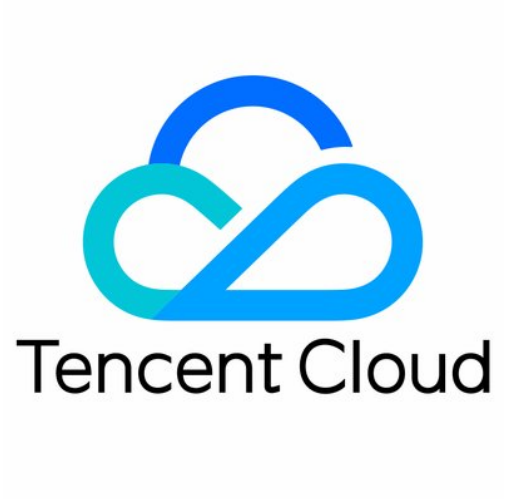
User Generated Short Video SDK

SDK Integration

Product Documentation



Contents

SDK Integration
SDK Integration (Xcode)
SDK Integration (Android Studio)

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

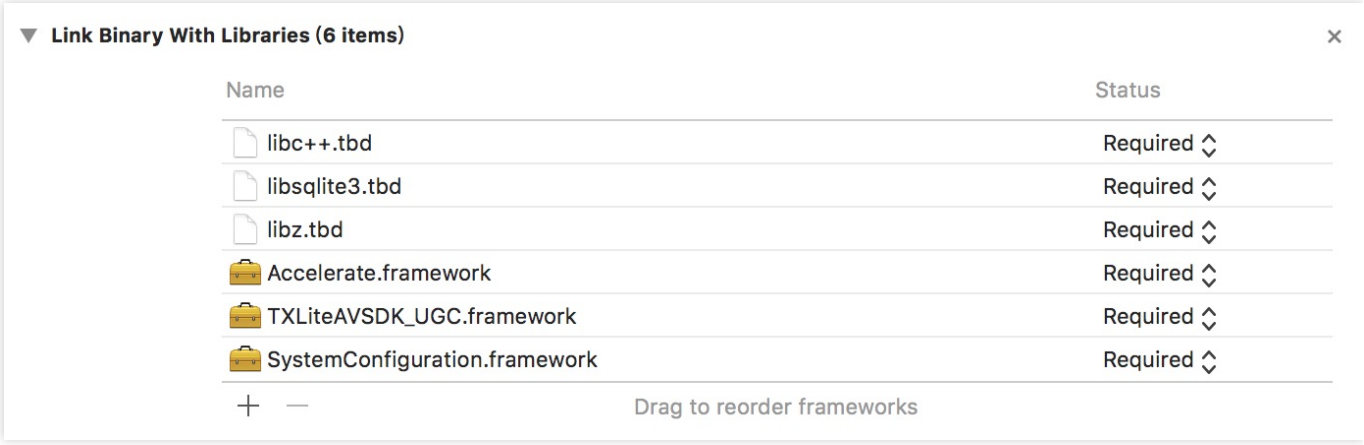
Directions

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:



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

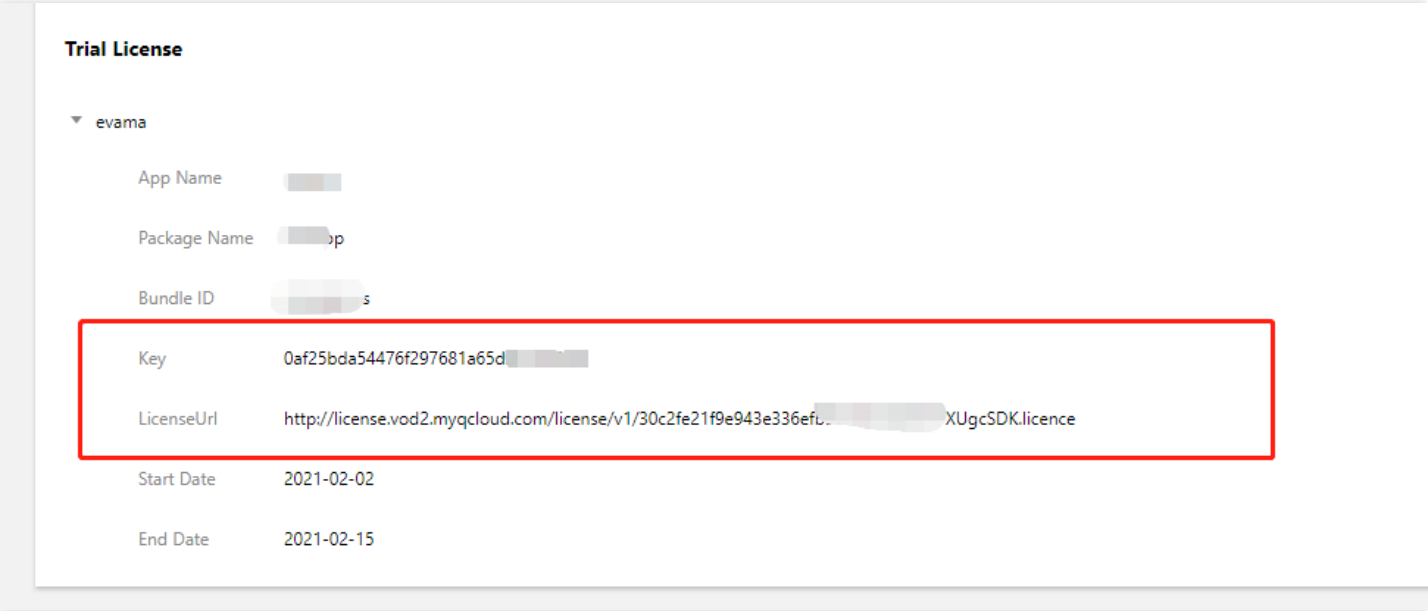
Step 2. Configure app permissions

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

Note：

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

Step 4. Configure logs

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

- **setConsoleEnabled**
  Sets whether to print the SDK output in the Xcode console.
- **setLogLevel**
  Sets whether to allow the SDK to print local logs. By default, the SDK writes logs to the **Documents/logs** folder of the current app.
  We recommend that you enable local log printing. You may need to provide log files if you run into a problem and need technical support.
- **Viewing log files**
  To reduce the storage space taken up by log files, the UGSV SDK encrypts local logs and limits their number. You need a log decompression tool to view the content of log files.

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

Step 5. Build and run the project

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

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

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
itWithMedia:media conifg: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 = [[UGCKitMediaPic
kerViewController alloc] initWithConfig:config theme:nil];
// Display the media picker view controller.
[self presentViewController:mediaPickerViewController animated:YES completio
n: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 process
ing.
[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] i
nitWithMedia:media theme:nil];
cutViewController.completion = ^(UGCKitResult *result) {
if (!result.cancelled && !result.error) {
[self editVideo:result.media];
} else {
[self.navigationController popViewControllerAnimated: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

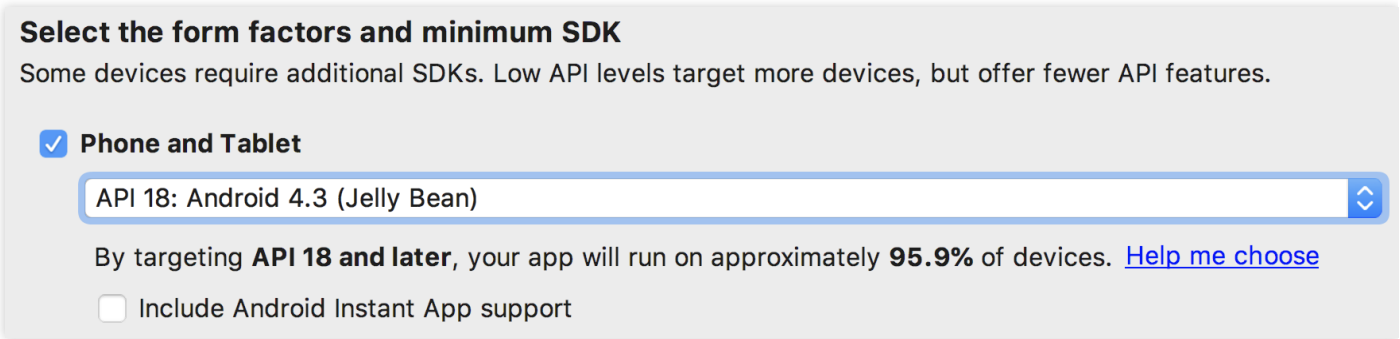
Below are the environment requirements for SDK development. You don't need to meet the same requirements for application development, but make sure that your application is compatible with the SDK.

- 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**



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` wi
th 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.

