener(this); // Set shoot callback
mVideoView = (TXCloudVideoView) findViewById(R.id.video_view); // Prepare a view for camera image preview
TXRecordCommon.TXUGCSimpleConfig param = new TXRecordCommon.TXUGCSimpleConfig();
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_LOW; // 360p
//param.videoQuality = TXRecordCommon.VIDEO_QUALITY_MEDIUM; // 540p
param.videoQuality = TXRecordCommon.VIDEO_QUALITY_HIGH; // 720p
```

```
param.isFront = true; // Whether to use the front camera
param.minDuration = 5000; // Minimum video shoot duration in milliseconds
param.maxDuration = 60000; // Maximum video shoot duration in milliseconds
param.touchFocus = false; // false: autofocus; true: manual focus
mTXCameraRecord.startCameraSimplePreview(param,mVideoView);
// End video image preview
mTXCameraRecord.stopCameraPreview();
```

## 2. Adjust preview parameters

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

```
// Switch the video shoot resolution to 540p
mTXCameraRecord.setVideoResolution(TXRecordCommon.VIDEO_RESOLUTION_540_960);
// Switch the video shoot bitrate to 6,500 Kbps
mTXCameraRecord.setVideoBitrate(6500);
// Get the maximum focal length supported by the camera
mTXCameraRecord.getMaxZoom();
// Set the focal length to 3. 1 indicates the furthest view (normal lens), and 5
indicates the nearest view (enlarging lens)
mTXCameraRecord.setZoom(3);
// Switch to the rear camera. true: switches to front camera; false: switches to
rear camera
mTXCameraRecord.switchCamera(false);
// Enable the flash. true: enables; false: disables
mTXCameraRecord.toggleTorch(false);
// If `param.touchFocus` is `true`, manual focus will be used. You can use the fo
llowing API to set the focus position
mTXCameraRecord.setFocusPosition(eventX, eventY);
// Set the callback for custom image processing
mTXCameraRecord.setVideoProcessListener(this);
```

## Capturing

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

```
// Photo capturing, which will take effect if called after `startCameraSimplePrev
iew` or `startCameraCustomPreview`
mTXCameraRecord.snapshot(new TXRecordCommon.ITXSnapshotListener() {
    @Override
    public void onSnapshot(Bitmap bmp) {
        // Save or display the photo
    }
});
```

```
}  
});
```

## Controlling Shoot Process

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

```
// Start shoot  
mTXCameraRecord.startRecord();  
// Start shoot. You can specify the addresses for the output video file and cover  
mTXCameraRecord.startRecord(videoFilePath, coverPath);  
// Start shoot. You can specify the addresses for the output video file, video segment storage, and cover  
mTXCameraRecord.startRecord(videoFilePath, videoPartFolder, coverPath);  
// Pause shoot  
mTXCameraRecord.pauseRecord();  
// Resume shoot  
mTXCameraRecord.resumeRecord();  
// End shoot  
mTXCameraRecord.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);
```

## Setting Shoot Attributes

### Set the video image

```
// Set the landscape or portrait mode for shoot
mTXCameraRecord.setHomeOrientation(TXLiveConstants.VIDEO_ANGLE_HOME_RIGHT);
// Set the video preview direction
// TXLiveConstants.RENDER_ROTATION_0 (normal portrait)
// TXLiveConstants.RENDER_ROTATION_90 (rotated 90 degrees leftwards)
// TXLiveConstants.RENDER_ROTATION_180 (rotated 180 degrees leftwards)
// TXLiveConstants.RENDER_ROTATION_270 (rotated 270 degrees leftwards)
// Note: it needs to be set before `startRecord` and will not take effect if set
during shoot
mTXCameraRecord.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
// Set the aspect ratio for shoot
// TXRecordCommon.VIDEO_ASPECT_RATIO_9_16 The aspect ratio is 9:16
// TXRecordCommon.VIDEO_ASPECT_RATIO_3_4 The aspect ratio is 3:4
// TXRecordCommon.VIDEO_ASPECT_RATIO_1_1 The aspect ratio is 1:1
// Note: it needs to be set before `startRecord` and will not take effect if set
during shoot
mTXCameraRecord.setAspectRatio(TXRecordCommon.VIDEO_ASPECT_RATIO_9_16);
```

### Set the 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)
mTXCameraRecord.setRecordSpeed(TXRecordCommon.VIDEO_RECORD_SPEED_NORMAL);
```

### Set the audio

```
// Set the mic volume level. It is used to adjust the mic volume level when backg
round music is played back
// Volume level. 1 indicates the normal volume level. The recommended value range
is 0-2. If you want to increase the volume level, you can set a higher value
```

```
mTXCameraRecord.setMicVolume(volume);  
// Set whether the shoot is muted in the `isMute` parameter. The shoot is unmuted  
by default  
mTXCameraRecord.setMute(isMute);
```

## Setting Effects

During video shoot, you can set various special effects for the shot video.

### Watermark

```
// Set a global watermark  
// TXRect: normalized value of watermark relative to video image. The SDK will au  
tomatically calculate the `height` according to the watermark aspect ratio  
// Suppose the video image dimensions are (540, 960), and three parameters in `TX  
Rect` are all set to 0.1  
// Then, the actual pixel coordinates of the watermark are (540 * 0.1, 960 * 0.1,  
540 * 0.1)  
// 540 * 0.1 * watermarkBitmap.height / watermarkBitmap.width)  
mTXCameraRecord.setWatermark(watermarkBitmap, txRect)
```

### Filter

```
// Set the color filter to Romantic, Fresh, Aesthetic, Rosy, Vintage, etc.  
// filterBitmap: color lookup table used for filter. Note: it must be in .png for  
mat  
// The filter color lookup table used by the demo is in the `RTMPAndroidDemo/app/  
src/main/res/drawable-xxhdpi/` directory  
mTXCameraRecord.setFilter(filterBitmap);  
// Set the filter mix effect  
// mLeftBitmap Filter on the left  
// leftIntensity Level of the filter on the left  
// mRightBitmap Filter on the right  
// rightIntensity Level of the filter on the right  
// leftRadio Ratio of the dimensions of the image on the left  
// You can use this API to implement the effect of switching filters by swipe. Fo  
r more information, please see the demo  
mTXCameraRecord.setFilter(mLeftBitmap, leftIntensity, mRightBitmap, rightIntensit  
y, leftRatio);  
// It is used to set the filter effect level. Value range: 0-1. The greater the v  
alue, the more obvious the effect. Default value: 0.5  
mTXCameraRecord.setSpecialRatio(0.5);
```

## Beauty filter

```
// Set the beauty filter type
mTXCameraRecord.setBeautyStyle(style);
// Set the eye enlarging effect level. Recommended value range: 0-9. If you want
a more obvious effect, you can set a greater value
mTXCameraRecord.setEyeScaleLevel(eyeScaleLevel);
// Set the face slimming effect level. Recommended value range: 0-9. If you want
a more obvious effect, you can set a greater value
mTXCameraRecord.setFaceScaleLevel(faceScaleLevel);
// Set the chin slimming effect level. Recommended value range: 0-9. If you want
a more obvious effect, you can set a greater value
mTXCameraRecord.setFaceVLevel(level);
// Set the chin lengthening/shortening effect level. Recommended value range: 0-
9. If you want a more obvious effect, you can set a greater value
mTXCameraRecord.setChinLevel(scale);
// Set the face shortening effect level. Recommended value range: 0-9. If you wan
t a more obvious effect, you can set a greater value
mTXCameraRecord.setFaceShortLevel(level);
// Set the nose narrowing effect level. Recommended value range: 0-9. If you want
a more obvious effect, you can set a greater value
mTXCameraRecord.setNoseSlimLevel(scale);
// Set the green screen keying file. It can be in formats supported by Android, s
uch as .jpg and .png for images and .mp4 and .3gp for videos, and can be looped
mTXCameraRecord.setGreenScreenFile(path, isLoop);
// Set `motionTplPath`, which is the path of the animated effect file of the ani
mated sticker. An empty string ("" ) indicates to cancel the animated effect
mTXCameraRecord.setMotionTmp(motionTplPath);
// Set whether to mute the animated sticker. true: mutes; false: unmutes
mTXCameraRecord.setMotionMute(true);
```

## Getting License Information

UGSV license verification is added in the new version of the SDK. If the verification fails, you can use the following API to query the specific information in the license:

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

## Advanced Features

[Multi-segment shoot](#)

[Shoot drafts](#)

[Adding background music](#)

[Voice changing and reverb](#)

[Customizing video data](#)

# Multi-Segment Shoot

## iOS

Last updated : 2022-05-24 15:59:37

The following is the basic usage process of multi-segment video shoot:

1. Start video image preview.
2. Start shoot.
3. Play back the background music.
4. Pause shoot.
5. Pause the background music.
6. Resume the background music.
7. Resume shoot.
8. Stop shoot.
9. Stop the background music.

```
// Start video image preview
recorder = [TXUGCRecord sharedInstance];
[recorder startCameraCustom:param preview:preview];
// Start shoot
[recorder startRecord];
// Set the background music
[recorder setBGM:BGMPath];
// Play back the background music
[recorder playBGMFromTime:beginTime toTime:_BGMDuration withBeginNotify:^(NSInteger errCode) {
// Start playback
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
// Playback progress
} andCompleteNotify:^(NSInteger errCode) {
// End playback
}]];
// After `pauseRecord` is called, a video segment will be generated, which can be
// obtained from and managed in `TXUGCPartsManager`
[recorder pauseRecord];
// Pause the background music
[recorder pauseBGM];
// Resume the background music
[recorder resumeBGM];
// Resume video shoot
[recorder resumeRecord];
```

```
// Stop video shoot and compose multiple video segments into one video
[recorder stopRecord];
// Stop the background music
[recorder stopBGM];
// Get the video segment management object
TXUGCPartsManager *partsManager = recorder.partsManager;
// Get the total duration of all video segments
[partsManager getDuration];
// Get the paths of all video segments
[partsManager getVideoPathList];
// Delete the last video segment
[partsManager deleteLastPart];
// Delete a specified video segment
[partsManager deletePart:1];
// Delete all video segments
[partsManager deleteAllParts];
// You can add videos except the one currently being shot
[partsManager insertPart:videoPath atIndex:0];
// Compose all video segments
[partsManager joinAllParts: videoOutputPath complete:complete];
```

# Android

Last updated : 2022-05-24 16:00:18

The following is the basic usage process of multi-segment video shoot:

