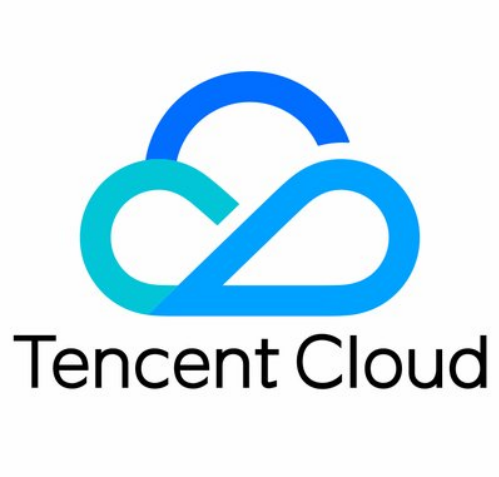
User Generated Short Video SDK

Single Feature Integration (iOS)

Product Documentation



Contents

Single Feature Integration (iOS)
SDK Integration (Xcode)

Last updated：2022-03-07 10:23:07

Supported Platforms

The SDK is supported on iOS 8.0 and above.

Environment Requirements

- Xcode 9 or above
- OS X 10.10 or above

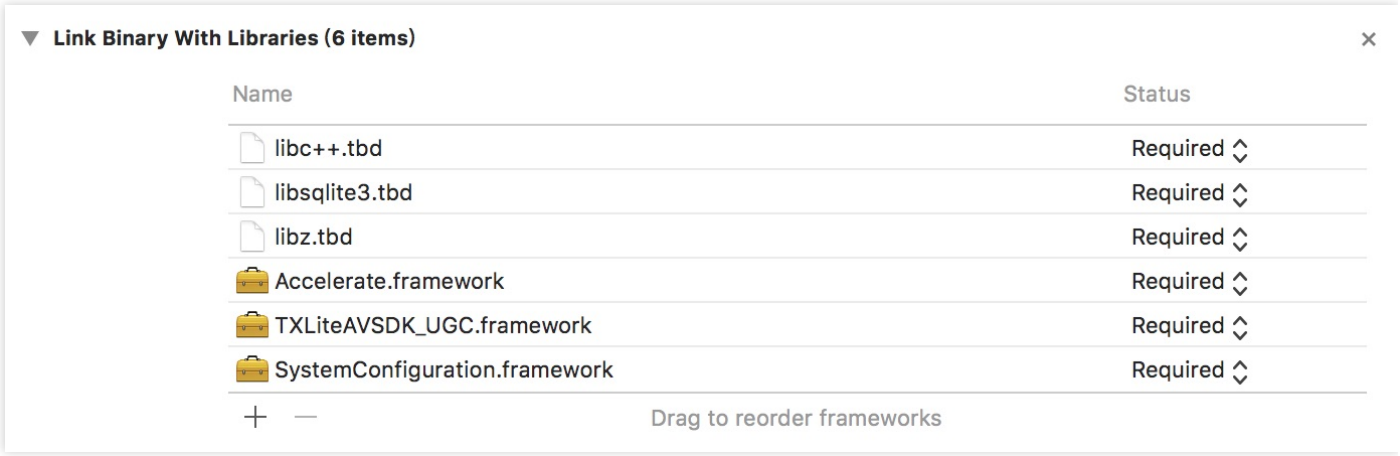
Directions

Step 1. Link the SDK and system libraries

1. Decompress the downloaded SDK package, copy framework files whose names start with `TXLiteAVSDK¥_` (e.g. TXLiteAVSDK_UGC.framework) in the SDK folder to your project folder, and drag them to the project.
2. Select your project’s target, and add the following system libraries.

- Accelerate.framework
- SystemConfiguration.framework
- libc++.tbd
- libsqlite3.tbd

  The library dependencies of the project after the addition are as shown below:



3. Select your project's target, search for `bitcode` in **Building Settings**, and set **Enable Bitcode** to **No**.

Step 2. Configure app permissions

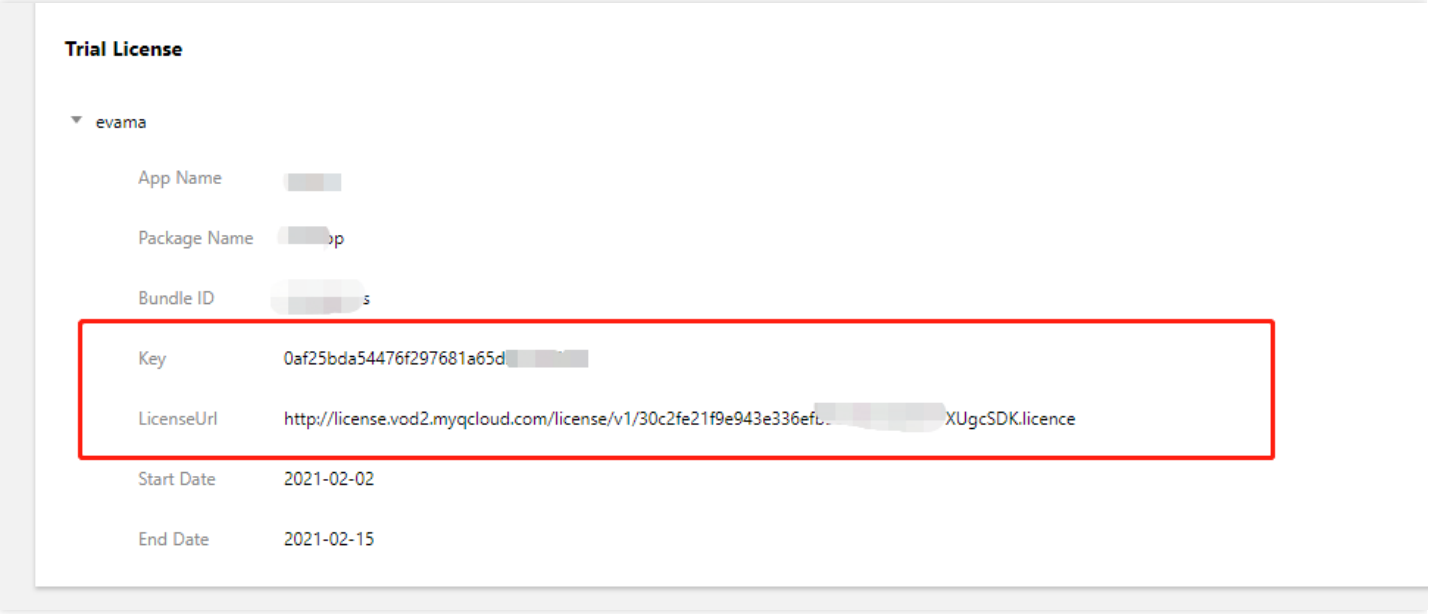
The app needs access to the photo album, which can be configured in `Info.plist`. Right-click `Info.plist`, select **Open as** > **Source Code**, and copy and modify the code below.

```
<key>NSAppleMusicUsageDescription</key>
<string>Video Cloud Toolkit needs to access your media library to obtain music files. It cannot a
dd music if you deny it access.</string>
<key>NSCameraUsageDescription</key>
<string>Video Cloud Toolkit needs to access your camera to be able to shoot videos with images.</
string>
<key>NSMicrophoneUsageDescription</key>
<string>Video Cloud Toolkit needs to access your mic to be able to shoot videos with audio.</stri
ng>
<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.



Before using UGSV features in your app, we recommend that you add the following code to `-[AppDelegate application:didFinishLaunchingWithOptions:]`.

```
@import TXLiteAVSDK_UGC;
@implementation AppDelegate
- (BOOL)application:(UIApplication*)applicationdidFinishLaunchingWithOptions:(NSDictinoary*)optio
ns {
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 **v4.7 license** and have updated 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 on v4.9 or above and should be configured as described above.

- For the Enterprise Edition, see Animated Effects and Face Changing.

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 Feature-based Integration

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.

Environment Requirements

- Xcode 10 or above
- iOS 9.0 or above

Step 1. Integrate UGCKit

1. **Configure the project**:

   i. 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 **BeautySettingKit** and **UGCKit** to the root directory of your project (the same directory as the Podfile).

   ii. Open the Podfile and add the following:

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

   iii. Run **pod install** and open `project name.xcworkspace`. You will find a `UGCKit BeautySettingKit` file in `Pods/Development Pods`.

2. **Import resources of the Enterprise Edition (required only if you use the Enterprise Edition)**:
   Drag `EnterprisePITU` (in `App/AppCommon` of the ZIP file of the Enterprise Edition SDK) to your project, click **Create groups**, select your target, and click **Finish**.

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] initWit
hConfig:nil theme:nil];
[self.navigationController pushViewController:recordViewController]
<dx-code-holder data-codeindex="5"></dx-code-holder>
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] initWithMedia:med
ia conifg:nil theme:nil];
// Display the editing view controller.
[self.navigationController pushViewController:editViewController animated:YES];
<dx-code-holder data-codeindex="6"></dx-code-holder>
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**