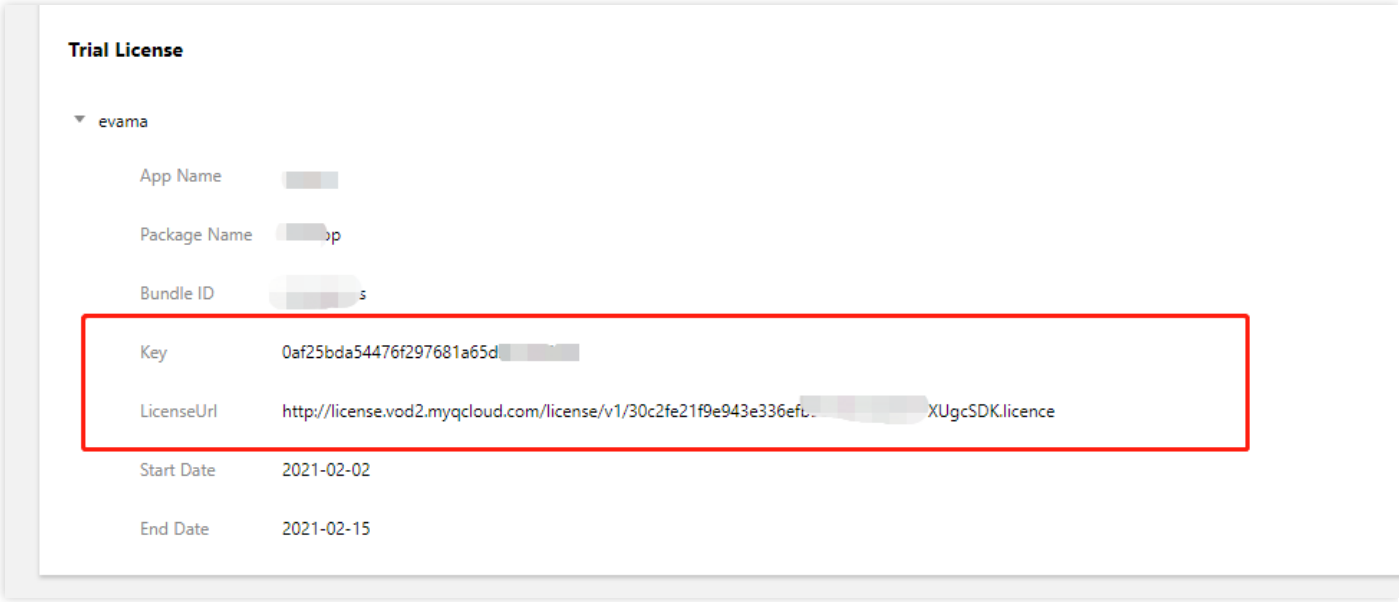
Step 2. Configure app permissions

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:



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 `getSDKVersioin` 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.