1. Start video image preview.
2. Start shoot.
3. Play back the background music.
4. Pause shoot.
5. Pause the background music.
6. Resume shoot.
7. Resume the background music.
8. Stop shoot.
9. Stop the background music.

```
// Start shoot
mTXCameraRecord.startRecord();
// After `pauseRecord` is called, a video segment will be generated, which can be
// obtained from `TXUGCPartsManager`
mTXCameraRecord.pauseRecord();
mTXCameraRecord.pauseBGM();
// Resume video shoot
mTXCameraRecord.resumeRecord();
mTXCameraRecord.resumeBGM();
// Stop video shoot and compose multiple video segments into one video
mTXCameraRecord.stopBGM();
mTXCameraRecord.stopRecord();
// Get the video segment management object
mTXCameraRecord.getPartsManager();
// Get the total duration of all video segments
mTXUGCPartsManager.getDuration();
// Get the paths of all video segments
mTXUGCPartsManager.getPartsPathList();
// Delete the last video segment
mTXUGCPartsManager.deleteLastPart();
// Delete a specified video segment
mTXUGCPartsManager.deletePart(index);
// Delete all video segments
mTXUGCPartsManager.deleteAllParts();
// You can add videos except the one currently being shot
mTXUGCPartsManager.insertPart(videoPath, index);
```

# Shoot Drafts

## iOS

Last updated : 2022-05-24 16:02:12

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 sharedInstance];
//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 sharedInstance];
//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

Last updated : 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 curr
ently 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

Last updated : 2022-05-24 16:03:54

### Adding Background Music During Shooting

```
// Get the `recorder` object
TXUGCRecord *recorder = [TXUGCRecord sharedInstance];
// Set the background music file path
[recorder setBGMAsset:path];
// Set the background music. If the music file is loaded from the system media li
brary, you can directly pass in the corresponding `AVAsset`
[recorder setBGMAsset:asset];
// Play back the background music
[recorder playBGMFromTime:beginTime
toTime:endTime
withBeginNotify:^(NSInteger errorCode) {
// Callback for the start of playback. If `errorCode` is 0, it is a success; otherw
ise, it is a failure
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
// progressMS: duration of the part that has been played back. durationMS: total
duration
} andCompleteNotify:^(NSInteger errorCode) {
// Callback for the end of playback. If `errorCode` is 0, it is a success; otherwis
e, it is a failure
}]];
// Stop the background music
[recorder stopBGM];
// Pause the background music
[recorder pauseBGM];
// Resume the background music
[recorder resumeBGM];
// Set mic volume level. It is used to adjust the mic volume level when backgroun
d music is played back
// volume: volume level. 1 indicates the normal volume level. The recommended val
ue range is 0-2. If you want to increase the volume level, you can set a higher v
alue
[recorder setMicVolume:1.0];
// `setBGMVolume` is used to set the background music volume level. It controls t
he background music volume level during playback.
// volume: volume level. 1 indicates the normal volume level. The recommended val
ue range is 0-2. If you want to increase the background music volume level, you c
```

```
an set a higher value  
[recorder setBGMVolume:1.0];
```

## Adding Background Music During Editing

```
// Initialize the editor  
TXPreviewParam *param = [[TXPreviewParam alloc] init];  
param.videoView = videoView;  
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;  
ugcEdit = [[TXVideoEditor alloc] initWithPreview:param];  
// Set the background music file path  
[ugcEdit setBGMAAsset:fileAsset result:^(int result) {  
}];  
// Set the start and end time for background music playback  
[ugcEdit setBGMStartTime:0 endTime:5];  
// Set whether to loop the background music  
[ugcEdit setBGMLoop:YES];  
// Set the start position where the background music is to be added in the video  
[ugcEdit setBGMAAtVideoTime:0];  
// Set the video volume level  
[ugcEdit setVideoVolume:1.0];  
// Set the background music volume level  
[ugcEdit setBGMVolume:1.0];
```

After you set the background music, when you start the editor for video preview, the background music will be played back according to the configured parameters. After you start the editor for video generation, the background music will be composed into the generated video according to the configured parameters.

# Android

Last updated : 2022-05-24 16:04:08

## Adding music during shooting

```
// Specify the path of the music to add
mTXCameraRecord.setBGM(path);

// Set the callback (TXRecordCommon.ITXBGMNotify) for music playback
mTXCameraRecord.setBGMNotify(notify);

// Play the music
mTXCameraRecord.playBGMFromTime(startTime, endTime)

// Stop the music
mTXCameraRecord.stopBGM();

// Pause the music
mTXCameraRecord.pauseBGM();

// Resume the music
mTXCameraRecord.resumeBGM();

// Set the volume of the music mixed into other audio of the video
// Volume. The normal volume is 1. We recommend 0-2, but you can set it to a larger value if you want louder music.
mTXCameraRecord.setBGMVolume(x);

// Set the start and end time for music playback. This API must be called before startPlay to take effect.
mTXCameraRecord.seekBGM(startTime, endTime);
```

## Adding music during editing

```
// Set the path of the music to add. If 0 is returned, the setting is successful. Other values indicate failure to set the path due to reasons such as unsupported audio format.
public int setBGM(String path);
```

```
// Set the start and end time (ms) for music playback
public void setBGMStartTime(long startTime, long endTime);

// Set whether to loop the music. true: loop; false: do not loop
public void setBGMLoop(boolean looping);

// Set where to start adding the music
public void setBGMAtVideoTime(long videoStartTime);

// Set the audio volume of the video. The value range of the volume parameter is
0-1. 0 means to mute the video, and 1 means to use the original volume.
public void setVideoVolume(float volume);

// Set the volume of the music. The value range of the volume parameter is 0-1.
0 means to mute the music, and 1 means to use the original volume.
public void setBGMVolume(float volume);
```

Note :

After you complete the settings, the music will be played as configured when the video is previewed and will also be added to the video generated.

# Voice Changing and Reverb

## iOS

Last updated : 2022-05-24 16:06:06

### Voice changing and reverb for video shooting:

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

Last updated : 2022-05-24 16:06:34

## 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_TYPE_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

Last updated : 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
TXVideoEditor.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
{
    TXVideoEditor *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
    TXVideoEditor *editor = [[TXVideoEditor 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 gene
    // ration 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 interval
 * s)
 * @param maxSize Maximum thumbnail dimensions. The dimensions of the generated th
 * umbnails 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 `TXVideoEditor` 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 `TXVideoEditor` 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 `TXVideoEditor` 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:

```
TXVideoEditor *_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:CGRectMakeMake(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.

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

Last updated : 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
<code>TXVideoInfoReader</code>	Media information obtaining
<code>TXVideoEditor</code>	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().getVideoF
ileInfo(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:

```
mTXVideoEditor.getThumbnail(TCVideoEditorWrapper.mThumbnailCount, 100, 100, false, mThumbnailListener);  
private TXVideoEditor.TXThumbnailListener mThumbnailListener = new TXVideoEditor.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);  
  
TXVideoEditor txVideoEditor = new TXVideoEditor(TCVideoPreviewActivity.this);  
txVideoEditor.setVideoPath(mVideoPath);  
txVideoEditor.setThumbnailListener(new TXVideoEditor.TXThumbnailListener() {  
    @Override  
    public void onThumbnail(int index, long timeMs, Bitmap bitmap) {  
        Log.i(TAG, "bitmap:" + bitmap + ",timeMs:" + timeMs);  
        saveBitmap(bitmap, timeMs);  
    }  
});  
txVideoEditor.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
mTXVideoEditor.setThumbnail(thumbnail); // Configure the thumbnails generated dur
ing pre-processing
mTXVideoEditor.setThumbnailListener(mThumbnailListener); // Configure the callbac
k for thumbnail generation
mTXVideoEditor.setVideoProcessListener(this); // Configure the callback of video
pre-processing progress
mTXVideoEditor.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 ima
ge is kept intact, and there may be black bars if the aspect ratio does not matc
h.
```

## 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 `TXVideoEditor` 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;  
mTXVideoEditor.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);
```

`tailWaterMark` 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;
mTXVideoEditor.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 `mTXVideoEditor` 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

Last updated : 2022-05-24 15:47:45

## Reusing Existing UI

The video splicer has complicated interaction logic, which means that the UI is also complicated; therefore, we recommend you reuse the UI source code in the SDK. The `VideoJoiner` directory contains the UI source code of the short video splicer.

- **VideoJoinerController**: it is used to implement the video splicing list as shown above, which supports drag-up/drag-down for order adjustment.
- **VideoJoinerCell**: it is used to splice all video segments in the list.
- **VideoEditPrevController**: it is used to preview the spliced video.

## Implementing UI on Your Own

If you do not want to reuse the UI code in the SDK, you can implement the UI on your own as follows:

### 1. Select video files

In the demo, the `Q UIImagePickerController` open-source library is used to implement the multi-file selection feature. The relevant code can be found in `MainViewController` of the demo.

### 2. Set the preview view

You need to create a `TXVideoJoiner` object for video splicing. Just like `TXUGCEditor`, the preview feature also requires the upper layer to provide the preview `UIView`:

```
// Prepare the preview view
TXPreviewParam *param = [[TXPreviewParam alloc] init];
param.videoView = _videoPreview.renderView;
param.renderMode = PREVIEW_RENDER_MODE_FILL_EDGE;
// Create a `TXVideoJoiner` object and set the preview view
TXVideoJoiner* _videoJoin = [[TXVideoJoiner alloc] initWithPreview:param];
_videoJoin.previewDelegate = _videoPreview;
// Set the video file group `_composeArray` for splicing, i.e., multiple files selected in step 1
[_videoJoin setVideoPathList:_composeArray];
```

After setting the preview view and passing in the array of video files to be spliced, you can play back the preview. The splicing module provides some APIs for video playback preview:

- `startPlay` : starts the video.
- `pausePlay` : pauses the video.
- `resumePlay` : resumes the video.

### 3. Generate the final file

If the preview effect is satisfactory, you can call the generation API to generate the spliced file:

```
_videoJoin.joinerDelegate = self;  
[_videoJoin joinVideo:VIDEO_COMPRESSED_540P videoOutputPath:_outFilePath];
```

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

# Android

Last updated : 2022-05-24 15:51:14

## Reusing Existing UI

The video splicer has complicated interaction logic, which means that the UI is also complicated; therefore, we recommend you reuse the UI source code in the SDK. The `videojoiner` directory contains the UI source code of the short video splicer.

- `TCVideoJoinerActivity` is used to implement the video splicing list as shown above, which supports drag-up/drag-down for order adjustment.
- `TCVideoJoinerPreviewActivity` is used to preview the spliced video.

## Implementing UI on Your Own

If you do not want to reuse the UI code in the SDK, you can implement the UI on your own as follows:

### 1. Select video files

Implement the multi-file selection feature on your own.