`UGCKitMediaPickerViewController` is used to select and splice images or videos. When multiple videos are selected, it returns a spliced video. Below is an example:

```
// 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];
<dx-code-holder data-codeindex="7"></dx-code-holder>
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:me
dia 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
- Animated effects and face changing (Enterprise)

Capturing and Shoot
Capturing and Shoot (iOS)

Last updated : 2020-08-27 11:18:57

Feature Overview

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

Overview of Used Classes

Tencent Cloud UGC SDK provides the following APIs to implement short video shoot as detailed below:

API File	Feature
`TXUGCRecord.h`	Short video shoot
`TXUGCRecordListener.h`	Callback for short video shoot
`TXUGCRecordEventDef.h`	Event callback for short video shoot
`TXUGCRecordTypeDef.h`	Basic parameter definition
`TXUGCPartsManager.h`	Video segment management class, which is used to shoot multiple video segments and delete existing segments

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.

Sample code

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

@implementation VideoRecordViewController
- (void)viewDidLoad {
[super viewDidLoad];

// Create a view to display the camera preview
_videoRecordView = [[UIView alloc] initWithFrame:self.view.bounds];
[self.view addSubview:_videoRecordView];

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

// 2. Start preview. Set the relevant parameters and the view where the preview is to be displaye
d
[[TXUGCRecord shareInstance] startCameraSimple:param preview:_videoRecordView];

// 3. Set the shoot effects. Watermarking is used here as an example
UIImage *watermarke = [UIImage imageNamed:@"watermarke"];
[[TXUGCRecord shareInstance] setWaterMark:watermarke normalizationFrame:CGRectMake(0.01, 0.01, 0.
1, 0)];
}

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

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

// Callback for shoot completion
```

```
-(void) onRecordComplete:(TXUGCRecordResult*)result
{
if (result.retCode == UGC_RECORD_RESULT_OK) {
// The shoot is successful, and the video file is in `result.videoPath`
} else {
// Process the error. For the error code definition, please see the definition of `TXUGCRecordRes
ultCode` in `TXUGCRecordTypeDef.h`.
}
}

-(void)alert:(NSString *)title msg:(NSString *)msg
{
UIAlertView *alert = [[UIAlertView alloc] initWithTitle:title message:msg delegate:self cancelBut
tonTitle:@"OK" otherButtonTitles:nil, nil];
[alert show];
}
@end
```

Previewing Video Image

`TXUGCRecord` (in `TXUGCRecord.h`) 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 *record = [TXUGCRecord sharedInstance];
record.recordDelegate = self; // Set the shoot callback. For the callback method, please see `TXU
GCRecordListener`

// Configure the camera 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 video shoot duration to 5s
param.maxDuration = 60; // Set the maximum video shoot duration to 60s
param.enableBFrame = YES; // Enable B frame to improve the image quality at the same bitrate

// Display the camera preview image in `self.previewView`
[recorder startCameraSimple:param preview:self.previewView];
```

```
// End video image preview
[[TXUGCRecord shareInstance] stopCameraPreview];
```

2. Adjust preview parameters

After starting the camera, you can modify the parameters as follows:

```
// Switch the video shoot resolution to 540p
[recorder setVideoResolution: VIDEO_RESOLUTION_540_960];

// Switch the video shoot bitrate to 6,500 Kbps
[recorder setVideoBitrate: 6500];

// Set the focal length to 3. 1 indicates the furthest view (normal lens), and 5 indicates the ne
arest view (enlarging lens)
[recorder setZoom: 3];

// Switch to the rear camera. YES: switches to front camera; NO: switches to rear camera
[recorder switchCamera: NO];

// Enable the flash. YES: enables; NO: disables
[recorder toggleTorch: YES];

// Set the callback for custom image processing
recorder.videoProcessDelegate = delegate;
```

Controlling Shoot Process

Starting, pausing, and resuming shoot

```
// Start shoot
[recorder startRecord];

// Start shoot. You can specify the addresses for the output video file and cover
[recorder startRecord:videoFilePath coverPath:coverPath];

// Start shoot. You can specify the addresses for the output video file, video segment storage, a
nd cover
[recorder startRecord:videoFilePath videoPartsFolder:videoPartFolder coverPath:coverPath];

// Pause shoot
[recorder pauseRecord];

// Resume shoot
```

```
[recorder resumeRecord];

// End shoot
[recorder stopRecord];
```

The shoot process and result will be called back through the `TXUGCRecordListener` protocol (defined in `TXUGCRecordListener.h`):

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

```
@optional
(void)onRecordProgress:(NSInteger)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:(TXUGCRecordResult*)result;
```

- `onRecordEvent` is an API reserved for shoot event callback and is not used currently.

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

Setting Shoot Attributes

1. Set the video image

```
// Set the landscape or portrait mode for shoot
[recorder setHomeOrientation:VIDOE_HOME_ORIENTATION_RIGHT];

// Set the video preview direction
// rotation: degree for the video preview image to rotate rightwards. Valid values: 0, 90, 180, 270 (other values are invalid)
// Note: it needs to be set before `startRecord` and will not take effect if set during shoot
[recorder setRenderRotation:rotation];
```