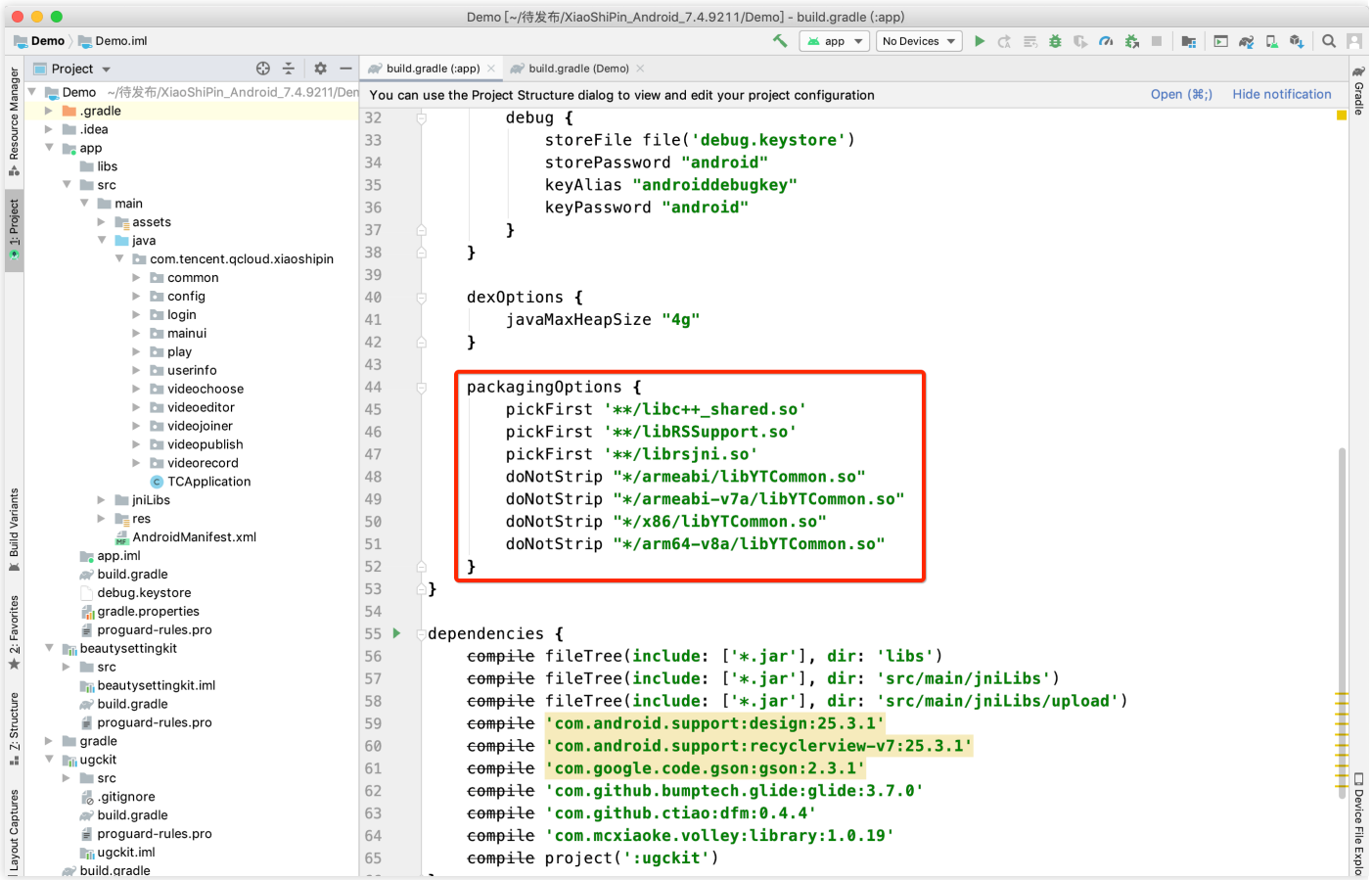
2. If you use the full edition integrated with the .aar file, check whether the .so libraries for the x64 architecture are filtered out in `defaultConfig` in `build.gradle` in the project directory.

```
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'), 'progu
ard-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.rqd: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`.

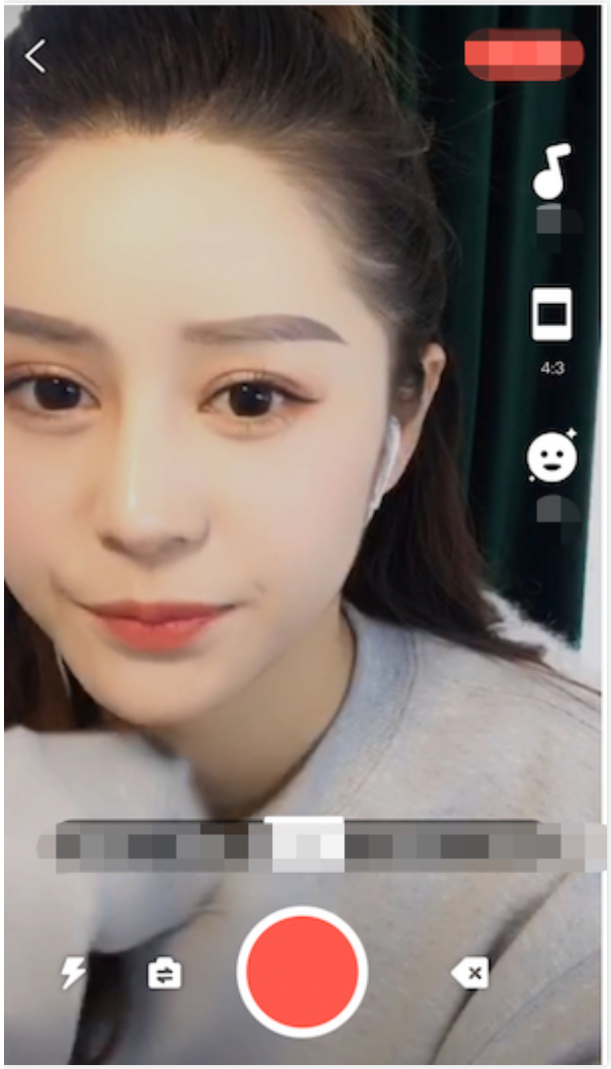
```
<style name="RecordActivityTheme" parent="UGCKitRecordStyle"/>
```

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 `AndroidManife
st` (android:theme).
setTheme(R.style.RecordActivityTheme);
setContentView(R.layout.activity_video_record);
// Get `UGCKitVideoRecord`
```

```
mUGCKitVideoRecord = (UGCKitVideoRecord) findViewById(R.id.video_record_layou
t);
// 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 de
tails, 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[] per
missions, @NonNull int[] grantResults) {
if (grantResults != null && grantResults[0] == PackageManager.PERMISSION_GRAN
TED) {
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" />
```