### 2. Set the preview

You need to create a `TXVideoJoiner` object for video splicing. Just like `TXVideoEditor`, the preview feature also requires the upper layer to provide the preview `FrameLayout`:

```
// Prepare the preview view
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreviewParam();
param.videoView = mView;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
// Create a `TXUGCJoiner` object and set the preview view
TXVideoJoiner mTXVideoJoiner = new TXVideoJoiner(this);
mTXVideoJoiner.setTXVideoPreviewListener(this);
mTXVideoJoiner.initWithPreview(param);
// Set the video file group `mVideoSourceList` for splicing, i.e., multiple files selected in step 1
mTXVideoJoiner.setVideoPathList(mVideoSourceList);
```

After setting the preview view and passing in the array of video files to be spliced, you can play back the preview. The splicing module provides some APIs for video playback preview:

- `startPlay`: starts the video.
- `pausePlay`: pauses the video.
- `resumePlay`: resumes the video.

### 3. Generate the final file

If the preview effect is satisfactory, you can call the generation API to generate the spliced file:

```
mTXVideoJoiner.setVideoJoinerListener(this);  
mTXVideoJoiner.joinVideo(TXVideoEditConstants.VIDEO_COMPRESSED_540P, mVideoOutput  
Path);
```

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

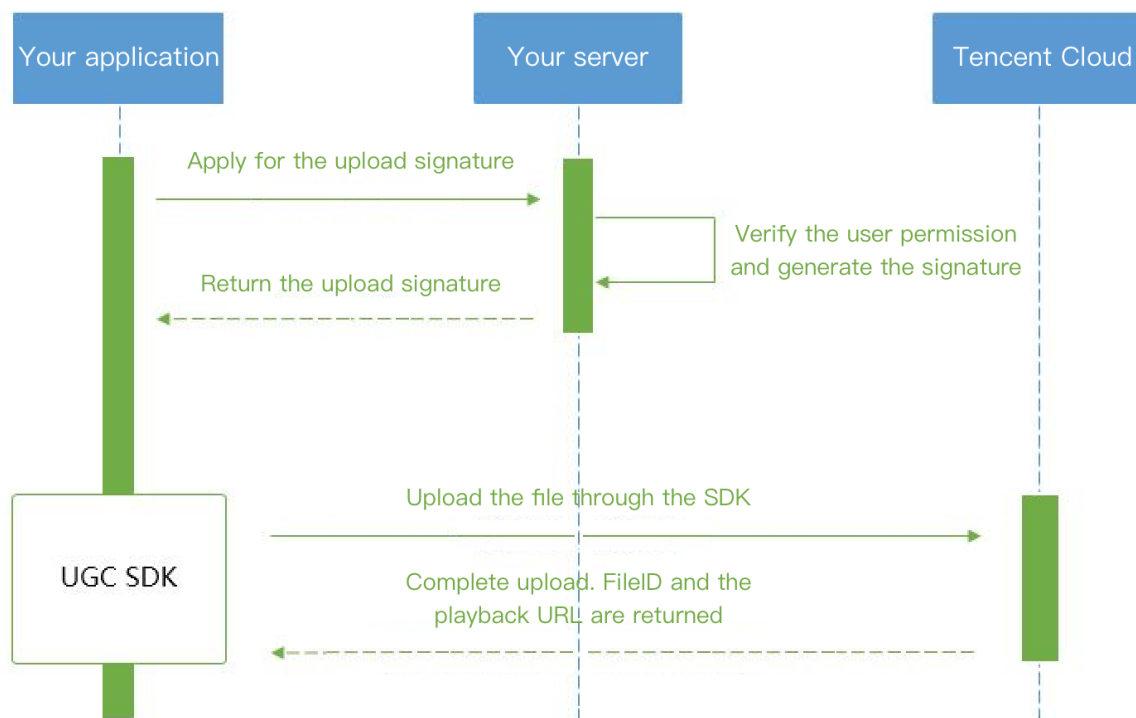
# Upload and Playback Signature Distribution

Last updated : 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 original = querystring.stringify(arg_list);
  let original_buffer = new Buffer(original, "utf8");
```

```
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(original_buffer).digest();
let signature = Buffer.concat([hmac_buffer, original_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