```
// Set the aspect ratio for shoot
// VIDEO_ASPECT_RATIO_9_16 The aspect ratio is 9:16
// VIDEO_ASPECT_RATIO_3_4 The aspect ratio is 3:4
// 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
[recorder setAspectRatio:VIDEO_ASPECT_RATIO_9_16];
```

2. Set the speed

```
// Set the video shoot speed
// VIDEO_RECORD_SPEED_SLOWEST, Ultra-slow
// VIDEO_RECORD_SPEED_SLOW, Slow
// VIDEO_RECORD_SPEED_NOMAL, Normal
// VIDEO_RECORD_SPEED_FAST, Fast
// VIDEO_RECORD_SPEED_FASTEST, Ultra-fast
[recorder setRecordSpeed:VIDEO_RECORD_SPEED_NOMAL];
```

3. Set the audio

```
// Set the mic volume level. It is used to adjust the mic volume level when background music is p
layed 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
[recorder setMicVolume:volume];

// Set whether the shoot is muted in the `isMute` parameter. The shoot is unmuted by default
[recorder setMute:isMute];
```

Capturing

```
// Photo capturing, which will take effect if called after `startCameraSimplePreview` or `startCa
meraCustomPreview`
[recorder snapshot:^(UIImage *image) {
// `image` is the capturing result
}];
```

Setting Effects

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

1. Watermark

```
// Set a global watermark
// normalizationFrame: normalized value of watermark relative to video image. The SDK will automa
tically calculate the `height` according to the watermark aspect ratio
// Suppose the video image dimensions are (540, 960), and `frame` is set to (0.1, 0.1, 0.1, 0)
// Then, the actual pixel coordinates of the watermark are:
// (540*0.1, 960*0.1, 540*0.1, 540*0.1*waterMarkImage.size.height / waterMarkImage.size.width)
[recorder setWaterMark:waterMarkImage normalizationFrame:frame)
```

2. Filter

```
// Set the style filter
// Set the color filter to Romantic, Fresh, Aesthetic, Rosy, Vintage, etc.
// filterImage: color lookup table used for filter. Note: it must be in .png format
// The filter color lookup table used by the demo is in `FilterResource.bundle`
[recorder setFilter:filterImage];

// It is used to set the filter effect level. Value range: 0-1. The greater the value, the more o
bvious the effect. Default value: 0.5
[recorder setSpecialRatio:ratio];

// 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. For more informati
on, please see the demo
[recorder setFilter:leftFilterImgage leftIntensity:leftIntensity rightFilter:rightFilterImgage ri
ghtIntensity:rightIntensity leftRatio:leftRatio];
```

3. Beauty filter

```
// Set the levels of the beauty, brightening, and rosy skin filters and the beauty filter style
// `beautyStyle` is defined as follows:
// typedef NS_ENUM(NSInteger, TXVideoBeautyStyle) {
// VIDOE_BEAUTY_STYLE_SMOOTH = 0, // Smooth
// VIDOE_BEAUTY_STYLE_NATURE = 1, // Natural
// VIDOE_BEAUTY_STYLE_PITU = 2, // Pitu beauty filter, which can be used only in Enterprise Editi
on
// };
// Value range: 0-9; 0 indicates that the filter is disabled, and the greater the value, the more
obvious the effect
```

```
[recorder setBeautyStyle:beautyStyle beautyLevel:beautyLevel whitenessLevel:whitenessLevel ruddin
essLevel:ruddinessLevel];
```

Advanced Features

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

Multi-Segment Shoot (iOS)

Last updated : 2020-08-27 11:18:58

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 shareInstance];
[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 a
nd 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];
```

Shoot Drafts (iOS)

Last updated : 2021-09-16 11:09:51

How the shooting drafts feature works

Starting a shooting

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

Resuming the shooting

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

```
//Get the object of the previous shooting
record = [TXUGCRecord shareInstance];
//Start shooting a video.
[record startRecord];
//Pause the shooting and cache the video segment
[record pauseRecord:^{
NSArray *videoPathList = record.partsManager.getVideoPathList;
//Set `videoPathList` to a local path.
}];
//Get the object of the resumed shooting.
record2 = [TXUGCRecord shareInstance];
//Preload the locally cached video segment.
[record2.partsManager insertPart:videoPath atIndex:0];
//Start the shooting
[record2 startRecord];
//End the shooting. The SDK will splice together the two video segments.
[record2 stopRecord];
```

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

Adding Background Music (iOS)

Last updated : 2020-08-27 11:18:59

Adding Background Music During Shooting

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

// Set the background music file path
[recorder setBGMAsset:path];

// Set the background music. If the music file is loaded from the system media library, you can d
irectly pass in the corresponding `AVAsset`
[recorder setBGMAsset:asset];