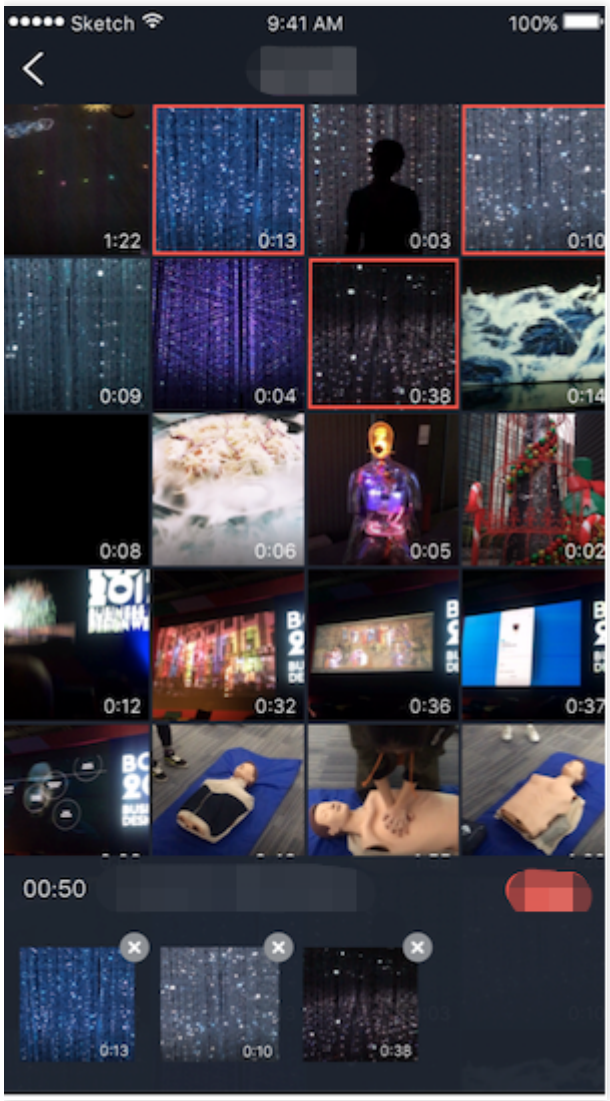
2. Create an empty theme in `res/values/styles.xml` and inherit the default video importing theme of `UGCKit`.

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

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

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
// 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`.

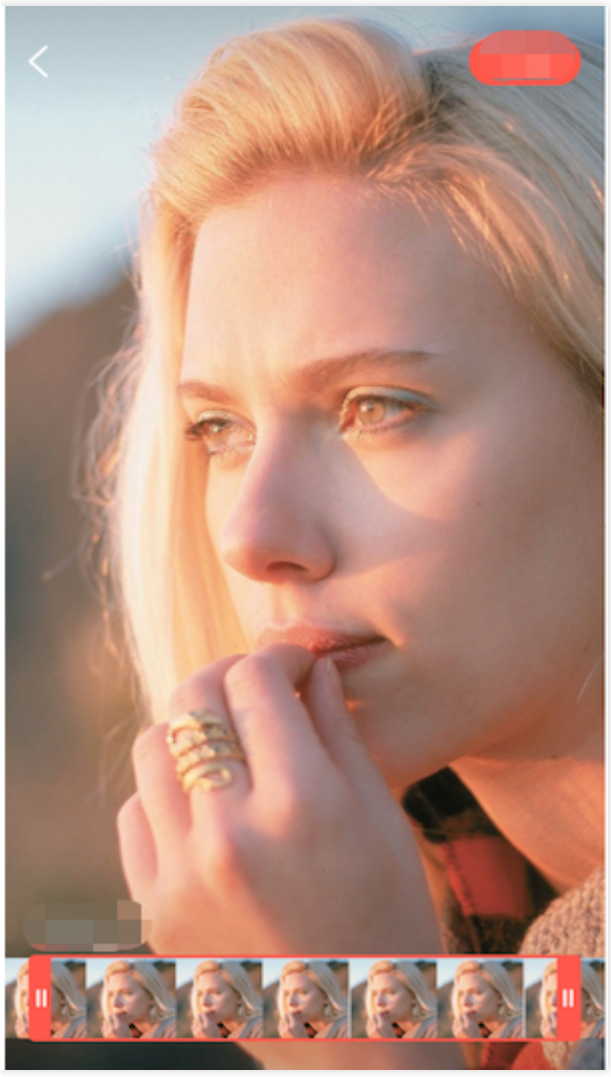
```
<style name="EditerActivityTheme" parent="UGCKitEditerStyle"/>
```