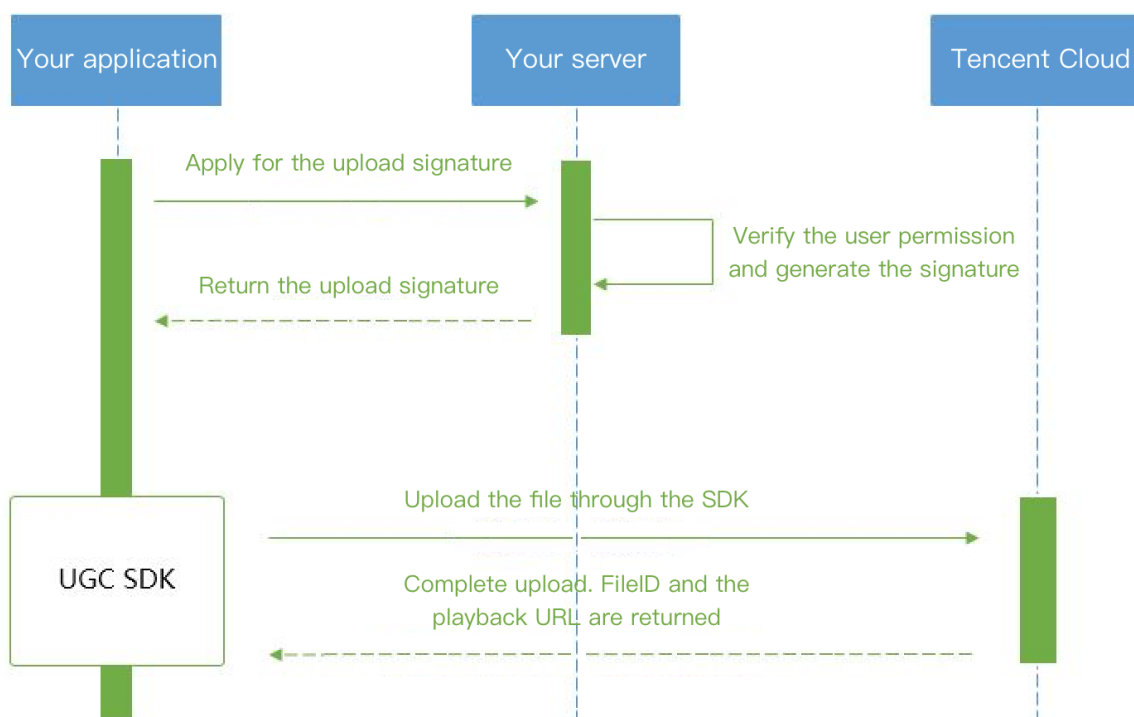
Last updated : 2022-05-24 16:08:49

## 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

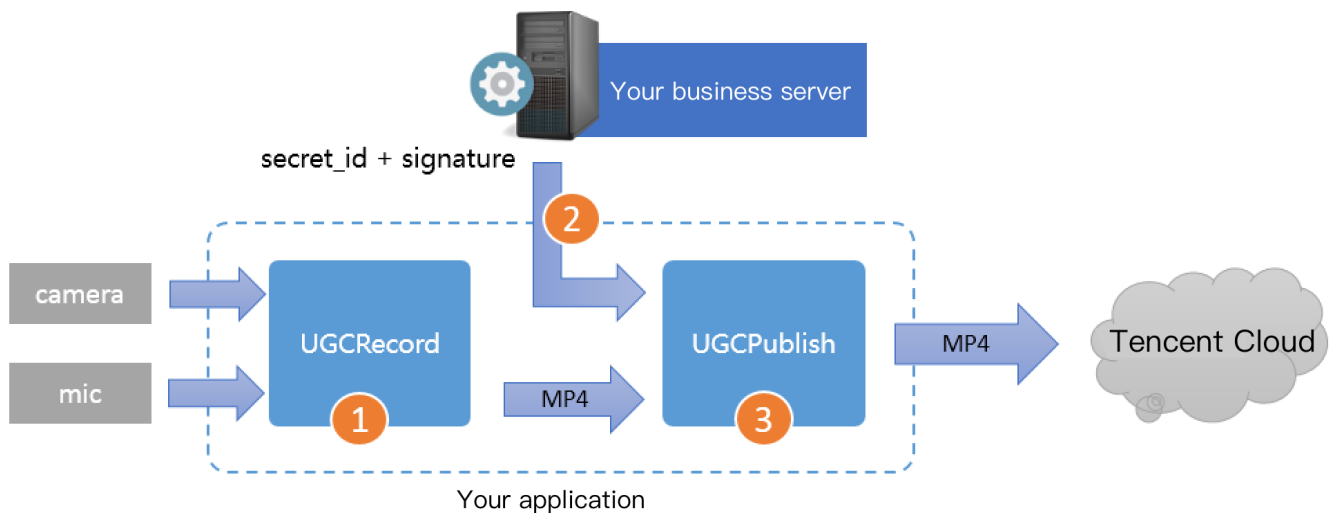
```
/**
 * Calculate a signature
 */
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId = 0, oneTimeValid = 0, sourceContext = '' }) {
  // Determine the generation time and expiration time of the signature
  let current = parseInt((new Date()).getTime() / 1000)
  let expired = 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 original = querystring.stringify(arg_list);
  let original_buffer = new Buffer(original, "utf8");
  let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
  let hmac_buffer = hmac.update(original_buffer).digest();
  let signature = Buffer.concat([hmac_buffer, original_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 `TXVideoEditor.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

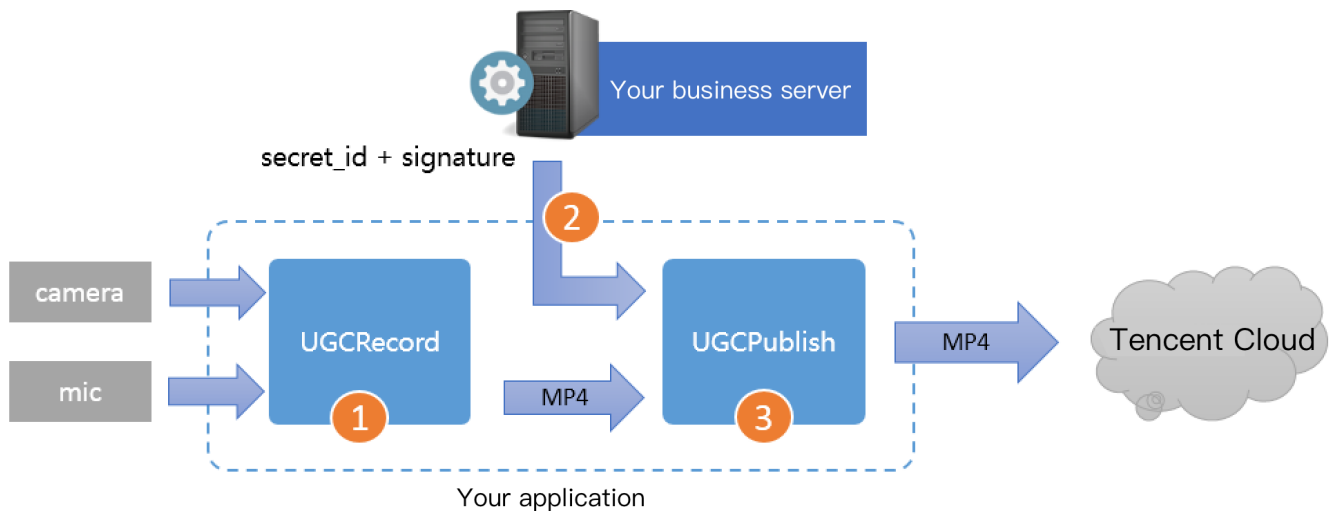
After the video is successfully uploaded in step 3, the video `fileId`, playback URL, and cover URL will be returned. You can directly pass in the `fileId` or playback URL to the [VOD player](#) for playback.

# Android

Last updated : 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 `TXVideoEditor.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
<code>errCode</code>	Error code.
<code>descMsg</code>	Error message.
<code>videoURL</code>	VOD address of short video.
<code>coverURL</code>	Cloud storage address of video cover.
<code>videoid</code>	Cloud storage ID of video file, through which you can call VOD's <a href="#">server APIs</a> .

- 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

Last updated : 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

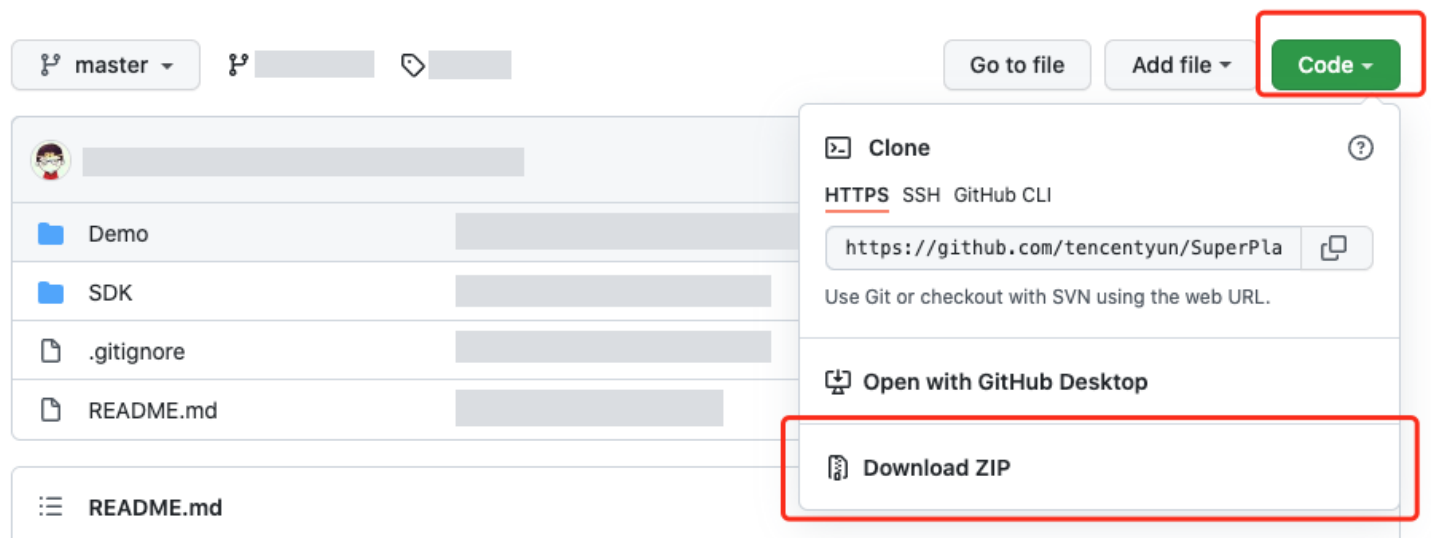
### Step 1. Download the player code package

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
- Download using a Git command

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



## 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
- Manually download the SDK

1. To install the component using CocoaPods, add the code below to the Podfile:

(1) 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` .

### 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
- Play by URL

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

- After a video is published from a client, the server will return a file ID to the client.
- 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.

<input type="checkbox"/>	Audio/Video name/ID	Audio/Video status	Audio/Video category	Uploading time	Expiration time	Storage region	Operation
<input type="checkbox"/>	 ID:38 00:19:22 <a href="#">Quick View</a>	 Normal	Other	2022-10-31 00:53:21	Permanent	Outside Chinese mainland	<a href="#">Manage</a> <a href="#">Delete</a> <a href="#">Download</a>
<input type="checkbox"/>	 ID:38770230 00:25:35	 Normal	Other	2022-10-31 00:52:45	Permanent	Outside Chinese mainland	<a href="#">Manage</a> <a href="#">Delete</a> <a href="#">Download</a>
<input type="checkbox"/>	 ID:38 00:45:44	 Normal	Other	2022-10-31 00:52:03	Permanent	Outside Chinese mainland	<a href="#">Manage</a> <a href="#">Delete</a> <a href="#">Download</a>

Total items: 3

10 / page 1 / 1 page

>!

>- 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];
```

- **Stop playback**

```
[_playerView resetPlayer];
```

## More Features

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.

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

- Back to windowed mode
- Enable screen locking
- On-screen comments
- Screenshot
- Change resolution