// Play back the background music
[recorder playBGMFromTime:beginTime
toTime:endTime
withBeginNotify:^(NSInteger errCode) {
// Callback for the start of playback. If `errCode` is 0, it is a success; otherwise, it is a fai
lure
} withProgressNotify:^(NSInteger progressMS, NSInteger durationMS) {
// progressMS: duration of the part that has been played back. durationMS: total duration
} andCompleteNotify:^(NSInteger errCode) {
// Callback for the end of playback. If `errCode` is 0, it is a success; otherwise, it is a failu
re
}];

// 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 background music is playe
d back
// volume: 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
[recorder setMicVolume:1.0];
```

```
// `setBGMVolume` is used to set the background music volume level. It controls the background music volume level during playback.
// volume: volume level. 1 indicates the normal volume level. The recommended value range is 0-2. If you want to increase the background music volume level, you can 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 = [[TXVideoEditer alloc] initWithPreview:param];

// Set the background music file path
[ugcEdit setBGMAsset: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 setBGMAtVideoTime: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.

Voice Changing and Reverb (iOS)

Last updated : 2020-08-27 11:18:59

Voice changing and reverb for video shooting:

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

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

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

Note :

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

Preview, Clipping, and Splicing Video Editing (iOS)

Last updated : 2021-03-05 15:19:29

Feature Overview

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

Overview of Relevant Classes

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

Use Instructions

The following is the basic usage process of video editing:

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

Sample

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

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

```
}

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

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

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

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

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

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

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

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

```
}
@end
```

Getting Video Information

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

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

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

The returned `TXVideoInfo` is defined as follows:

```
/// Video information
@interface TXVideoInfo : NSObject
/// Image of the first video frame
@property (nonatomic, strong) UIImage* coverImage;
/// Video duration in seconds
@property (nonatomic, assign) CGFloat duration;
/// Video size in bytes
@property (nonatomic, assign) unsigned long long fileSize;
/// Video frame rate in fps
@property (nonatomic, assign) float fps;
/// Video bitrate in Kbps
@property (nonatomic, assign) int bitrate;
/// Audio sample rate
@property (nonatomic, assign) int audioSampleRate;
/// Video width
@property (nonatomic, assign) int width;
/// Video height
@property (nonatomic, assign) int height;
/// Video image rotation angle
@property (nonatomic, assign) int angle;
@end
```

Getting Thumbnail

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

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

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

```
/** Get the list of thumbnails of the video
* @param count Number of the thumbnails to be obtained (at even sampling intervals)
* @param maxSize Maximum thumbnail dimensions. The dimensions of the generated thumbnails will no
t exceed the specified width and height.
* @param videoAsset Video file attributes
* @param sampleProcess Sampling progress
*/
+ (void)getSampleImages:(int)count
maxSize:(CGSize)maxSize
videoAsset:(AVAsset *)videoAsset
progress:(sampleProcess)sampleProcess;
```

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

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

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

Editing and Previewing

Video editing supports two effect preview modes: **pinpoint preview** (the video image is frozen at a specified point in time) and **range preview** (a video segment within a specified time range (A–B) is

looped). To use a preview mode, you need to bind a `UIView` to the SDK in order to display the video image.

1. Bind UIView

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

```
PREVIEW_RENDER_MODE_FILL_SCREEN - Fill mode, where the video image will cover the entire screen w
ith 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 w
ill exist if the aspect ratio of the video is different from that of the screen.
```

2. Use pinpoint preview

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

```
/** Render the video image at a specified point in time
* @param time Preview frame time in seconds
*/
- (void)previewAtTime:(CGFloat)time;
```

3. Use range preview

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

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

4. Pause and resume preview

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

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

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

5. Add a beauty filter

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

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

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

Demo:

```
TXVideoEditer *_ugcEdit;
NSString * path = [[NSBundle mainBundle] pathForResource:@"FilterResource" ofType:@"bundle"];
path = [path stringByAppendingPathComponent:@"langman.png"];
UIImage* image = [UIImage imageWithContentsOfFile:path];
[_ugcEdit setFilter:image];
```

6. Set a watermark

1. Set a global watermark

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

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

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

Demo:

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

2. Set a post-roll watermark

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

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

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

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

Compressing and Clipping

Setting video bitrate

```
/**
* Set the video bitrate
* @param bitrate Video bitrate in Kbps
* If the bitrate is set, it will be selected preferably when the SDK compresses videos. Please se
t an appropriate bitrate. If it is too low, the video image will be blurry; if it is too high, th
e 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 wi
ll automatically calculate the bitrate based on the compression quality
*/
- (void) setVideoBitrate:(int)bitrate;
```

Clipping video

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