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.EditerActivityTheme);
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.EditerActivityTheme);
setContentView(R.layout.activity_video_editer);
// 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.

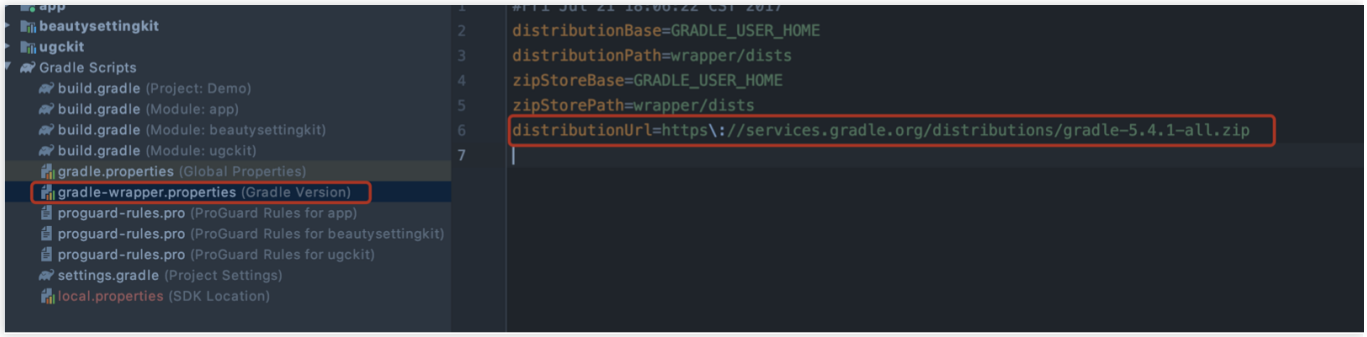
- Capturing and Shooting
- Video Editing
- Video Splicing
- Video Upload
- Player SDK

FAQs

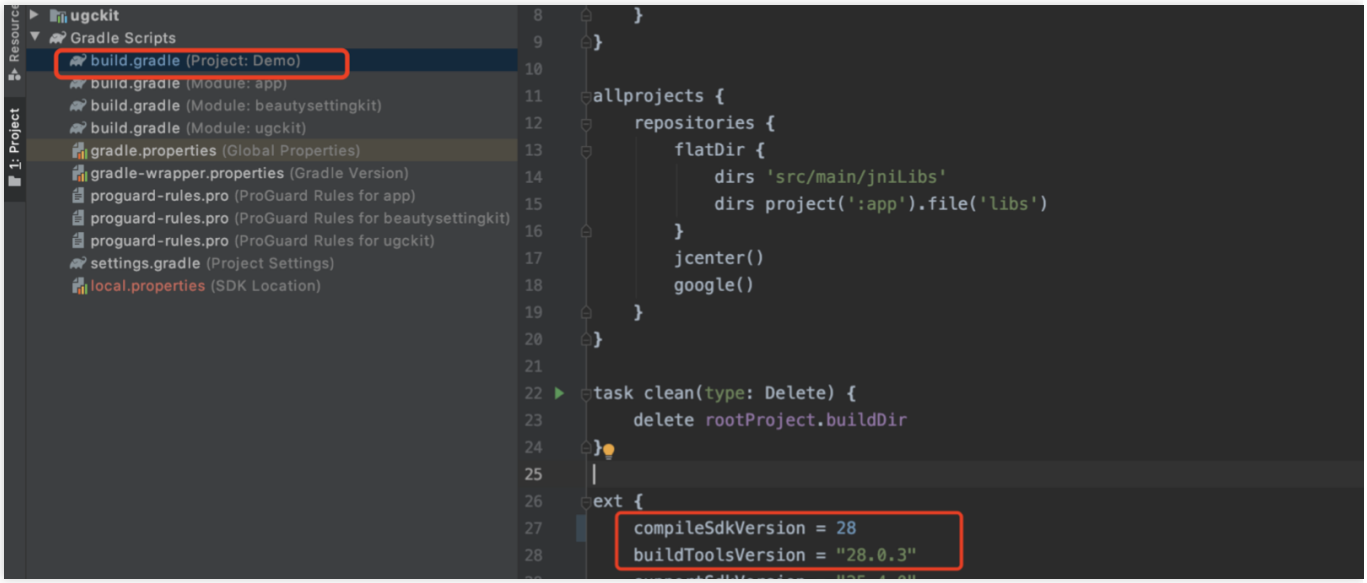
Can I use AndroidX?