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;
```

## 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/6c9a5118vodcq15
00005830/cc1e28208602268011087336518/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 start
Index: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**.

### Background Modes

Modes ☒ Audio, AirPlay, and Picture in Picture  
☐ Location updates  
☐ Voice over IP

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.tipTitle = @"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
tipTitle	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

Last updated : 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

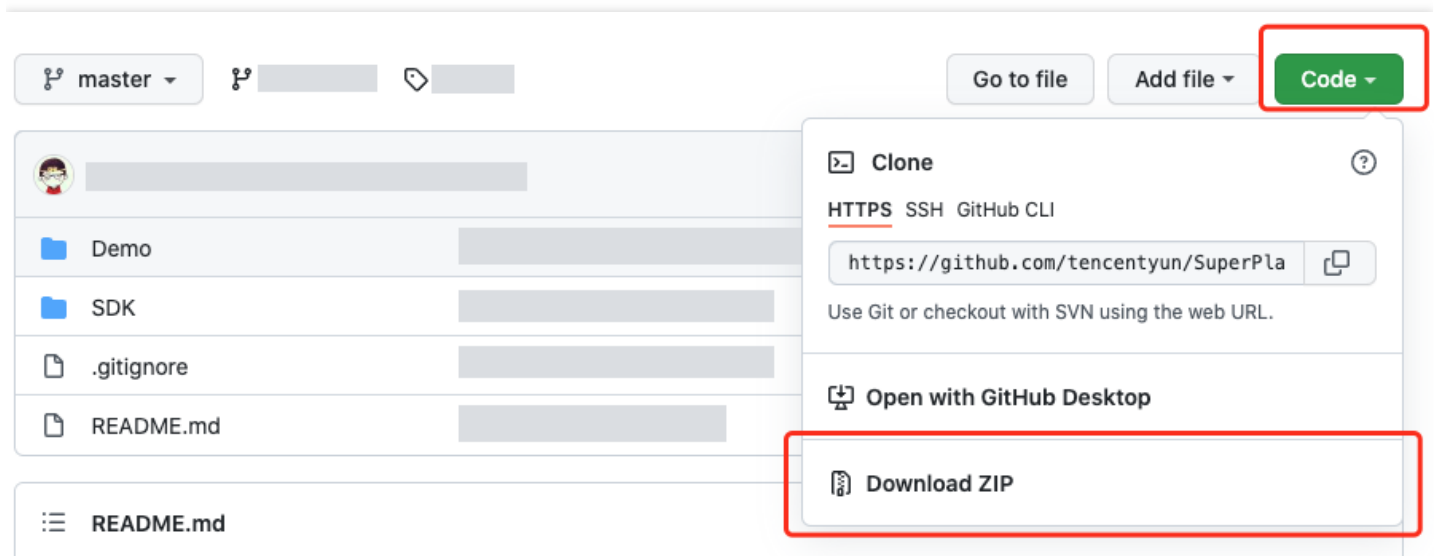
GitHub page: [LiteAVSDK/Player\\_Android](#)

You can download the Player for Android by [downloading the Player component ZIP package](#) or [running the Git clone command](#).

- Download the ZIP file

- Download using a Git command

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



## 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)
- Manual download in Gradle (AAR)
- SDK integration (jar + so)

1. Download the SDK + demo package for Android [here](#).

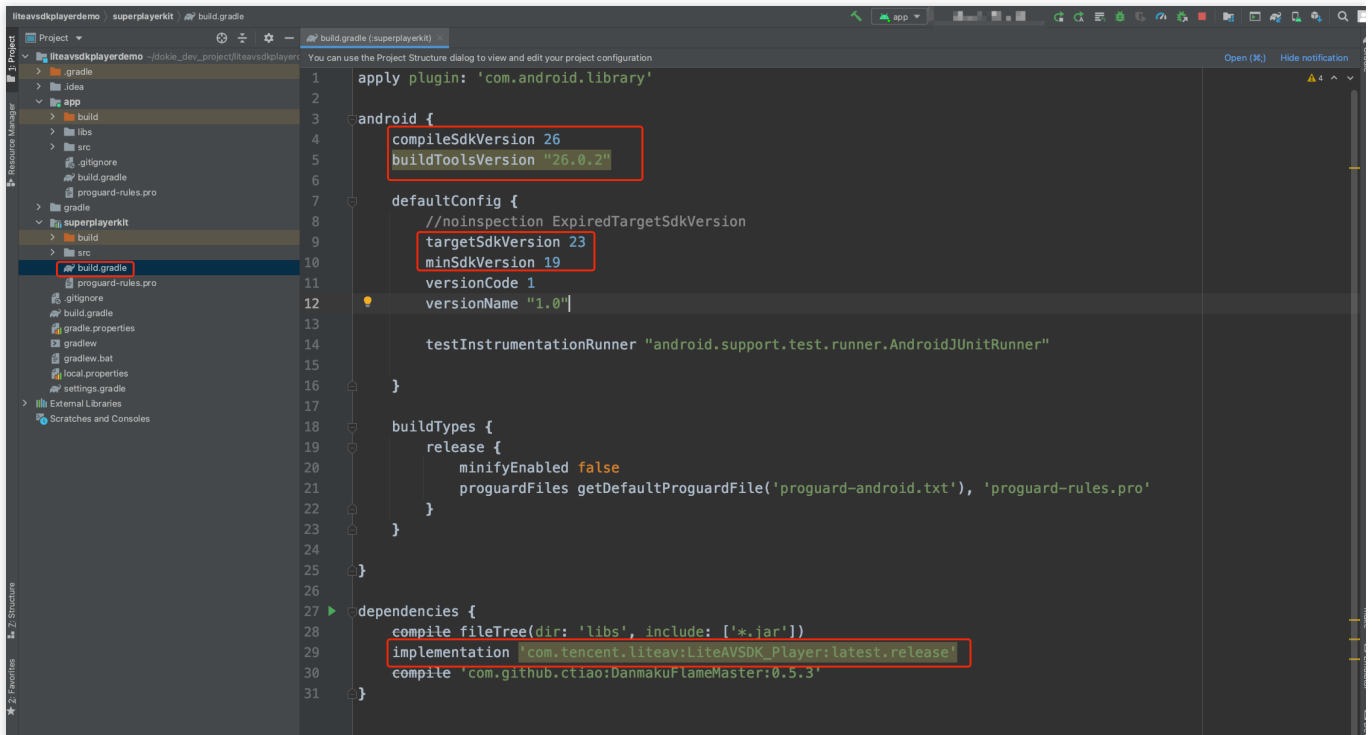
2. Copy the `Demo/superplayerkit` module to your project and then configure as follows:

- 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

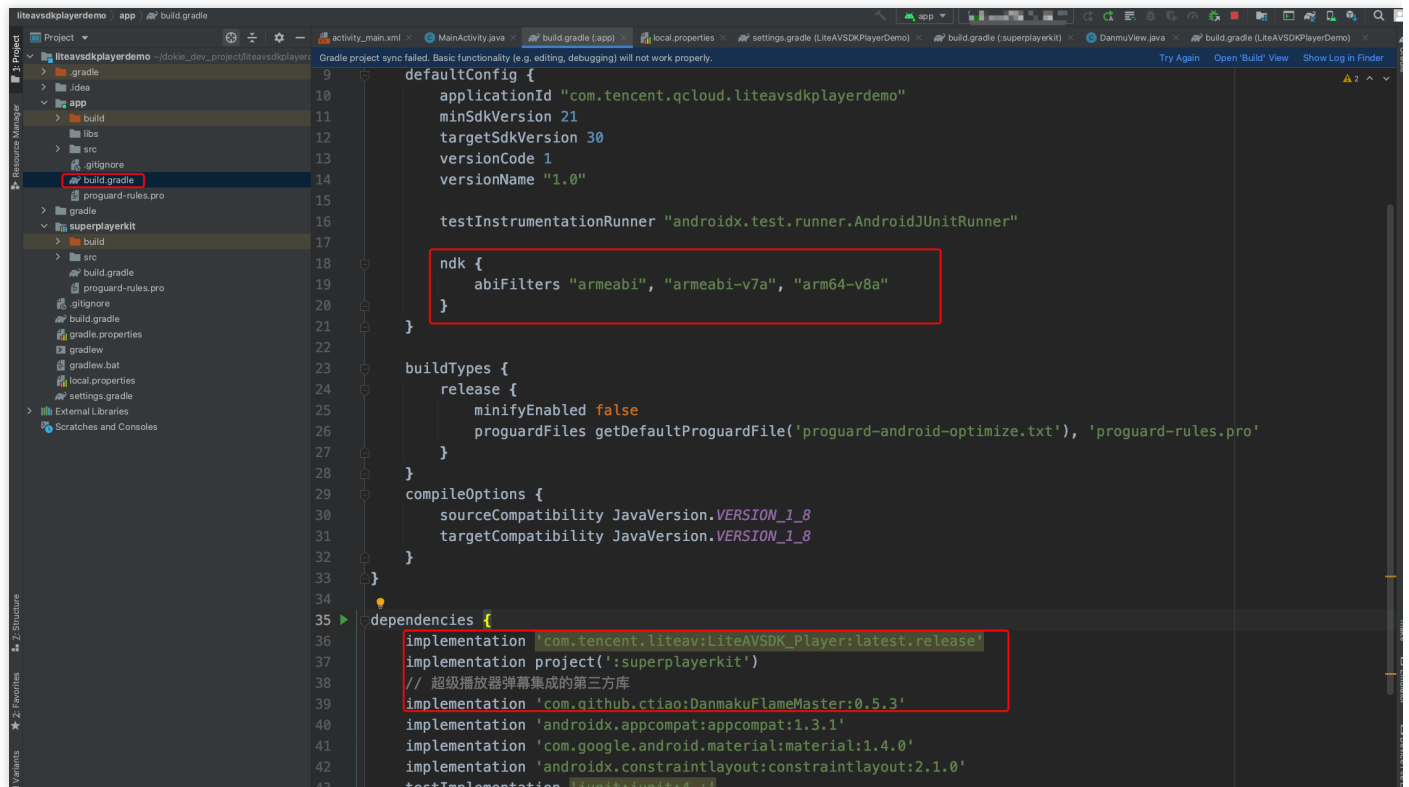
```
rootProject.ext.liteavSdk .
```



```
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:



- i. 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'
}

```

- ii. 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 `libijkhls-cache-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/libijkhls-cache-master.so"
    exclude "lib/armeabi-v7a/libijkhls-cache-master.so"
    exclude "lib/arm64-v8a/libijkhls-cache-master.so"
}
```

iii. 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.

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
  - VOD playback through `FileID`

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  
r playback  
mSuperPlayerView.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
- Screen lock
- On-screen comments
- Screenshot
- Change resolution

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

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

## 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/cc1e28208602268011087336518/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 <code>SuperPlayerModel</code> 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

Parameter	Type	Description
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

Last updated : 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	<b>Basic beauty filters</b> Brightening, skin smoothing, and blush	✓	✓	✓	✓	✓	✓
	<b>Image settings</b> Contrast, saturation, and sharpness	✓	✓	✓	✓	✓	✓
	<b>Basic beautification</b> Eye enlarging and face slimming (natural, attractive, and handsome)	✓	✓	✓	✓	✓	✓
	<b>Filters</b> 10 general filters by default	✓	✓	✓	✓	✓	✓
Extended	<b>Stickers</b> (10 2D general stickers for free)	-	✓	✓	✓	✓	✓
	<b>General beautification SDKs</b> (Face narrowing/Chin reshaping/Hairline adjustment/Nose narrowing)	-	-	✓	-	-	-
	<b>Gesture recognition</b> (One gesture sticker for free)	-	-	-	✓	-	-
	<b>Keying/Virtual background</b> (Three keying stickers for free)	-	-	-	-	✓	-

	Feature	Edition					
		A1 - 01	A1 - 02	A1 - 03	A1 - 04	A1 - 05	A1 - 06
	<b>Makeup</b> (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	<b>Basic beauty filters</b> Brightening, skin smoothing, and blush	✓	✓	✓	✓	✓
	<b>Image settings</b> Contrast, saturation, and sharpness	✓	✓	✓	✓	✓
	<b>Advanced beautification</b> 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	✓	✓	✓	✓	✓
	<b>Filters</b> (General filters by default)	✓	✓	✓	✓	✓



	Feature	Edition				
		S1 - 00	S1 - 01	S1 - 02	S1 - 03	S1 - 04
	<b>Stickers</b> (2D general stickers by default)	-	✓	✓	✓	✓
	<b>Advanced stickers</b> (3D general stickers by default)	-	✓	✓	✓	✓
	<b>Makeup</b> Full face makeup	-	✓	✓	✓	✓
Extended	<b>Gesture recognition</b> (One gesture sticker for free)	-	-	✓	-	✓
	<b>Keying/Virtual background</b> (Three keying stickers for free)	-	-	-	✓	✓
SDK Download	iOS & Android	-				

# SDK Integration Guide

## iOS

Last updated : 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 auth
result, 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 <code>URL</code> or <code>KEY</code> 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 <code>v_cube.license</code> file is empty, which may be caused by an I/O failure.
-6	The JSON field in the <code>v_cube.license</code> 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 <code>TELicense</code> 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 `YTPProcessInput` 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

Last updated : 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.E9.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()).setVideoProcessListener(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()).setVideoProcessListener(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

Last updated : 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

Last updated : 2021-09-15 17:15:24

## 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_DREAM, // Dark dream
    TXEffectType_SOUL_OUT, // Soul out
    TXEffectType_SCREEN_SPLIT, // Screen split
    TXEffectType_WINDOW_SHADOW, // Window blinds
    TXEffectType_GHOST_SHADOW, // Ghost shadow
    TXEffectType_PHANTOM, // Phantom
    TXEffectType_GHOST, // Ghost
    TXEffectType_LIGHTNING, // Lightning
    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(NSUInteger, TXSpeedLevel) {
    SPEED_LEVEL_SLOWEST, // Ultra-slow
    SPEED_LEVEL_SLOW, // Slow
    SPEED_LEVEL_NOMAL, // Normal
    SPEED_LEVEL_FAST, // Fast
    SPEED_LEVEL_FASTEST, // Ultra-fast
};
```

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

Last updated : 2020-09-01 14:52:06

## 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; // Filter 1
public static final int TXEffectType_SPLIT_SCREEN = 1; // Filter 2
public static final int TXEffectType_DARK_DRAEM = 2; // Filter 3
public static final int TXEffectType_ROCK_LIGHT = 3; // Filter 4
public static final int TXEffectType_WIN_SHADOW = 4; // Filter 5
public static final int TXEffectType_GHOST_SHADOW = 5; // Filter 6
public static final int TXEffectType_PHANTOM_SHADOW = 6; // Filter 7
public static final int TXEffectType_GHOST = 7; // Filter 8
public static final int TXEffectType_LIGHTNING = 8; // Filter 9
public static final int TXEffectType_MIRROR = 9; // Filter 10
public static final int TXEffectType_ILLUSION = 10; // Filter 11
```

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
mTXVideoEditor.startEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 1000);
mTXVideoEditor.stopEffect(TXVideoEditConstants.TXEffectType_SOUL_OUT, 2000);
// Use the second special effect filter between the third and fourth seconds
mTXVideoEditor.startEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 3000);
mTXVideoEditor.stopEffect(TXVideoEditConstants.TXEffectType_SPLIT_SCREEN, 4000);
// Delete the special effect filter set between the third and fourth seconds
mTXVideoEditor.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:
SPEED_LEVEL_SLOWEST - Ultra-slow
SPEED_LEVEL_SLOW - Slow
SPEED_LEVEL_NORMAL - Normal
SPEED_LEVEL_FAST - Fast
SPEED_LEVEL_FASTEST - Ultra-fast
```

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);

mTXVideoEditor.setSpeedList(list);
```

## Reverse Playback

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

### Note :

The legacy version of `setTXVideoReverseListener()` (below SDK 4.5) needs to be manually called for the first listening, while the new version can take effect without being called.