```
TXVideoEditer* _ugcEdit = [[TXVideoEditer alloc] initWithPreview:param];
// Set the clipping start and end time
```

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

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

Advanced Features

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

Video Splicing (iOS)

Last updated : 2021-12-23 14:53:46

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 `QBImagePicker` 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 `TXUGCEditer`, 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` .

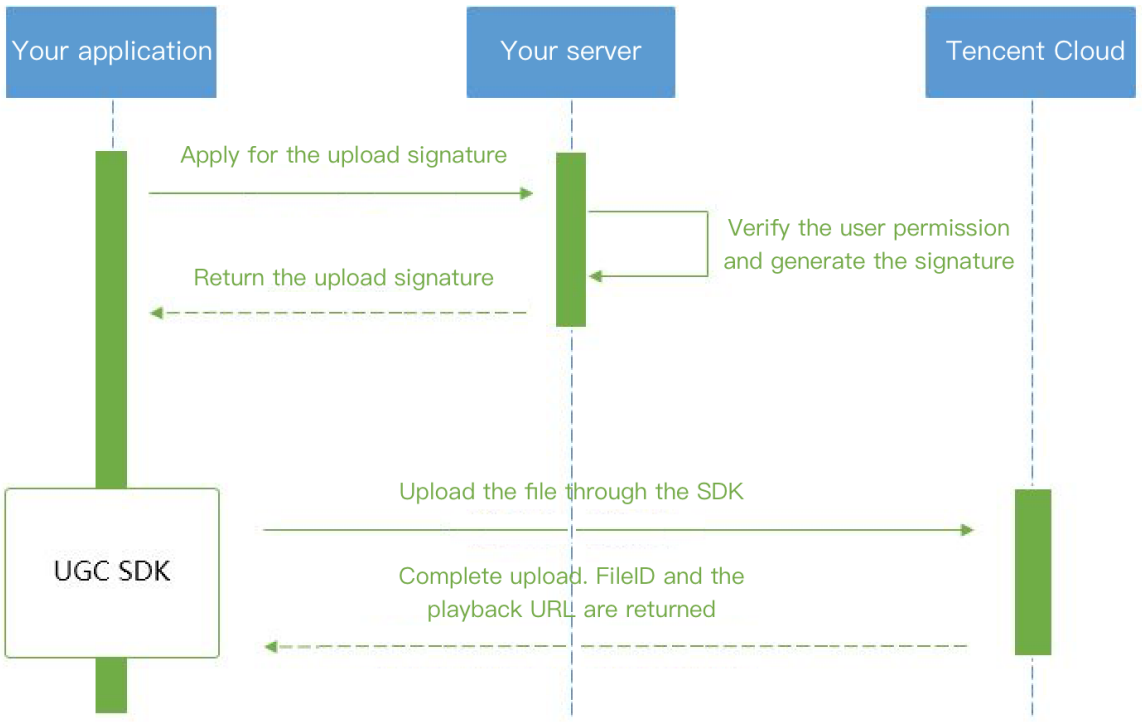
Upload and Playback Signature Distribution

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

Video Upload (iOS)

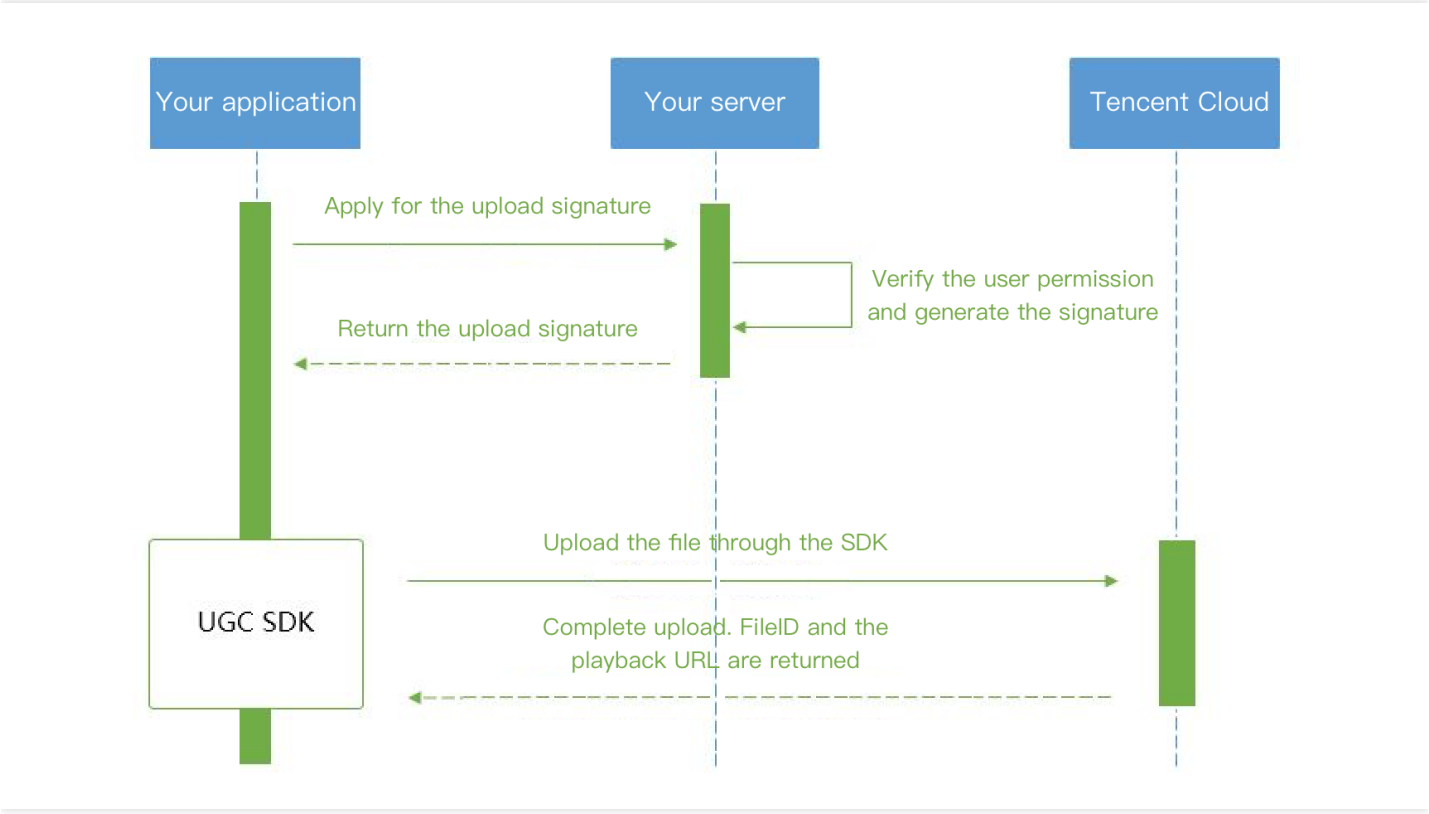
Last updated : 2020-09-22 10:06:22

Calculating Upload Signature

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

Overview

The overall process for upload from client is as follows:



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

- The client first requests an upload signature from the signature distribution service.
- The signature distribution service verifies whether the client user has the upload permission. If the verification passes, a signature will be generated and delivered; otherwise, an error code will be returned, and the upload process will end.

- After receiving the signature, the client will use the upload feature integrated in the UGSV SDK to upload the video.
- After the upload is completed, the VOD backend will send an upload completion event notification to your event notification receipt service.
- If the signature distribution service specifies a video processing task flow in the signature, the VOD service will automatically process the video accordingly after the video is uploaded. Video processing in UGSV scenarios is generally AI-based porn detection.
- After the video is processed, the VOD backend will send a task flow status change event notification to your event notification receipt service.

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

Signature generation

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

Signature distribution service implementation sample

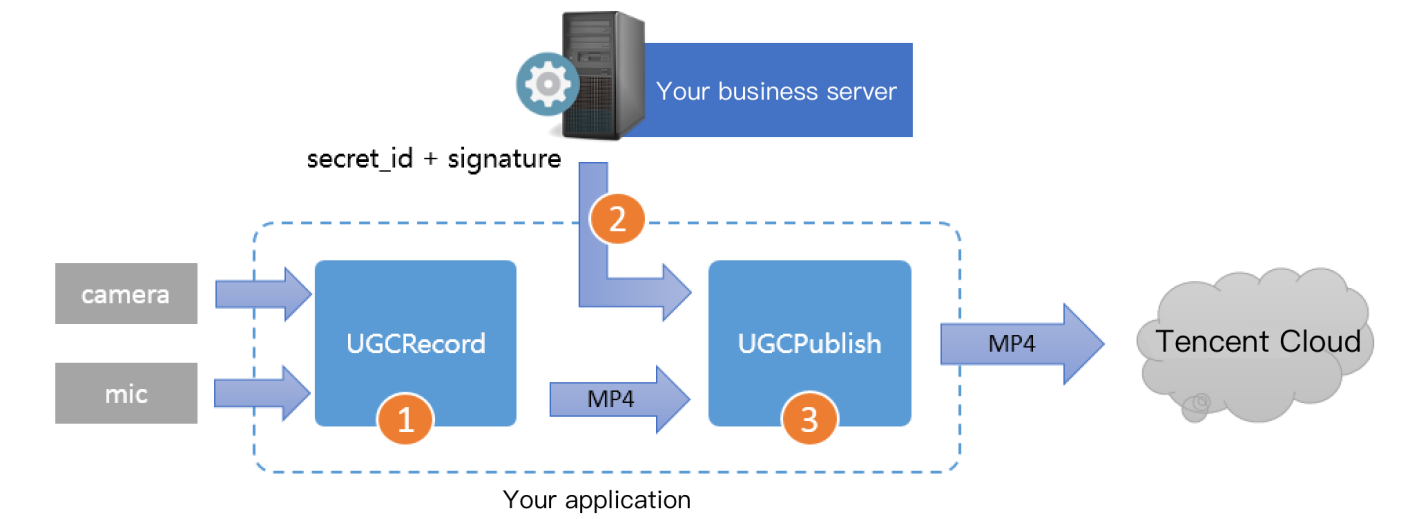
```
/**
* Calculate a signature
*/
function createFileUploadSignature({ timeStamp = 86400, procedure = '', classId = 0, oneTimeValid
= 0, sourceContext = '' }) {
// Determine the generation time and expiration time of the signature
let current = parseInt((new Date()).getTime() / 1000)
let expired = current + timeStamp; // Signature validity period: 1 day
// Enter the parameters into the parameter list
let arg_list = {
//required
secretId: this.conf.SecretId,
currentTimeStamp: current,
expireTime: expired,
random: Math.round(Math.random() * Math.pow(2, 32)),
//opts
procedure,
classId,
oneTimeValid,
sourceContext
}
// Calculate the signature
let orignal = querystring.stringify(arg_list);
let orignal_buffer = new Buffer(orignal, "utf8");
let hmac = crypto.createHmac("sha1", this.conf.SecretKey);
let hmac_buffer = hmac.update(orignal_buffer).digest();
```