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.

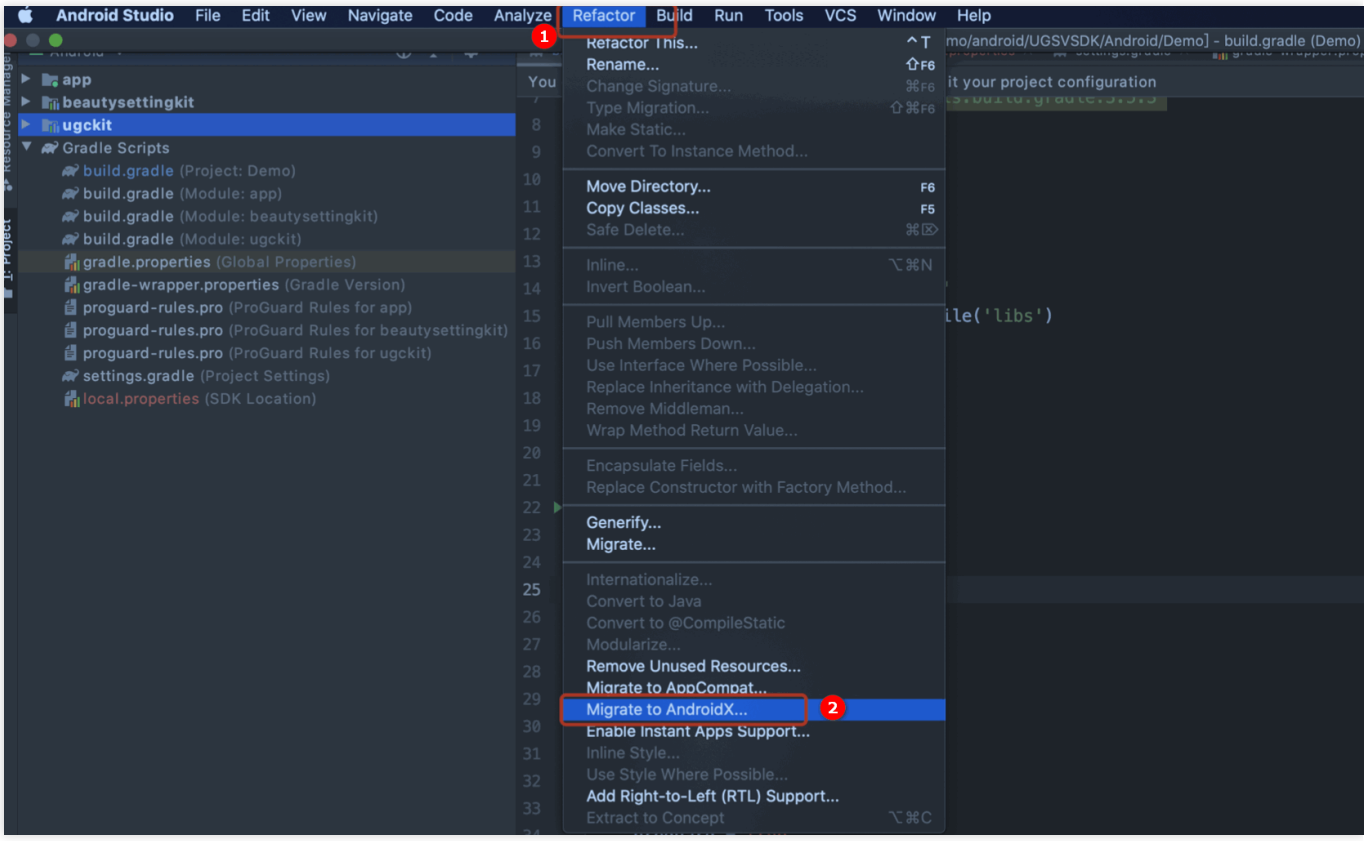


   - Update `compileSdkVersion` to 28 or later.
   - Update `buildToolsVersion` to 28.0.2 or later.

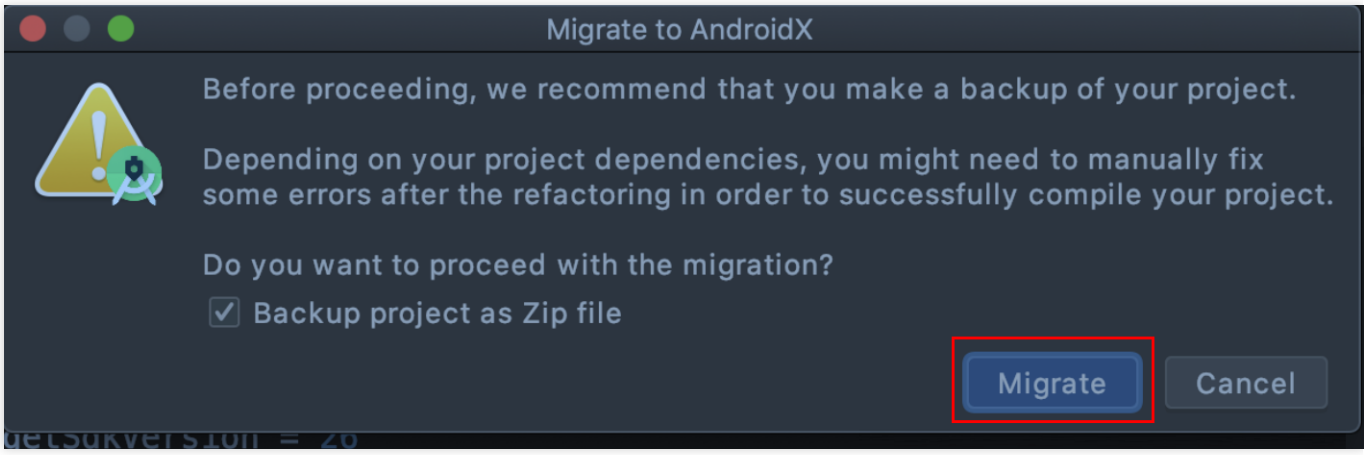


2. **Migrate to AndroidX:**