Demo:

```
mTXVideoEditor.setTXVideoReverseListener(mTxVideoReverseListener);
mTXVideoEditor.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 th
rice.
repeatList.add(repeat); // Currently, only one video segment can be looped.
mTXVideoEditor.setRepeatPlay(repeatList);
```

# Stickers and Subtitles

## iOS

Last updated : 2020-09-01 14:52:07

### 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];
    }
}
[_ugcEditor setAnimatedPasterList:animatePasters];
[_ugcEditor setPasterList:staticPasters];
}

```

## Custom Animated Sticker

Adding an animated sticker is actually to arrange **a series of images** in **a certain sequence** at **a certain interval** and insert them to the video to implement an animated sticker effect.

How do I customize a sticker?

Here, an animated sticker in the SDK demo is used as an example:

```

{
    "name": "glass", // Sticker name
    "count": 6, // Number of stickers
    "period": 480, // Playback period: time taken for playing back the sticker once in
    ms
    "width": 444, // Sticker width
    "height": 256, // Sticker height
    "keyframe": 6, // Key image, which can represent the animated sticker
    "frameArray": [ // Set of all images
        {"picture": "glass0"},
        {"picture": "glass1"},

```

```
{ "picture": "glass2" },
{ "picture": "glass3" },
{ "picture": "glass4" },
{ "picture": "glass5" }
]
```

The SDK will get the corresponding `config.json` of the animated sticker and display it in the format defined in the `.json` file.

### Note :

As this encapsulation format is required by the SDK, please describe animated stickers strictly in this format.

## Adding Subtitles

### 1. Bubble 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
}

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

## 2. How do I customize bubble subtitles?

### Parameters required by bubble subtitles

- Font area dimensions: top, left, right, bottom
- Default font size
- Width and height

#### Note :

The parameters above are all in px.

### Encapsulation format

As bubble subtitles contain many parameters, we recommend you encapsulate relevant parameters at the demo layer. For example, the Tencent Cloud demo uses the JSON format for encapsulation:

```
{
  "name": "boom", // Bubble subtitle name
```



```
"width": 376, // Width
"height": 335, // Height
"textTop":121, // Top margin of text area
"textLeft":66, // Left margin of text area
"textRight":69, // Right margin of text area
"textBottom":123, // Bottom margin of text area
"textSize":40 // Font size
}
```

### **Note :**

You can determine the encapsulation format by yourself, which is optional for the SDK.

## **Overlong subtitle**

### **If a subtitle is too long, how do I arrange the layout to make the subtitle neatly fit in the bubble?**

An automatic layout control is provided in the demo. If a subtitle is too long, the control will automatically decrease the font size until the entire subtitle can fit in the bubble.

You can also modify the relevant control source code to meet your unique business needs.

# Android

Last updated : 2020-09-01 14:52:08

## 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.getOperationVi
```

```
ew(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 TXVideoEditConstant
s.TXAnimatedPaster();

txAnimatedPaster.animatedPasterPathFolder = mAnimatedPasterSDcardFolder + view.ge
tPasterName() + 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 : " + txPaste
r.startTime + ", " + txPaster.endTime);
}
}

mTXVideoEditor.setAnimatedPasterList(animatedPasterList); // Set an animated stic
ker
mTXVideoEditor.setPasterList(pasterList); // Set a static sticker
```

## Custom Animated Sticker

Adding an animated sticker is actually to arrange **a series of images** in **a certain sequence** at **a certain interval** and insert them to the video to implement an animated sticker effect.

### Encapsulation format

Here, an animated sticker in the demo is used as an example:

```
{
  "name": "glass", // Sticker name
  "count": 6, // Number of stickers
  "period": 480, // Playback period: time taken for playing back the sticker once in ms
  "width": 444, // Sticker width
  "height": 256, // Sticker height
  "keyframe": 6, // Key image, which can represent the animated sticker
  "frameArray": [ // Set of all images
    {"picture": "glass0"},
    {"picture": "glass1"},
    {"picture": "glass2"},
    {"picture": "glass3"},
    {"picture": "glass4"},
    {"picture": "glass5"}
  ]
}
```

The SDK will get the corresponding `config.json` of the animated sticker and display it in the format defined in the .json file.

#### Note :

As this encapsulation format is required by the SDK, please describe animated stickers strictly in this format.

## Adding Subtitles

### 1. Bubble 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);
}
mTXVideoEditor.setSubtitleList(mSubtitleList); // Set the subtitle list
```

## 2. How do I customize bubble subtitles?

## Parameters required by bubble subtitles

- Font area dimensions: top, left, right, bottom
- Default font size
- Width and height

### Note :

The parameters above are all in px.

## Encapsulation format

As bubble subtitles contain many parameters, we recommend you encapsulate relevant parameters at the demo layer. For example, the Tencent Cloud demo uses the JSON format for encapsulation:

```
{
  "name": "boom", // Bubble subtitle name
  "width": 376, // Width
  "height": 335, // Height
  "textTop": 121, // Top margin of text area
  "textLeft": 66, // Left margin of text area
  "textRight": 69, // Right margin of text area
  "textBottom": 123, // Bottom margin of text area
  "textSize": 40 // Font size
}
```

### Note :

You can determine the encapsulation format by yourself, which is optional for the SDK.

## Overlong subtitle

### If a subtitle is too long, how do I arrange the layout to make the subtitle neatly fit in the bubble?

An automatic layout control is provided in the demo. If a subtitle is too long, the control will automatically decrease the font size until the entire subtitle can fit in the bubble.

You can also modify the relevant control source code to meet your unique business needs.

# Video Karaoke

## iOS

Last updated : 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.

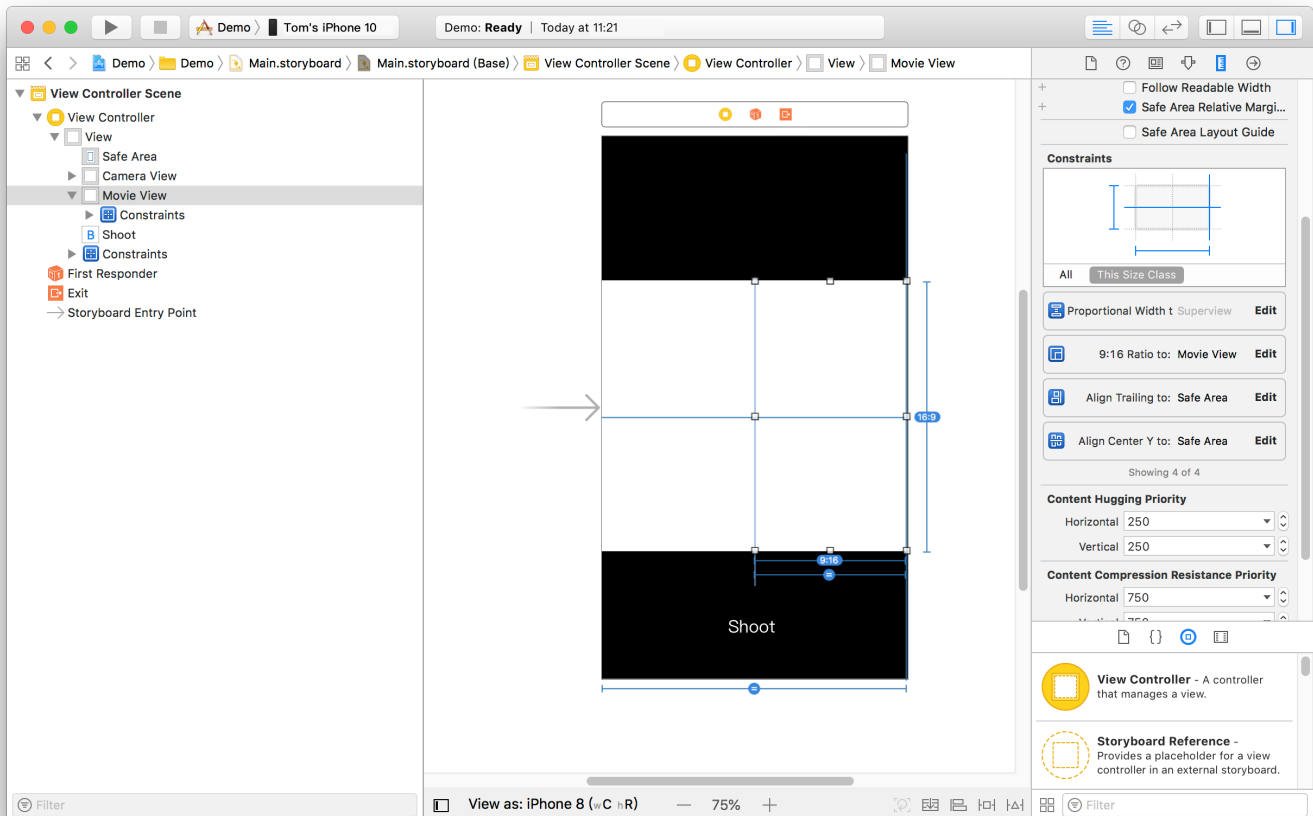
## UI Construction

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"

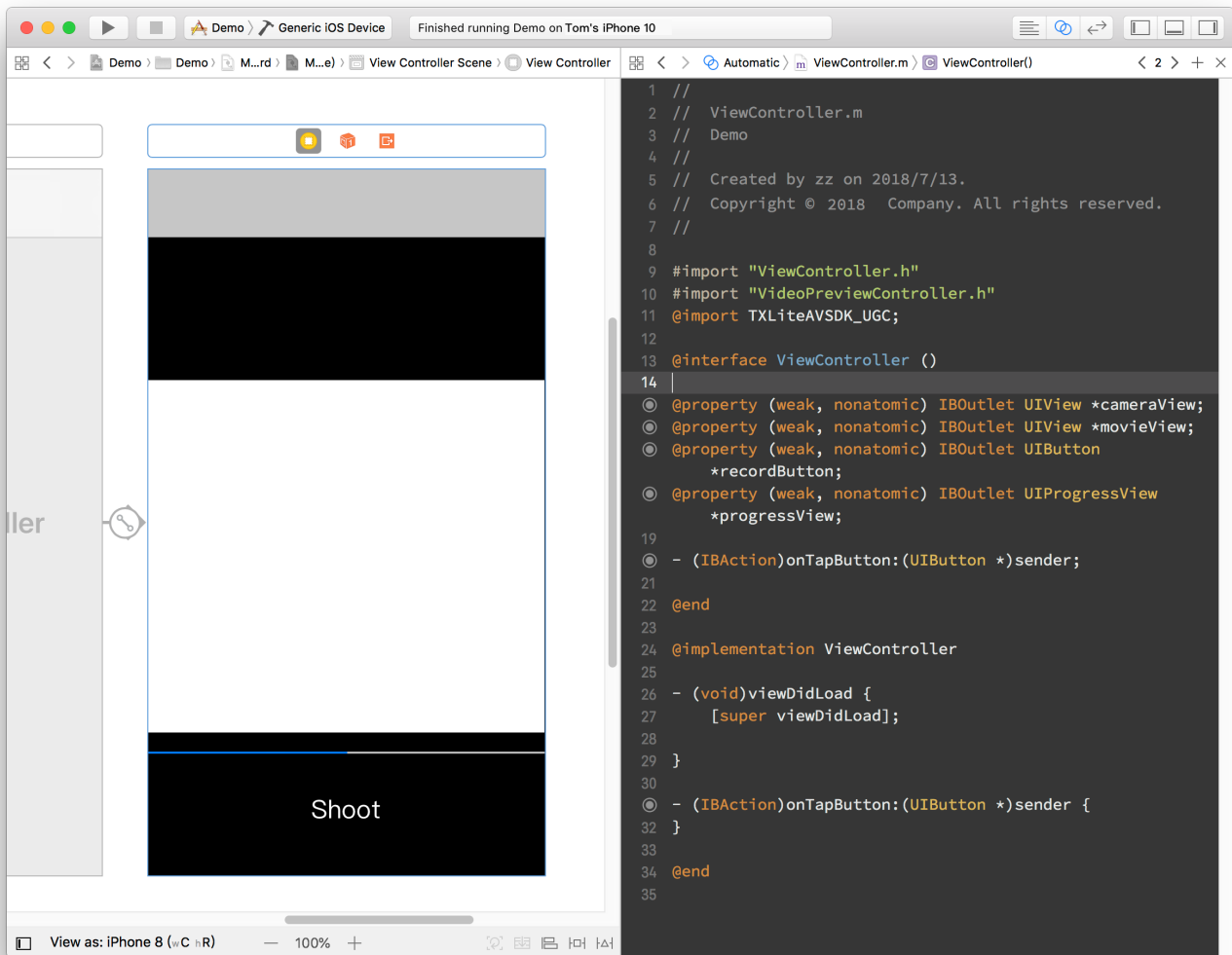
Configure a simple shoot page. Open `Main.storyboard` , drag two `UIView` objects into it, configure their width to 0.5 time of the `superview` , and set their aspect ratio to 16:9.



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 `TXVideoEditor`, `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 `ViewController.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>
{
    TXVideoEditor *_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 = [[TXVideoEditor alloc] initWithPreview:param];
[_editor setVideoPath:mp4Path];
_editor.previewDelegate = self;

// Initialize the shoot parameters
_recorder = [TXUGCRecord sharedInstance];
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 stringByAppending
String:@".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>
{
    TXVideoEditor *_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 = [[TXVideoEditor 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

Last updated : 2020-09-01 14:57:25

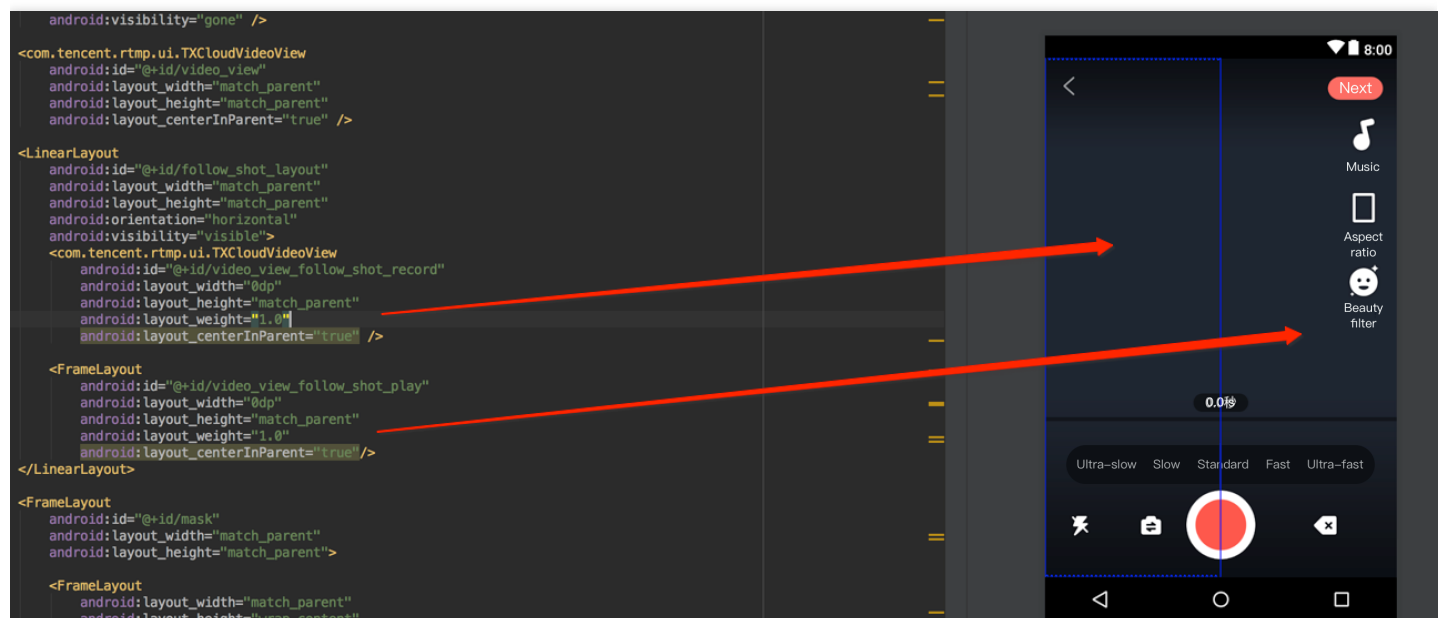
This document describes how to implement basic duet features.

## 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 `TXVideoEditor`, `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_EDITOR_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, `TXVideoEditor` is used. You can also use `TXVodPlayer`
mTXVideoEditor = new TXVideoEditor(this);
mTXVideoEditor.setVideoPath(mFollowShotVideoPath);
TXVideoEditConstants.TXPreviewParam param = new TXVideoEditConstants.TXPreviewP
```



```
aram();
param.videoView = mVideoViewPlay;
param.renderMode = TXVideoEditConstants.PREVIEW_RENDER_MODE_FILL_EDGE;
mTXVideoEditor.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_ADJUST_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 video's 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.TXAbsolute
Rect();
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_540P, 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;
startEditorPreview(mFollowShotVideoOutputPath);
if(mTXVideoEditor != null){
mTXVideoEditor.release();
mTXVideoEditor = 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

Last updated : 2020-09-01 14:52:09

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:

```
/*
 *pictureList: 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 dem
 o); 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 im
 ages is greater than or equal to 3, and the frame rate (`fps`) is normal
 */
- (int)setPictureList:(NSArray<UIImage *> *)pictureList fps:(int)fps;

/*
 *transitionType: transition type. For more information, please see `TXTransitionT
 ype`
 * 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(^)(C
GFloat))duration;
```

- 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

Last updated : 2020-09-01 14:52:09

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 dem
 o); 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)
```

- 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

Last updated : 2022-06-24 14:56:48

### 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 a
 lso be the texture type returned by the API. This callback follows the calling o
 f 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

Last updated : 2020-09-01 14:52:09

## 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 `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 face coordinates in Enterprise Edition  
     * @param points Normalized face coordinates. Every two coordinates indicate the `X` and `Y` values of a point. The value range is [0.f,1.f]  
     */  
    void onDetectFacePoints(float[] points);  
  
    /**  
     * 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 `GL_ES20.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