```
let signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");
return signature;
}
/**
* Respond to a signature request
*/
function getUploadSignature(req, res) {
res.json({
code: 0,
message: 'ok',
data: {
signature: gVodHelper.createFileUploadSignature({})
}
});
}
```

Connection Process

Publishing short video

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



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

- Step 3. Use the `TXUGCPublish` API to publish the video. After the video is successfully published, the SDK will call back the watch URL to you.

Notes

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

Connection directions

1. Select a video

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

2. Compress the video

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

3. Publish the video

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

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

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

// Video file path generated by shoot, which can be obtained through the `onRecordComplete` callb
ack of `TXVideoRecordListener`
param.videoPath = _videoPath;
// Path of the first-frame video preview image generated by shoot. The value is the file path spe
cified by calling `startRecord`. You can also specify a path and save the `UIImage` obtained in t
```

```
he `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.

Player SDK for iOS

Last updated : 2020-08-27 11:31:45

Overview

The Superplayer SDK 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 offers advanced capabilities like instant playback on splash screen, low latency, and video thumbnail.

SDK Download

The VOD Superplayer SDK for iOS can be downloaded here.

Target Audience

This document describes Tencent Cloud's proprietary capabilities. Please make sure that you have activated the relevant Tencent Cloud services before reading it. If you haven't registered an account, please sign up first.

Quick Integration

This project supports installation through CocoaPods. You only need to add the following code to the `Podfile` :