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**:

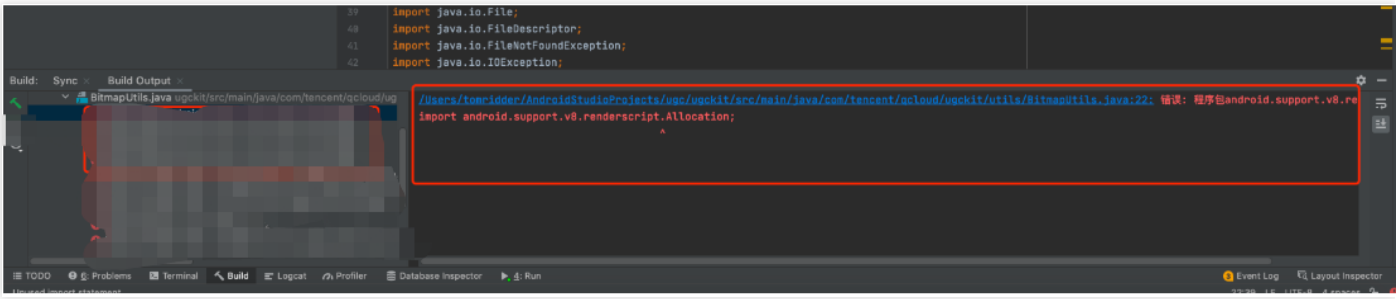
```
ERROR: Unable to find method 'org.gradle.api.tasks.compile.CompileOptions.setBo
otClasspath(Ljava/lang/String;)V'.
```

```
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.