Last updated : 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

Last updated : 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

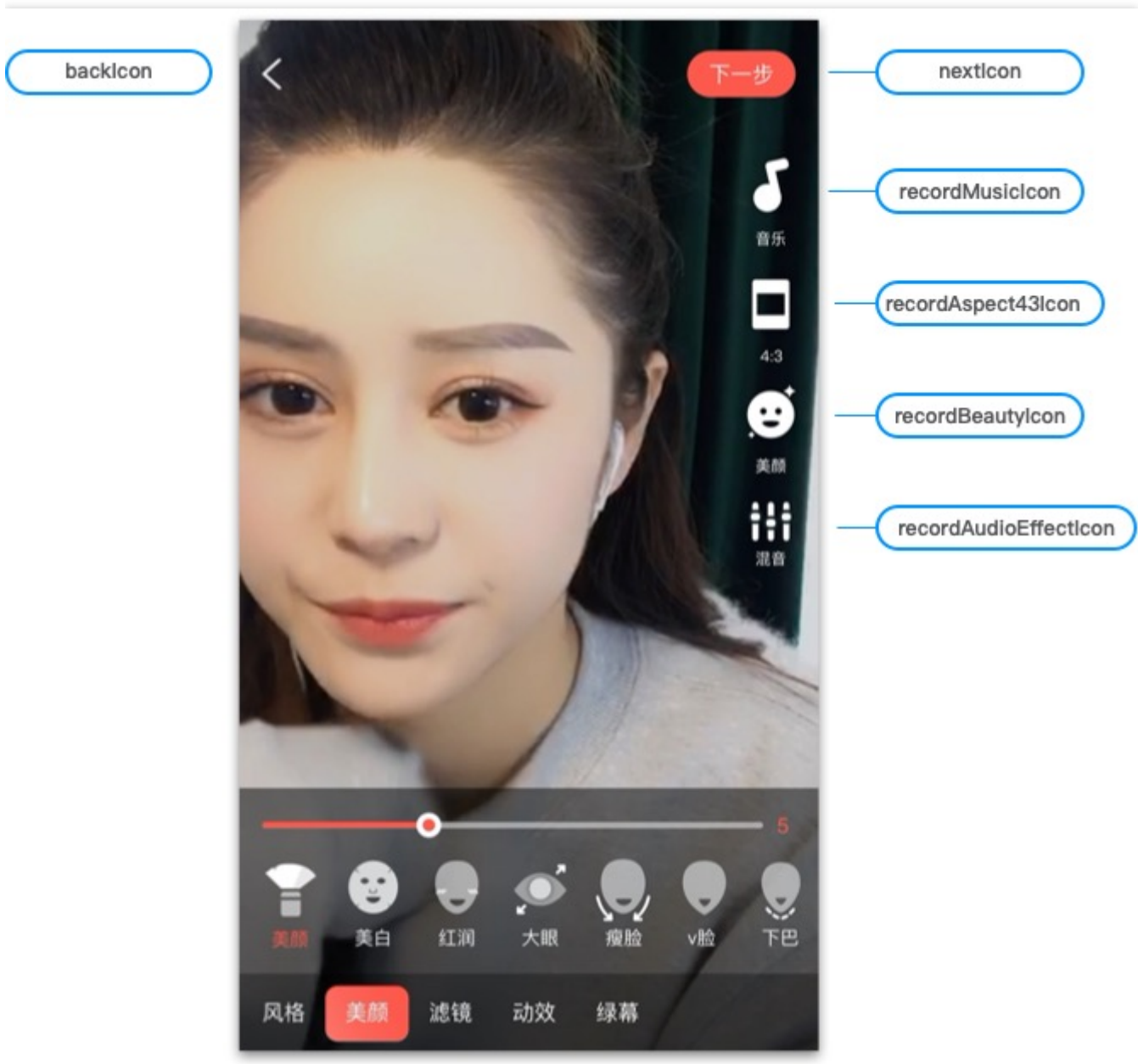
The methods of getting the colors on all UIs of `UGCKit` are defined in the `UGCKitTheme` class. You can modify the attribute values to change colors. For the specific resource names, see the comments in `UGCKitTheme.h` .

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 imageNamed:@"myConfirmIcon"]; // Set the icon used to complete recording
theme.recordMusicIcon = [UIImage imageNamed:@"myMusicIcon"]; // Set the icon of the music feature
theme.beautyPanelWhitnessIcon = [UIImage imageNamed:@"beauty_whitness"]; // Set t
```

*he icon of the brightening filter*

```
UGCKitRecordViewController *viewController = [[UGCKitRecordViewController alloc]  
initWithConfig:nil theme:theme]; // Use the custom theme to create a controller
```

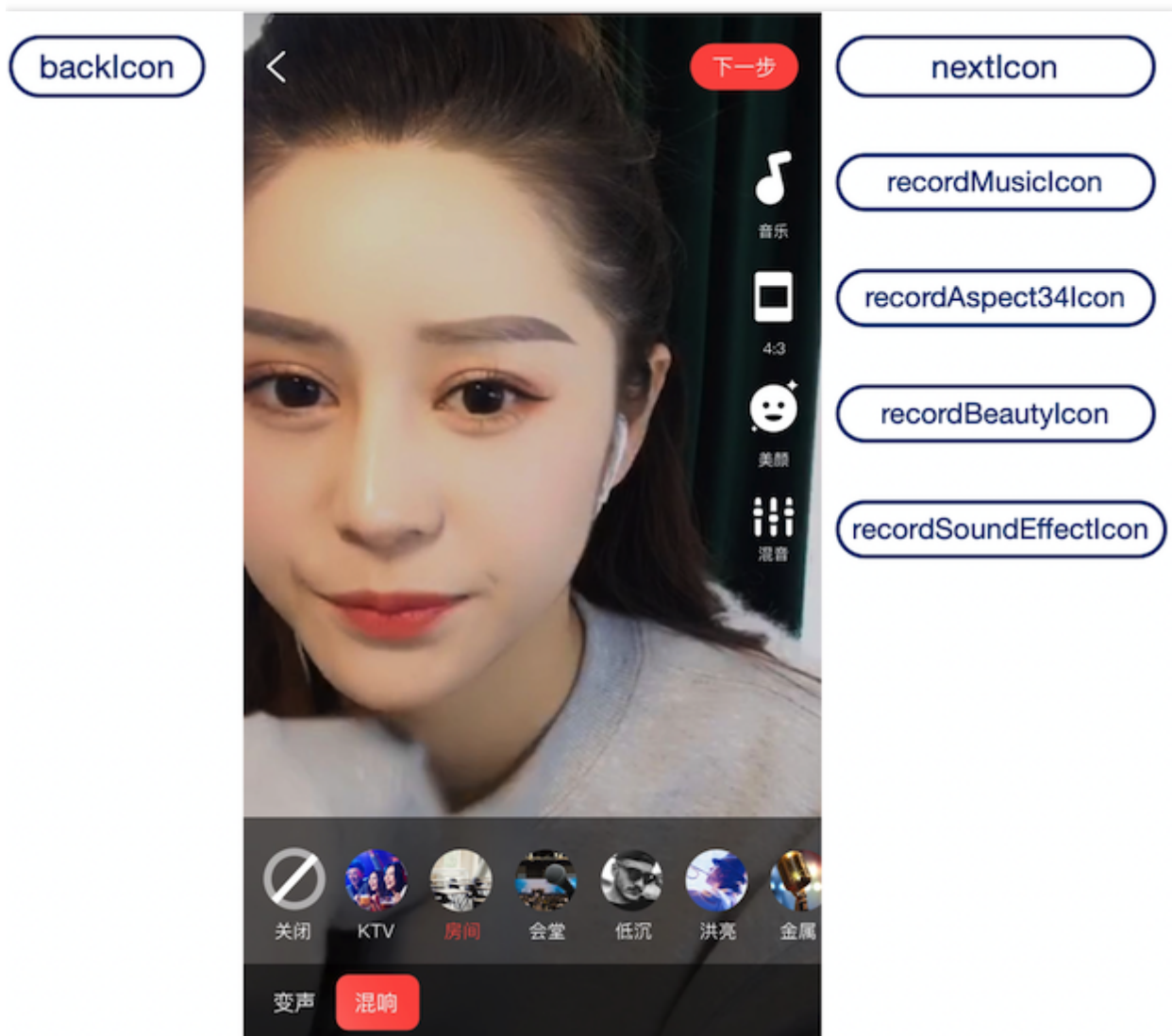
# Android

Last updated : 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



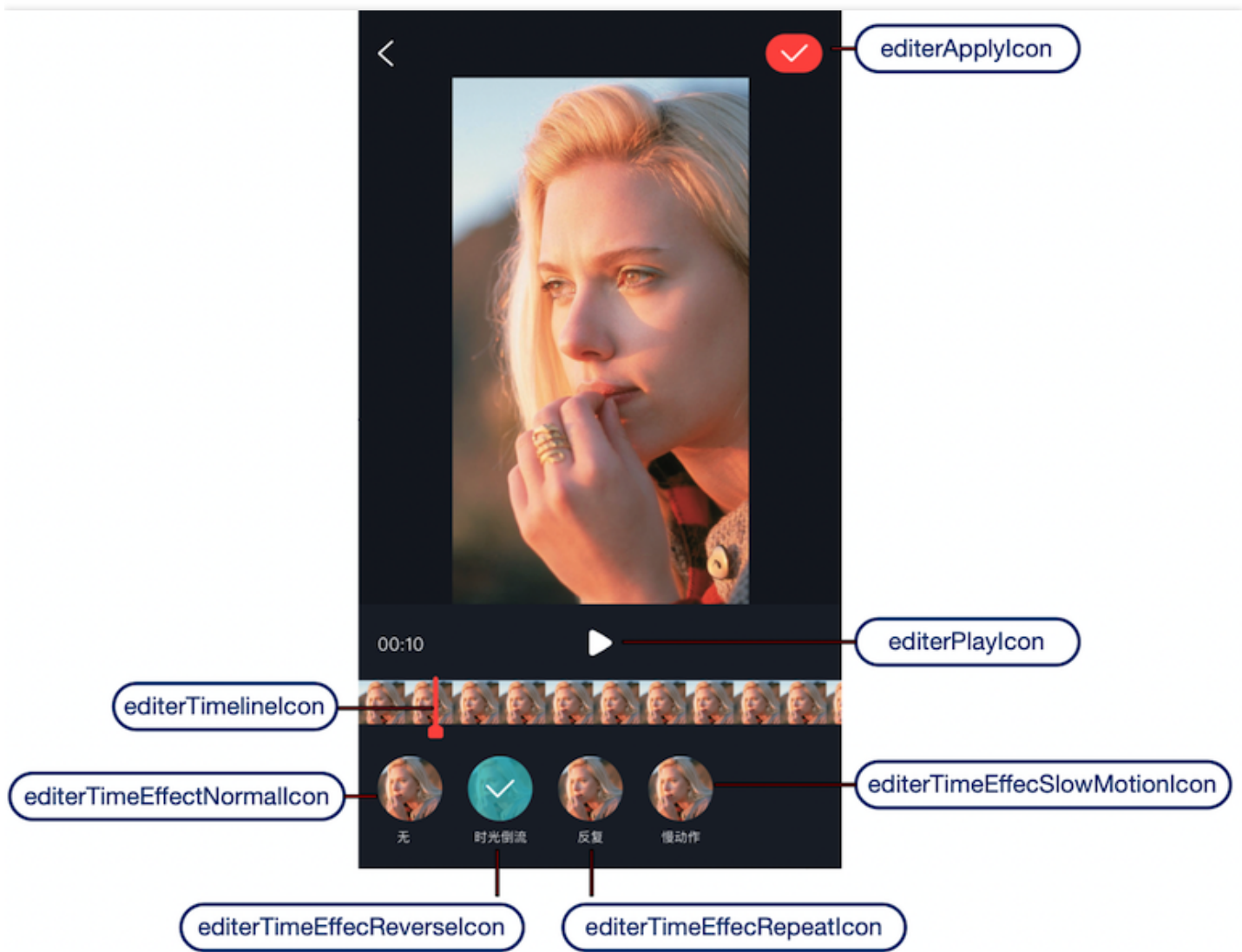
```
app/ugckit/res/values.theme_style.xml .
```

Below is the sample code for replacing the music and beauty filter icons on the recording UI:

2. Declare the custom theme in `AndroidManifest.xml` :

## 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 `EditorStyle` , 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 playback and pause icons on the editing UI:

2. Declare the custom theme in `AndroidManifest.xml` :