```
pod 'SuperPlayer'
```

Run `pod install` or `pod update` .

Using player

The main class of the player is `SuperPlayerView` , and videos can be played back after it is created.

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

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

```
// Hotlink protection disabled
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329073;// Configure `AppId`
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = "5285890799710670616"; // Configure `FileId`
[_playerView playWithModel:model];
```

```
// To enable hotlink protection, you need to enter the `psign`, which is a signature for superpla
yer. For more information on the signature and how to generate it, please see https://intl.cloud.
tencent.com/document/product/266/38099
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// Configure `AppId`
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = "5285890799710173650"; // Configure `FileId`
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkI
joiNTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI6MjE0NzQ4MzY0Ny
widXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ
3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo6liXws-LzU";
[_playerView playWithModel:model];
```

Run the code and you can see that the video is played back on the phone and most of the features in the UI are available.

Selecting FileId

A video `FileId` is usually returned by the server after the video is uploaded:

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

If the file already exists in Tencent Cloud, you can go to Media Assets, find it, and view its `FileId`.

Timestamping

When lengthy videos are played back, timestamps on the progress bar can help viewers find the points of interest easily. You can add timestamps by using the `AddKeyFrameDescs.N` parameter in the ModifyMediaInfo API.

After the call is made, new elements will be displayed in the player UI.

Small window playback

A small window is a player that floats over the main window within the application. Small window playback is very simple to implement. You just need to call the following code in the appropriate position:

```
[SuperPlayerWindow sharedInstance].superPlayer = _playerView; // Set the player for small window playback
[SuperPlayerWindow sharedInstance].backController = self; // Set the returned view controller
[[SuperPlayerWindow sharedInstance] show]; // Floating display
```

Exiting playback

When the player is no longer needed, call `resetPlayer` to clear the internal state of the player and free up the memory.

```
[_playerView resetPlayer];
```

More Features

To try out the complete features, scan the QR code below to download the Tencent Video Cloud toolkit or run the project demo directly.